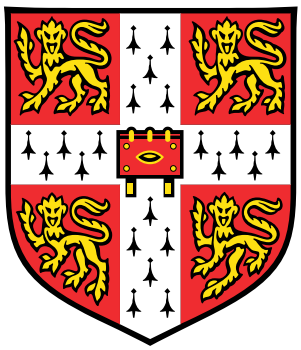
Efficient and Composable Adaptation for Cross-Lingual Transfer



Alan John Ansell

Supervisors: Prof. Anna Korhonen
Dr. Ivan Vulić
Dr. Edoardo M. Ponti

Department of Theoretical and Applied Linguistics
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

Churchill College July 2024

Declaration

This thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the preface and specified in the text. It is not substantially the same as any work that has already been submitted, or is being concurrently submitted, for any degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the preface and specified in the text. It does not exceed the prescribed word limit for the relevant Degree Committee.

Alan John Ansell
July 2024

Abstract

Parameter-efficient fine-tuning (PEFT) has emerged as a important technique for moderating the growing cost of fine-tuning state-of-the-art pre-trained language models. The modular properties of some PEFT techniques, such as reusability, composability and resistance to overfitting, lend them to applications in *cross-lingual* transfer, where data from "high-resource" *source* language(s) is employed to improve performance of NLP systems for (typically lower-resource) *target* languages. For instance, two independently trained PEFT modules, one for a specific task and one for a specific target language, can be *composed* in zero-shot fashion to perform transfer to that task-language combination.

The main goal of this thesis is to advance our knowledge of, and techniques for, PEFT and cross-lingual transfer, both independently and especially where the former is used as a tool for the latter. I pay particular attention to three aspects of this problem: (i) how best to make use of the various sources of *data* available for cross-lingual transfer; (ii) how to devise PEFT methods which deliver the best cross-lingual transfer *performance*; and (iii) how to improve the *efficiency* of PEFT both in and beyond the context of cross-lingual transfer.

The main contributions of this thesis are as follows: we first propose MAD-G, a hypernetwork method which learns to *generate* bottleneck adapters (a form of PEFT module) for specific languages from vectors of their typological features (such as word order and language family). In addition to the significant efficiency benefit over training independent adapters for single languages, this opens the possibility of generating modules for the many languages for which typological features are available but which lack any significant text data.

We then propose *sparse fine-tuning* (SFT) as an alternative PEFT method for cross-lingual transfer to address the inexpressivity of bottleneck adapters. We present Lottery-Ticket Sparse Fine-Tuning (LT-SFT), a new SFT method, and show that SFTs trained with this method can be composed to obtain better cross-lingual transfer performance than with adapters.

Next, we propose "BiStillation," a distillation-based method for creating more efficient models for specific cross-lingual transfer pairs with minimal loss of performance compared to the base multilingual model, and which outperforms multilingually-distilled models.

Whereas much work in cross-lingual transfer confines itself to considering one or two data sources, we then consider a more general class of scenarios, where a mixture of labelled, unlabelled and parallel data is available for the source-target pair(s) of interest. We show how these data sources can be incorporated into a unified training setup in a complementary manner using a range of cross-lingual transfer techniques, yielding large performance improvements for low-resource languages in our evaluation.

Finally, we address the time- and memory-inefficiency of previously proposed SFT methods, including LT-SFT, by proposing SpIEL, which preserves memory efficiency by maintaining a fixed-size set of trainable parameters which is periodically updated by replacing those which have changed the least from their pre-trained values with others which appear most likely to reduce the loss in the future. We show that SpIEL performs favourably compared to state-of-the-art PEFT methods for instruction tuning of large language models and with similar efficiency.

Acknowledgements

This journey of growth and discovery would never have begun without the invitation of my supervisor, Anna Korhonen. Thank you, Anna, for seeing potential in me despite my modest research experience and obscure origin in a far-flung corner of the map. When I first received the notification that I had been accepted to the PhD programme at the LTL, I was so excited to share the news that my legs couldn't keep up with me and I fell over on my way out of the room! I am also enormously grateful to David and Claudia Harding, whose patronage via the Harding Scholarship made it possible for me to take up this fantastic opportunity.

I would like to express my profound gratitude to, and admiration for, Ivan Vulić and Edoardo Ponti, my day-to-day supervisors, for being so generous with your knowledge, wisdom, time and support over the past four years whilst under so many other demands and accomplishing so much besides. It has been a pleasure learning the craft of research from you.

I have been blessed with a succession of wonderful mentors who influenced my life towards a much more interesting and rewarding direction than it might have otherwise taken. I would particularly like to recognise Golbon Zakeri, Margaret Atkinson and Deborah Williams, who gave me the most valuable gift a teacher can give by kindling my intellectual curiosity. I would also like to thank my Masters supervisors Bernhard Pfahringer and Felipe Bravo-Marquez for providing me with an excellent preparation for PhD study.

Thank you to my fellow LTL students for your companionship on this journey, especially regular lab co-attendees Benjamin, Songbo and Panos; Tiancheng, for all the fascinating conversations on topics as diverse as snake venom, geographically displaced peoples, and mushrooms; my doctoral twin Genie, for the therapeutic discussions on the ailments of late-stage PhD students; and Hannah and Marinela for the fruitful collaborations. It was also a pleasure to get to know and discuss research and life with the visiting students at the lab: Teemu Vahtola, Andrea Pedrotti, Chen Cecilia Liu and (briefly) Philipp Borchert and Fabian David Schmidt. I wish you all the best of luck!

I wish to thank Nigel Collier, whose efforts towards maintaining and improving the LTL's IT infrastructure have contributed greatly to the lab's and my own productivity.

I would also like to acknowledge my other co-authors, Sebastian Ruder, Goran Glavaš, and particularly Jonas Pfeiffer, whose work served as something of an inspiration in my PhD research; and Alessandro Sordoni and Lucas Caccia for your warm hospitality and engaging collaboration at Microsoft Research Montréal.

I give special thanks to the Examiners of this thesis, Andreas Vlachos and Jörg Tiedemann, whose careful reading and thoughtful feedback have brought about a significant improvement in the final product.

Lastly and above all, I wish to thank my family, Mel, Grandma, Mum and Dad, my first and greatest intellectual-curiosity-kindler; and Shibani. Your love and support have been a blessing of incomparable value in my life.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Motivation

The story of modern Natural Language Processing (NLP) has been one of increasingly large models trained on increasing amounts of data. The resulting systems have become increasingly powerful and general, at first being able to learn a specific skill only implicitly by studying a large set of examples, but later becoming better able to solve tasks specified explicitly in words rather than through example (Radford et al., 2018, 2019; Brown et al., 2020). Throughout this evolution, however, the need to *adapt* models, that is, to modify them to display certain desired behaviours, has remained constant. This need arises from the fact that language models are typically "pre-trained" on large text corpora by learning to predict the next word in a sequence, or a missing word which has been removed from the sequence. While performing these tasks accurately requires the model to learn certain transferable skills and internal representations of the world, these are latent and must be exposed before they can be useful for other applications (Devlin et al., 2019; Wei et al., 2022; Ouyang et al., 2022). Furthermore, the pre-training corpus may lack some of the required knowledge, and may not be representative of the kinds of values or behaviours we want the model to display (Askell et al., 2021).

Models are typically adapted through a process known as "fine-tuning," where some or all of the parameters are updated through further training on a dataset specific to the skills, tasks or values we want the model to acquire. However, fine-tuning is a costly process in terms of time, memory and energy. This has generated significant interest in ways of improving the efficiency of the fine-tuning process. One prominent class of such methods are *parameter-efficient* fine-tuning (PEFT) methods (Houlsby et al., 2019; Pfeiffer et al., 2023b; Lialin et al., 2023), which aim to improve training efficiency by fine-tuning only a small subset of the pre-trained model's parameters.

It is often desirable for fine-tuning to be *modular* as well as efficient, meaning that the skills or knowledge learned through fine-tuning are isolated in specific neural network components which can be reused or combined with other modular fine-tunings. Modular neural networks can support many target tasks or domains with minimal duplication of components, when an entire, independently trained network would otherwise be required for each desired application. One important application of modular fine-tuning which is a focus of this thesis is *cross-lingual transfer*, where, given training data for a task in a certain *source* language, we wish to adapt a model to perform the task in some other *target* language. Here, there are two abilities being combined: the ability to perform the task in question, and to understand (and possibly generate) the target language.

In this thesis, I investigate the most effective ways of applying parameter-efficient and modular fine-tuning in combination with various types of data to cross-lingual transfer, and propose new modular PEFT techniques and efficient methods for cross-lingual transfer.

1.1.1 Modular and Efficient Fine-Tuning

The cost of pre-training and fine-tuning neural language models through the standard approach of stochastic gradient descent with backpropagation can be formidable. The large matrix multiplications involved in computing neural functions generally require expensive hardware such as graphics processing units (GPUs) to execute efficiently. Backpropagation requires the gradient of the loss function to be computed and stored with respect to all model parameters, which further requires the storage of intermediate activations calculated during the forward pass; adaptive gradient descent optimisers such as Adam (Kingma and Ba, 2015) often store two further "state" values per parameter. It is often necessary to distribute computational and memory costs over several computational devices; for instance the recent Llama 3 family of models was trained on a cluster of 24 *thousand* GPUs (Meta AI, 2024). However, in the case of fine-tuning, there is an alternative to this dramatic scaling of resource expenditure. Since, in comparison to pre-training, fine-tuning is more focused on exposing already-learned skills and knowledge rather than learning them from scratch, it has been shown that it is possible to achieve strong fine-tuning performance by modifying only a small portion of a model's parameters; this class of techniques is known collectively as *parameter-efficient* fine-tuning (PEFT). PEFT techniques significantly reduce the memory requirement of fine-tuning, as fewer gradients, optimiser state values and intermediate representations must be stored when fewer parameters are being trained, and they also reduce the computational cost of the

backward pass, as fewer gradients must be computed. This helps to reduce the cost of fine-tuning and make it more accessible to those with fewer computational resources.

Parameter-efficient fine-tuning approaches also tend to exhibit properties of *modularity*. This means that they can be used in ways and contexts beyond the precise ones in which they were trained. For instance, Pfeiffer et al. (2021a) show how the knowledge contained in several independently trained PEFT modules can be "fused" to learn better fine-tunings for new tasks; this is enabled by the fact that this knowledge is isolated in discrete modules, whose small size allows several of them to be loaded into the model simultaneously. Ostapenko et al. (2024) and Muqeeth et al. (2024) take the idea of module reuse further, proposing means of selecting and jointly applying relevant modules for a given input on-the-fly. Another desirable property often possessed by PEFT modules is *composability*, where two independent fine-tunings can be combined in such a way that the resulting model inherits the skills and knowledge present in both without harmful "destructive interference" (Nayak et al., 2023). This composability property is especially useful in the context of *cross-lingual transfer*, where we wish to learn to perform a given task in a language for which we have few or no training examples of that task; consequently, we may wish to *compose* independent fine-tunings for the given task and language (Pfeiffer et al., 2020b; Ansell et al., 2022).

1.1.2 Cross-Lingual Transfer

There are more than 7,000 languages spoken across the world (Eberhard et al., 2024; Hammarström et al., 2024), but the distribution of usable data across these languages is far from uniform: a handful of languages have ample data whilst the majority have almost no accessible data (Joshi et al., 2020). There are three main kinds of data of interest here: *unlabelled* data, which is general-domain text from the language in question, which can be obtained from books or online sources such as Wikipedia; *labelled* data, which consists of examples of a certain language task, such as sentiment classification, where (for instance) a movie review might be accompanied by a "label" denoting that it is a positive or negative review; and *parallel* data, which consists of text in some *source* language and its translation into one or more *target* languages. Of these, unlabelled data is typically the most plentiful, as it is produced organically by literate speakers of the language in question, whereas labelled and parallel data requires deliberate effort, often by an expert linguist.

Presently, the default approach to producing an NLP system which is highly capable in a single language involves pre-training a large neural language model on an enormous corpus of unlabelled data belonging to that language, followed by fine-tuning on a set of

labelled data. It has been found that larger models trained on larger pre-training corpora require less fine-tuning data to reach the same level of performance in a given task, and at a certain scale even become capable of understanding tasks explicitly specified in words with few or no examples (Radford et al., 2017, 2018; Brown et al., 2020). Nevertheless, these larger-scale models still require labelled "instruction tuning" data to expose their general problem-solving ability and desirable attributes such as "helpfulness, honesty and harmlessness" (Askell et al., 2021).

Such data-hungry methods are infeasible for all but the aforementioned handful of "high-resource" languages; yet all hope is not lost for less data-rich languages. While different languages have different vocabularies and morphological and grammatical rules, they are all used to refer to a similar set of objects and concepts in the world which have similar relations to each other (though modulated by some degree of cross-cultural variation), and linguists generally agree that human languages share some degree of common structure (Chomsky, 1965; Croft, 2002; Evans and Levinson, 2009). Therefore a model which has primarily been exposed to one language can potentially learn knowledge which generalises to another. The act of exploiting knowledge learned from data in one language to aid performance in another is called *cross-lingual transfer*.

Perhaps the most obvious technique for cross-lingual transfer is machine translation (MT). A neural MT (NMT) system learns to translate between a given source and target language by training on parallel data for the given language pair. This enables task-specific training data to be translated into a language in which it is not already available (this is known as the *translate-train* approach to cross-lingual transfer (Conneau et al., 2020)), or alternatively, examples on which we wish to perform inference to be translated into a language where a system for the relevant task is already available (*translate-test*). While these approaches work well in some circumstances, they have several significant drawbacks: the translation step introduces an additional source of error into the pipeline; the quality of translation and thus degree of error introduced is dependent on the amount of parallel data available, which tends to be low in comparison to the amount of unlabelled data, as mentioned above; translated texts are known to suffer from "translationese," or artifacts of the translation process which induce a domain shift which may result in poorer than expected results when systems trained on translated texts are deployed on "naturally occurring" language examples (Artetxe et al., 2020a).

Unsupervised cross-lingual transfer techniques, by contrast, are those which do not make use of any explicit cross-lingual signal such as parallel data. Currently, the most potent such tool is the "massively multilingual Transformer" (MMT), which is similar to a typical Transformer but is pre-trained on a corpus consisting of text from many

languages. An MMT typically delivers much better in-language performance[1] for a given low-resource language than would be expected of a model trained on the available data of that language alone due to positive transfer from higher-resource languages. Importantly, MMTs also display impressive zero-shot cross-lingual transfer abilities: that is, an MMT fine-tuned for a given task on examples from one language can often retain a reasonable level of performance on examples from another language (Wu and Dredze, 2019; Pires et al., 2019).

However, MMTs' high degree of multilinguality is a double-edged sword: while on the one hand there can be positive transfer between languages, on the other, the more languages a model covers, the less capacity it can dedicate to each language. Conneau et al. (2020) find that MMTs deliver suboptimal cross-lingual transfer performance for high-resource languages, and that performance on low-resource languages starts to degrade beyond a certain number of pre-training languages; they term this effect the "curse of multilinguality". Pfeiffer et al. (2020b) propose learning language modules in the form of *bottleneck adapters* (Rebuffi et al., 2017; Houlsby et al., 2019) through continued pre-training to recover performance lost on a specific language due to the curse of multilinguality, or to adapt the model to a new language unseen during pre-training. Bottleneck adapters are lightweight layers consisting of a shallow multi-layer perceptron inserted into each Transformer block which are trained while the remaining model parameters are frozen. Pfeiffer et al. (2020b) show that their language adapters can be composed with task-specific adapters to perform zero-shot cross-lingual transfer, terming this approach "MAD-X".

PEFT modules are a natural choice for language- and task-specific adaptation in cross-lingual transfer for several reasons. We are often interested in supporting multiple tasks and languages. PEFT-based training keeps the time and memory requirements of developing and running such a system manageable, with the composability property allowing a single module to be trained for each task and language when a separate model for each (task, language) pair would otherwise be required. Another benefit is that the limited capacity of PEFT modules reduces the tendency of knowledge learned during pre-training to be "forgotten" during fine-tuning, which might be especially hazardous during cross-lingual transfer when specialising to the target language at the possible expense of proficiency in the source language. Furthermore, this limited capacity makes it harder to overfit to the fine-tuning data, which is more likely to occur on small datasets such as those which are typical for low-resource languages. Since the target language module is typically applied at inference time but not during training, parameter-efficiency

[1]That is, performance when the model is fine-tuned and evaluated on data from the same language.

also reduces the mismatch between the training- and inference-time model variants, which may help to preserve cross-lingual alignment and prevent interference between the language and task adaptations.

1.2 Thesis Outline

1.2.1 Aims

The aim of my PhD research has been to advance our understanding of and techniques for modular and parameter-efficient fine-tuning and cross-lingual transfer, both separately and especially where the former is used as a tool for the latter. I take Pfeiffer et al. (2020b)'s modular, adapter-based "MAD-X" framework for cross-lingual transfer as an inspiration and point of departure, and investigate the following research questions (RQs):

1. How can cross-lingual transfer benefit from additional sources of data not utilised by the zero-shot MAD-X approach, including typological data, parallel data, and "few-shot" task data in the target language?

2. How can bottleneck adapters be improved on as a PEFT method for learning composable language- and task-specific modules?

3. How can we improve the efficiency of cross-lingual transfer methods, including PEFT-based methods?

1.2.2 Overview

The remaining chapters of this thesis can be summarised as follows:

- Chapter 2 reviews the background necessary to understand the subsequent chapters and place them in context. The areas covered are the evolution of transfer learning in NLP; transfer learning with parameter-efficient and modular fine-tuning; and multilinguality in NLP and cross-lingual transfer specifically.

- Chapter 3 addresses RQ1 from above, and in particular how language typology data (containing information about a language's syntactic, phonological and phonetic features) can be used to improve cross-lingual transfer. We propose "MAD-G", where a hypernetwork is trained to *generate* a bottleneck adapter for a given target language from a vector of its typological features. This opens up the possibility of adaptation to target languages without unlabelled data. We also find that

MAD-G requires far less training per target language than independent MAD-X style adapters, touching on RQ3 on efficiency.

- Chapter 4 addresses RQ2. We identify a lack of expressivity and loss of inference-time efficiency as drawbacks of bottleneck adapters. We propose *sparse fine-tunings* (SFTs), where a small, strategically selected subset of the base model's parameters are fine-tuned, as an alternative composable PEFT method for cross-lingual transfer. We introduce Lottery Ticket Sparse Fine-Tuning (LT-SFT), a novel sparse fine-tuning method inspired by the Lottery Ticket Hypothesis (Frankle and Carbin, 2019).

- Chapter 5 addresses RQ3. We note the inherent parameter- and compute-inefficiency of massively multilingual Transformers for cross-lingual transfer in the common scenario where we are only interested in one target language. We show that by distilling smaller *bilingual* models covering just the source and target language of interest, we can achieve a similar level of performance to the base MMT with much better inference efficiency. We build on the techniques developed in Chapter 4, employing LT-SFT during both language- and task-specific distillation processes.

- Chapter 6 returns to RQ1. We propose a framework for exploiting any mixture of labelled, unlabelled and parallel data for cross-lingual transfer, with a focus on maximising performance for low-resource languages.

- Chapter 7 again addresses RQ3. While sparse fine-tuning has the advantage of expressivity and inference-time efficiency, the Lottery Ticket method and other previously proposed methods for learning SFTs are costly in terms of training time and memory requirement. We propose SpIEL, which is competitive with LoRA (Hu et al., 2022, the most established PEFT method at the time of writing) in terms of training efficiency, and show that it outperforms LoRA in most of our instruction tuning experiments. We leave cross-lingual transfer experiments with SpIEL to future work.

- Chapter 8 summarises the findings of the thesis, draws general conclusions and proposes future research directions for PEFT and cross-lingual transfer both independently and together.

1.3 Publications

Chapters 3-7 (and to a lesser extent 1, 2 and 8) are based on the following publications:

- Chapter 3: **MAD-G: Multilingual Adapter Generation for Efficient Cross-Lingual Transfer** (Ansell et al., 2021). Alan Ansell, Edoardo M. Ponti, Jonas Pfeiffer, Sebastian Ruder, Goran Glavaš, Ivan Vulić, and Anna Korhonen. In Findings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP 2021), Nov 2021. I wrote the code and ran the experiments and analysis for this work, and was the primary writer of the paper with assistance from the other co-authors.

- **Composable Sparse Fine-Tuning for Cross-Lingual Transfer** (Ansell et al., 2022). Alan Ansell, Edoardo M. Ponti, Anna Korhonen and Ivan Vulić. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL 2022), May 2022. I wrote the code and ran the experiments and analysis for this work, and was the primary writer of the paper with assistance from the other co-authors.

- **Distilling Efficient Language-Specific Models for Cross-Lingual Transfer** (Ansell et al., 2023b). Alan Ansell, Edoardo M. Ponti, Anna Korhonen and Ivan Vulić. In Findings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL 2023), July 2023. I wrote the code and ran the experiments and analysis for this work, and was the primary writer of the paper with assistance from the other co-authors.

- **Unifying Cross-Lingual Transfer across Scenarios of Resource Scarcity** (Ansell et al., 2023a). Alan Ansell*, Marinela Parović*, Ivan Vulić, Anna Korhonen and Edoardo M. Ponti. In Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP 2023), Dec 2023. Marinela and I made an equal contribution to the code, experiments and analysis and were the primary writers of the paper with assistance from the other co-authors.

- **Scaling Sparse Fine-Tuning to Large Language Models** (Ansell et al., 2024). Jan 2024. Alan Ansell, Ivan Vulić, Hannah Sterz, Anna Korhonen and Edoardo M. Ponti. I assumed the primary role in writing code and running experiments, with significant assistance from Edoardo and additional assistance from Hannah. I also took the primary role in writing the paper with significant assistance from the other co-authors.

Chapter 2

Background

Over the course of my PhD study, NLP technology has undergone transformational change. In this chapter, I divide the relevant background into three sections, with each section organised chronologically to help place each of my research projects into its proper temporal context. The sections are "Transfer Learning in NLP" (§2.1), in which I describe the evolution of fundamental models for NLP and their adaptation to specific tasks and domains; "Parameter-Efficient and Modular Fine-Tuning" (§2.2), in which I discuss adaptation methods which are more efficient and enable the combination of more than one facet of knowledge; and "Multilinguality and Cross-Lingual Transfer," in which I introduce the field of multilingual NLP.

2.1 Transfer Learning in NLP

Machine learning algorithms aim to derive generalisable knowledge or skills from a finite set of observations and apply them to novel situations. Traditionally, NLP practitioners aimed to create a system for one specific task; typical examples would be part-of-speech tagging, parsing, sentiment analysis or question answering, among many others. Creating a dataset for such a task is a major undertaking requiring careful design and the laborious exercise of data gathering and annotation (Marcus et al., 1993; Socher et al., 2013; Rajpurkar et al., 2016). However, a model trained on even a large task-specific dataset but without any other exposure to language would struggle to differentiate between signal and noise. Consider for instance a dataset consisting of sentences with binary sentiment labels, that is, labels denoting each sentence as having positive or negative sentiment. Each label provides only a single bit of information which the model may use to extract generalisable sentiment-related features from a sea of other irrelevant linguistic features and learn how they combine to form a sentence's overall sentiment.

One way of dealing with this problem is to "guide" the model by using human knowledge to craft features relevant to the task. For instance, one of the inputs to a model for named entity recognition might be whether a given word begins with a capital letter, as capitalised words are more likely to be named entities. High-performance systems might use hundreds of such hand-crafted features (Curran and Clark, 2003; Nadeau and Sekine, 2007). However, this approach has become a casualty of the "bitter lesson of AI research" (Sutton, 2019). The factors which influence humans' linguistic judgements are complex and subtle and not easily reduced to a fixed set of categorical features. By attempting to do so, the model's designer provides it with a signal that is impoverished, even if its "information density" is higher. Furthermore, features engineered for one task may be far less useful for another: word capitalisation probably tells us very little about a sentence's sentiment for example. Therefore, the enormous effort required to optimise the feature set for a model might yield very little benefit outside the narrow task and domain for which it is intended.

Transfer learning, on the other hand, seeks to alleviate the problem of insufficient data by removing the assumption that we must learn only from data belonging to the *target* distribution, that is, the distribution we are ultimately interested in drawing inferences on. To borrow Pan and Yang (2010)'s formalisation, transfer learning aims to use knowledge obtained from a *source* domain and task $\mathcal{D}_S$ and $\mathcal{T}_S$ to improve the learning of predictive function $f_T(\cdot)$ for *target* task $\mathcal{T}_T$ on domain $\mathcal{D}_T$. Here, a domain $\mathcal{D} = \{\mathcal{X}, p(x)\}$ consists of a *feature space* of possible inputs $\mathcal{X}$, and a probability distribution over the feature space $p(x)$. A task $\mathcal{T} = \{\mathcal{Y}, p(y|x)\}$ consists of a *label space* $\mathcal{Y}$ and a conditional probability distribution $p(y|x)$. The underlying probability distributions $p(x)$ and $p(y|x)$ are both unobservable; we must learn a function $f(x, y)$ to estimate $p(y|x)$ given a training set consisting of pairs $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$.

Transfer learning scenarios can be categorised according to the relationships between the source and target domains and tasks. The case where the source and target domains share the same feature space $\mathcal{X}$ but have a different distribution $p(x)$ over inputs is often referred to as *domain adaptation* (Ruder, 2017); for instance, transfer from sentiment classification on film reviews to sentiment classification on product reviews. Some more distant transfer scenarios involve differing feature spaces $\mathcal{X}_S$ and $\mathcal{X}_T$; a prominent example is *cross-lingual transfer*, where the source and target feature spaces consist of text from different languages. On the other hand, we sometimes wish to employ data from one task to improve performance on another. *Multi-task learning*, for instance, aims to exploit the commonalities between several tasks by training on their data simultaneously (Caruana, 1997). However, the most influential concept in task transfer learning is likely that of

pre-training, where a model is first trained to perform a very general task for which data is plentiful, and is then taken as a starting point for specialisation to a more specific, less data-rich task. The first instance of this approach achieving widespread success was in computer vision, where it was found that deep convolutional neural networks (Krizhevsky et al., 2012) trained on the ImageNet (Deng et al., 2009) image classification dataset yield representations that are highly transferable to other domains and computer vision tasks (Razavian et al., 2014), or that they can be fine-tuned to perform such tasks at an impressive level of performance (Yosinski et al., 2014).

2.1.1 Word Embeddings

While the "ImageNet moment" of transfer learning breakthrough in NLP is not generally considered to have arrived until around 2018 (Ruder, 2018), pre-training in NLP has an extensive history. Much of this work was motivated by two NLP-specific challenges:

1. Language (in its written form) consists of sequences of discrete words belonging to a high-dimensional vocabulary, and the relationship between words and their meanings is largely arbitrary. Thus the training signal for any one word is sparse and not straightforwardly transferable to other words, even closely related ones.

2. Unlike computer vision, where models pre-trained on image classification tasks seem to learn sufficiently general representations to transfer well to many other tasks, NLP lacks a canonical supervised pre-training task. This can perhaps be attributed to the fact that language has multiple layers (syntactic, semantic, pragmatic) which operate at differing levels of granularity (word, sentence and document level), and most tasks target a small subset of these layers and levels. Part-of-speech tagging, for instance, is a word-level syntactic task, whereas sentiment analysis is a sentence- or document-level semantic task.

The solution to the first of these problems was found in *word embeddings*, also known as *word vectors*. The naive, "sparse" vector representation of the ith word of a vocabulary V consists of a single 1 at position i in a $|V|$-dimensional vector consisting otherwise of zeros, so that every word's vector is orthogonal to every other's. A *dense* word vector, or embedding, on the other hand, is a vector of real numbers of dimension $d \ll |V|$, with the intuition that the distance between two words' embeddings in the vector space corresponds to their degree of semantic or syntactic relatedness. Reducing the dimensionality of a word's representation from $|V|$ to d may be beneficial from the point of view of computational efficiency, but more importantly, it enables better generalisation through

transfer between related words. To paraphrase Bengio et al. (2000), "*If we knew that* `dog` *and* `cat` *played similar roles (semantically and syntactically), and similarly for (*`the`*,* `a`*), (*`bedroom`*,* `room`*), (*`is`*,* `was`*), (*`running`*,* `walking`*), we could naturally generalise (i.e. transfer probability mass) from*

	`The cat is walking in the bedroom`
to	`A dog was running in a room`
and likewise to	`The cat is running in a room`
	`A dog is walking in a bedroom`
	`The dog was walking in the room`

and many other combinations."

But how can one learn an effective set of word vectors in the first place? In view of the aforementioned lack of a suitable general-purpose supervised task, NLP research turned to unsupervised learning methods dependent only on unlabelled data, i.e. plain text, which unlike data annotated by expert linguists, is readily available in great quantity and diversity on the internet. Unsupervised word embedding approaches were largely inspired by the observation in distributional semantics that words which often occur near to each other tend to be related in some way, a principle summarised by Firth (1957) in the statement "*You shall know a word by the company it keeps.*"

A prominent example of a distributional word embedding learning algorithm is `word2vec` (Mikolov et al., 2013c), which was the first to achieve widespread adoption across many tasks. Each word is assigned an "input" and an "output" vector. The training examples are drawn from a large corpus and consist of a center word and its "context", the $c = 5$ words appearing on either side of the center. The objective is to maximise the estimated probability of the center word occurring given the context. The conditional probability of the center word occurring is calculated for each context word independently. To avoid the computational expense of computing a full probability distribution over the vocabulary, `word2vec`'s *negative sampling* approach substitutes these conditional probabilities with scores obtained from the dot product of the input vector of the center word and the output vector of the context word. It simultaneously maximises the scores for words that did co-occur in the corpus and minimises the scores for "fake" context words sampled from the full vocabulary. The popular `GloVe` method (Pennington et al., 2014), on the other hand, explicitly calculates a "word-word co-occurrence matrix," containing the frequency with which every pair of words co-occurs in the corpus. It computes word vectors by efficiently factorising this matrix. The impressive efficiency of `word2vec` and `GloVe` enabled them to learn superior word representations by leveraging multi-billion word corpora at relatively modest computational cost.

Pre-trained embeddings were typically used as inputs to neural models with architectures tailored specifically to a given target task, unlocking significant improvements across many areas of NLP (Chen and Manning, 2014; Seo et al., 2016; Wang et al., 2016; Lee et al., 2017). Ma and Hovy (2016) for instance find that initialising the embedding layer of their end-to-end neural sequence labelling model with `GloVe` (Pennington et al., 2014) vectors improves the F1 score they obtain on a named entity recognition (NER) task by more than 10 points. Such results demonstrate the promise of transfer learning for NLP by leveraging the information contained in unannotated text. However, pre-trained word vectors fall far from exploiting the full potential of this approach. `word2vec` and `GloVe` adopt a notion of context which treats the words it contains independently and without respect to their order, and the final embeddings it learns are fixed and insensitive to the context in which they appear. Such a system has no opportunity to learn linguistic phenomena above the level of lexical semantics. To achieve the goal of maximising the generalisable knowledge learned from unannotated text, we require an architecture capable of modelling the interactions between words in sequences of arbitrary length, and a pre-training task of sufficient difficulty and generality to require the model to understand a wide range of linguistic phenomena, and even acquire world knowledge.

2.1.2 Pre-Trained Language Models

Language modelling (LM), where the model is required to predict the next word in a sequence of words, has emerged as the primary task for pre-training of NLP models. It can be considered a *self-supervised* task, because it involves predicting explicit labels, but these labels are produced procedurally from unannotated data rather than manually by humans. One of the earliest experiments in transfer learning with language model pre-training was by Collobert and Weston (2008), who pre-train a Time-Delay Neural Network (Waibel et al., 1989) on a simplified language modelling task where the model is required to discriminate between real and "fake" words in a sentence. Specifically, the model is supplied with a fixed-length sequence of words whose middle word has been replaced with a random word with 50% probability; the label is positive if the middle word is the original and negative if it has been replaced. Unlike the distributional embeddings learning methods discussed above, Collobert and Weston's neural architecture and training task take into account the order of words in the context. However, they transfer only the embeddings learned during the self-supervised pre-training to the main target task, semantic role labelling (SRL), as they use a slightly different architecture for SRL than for LM. Nevertheless, they find that using the embeddings pre-trained with

the LM objective significantly reduces the downstream SRL error rate compared to using randomly initialised embeddings.

While there was some successful exploration of LM-based pre-training in the intervening years (e.g. Dai and Le, 2015; Radford et al., 2017), the "ImageNet moment" (Ruder, 2018) of transfer learning in NLP arrived in 2018 with the release of a succession of models which fully embraced the pre-training/fine-tuning paradigm, notably ULMFit (Howard and Ruder, 2018), ELMo (Peters et al., 2018), GPT (Radford et al., 2018) and BERT (Devlin et al., 2019). These models were successful for several reasons. Their flexible and powerful architectures, based on either the LSTM gated recurrent neural network (Hochreiter and Schmidhuber, 1997) or the Transformer (Vaswani et al., 2017) are capable of modelling long-range dependencies over arbitrarily long sequences, and the same architecture can accommodate both LM and a wide range of downstream tasks with minimal modification, typically requiring only the addition of a task-specific classification head. This enables the high-level information learned during pre-training such as sentence-level syntax and semantics to be transferred to the downstream task either by directly utilising the representations learned across the model layers as in ELMo, or by exposing the model's latent knowledge through fine-tuning its parameters, as in ULMFit, GPT and BERT. These models also benefited from greater scale than their predecessors, both in terms of capacity as measured by number of parameters and amount of data seen during training. GPT and BERT, the most powerful of this first generation of pre-trained models, achieved sizable performance gains across almost every task in NLP. Howard and Ruder (2018) point out that under their experimental settings, ULMFiT with only 5% of the training data could deliver a similar level of performance to a non-pre-trained model using the entire training set.

Another contributor to GPT and BERT's success was their use of the new Transformer architecture (Vaswani et al., 2017). The Transformer has at least two significant advantages over the previously predominant recurrent neural network (RNN) variants such as the LSTM. Whereas RNNs sequentially compute representations for each word in a sequence, the Transformer's self-attention mechanism allows it to pass information between every pair of words at each layer. This enables better parallelisation for greater computational efficiency during model execution and aids learning long-range dependencies by reducing the length of the computational path between distant words in the sequence (Hochreiter et al., 2001). I discuss the Transformer architecture in detail in §2.1.5.

2.1.3 In-Context Learning

In contrast with non-pre-trained, task specific models whose performance was typically limited by the availability of annotated training data, pre-trained Transformers exhibit impressive scaling with respect to model size and pre-training duration. Kaplan et al. (2020) showed that Transformer language modelling performance has a power law relationship to model and dataset size and training compute which holds over at least six orders of magnitude on the efficient frontier. This is reflected in the performance of pre-trained Transformers on downstream tasks. For instance, Raffel et al. (2020) consistently find significant improvements in performance of their T5 models across a wide range of tasks at each of 4 increments in size from 220 million to 11 billion parameters.

In addition to exhibiting improved performance after task-specific fine-tuning, larger pre-trained language models appear to gain qualitatively different and more powerful transfer abilities. In particular, larger models are increasingly capable of performing tasks which they have not been explicitly trained for. Radford et al. (2019) demonstrate that their 1.5 billion parameter GPT-2 model achieves a reasonable level of performance on several common tasks without using any task-specific training data. They elicit the desired behaviour for a given task by presenting test examples to the model in a task-specific natural language template. For instance, for news article summarisation, the text "`TL;DR:`"[1] is appended to the article, and the model is used to predict the subsequent words; for English-to-French translation, the model input consists of a sequence of English-French translation pairs in the form "`english sentence = french sentence`" followed by "`english sentence =` ", where the final `english sentence` is the one we actually want to translate. Using the model input itself to specify the task in this manner is known as "in-context learning."

While in-context learning was originally far inferior to fine-tuning performance-wise in most cases, further scaling allowed few-shot in-context prompting performance to make significant progress toward closing the gap with the fine-tuning state-of-the-art for many tasks (Brown et al., 2020). The success of in-context learning implies that large language models (LLMs) learn many of the high-level concepts needed to solve typical NLP tasks despite not receiving any explicit instruction in these concepts; rather, they learn these concepts implicitly because they happen to be useful for the next word prediction task on which the model *is* explicitly trained. Larger, more thoroughly trained models seem to be capable of learning increasingly abstract, implicit concepts. For instance, with zero-shot prompting the 175-billion parameter GPT-3 model (Brown et al., 2020) achieves 91%

[1] Common internet slang for "Too long, didn't read", typically followed by a summary of the preceding text.

accuracy on the Choice of Plausible Alternatives (COPA; Roemmele et al., 2011) dataset, which requires understanding of cause and effect in real-world scenarios; their models also become capable of simple arithmetic at this scale. Remarkably, some advanced models can even play chess to a reasonable standard without explicit instruction (Carlini, 2023), presumably because their training data includes a large number of notated chess games, and they learn some form of internal model of the game in order to be able to accurately predict next moves.

2.1.4 Instruction Tuning

Although LLMs display impressive zero- and few-shot in-context learning abilities, they are not optimised for usefulness or correctness in their raw, pre-trained form. Rather, they are trained to produce output which simulates their training data, which may differ greatly from the output we would like an NLP system to produce. Consider the example of chess: the training corpus likely contains chess games of a wide variety of play quality. To be able to make accurate next move predictions, the LLM will not only need to model the game situation, but also the style and strength of the players. Therefore, while the model may learn the capability to play chess well, it is unlikely to display this ability unless the context seems to call for it. Carlini (2023) found that the strength of an LLM-based chess agent could be modulated by changing the names of the players in the header of the game record passed as input to the model; the best performance was obtained when the game was represented as being between two world champion players.

In order to elicit the desired "personality" (Askell et al. (2021) propose "helpful, honest and harmless" as ideal attributes) from the LLM, a supervised fine-tuning step is often employed after pre-training. However, since we are aiming to produce a model which excels at in-context learning, or to use a more intuitive term, *instruction following* on as broad a range of instructions as possible, *instruction tuning* datasets tend to prioritise diversity of task types to promote generalisation to novel instructions. This stands in contrast to the pre-training/fine-tuning paradigm we have considered so far, where the fine-tuning step serves to *specialise* the model for a particular task.

2.1.5 The Transformer Architecture

The Transformer architecture was originally envisioned by Vaswani et al. (2017) as a sequence-to-sequence model for machine translation. It consists of an *encoder*, which encodes a sequence of tokens into a corresponding sequence of vector representations, and a *decoder*, which, given the encoder output and the tokens generated so far, predicts

a probability distribution over the next token in the sequence. The distinctive feature of the Transformer is that it relies entirely on *attention* mechanisms to model interactions between different positions in the sequence.

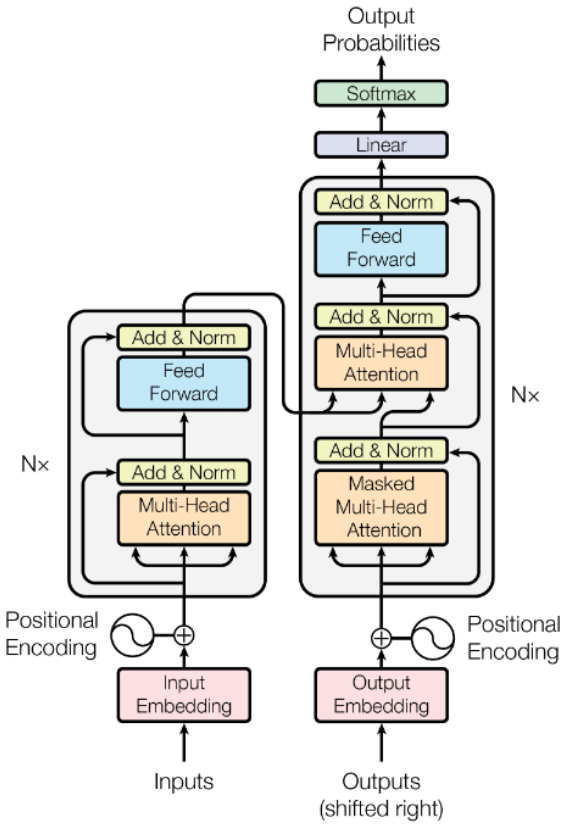


Fig. 2.1 The arrangement of components of a Transformer encoder (left) and decoder (right). Reproduced from Vaswani et al. (2017).

A Transformer encoder or decoder consists of several blocks, each containing an attention component followed by a dense feed-forward component. The attention component of the encoder employs only self-attention, where the representations output by the previous layer interact with each other. The decoder additionally employs cross-attention, where the representations output by the decoder's self-attention interact with the final representations produced by the encoder. To enforce the sequential nature of the output and avoid "information leakage," the decoder's self-attention is masked to prevent it from "seeing" later tokens in the sequence than the one it is currently predicting during training. Figure 2.1 illustrates the arrangement of components within Transformer blocks and the interaction between the encoder and decoder layers.

The specific variety of attention used in Vaswani et al. (2017)'s Transformer formulation is known as *scaled dot-product attention*. Each representation in the "source"

sequence is considered to pose a "query" in the form of a vector to the "target" sequence, which is the source sequence itself in the case of self-attention, or the encoder sequence in the case of cross-attention. For each position in the target sequence, there is a "key" vector and corresponding "value" vector. For a source and target sequence of lengths m and n respectively, the query and key vectors of dimension d_q and the value vectors of dimension d_k can be stacked to form matrices $Q \in \mathbb{R}^{m \times d_k}$, $K \in \mathbb{R}^{n \times d_k}$ and $V \in \mathbb{R}^{n \times d_v}$. The attention score between a given query and key is defined to be the dot product of their vectors, thus the attention score for all pairs can be calculated as $QK^\top$. These are then scaled by dividing by $\sqrt{d_k}$ and normalised to a probability distribution over keys for each query using the softmax function. The output of the attention calculation is a linear combination of the values weighted by the distribution of the corresponding keys:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V. \tag{2.1}$$

However, instead of having a single set of queries, keys and values, Vaswani et al. (2017) pioneer "multi-head attention," where there are several independent sets of queries, keys and values, each of which feeds into its own scaled dot-product attention calculation, the results of which are concatenated and combined with a linear layer. This allows the attention mechanism to attend to several different relevant features of the sequence simultaneously.

$$\text{MultiHead}(S, T) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \tag{2.2}$$

$$\text{where head}_i = \text{Attention}(SW_i^Q, TW_i^K, TW_i^V), \tag{2.3}$$

where $S \in \mathbb{R}^{m \times d_{\text{model}}}, T \in \mathbb{R}^{n \times d_{\text{model}}}$ are the source and target sequence representations, $W_i^Q, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$ and $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ project the raw representations to their queries, keys and values respectively, h is the number of attention heads and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

A Transformer block consists of a self-attention block, followed by a cross-attention block (decoder only), followed by a feed-forward layer. Each of these sub-components has a skip connection and is followed by a layer normalisation (Ba et al., 2016). The feed-forward layer consists of two sequential linear layers with a non-linearity:

$$\text{FFN}(\boldsymbol{x}) = W_2\alpha(W_1\boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2. \tag{2.4}$$

In Vaswani et al. (2017)'s original formulation, the non-linear activation function α is the rectified linear unit (ReLU; Fukushima, 1975). Later Transformer variants have a

variety of alternative activation functions, such as GeLU (Hendrycks and Gimpel, 2016) in BERT and GPT (among others). More recently, variants of the Gated Linear Unit (Dauphin et al., 2017) have been popular. For instance, the SwiGLU FFN (Shazeer, 2020)

$$\text{SwiGLU}(\boldsymbol{x}, W, V, W_2) = W_2(\text{Swish}(W\boldsymbol{x}) \odot V\boldsymbol{x}), \tag{2.5}$$

where $\text{Swish}(\boldsymbol{x}) = \boldsymbol{x} \odot \sigma(\boldsymbol{x})$, has been used by LLaMA 1 and 2 (Touvron et al., 2023a,b) and Mistral (Jiang et al., 2023).

The input representations to the first encoder and decoder layer are taken from an embedding matrix $E_{\text{in}} \in \mathbb{R}^{V \times d_{\text{model}}}$. Because the Transformer's attention mechanism does not explicitly model the order of the tokens in the sequence, positional information is often incorporated by adding a position-specific embedding to the representation of each token. Vaswani et al. (2017)'s Transformer implementation uses *sinusoidal* positional embeddings, where the positional embedding for position *pos* is given by

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \tag{2.6}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right). \tag{2.7}$$

Vaswani et al. (2017) point out that this choice of positional embeddings assists the model in "learning to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}". Some Transformer models such as BERT have opted for fully learnable position embeddings for each position. More recently *rotary position embeddings* (Su et al., 2023), which incorporate relative positional information in the cross-attention mechanism itself at every Transformer block have become commonplace.

While originally envisioned as a sequence-to-sequence architecture, the encoder and decoder components of the Transformer are often used independently. BERT, RoBERTa (Liu et al., 2019) and ELECTRA (Clark et al., 2020b) are examples of encoder-only models. They produce token-level representations which can be used for lexical tasks such as part-of-speech tagging and named entity recognition, as well as a pooled sequence-level representation, which can be used for sentence-level tasks such as sentiment analysis. Since Transformer encoders do not employ masked attention, they cannot be pre-trained with the standard language modelling objective. BERT for instance uses a *masked* language modelling objective, where 15% of the input tokens are replaced at random with either a special `[MASK]` token, a random token or the original token. The model

must then predict the masked tokens. One benefit of masked language modelling is that the model can learn from bidirectional context, whereas in standard language modelling only left-side context is available for prediction. However, since only 15% of tokens can be used for prediction, masked language modelling may be less sample efficient. ELECTRA circumvents this problem by framing pre-training as a discrimination problem. It simultaneously trains a small "generator" encoder with a masked language modelling objective and a discriminator model which must discern whether each token in the input is genuine or has been replaced by a "fake" token sampled from the generator. This enables the model to learn from a prediction on every token.

Unsupervised pre-training for sequence-to-sequence Transformers (those which have both an encoder and a decoder) typically employs a *denoising* objective, where the input sequence is corrupted in some way and the decoder must reconstruct its original form. Lewis et al. (2020a), for instance, pre-train their BART model by replacing randomly selected spans of the input with a special token and requiring the model to predict the entire input sequence including the masked-out spans. Raffel et al. (2020) propose a more efficient variant of this objective for pre-training their T5 models, where the ith masked span in the input sequence is replaced by a special "sentinel" token specific to the ith position, and the decoder is required to output only the masked spans themselves, separated by the corresponding sentinel tokens. Their generative abilities allowed sequence-to-sequence pre-trained Transformers to excel on tasks which require a sentence or document to be conceptually transformed into another, such as summarisation and language translation.

Most recent large language models are decoder-only models (Brown et al., 2020; Touvron et al., 2023a,b; Jiang et al., 2023, among many others). Since sequence-to-sequence tasks can generally be recast as causal language modelling tasks by providing the input as a prompt to the language model (i.e. a prefix to the decoder output, perhaps with some additional framing words or tokens), the lack of an encoder is not much of a limitation in practice. Wang et al. (2022b) systematically compare zero-shot generalisation of encoder-decoder and decoder-only models, finding no advantage for encoder-decoder models when decoder-only models are pre-trained with both causal and masked language modelling; decoder-only models may therefore have become favoured due to their greater simplicity.

2.2 Parameter-Efficient and Modular Fine-Tuning

2.2.1 Efficient Fine-Tuning

While it has evolved from a technique for specialising pre-trained models for a specific task to a more general-purpose means of optimising in-context learning performance through instruction tuning, fine-tuning remains an essential component of the NLP pipeline. The default, "full" fine-tuning approach is similar to pre-training, that is, to update all model parameters using some variant of stochastic gradient descent over mini-batches of the training data. While this allows the model's full capacity to be applied to learning the target task, it has several drawbacks:

- It is **slow**: training large neural networks is computationally intensive and time-consuming. Part of the appeal of the pre-training/fine-tuning paradigm is that those who lack the resources to pre-train language models themselves can still apply them to their problems through fine-tuning. Much of this strength is negated if even fine-tuning is prohibitively slow.
- It is **parameter-** and **memory-inefficient**: the amount of GPU memory required during full fine-tuning is also similar to the requirement for pre-training. This may make it inaccessible to those who do not have multiple high-end GPUs at their disposal, especially for the most recent multi-billion parameter LLMs. The amount of permanent storage required for LLM fine-tunings and the time needed to load them may also be onerous, especially when there is a separate fine-tuning for each of multiple target tasks.
- It is **prone to overfitting**: considering that fine-tuning datasets tend to be orders of magnitude smaller than pre-training datasets, large models may overfit on their fine-tuning data rather quickly.

Aspects of Efficiency

There are several salient aspects of fine-tuning efficiency:

1. Time/computation efficiency: this can be measured in many ways. For instance, raw computational cost could be measured in floating-point operations (FLOPs) per token - though note that since the computational complexity of Transformer execution is proportional to the square of the sequence length, this value is dependent on the sequence length. FLOP-based cost correlates imperfectly with time-based cost (e.g. seconds required to process a batch of a certain size and sequence

length), which is hardware-dependent. For instance, a sparsity-based fine-tuning method which on paper requires far fewer FLOPs than a dense method may not in fact execute any faster on typical hardware which has limited support for sparse operations (Hooker, 2021). A third consideration is energy usage, which again is hardware-dependent and imperfectly correlated with FLOPs and time consumption. Schoonhoven et al. (2022), for example, suggest that capping a GPU's power limit may improve energy efficiency despite slowing processing down. In this thesis, I focus on time efficiency, as this is typically the most practically important of these considerations in a small-to-medium scale setting.

2. Memory efficiency: the form of memory which is most often a bottleneck for deep learning performance is GPU RAM. However, in the case of fine-tuning, the amount of memory it takes to store the parameter updates learned during the training procedure may also matter, as this affects the amount of time required to load the fine-tuning from disk or CPU RAM to the GPU. While this is usually a one-off cost, in certain cases where a system must support many different fine-tunings simultaneously but there is insufficient space to store all of them in GPU RAM, it may be necessary to frequently load and unload fine-tunings. In this work, I mostly treat memory efficiency as secondary to time- and parameter-efficiency, though it is a focus in Chapter 7. There, I mostly emphasise GPU memory efficiency, although the methods described throughout are naturally storage-efficient as a natural consequence of their parameter-efficiency.

3. Parameter efficiency: closely related to memory efficiency is parameter efficiency, which relates to how great a proportion of a model's total parameters are updated during the fine-tuning process. A fine-tuning method which is parameter-efficient is generally memory-efficient in terms of storage, though not always in terms of GPU RAM - in Chapter 4, we propose a fine-tuning technique which makes sparse updates to the model's pre-trained parameters, but these updates are represented as dense matrices in GPU memory. Parameter-efficiency implies a reduction in a fine-tuning's representational capacity, which may be desirable for several reasons: Hu et al. (2022) suggest that fine-tuning tasks tend to have relatively low *intrinsic dimension* (Li et al., 2018) and therefore require only a fraction of the capacity of full fine-tuning; furthermore, limiting the capacity helps to prevent forgetting (Biderman et al., 2024) and overfitting. However, there is some evidence from recent work that this limitation of capacity may sometimes cause worse performance compared to full fine-tuning, especially for difficult tasks: Biderman et al. (2024) show that

on a set of coding and maths tasks, full fine-tuning significantly outperforms the PEFT method LoRA with a high rank, which in turn outperforms LoRA with lower ranks, and posit a trade-off between performance in the fine-tuning domain and remembering other abilities, with limited capacity favouring the latter over the former. Shuttleworth et al. (2024) further investigate the difference in behaviour between LoRA and full fine-tuning, finding that unlike full fine-tuning, LoRA has a tendency to introduce "intruder dimensions" in a model's weight matrices, that is, singular dimensions which differ greatly from those of the pre-trained matrix's. They hypothesise that intruder dimensions arise as a result of the model having to learn drastic alterations to its pre-trained dimensions to fit the fine-tuning data with limited capacity, and that intruder dimensions are associated with forgetting. This accords with their finding that *more* forgetting appears at very low LoRA ranks, and that forgetting is generally minimised at a "sweet spot" of intermediate LoRA ranks.

Another crucial distinction is that fine-tuning methods can have different efficiency properties at *training* and *inference* time. For instance, in Chapter 4, we will compare two different fine-tuning methods, bottleneck adapters and "lottery ticket" sparse fine-tuning. The former inserts new layers into a pre-trained model and then updates only these layers during fine-tuning, which is very time and memory efficient during training but less so during inference, while the latter has a relatively costly two-step training procedure but incurs no extra cost in time or GPU memory at inference. Chapter 7 is motivated by the desire to improve the training efficiency of sparse fine-tuning.

Computational Costs of Deep Learning

To get a better handle on the training efficiency-related aspects of fine-tuning, it is helpful to consider the main uses of time and memory during stochastic gradient descent with backpropagation, employing the example of a linear layer $Y = X^\top W$ for concreteness when required:

1. Storage of the parameter values (W in this case).
2. Computation and storage of intermediate representations during the forward pass. For a linear layer, it is necessary to compute the matrix multiplication $X^\top W$ and store the input matrix X in order to calculate gradients during the backward pass.
3. Computation and storage of gradients during the backward pass. For a linear layer, it is typically necessary to calculate both $\frac{\partial \mathcal{L}}{\partial W} = X \frac{\partial \mathcal{L}}{\partial Y}$ (hence why we needed to store X during the forward pass) and $\frac{\partial \mathcal{L}}{\partial X} = W \frac{\partial \mathcal{L}}{\partial Y}$.

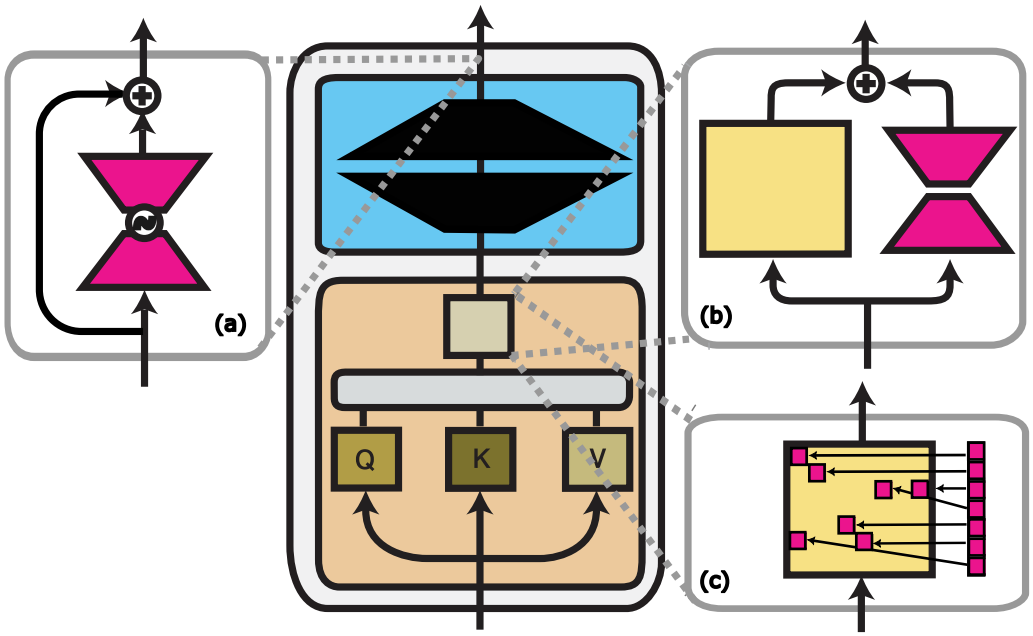


Fig. 2.2 An illustration of several kinds of PEFT modules as applied to a Transformer layer: (a) a Bottleneck Adapter (Houlsby et al., 2019; Pfeiffer et al., 2021a); (b) a Low-Rank Adapter (Hu et al., 2022) and (c) Sparse Fine-Tuning as applied to the output projection of self-attention. Note that dropout and layer normalisation components are omitted for simplicity.

4. Storage and updating of the optimiser state. The most widely used adaptive optimisers typically store additional state which can be quite memory-intensive; Adam (Kingma and Ba, 2015) for instance stores two momentum values per parameter.

Parameter-efficient fine-tuning (PEFT) alleviates the cost of parameter-efficient fine-tuning through reducing requirements (3) and (4) by fine-tuning only a small proportion of the model parameters, as only the tunable parameters require gradients to be calculated and stored and optimiser state to be stored and updated. It also addresses the risk of overfitting by reducing the degree of model capacity exercised during fine-tuning.

PEFT with Bottleneck Adapters

One of the first applications of PEFT in NLP was by Houlsby et al. (2019), who inserted specialised "bottleneck adapter" modules into each Transformer block of the pre-trained $\text{BERT}_{\text{LARGE}}$ model and trained only the parameters of these modules during task-specific fine-tuning. Despite the adapter modules accounting for only 3.6% of the total model

parameters, the resulting models achieved only marginally worse results than full fine-tuning on the GLUE benchmark (Wang et al., 2018). Adapter-based fine-tuning saw widespread adoption, likely as a result of the ease and efficiency of training and sharing adapters (Pfeiffer et al., 2020a). We will mostly consider bottleneck adapter modules of the type proposed by Pfeiffer et al. (2021a), where the adapter consists of an down- and up-projection (hence the term "bottleneck") with non-linear activation function inserted after the feed-forward layer of each Transformer block:

$$\text{Adapter}_l(\boldsymbol{h}, \boldsymbol{r}) = U_l \alpha(D_l \boldsymbol{h}) + \boldsymbol{r}, \tag{2.8}$$

where $\boldsymbol{h}$ and $\boldsymbol{r}$ are the post-norm output and residual of the feed-forward layer respectively, $U_l \in \mathbb{R}^{d_{\text{model}} \times d_b}$ and $D_l \in \mathbb{R}^{d_b \times d_{\text{model}}}$ are the trainable up- and down-projection at block l, $d_b \ll d_{\text{model}}$ is the dimension of the intermediate "bottleneck" layer, and α is a non-linear activation function. The bottleneck adapter is illustrated in Figure 2.2a.

PEFT with Low-Rank Adaptation

Bottleneck adapter fine-tuning is both faster and less memory-intensive than full fine-tuning for the reasons discussed above. Rücklé et al. (2021) find that the speedup in training throughput from using bottleneck adpaters is between 1.3 and 1.6 times compared to full fine-tuning. On the other hand, introducing additional layers into the model slows down the forward pass, resulting in an *inference* throughput of 0.95 times that of the original model in Rücklé et al.'s experiments. This issue is addressed by the *low-rank adaptation* (LoRA) PEFT method of Hu et al. (2022). LoRA is predicated on the intuition that fine-tuning tasks typically have low *intrinsic dimension* (Li et al., 2018), meaning that there is a set of acceptable solutions $\boldsymbol{\theta}_{\text{fine-tuned}} = \boldsymbol{\theta}_{\text{pre-trained}} + \Delta\boldsymbol{\theta}$ where $\Delta\boldsymbol{\theta}$ belongs to a subspace whose rank is much lower than the number of parameters $|\boldsymbol{\theta}|$. This seems to be a reasonable assumption if we think of fine-tuning as primarily exposing skills the model learned during pre-training rather than learning new skills from scratch. LoRA learns a low-rank update to a subset of the model's weight matrices as follows:

$$W' = W + \frac{\alpha}{r} BA, \tag{2.9}$$

where $W \in \mathbb{R}^{m \times n}$ is the frozen pre-trained matrix, $B \in \mathbb{R}^{m \times r}$ and $A \in \mathbb{R}^{r \times n}$ are trainable up- and down-projections, $r \ll m, n$ is the rank of the update, and α is a hyperparameter. For standard linear layers, which account for the vast majority of floating point operations in a Transformer, the forward pass takes the following form

when augmented with low-rank adaptation:

$$\boldsymbol{y} = W'\boldsymbol{x} = (W + \frac{\alpha}{r}BA)\boldsymbol{x} \tag{2.10}$$

$$= W\boldsymbol{x} + \frac{\alpha}{r}BA\boldsymbol{x}. \tag{2.11}$$

When $(A\boldsymbol{x})$ is computed first in the product $BA\boldsymbol{x}$, it is not necessary to materialise the full LoRA update $\frac{\alpha}{r}BA$ during training. Though LoRA incurs the additional cost of the forward and backward pass for A and B, this is greatly outweighed by the saving of no longer needing to compute or store the gradient of the frozen W, since A and B together have $(m+n)r$ elements, whereas W has mn elements and $r \ll m, n$. LoRA as applied to a linear layer is illustated in Figure 2.2b.

While during training it is more efficient to avoid materialising the LoRA update $\frac{\alpha}{r}BA$, the opposite is true at inference time, as it is possible to compute $W' = W + \frac{\alpha}{r}BA$ once and then use it in place of W for the entire inference run, achieving the same speed as with the original model. This advantage and the typically very strong performance LoRA displays have led to it becoming one of the predominant PEFT techniques.

Prompt and Prefix Tuning

Brown et al. (2020) showed that natural language "prompts" prepended or appended to a pre-trained language model's input can be used to elicit a desired behaviour. For instance, they found that inputting a document followed by the string "`TL;DR`" would encourage the model to generate a summary of the document. However, finding effective natural language prompts is not easy and is a field of study in itself (Jiang et al., 2020; Schick and Schütze, 2021; Shin et al., 2020). Prompt (Lester et al., 2021) and prefix (Li and Liang, 2021) tuning simulate a natural language prompt with a sequence of trainable vectors prepended to the model input. Using a continuous representation of the prompt allows it to be learned using standard SGD with backpropagation, at the cost of making it less interpretable. Whereas prompt tuning only tunes vectors prepended at the embedding layer, prefix tuning takes this idea further by prepending such trainable vectors to all sequences of hidden representations the model computes.

A significant advantage of prefix and prompt tuning is that because they conceptually intervene on the model input instead of the model parameters, it is possible to efficiently run inference with a different prompt for each example in a batch, allowing a system to support multiple tasks simultaneously without needing to load a new set of weights when switching to a new task. It is more difficult to support this behaviour with a fine-tuning method that intervenes on model parameters, since the model parameters are typically

fixed while a batch of inputs is processed. On the other hand, prepending virtual tokens to the input extends the sequence length and thereby slows down inference, with longer, more expressive prompts being increasingly costly. Therefore, prefix and prompt tuning may underperform more highly parameterised methods on complex tasks or those with relatively large fine-tuning datasets.

Other PEFT Methods

BitFit (Ben Zaken et al., 2022) pushes PEFT to an extreme by updating only a model's bias parameters during training. These account for less than 0.1% of the total parameters of the BERT and RoBERTa models used in Ben Zaken et al.'s experiments, yet they found that this was generally sufficient to match the performance of full fine-tuning on the GLUE benchmark (Wang et al., 2018). However, since BitFit is restricted to fine-tuning a very small subset of parameters, it may lack the capacity for more difficult tasks, with subsequent research showing its performance falling behind other approaches (Ansell et al., 2022; Liu et al., 2022). Furthermore, many recent LLMs do not have bias parameters.

Liu et al. (2022) propose $(IA)^3$ ("Infused Adapter by Inhibiting and Amplifying Inner Activations"), which intervenes on selected intermediate Transformer representations by scaling them elementwise. Specifically, $(IA)^3$ introduces three new trainable parameter vectors per Transformer block, $l_k \in \mathbb{R}^{d_k}$, $l_v \in \mathbb{R}^{d_v}$ and $l_{ff} \in \mathbb{R}^{d_{ff}}$, where d_{ff} is the dimension of the activation vector in the FFN layer. These vectors are multiplied elementwise with the attention keys and values and the FFN activations respectively. This can be formulated as

$$\text{Attention}_{(IA)^3}(Q, K, V) = \text{softmax}\left(\frac{Q(l_k \odot K^\top)}{\sqrt{d_k}}\right)(l_v \odot V) \tag{2.12}$$

$$\text{FFN}_{(IA)^3}(\boldsymbol{x}) = W_2(l_{ff} \odot \alpha(\boldsymbol{x}W_1 + \boldsymbol{b}_1)) + \boldsymbol{b}_2. \tag{2.13}$$

Since $(IA)^3$ only learns three vectors per layer, it is highly parameter-, memory- and compute-efficient, and Liu et al. (2022) find it to be highly effective in their evaluation, outperforming all other PEFT methods they tested including some which fine-tune an order of magnitude more parameters. In practice, however, $(IA)^3$ has not superseded LoRA as PEFT method of choice for large-scale fine-tuning tasks such as instruction tuning. This is likely because its limited capacity makes it more suited to tasks with small datasets such as in the range of 20-70 examples used in Liu et al.'s evaluation.

Gradient Accumulation

Gradient accumulation is a technique for reducing the memory usage during stochastic gradient descent. Each batch of B training examples is divided into G "micro-batches" of size M such that $B = GM$. The forward and backward pass are computed for each micro-batch sequentially, with the gradient calculated for each parameter during the backward pass being added to a cumulative total. Thus the memory required to store parameter values and gradients and optimiser state is the same as for full batches of size B, but the memory required to store the intermediate representations is reduced by a factor of G, since the batch size is typically the first dimension of these representation tensors.

Activation Checkpointing

Activation checkpointing (Chen et al., 2016), also sometimes known as gradient checkpointing, is another technique for reducing the cost of storing intermediate representations during backpropagation. It works by storing only selected intermediate representations during the initial forward pass and then recalculating the missing representations when needed during the backward pass. Consider a network with N layers. Activation checkpointing stores only the input to every $\sqrt{N}$th layer. When it is time to compute the gradients of the ith block of $\sqrt{N}$ layers, the full set of intermediate representations for this block is computed using a *second* forward pass originating from the "checkpoint," the saved input to the block, and is then discarded once the gradients have been calculated. Since $\sqrt{N}$ checkpoints are saved in the first forward pass, and each second forward pass calculates intermediate representations for a block of $\sqrt{N}$ layers which are then discarded before moving onto the next block, the total memory used to save intermediate representations is $\mathcal{O}(\sqrt{N})$, compared to $\mathcal{O}(N)$ without activation checkpointing. This comes at the cost of doubling the amount of time spent on forward passes.

Quantisation

While parameter efficiency reduces the memory required to store gradients and optimiser state during fine-tuning, and gradient accumulation and activation checkpointing target storage of intermediate representations, *quantisation* aims to reduce the cost of storing the model's parameters and/or activations by using a lower-precision format. On hardware which supports faster operations with the lower-precision format, such as INT8 product-sums, quantisation may also speed up computation. For example, Bhandare et al. (2019), who were among the first to quantise a Transformer, convert the data type of the

parameter and activation tensors involved in matrix multiplication operations from the FP32 used in pre-training to INT8 with a zero-offset followed by a scaling factor and rounding operation:

$$scale = \frac{\text{target range}}{\text{source range}} \tag{2.14}$$

$$A_{quantized} = round((A_{float} - zero_{offset}) \cdot scale), \tag{2.15}$$

where "source range" and "target range" refer to the range of values covered in the original and quantisation data types respectively - the target range for an unsigned INT8 value for instance would be 255 - 0 = 255. Mapping values from a higher- to a lower-precision data type inevitably introduces error and causes the quantised model to behave differently from its original-precision equivalent. However, the choice of source range has an effect on the size of this error. The source range can be naively chosen as the difference between the global maximum and minimum values of the tensor. This prioritises preserving the exact values of outliers at the expense of achieving high precision in the representation of values closer to the centre of the distribution. Bhandare et al. (2019) instead choose a source range which minimises the Kullback-Leibler divergence between the histograms of the quantised and unquantised tensors. To further reduce the quantisation error, it is now common to divide a tensor into a number of groups and apply a different scale factor to each depending on its distribution (Shen et al., 2020). For instance, when quantising a tensor of activations, Yao et al. (2022) dynamically calculate the source range and apply the resulting rescaling on a per-token basis. Dettmers et al. (2022), on the other hand, identify outlier values and store them in 16-bit precision while quantising the remaining values in 8 bits.

Quantisation methods can broadly be categorised as "quantisation-aware training" (QAT), where the model weights are adjusted through further training to ameliorate the error introduced by the precision reduction resulting from quantisation (Shen et al., 2020; Zafrir et al., 2019; Zhang et al., 2020; Tao et al., 2022; Xiao et al., 2023; Frantar et al., 2023, *inter alia*), and post-training quantisation (PTQ), where no adjustment to the model weights is made other than changing their data type (Bondarenko et al., 2021; Dettmers et al., 2022). The former requires access to the model's training data and is inefficient as it may require many additional training steps; the latter generally requires little additional computation (sometimes a small amount of "calibration data" is used to establish the distribution of the activation tensors) but tends to result in great performance degradation since there is no explicit counteraction of the introduced error, and thus admits lower compression ratios than QAT.

Due to the tendency for low-precision training to be unstable, quantisation has almost always been applied as a separate step after a model has been trained in full precision. Contrary to this trend, and of particular interest to us, is the work of Dettmers et al. (2023), who apply quantisation to reduce memory usage during the fine-tuning process itself: they propose quantised LoRA (qLoRA), a variant of LoRA that stores the pre-trained model weights in a 4-bit data type called NormalFloat4, temporarily dequantising each parameter tensor into a higher-precision data type only when required. Since the pre-trained weights are kept frozen and the LoRA weights are stored in 16- or 32-bit precision, there is minimal risk of instability due to underflow or overflow. A typical qLoRA linear layer can be expressed as

$$\boldsymbol{y}^{\text{BF16}} = \text{DEQUANT}(W^{\text{NF4}})\boldsymbol{x}^{\text{BF16}} + \frac{\alpha}{r} B^{\text{BF16}} A^{\text{BF16}} \boldsymbol{x}^{\text{BF16}}, \tag{2.16}$$

where $^{\text{BF16}}$ denotes tensors with the BrainFloat16 data type, $^{\text{NF4}}$ denotes the NormalFloat4 data type, $\text{DEQUANT}(\cdot)$ denotes the dequantisation operation which converts tensors from NF4 to BF16. Here, $W^{\text{NF4}} = \text{QUANT}(W^{\text{BF16}})$, where W^{BF16} is the pre-trained form of the weight matrix, and $\text{QUANT}(\cdot)$ denotes the quantisation operation which converts tensors from BF16 to NF4. Note that $\text{DEQUANT}(\text{QUANT}(W))$ is not exactly equal to W since it is not possible represent all values of a 16-bit value in 4 bits without some loss of precision. However, Dettmers et al. (2023) find that barely any performance penalty results from this loss of precision in their instruction tuning evaluation. While qLoRA is significantly more memory-efficient than unquantised LoRA, qLoRA training is slightly slower as it incurs the additional cost of dequantising its weights every time they are used.

2.2.2 Modular Fine-Tuning

The fine-tuning process can be thought of as endowing the pre-trained model with specific skills or knowledge. Often, these skills or knowledge are most useful not in isolation, but when combined with other knowledge or skills. For instance, we sometimes wish to adapt simultaneously to a specific task *and* domain; sometimes we wish to exploit already learned skills to better learn new, related skills (Pfeiffer et al., 2021a; Ostapenko et al., 2024; Muqeeth et al., 2024). The purpose of *modular* fine-tuning is to enable these skills and knowledge to be reused in this way by concentrating them in dedicated *modules*. The benefits most often associated with modular fine-tuning are:

- Transferability: when skills are concentrated in modules, it is often possible for several modules to be loaded in the model at once. This allows several skills to be

either learned or exploited simultaneously such that *positive transfer* between the skills results in better downstream performance. For instance, Pfeiffer et al. (2021a) propose a modular approach to multi-task learning, where an independent adapter module is first learned for each of N tasks, and the model is specialised to a specific task by learning the parameters of an additional module which fuses the outputs of the N adapters. Ponti et al. (2022) propose a multi-task learning framework which explicitly models a set of latent skills, for which it learns corresponding modules and a "task-skill allocation matrix," which determines which skills are required for each task.

- Composability: independently trained modules can often be *composed* to combine the skills or knowledge they represent without further training. Pfeiffer et al. (2020b) train bottleneck adapters to specialise models to specific tasks and languages, and show that an arbitrary task and language adapter can be stacked on top of each other, delivering improved performance in the task in the given language compared to that given by the task adapter alone. Chronopoulou et al. (2023) on the other hand adapt a model to an unseen domain by composing several relevant adapters for seen domains through averaging their weights. Nayak et al. (2023) compose soft prompts corresponding to concepts such as "small," "furry," and "cat" to adapt to new classes for image classification.

- Efficiency: modules are typically parameter-efficient, that is, they consist of a small proportion of the total model parameters, and are trained while the remaining parameters are frozen. As discussed in §2.2.1, this generally results in significant improvements in training speed and memory usage.

Pfeiffer et al. (2023b) identify four primary elements of a modular deep learning model: the *computation function*, which describes the implementation of each module in isolation; the *routing function*, which determines which modules are *active*, that is, included in the computational graph, for any given input to the model; the *aggregation function*, which combines the output of the active modules; and the *training setting*, which describes the order and details of module training. I refer the reader to Pfeiffer et al. (2023b) for a comprehensive discussion of this taxonomy.

2.3 Multilinguality and Cross-Lingual Transfer

The world is home to at least 7,000 languages (Eberhard et al., 2024; Hammarström et al., 2024). There are a number of factors limiting the availability of high-quality

NLP systems for the vast majority of these languages. Perhaps the most acute of these limitations is a lack of data: Joshi et al. (2020) describe 88% of the languages they consider as having "exceptionally limited resources," with "virtually no unlabelled data to use," let alone labelled data of the kind needed for task-specific fine-tuning. For context, the first-generation pre-trained Transformer BERT was trained on a corpus of 3.3 billion English words; as of 2024, this would be considered a very modest training corpus, with LLaMA 2 for instance being trained on 2 *trillion* tokens (Touvron et al., 2023b). Bearing in mind the importance of corpus size for language model scaling (Kaplan et al., 2020), high-performance, monolingually pre-trained models are not feasible for most languages, as a 3.3 billion token corpus is accessible for only a handful. Even if this approach to multilingual NLP were possible, it might not be practical: the cost and inconvenience of training, maintaining and deploying a hundred or more independent language models would be considerable.

In the face of this severe imbalance in data availability across languages, NLP practitioners have turned to transfer learning. Specifically, *cross-lingual transfer* aims to use the knowledge contained in data belonging to one or more *source languages* to learn a better model for tasks in one or more *target languages*. For the reasons discussed above, transfer is typically from higher to lower resource languages. Theoretical and comparative linguistics provide good reason to believe that data from one language can contain knowledge that generalises to other languages: while there is disagreement about the extent to which languages share a universal structure (Chomsky, 1965; Croft, 2002; Evans and Levinson, 2009), it is widely agreed that certain features such as recursion and some parts of speech (such as nouns and verbs) are common to all languages. Geographically or genealogically related languages tend to share typological features and similar vocabularies, suggesting increased transfer potential between such languages. Furthermore, all languages make reference to objects and concepts which exist in the same universe, and while there may be significant cross-cultural variation in how people think of and interact with the world, there is a great degree of similarity in human experience across time, location and linguistic background. For instance, the idea *"the tired children sleep soundly"* is far more plausible than the idea *"colourless green ideas sleep furiously"* no matter what language it is expressed in. Thus cross-lingual transfer offers the opportunity to learn a better model not just of superficial linguistic features, but of the world itself.

2.3.1 Machine Translation

The main focus of early research in multilingual NLP was machine translation (MT), the task of translating text from one language into another. While a useful end in itself, MT can also be considered a technique for cross-lingual transfer, because it enables a model designed to perform a task in language A to be applied to text in language B by first translating the text into language A. Early MT systems generally relied on bilingual lexicons and sets of rules to transform the source into the target language (Wang et al., 2022a). However, the subsequent statistical (SMT) and neural (NMT) approaches relied on learning translation models from *parallel corpora*, that is, datasets consisting of a large number of sentences in the source language and their translations into the target language. Since these translations are produced by expert human translators, they are typically much scarcer than unlabelled data, and are rarely available in significant quantity other than for a handful of the highest-resource languages. This has spurred the development of *unsupervised* techniques for multilingual NLP.

As I make limited use of MT in this thesis, I will focus on the application of MT to cross-lingual transfer rather than providing a detailed background of MT techniques themselves. There are two main approaches: *translate-train*, where an MT model is used to translate the training data from the source language into the target language, and this translated data is then used to fine-tune a pre-trained language model which supports the target language; and *translate-test*, where a pre-trained language model which supports the *source* language is fine-tuned on the original training data, and at inference time, examples are translated from the target into the source language before being sent to the fine-tuned model for prediction. These are generally effective approaches (Hu et al., 2020), but they have some limitations: firstly, they rely on labels being preserved after translation. For instance, while a sentence with positive sentiment in language A should remain positive when translated into language B, its dependency parse tree may change completely due to different word order, a different choice of grammatical constructions being more natural in language B and so on. Secondly, the performance is highly dependent on the quality of the MT model (Artetxe et al., 2023), which may be poor for low-resource languages given the typical shortage of parallel data. Furthermore, the need to balance "fluency in the target language" with "fidelity to the source text" (Volansky et al., 2013) often results in "translationese", or text that sounds unnatural in the target language. A system which is trained on "translationese" text but receives natural text for inference may suffer degradation in performance due to this distribution shift. Artetxe et al. (2020a) showed that this issue can be somewhat alleviated by always using translated data for both training and inference: for instance, in the translate-test

approach, the source language data can be "translated into translationese" by translating into some other pivot language and then back into the source language.

2.3.2 Unsupervised Cross-Lingual Transfer

Bilingual word embeddings (Mikolov et al., 2013b; Artetxe et al., 2016) were one of the first forays into unsupervised learning for multilingual NLP. The underlying idea is inspired by the observation that monolingual word embedding spaces exhibit linear regularities such that pairs of words which hold a fixed semantic relationship to each other tend to share a similar offset vector (Mikolov et al., 2013a) (for example, the offset between the embeddings for "man" and "woman" is similar to that for "King" and "Queen"). Assuming that two languages share a similar set of concepts and relationships between them, this suggests that their embedding spaces ought to be roughly isomorphic and could be mapped onto each other with a linear transformation. This linear mapping W can be learned by solving an optimisation problem of the form

$$\min_{W} \sum_{i=1}^{n} d(W\boldsymbol{x}_i, \boldsymbol{z}_i), \tag{2.17}$$

where $\boldsymbol{x}_i$ and $\boldsymbol{z}_i$ are the embeddings of a word in the source language and its translation in the target language respectively, and d is a distance metric. While Mikolov et al. (2013b)'s initial method relied on a bilingual dictionary of 5,000 common words, Artetxe et al. (2017) showed that their refined method could perform well with as few as 25 word pairs, or in a completely unsupervised manner by using only numerals, which have a consistent meaning across most languages. Exploiting the knowledge contained in monolingual corpora and the isomorphism in concepts across languages with this technique enabled significant advances in low-resource scenarios in MT (Artetxe et al., 2018) and other tasks including dependency parsing (Ammar et al., 2016), part-of-speech tagging (Fang and Cohn, 2017) and semantic parsing (Johannsen et al., 2015).

The pre-trained Transformer revolution opened new frontiers in cross-lingual transfer, starting with multilingual BERT (mBERT; Devlin et al., 2019). mBERT utilises the same architecture[2] and pre-training objective as BERT, but was pre-trained on a corpus consisting of Wikipedia dumps of the 100 languages with largest Wikipedias. Despite the lack of any explicit cross-lingual signal during training, mBERT learns representations which exhibit a surprising degree of cross-lingual transferability. Wu and Dredze (2019) investigate mBERT's performance in zero-shot cross-lingual transfer, where no task-

[2]mBERT shares the same dimensions as $\text{BERT}_{\text{BASE}}$ other than its vocabulary size which is much larger to accommodate the various lexicons and scripts of its pre-training languages.

specific data is available in the target language, finding that when fine-tuned on the source language and applied directly to the target language, it performs strongly on a range of tasks. Pires et al. (2019) find similar evidence of mBERT offering strong zero-shot cross-lingual transfer, and speculate that this may be partially a result of tokens in the model's multilingual vocabulary being shared across languages. Many tokens have similar meanings across languages (again consider numerals, for instance), which may act as "seeds" for aligning other tokens cross-lingually. Pires et al. (2019) show that mBERT is nevertheless capable of high-performance zero-shot cross-lingual transfer between languages which do not even share the same script: they obtain a 91% accuracy score on part-of-speech tagging (POS) for Hindi (written in Devanagari script) using mBERT fine-tuned only on POS examples in Urdu (written in Arabic script), suggesting that mBERT learns cross-lingual representations at a deeper level than the embedding layer alone. Despite lacking a common script, Hindi and Urdu are very closely related genealogically and typologically; Pires et al. (2019) also consider the relationship between language similarity and transferability, finding that languages which have more typological similarities (such as sharing SVO or SOV word order) tend to be more effective transfer pairs. Artetxe et al. (2020b) challenge the importance of token overlap for cross-lingual alignment by learning a new set of token embeddings for a target language through continued pre-training of a mBERT-like model which was *monolingually* pre-trained on the source language. Surprisingly, the resulting model exhibits only slightly diminished transfer performance relative to one trained jointly on the source and target language, despite the body of the former model being trained only on source language data. This suggests that even monolingual models learn abstractions which generalise well across languages.

Other notable massively multilingual Transformers (MMTs) include XLM-RoBERTa (XLM-R; Conneau et al., 2020), which uses a similar architecture to mBERT but is more thoroughly trained and has both an mBERT-sized XLM-R_{BASE} and a larger XLM-R_{LARGE} variant, and mDeBERTa (He et al., 2021), which is trained using a generator-discriminator setup similar to ELECTRA's (Clark et al., 2020b). Liu et al. (2020) trained mBART and Xue et al. (2021) trained mT5, multilingual versions of the sequence-to-sequence Transformers BART and T5, which were pre-trained on 25 and 100 languages respectively. mBART demonstrates impressive performance on machine translation even when the size of the parallel corpus for fine-tuning on the MT task is relatively small; it likely benefits from the cross-lingual alignment learned implicitly during its unsupervised pre-training as described above for mBERT.

The trend towards increasingly large decoder models and pre-training datasets has unfortunately not been accompanied by a proliferation of highly multilingual counterparts. The few examples of LLMs designed with multilinguality in mind include XGLM (Lin et al., 2022), a 7.5 billion parameter model pre-trained on a 500 billion token corpus of 30 languages; PolyLM (Wei et al., 2023), a 13 billion parameter model pre-trained on a 640 billion token corpus of 18 languages; and BLOOM (BigScience Workshop et al., 2023), which comes in a range of sizes from 560 million to 176 billion parameters and is trained on 46 natural languages and 13 programming languages, but may be significantly under-trained at 176 billion parameter scale given its corpus is only 384 billion tokens in total. Some recently released "frontier" models boast a high level of proficiency in several languages, but this is typically confined to a small number of very high-resource languages: Mixtral 8x22B (Jiang et al., 2024) covers English, French, Italian, German and Spanish, while Command-R+ (Cohere for AI, 2024) covers six European languages plus Japanese, Korean, Arabic and Chinese. Despite their lack of intentional multilinguality, some English-centric LLMs display impressive multilingual abilities (Armengol-Estapé et al., 2022; Holtermann et al., 2024), likely due to trace amounts of non-English data appearing in their pre-training corpora.

2.3.3 Few-Shot Cross-Lingual Transfer

While zero-shot cross-lingual transfer holds theoretical and some practical interest, often more realistic is the few-shot case, where a small number of gold-standard target language examples are available during task-specific fine-tuning. Though it may be expensive to annotate target language data, especially for low-resource languages where native speakers are hard to access, prior work has shown that using even a small amount during training can yield significant gains in performance (Zhao et al., 2021). One of the earliest approaches to few-shot cross-lingual transfer by Lauscher et al. (2020) involves first fine-tuning a multilingual pre-trained model on the source language data, then separately on the few target language shots, with the few-shot training step yielding up to 15 point improvements in test performance. However, recent work has suggested that it is more effective to jointly train on both the source and target data simultaneously (Xu and Murray, 2022; Schmidt et al., 2022), perhaps because the separate target language training phase of the sequential approach is highly prone to overfitting to the very small number of examples, in addition to forgetting what was learned in the source language training phase.

2.3.4 Language Adaptation

The most widely used multilingual Transformer encoders such as mBERT and XLM-R are generally pre-trained on around 100 languages, and more recent models tend to be less multilingual, as discussed above. This leaves the vast majority of the world's languages unsupported, and even those languages which are covered may display suboptimal performance due to the "curse of multilinguality" (Conneau et al., 2020), the decline in performance that occurs as a model becomes more multilingual and hence the capacity that can be dedicated to each language diminishes. We can attempt to circumvent this problem by specialising the multilingual model for a specific language with an additional pre-training step.

Pfeiffer et al. (2020b) propose the adaptation of multilingual Transformer encoders to specific languages through *continued pre-training*, that is, fine-tuning with the original objective (masked language modelling) but on a corpus consisting purely of text in the *target* language. They consider two implementations of an NLP system which must support N_T tasks in N_L target languages given task training data only in some other *source language*:

1. A baseline full fine-tuning approach. For each target language $l \in \{1, 2, ..., N_L\}$, a language-adapted variant $M^{(l)}$ of the pre-trained multilingual model M is created through continued pre-training of M on a corpus of language l. Then for each task $t \in \{1, 2, ..., N_T\}$, a model variant $M^{(l,t)}$ specific to the task-language pair (l, t) is created by fine-tuning $M^{(l)}$ on the training data for t.[3]

2. "MAD-X", a bottleneck adapter-based fine-tuning and composition approach. For the source language $l = 0$ and each target language $l \in \{1, 2, ..., N_L\}$, a language adapter $A^{(l)}_{\text{lang}}$ for M is created through continued pre-training on l's corpus. Then for each task $t \in \{1, 2, ..., N_T\}$, a task adapter $A^{(t)}_{\text{task}}$ is trained on top of $M \odot A^{(0)}_{\text{lang}}$, where $\odot$ denotes adapter insertion. That is, the task adapters are trained on top of the frozen source language adapter. Crucially, inference for task t in target language l is performed using $(M \odot A^{(l)}_{\text{lang}}) \odot A^{(t)}_{\text{task}}$, meaning the relevant task adapter is stacked on top of the relevant language adapter within each Transformer layer.

MAD-X's language adapters consist of both the conventional bottleneck adapter layers plus an additional "invertible adapter" to modify the behaviour of the embedding layer, which helps to learn the vocabulary of a new language.

While Pfeiffer et al. found that target language adaptation resulted in improvements in downstream performance with both approaches, there are several reasons to prefer the

[3]For brevity, I only describe Pfeiffer et al. (2020b)'s strongest "MLM-TRG" baseline here.

modular MAD-X approach: sequentially fully fine-tuning the base model on a language and then a task could result in forgetting pre-trained or language-specific knowledge by overwriting it with task-specific knowledge, which may be reflected in the better evaluation performance of the adapter-based approach; the full fine-tuning approach trains $N_L N_T$ independent models, requiring $\mathcal{O}(N_L N_T)$ time and storage space, whereas MAD-X trains $N_L + N_T + 1$ adapters, requiring only $\mathcal{O}(N_L + N_T)$ time and storage space; furthermore, each adapter is faster to train and much smaller than a fully fine-tuned model.

Chapter 3

Multilingual Adapter Generation for Efficient Cross-Lingual Transfer

3.1 Introduction

Although massively multilingual Transformers (MMTs) such as mBERT (Devlin et al., 2019) and XLM-R (Conneau et al., 2020) display impressive (zero-shot) cross-lingual transfer abilities (Pires et al., 2019; Wu and Dredze, 2019), their performance has been shown to drop when the target language is typologically distant to the source language, or the size of its pre-training data is limited (Hu et al., 2020; Lauscher et al., 2020). In addition, their coverage of the world's languages - and consequently the range of language technology applications they can support - remains insufficient.[1]

Bottleneck adapters (Rebuffi et al., 2017; Houlsby et al., 2019), introduced in §2.2.1, have been proposed as a parameter-efficient means to extend multilingual models to under-represented languages (Bapna and Firat, 2019; Üstün et al., 2020). The general practice is to train a language adapter on unlabelled data for each language (Pfeiffer et al., 2020b) via masked language modeling (MLM), as described in §2.3.4. However, this generally requires substantial amounts of monolingual data, which prevents adapters from serving under-resourced languages where such additional language-specific capacity would be most useful.

To address this deficiency, we propose *multilingual adapter generation* (MAD-G), a novel paradigm that enables the generation of bottleneck adapters for low-resource languages by *sharing information across languages*. Instead of learning separate adapters

[1] mBERT and XLM-R have been trained on corpora from 104 and 100 languages, respectively. According to Glottolog (Hammarström et al., 2024), however, there are over 7,000 languages spoken around the world.

for each language, MAD-G leverages contextual parameter generation (CPG; Platanios et al., 2018; Ponti et al., 2019b), that is, it learns a single model that can generate a language adapter for an arbitrary target language. At the core of MAD-G is a contextual parameter generator which takes the typological vector of a language as input and outputs the parameters of the language-specific adapter. The generator's parameters are trained via MLM on the Wikipedias of 95 languages, selected to maximise linguistic diversity. Unlike prior CPG work (Platanios et al., 2018; Üstün et al., 2020), MAD-G generates language adapters that are task-agnostic, thus allowing for efficient and modular cross-lingual transfer across the board, i.e., the MAD-G language adapters can be leveraged in arbitrary downstream tasks (Pfeiffer et al., 2020b).

MAD-G shares information across languages (i) at the level of hidden representations by sharing the parameters of the adapter generator as well as (ii) at the typological level by conditioning on features from the URIEL database (Littell et al., 2017). The latter additionally enables zero-shot transfer to unseen languages. Further, we propose a variant of MAD-G in which we generate adapters also conditioned on their Transformer layer position (see §3.3.2), allowing MAD-G to be much more parameter-efficient than adapter-based transfer methods of prior work.

In experiments on zero-shot cross-lingual transfer on part-of-speech tagging (POS), dependency parsing (DP) and named entity recognition (NER), MAD-G demonstrates competitive performance to training more expensive language-specific adapters and shows strong performance in low-resource scenarios, e.g., in the NER task for African languages. What is more, we show that transfer performance can be further improved by (a) multilingual training of task adapters and (b) fine-tuning of generated MAD-G adapters, via MLM, on small amounts of monolingual data. Finally, we provide a nuanced analysis of transfer performance to unseen languages, highlighting the importance of the diversity of the language sample selected for pre-training.

3.2 Background

Before introducing MAD-G in detail in §3.3, we recapitulate its key components adopted from previous work. In particular, we summarise the discussion of language adaptation from Chapter 2, and introduce Contextual Parameter Generation (CPG).

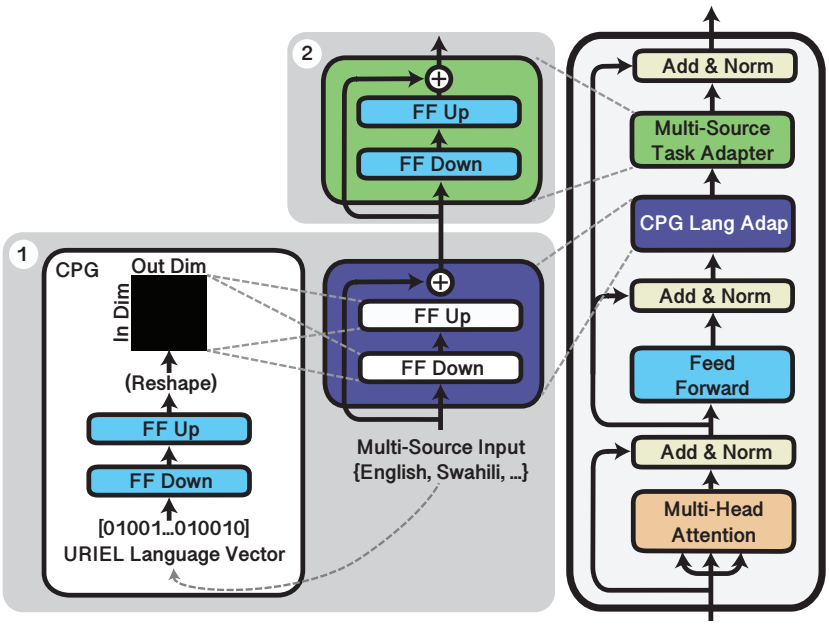


Fig. 3.1 Cross-lingual transfer with MAD-G. ① MAD-G training: the generator component learns to generate language-specific bottleneck adapters given URIEL vectors of input languages; the parameters of the generator are trained with an MLM objective, where instances of the respective language are passed through the frozen Transformer layers and the generated adapter parameters. ② In the downstream task fine-tuning, both the Transformer weights as well as the weights of the generated source-language adapter are frozen; an additional task adapter with randomly initialised weights is placed on top of the generated source language adapter. During target language downstream inference, the generated source language adapters are replaced with the generated target language adapters.

3.2.1 Language Adaptation

Massively multilingual Transformers (MMTs) infamously suffer from the "curse of multilinguality" (Arivazhagan et al., 2019; Conneau et al., 2020): for a fixed model capacity, their performance decreases as they cover more languages. Extending them to under-represented and unseen languages is far from trivial: additional training (of all model parameters) for such languages can lead to catastrophic forgetting of the previously acquired knowledge (McCloskey and Cohen, 1989; Santoro et al., 2016). A common remedy for both their coverage–performance trade-off and limited flexibility is to allocate *additional* model parameters for individual languages. This is often achieved through the use of bottleneck adapter layers (Houlsby et al., 2019; Pfeiffer et al., 2020b). We adopt the adapter variant of Pfeiffer et al. (2021a) as described in §2.2.1, and the continued pre-training approach for learning language adapters (LAs) discussed in §2.3.4.

3.2.2 Contextual Parameter Generation

Language adapters are an instance of a common design pattern in multilingual NLP: training a separate model or model components for each target language.[2] This approach based on a separate instance per language has two crucial drawbacks: (i) the total training time and number of parameters learned increase linearly with the number of languages; (ii) a lack of information sharing across languages due to the complete independence of learned parameters, which prevents low-resource languages from benefiting from their typological and genealogical ties to high(er)-resource languages.

CPG is a technique introduced by Platanios et al. (2018) to address these drawbacks. While originally conceived for neural machine translation (NMT), CPG can be applied to any neural model f parameterised by $\boldsymbol{\theta}$, for which we aim to learn parameterisations for a number of different *contexts*; in multilingual NLP, these "contexts" are languages. In the instance-per-language approach, an independent parameterisation $\boldsymbol{\theta}^{(l)}$, $l \in \{1, \ldots, n_l\}$, is learned for each of the n_l languages of interest. In CPG, the only language-specific parameters that we learn are the low-dimensional *language embeddings* $\boldsymbol{\lambda}^{(l)} \in \mathbb{R}^{d_l}$. These are used by the generator g, a hyper-network (Ha et al., 2017) component[3] with its own parameterisation $\boldsymbol{\phi}$, to produce the language-specific parameterisation of the main model: $\boldsymbol{\theta}^{(l)} = g_{\boldsymbol{\phi}}(\boldsymbol{\lambda}^{(l)})$. While g can in principle be any differentiable function (i.e., arbitrarily deep neural model), in practice it is typically set to a simple linear projection (i.e., $\boldsymbol{\phi} = W$):

$$g_W(\boldsymbol{\lambda}^{(l)}) = W\boldsymbol{\lambda}^{(l)}, \tag{3.1}$$

where $W \in \mathbb{R}^{n_p \times d_l}$ is a learnable weight matrix, n_p being the number of parameters of f.

The total number of parameters learned when training n_l independent models is $n_l n_p$, whereas the number of parameters in the W matrix is $d_l n_p$. Therefore, neglecting the small number of parameters dedicated to language embeddings, the CPG approach uses fewer parameters when $d_l < n_l$.[4] More importantly, in multilingual training the generator matrix W is shared across all languages, which enables knowledge sharing across languages and leads to improved transfer performance.

[2] Other examples include the training of language-specific pre-trained language models (Rust et al., 2021) as well as language pair-specific encoder–decoder models for machine translation (Luong et al., 2016; Firat et al., 2016).

[3] A hyper-network is a neural model that generates the parameters of another (main) neural model.

[4] Training MAD-G on 95 languages with $d_l = 32$ (this work) achieves roughly a threefold saving in parameter size.

Platanios et al. (2018) and Ponti et al. (2021a) opt for randomly initialising language embeddings $\boldsymbol{\lambda}^{(l)}$ and learning them end-to-end. Specified like this, however, CPG cannot generalise to languages unseen in training, as it would lack embeddings for those languages at inference. To support generalisation to arbitrary new languages, one must ground language embeddings in some external language representation, available for many languages. To this end, Ponti et al. (2019b) exploit typological language vectors from the URIEL database (Littell et al., 2017) directly as language embeddings to generate a full set of model parameters. In a similar vein, Üstün et al. (2020) use the typological language vectors from URIEL to generate task- and language-specific bottleneck adapters for dependency parsing: they learn the parameters $\boldsymbol{\phi}$ of the generator g via multilingual dependency parsing training on 13 languages. In contrast, MAD-G's multilingual MLM training allows the generation of task-agnostic LAs that can support downstream cross-lingual transfer for arbitrary NLP tasks.

3.3 MAD-G: Methodology

MAD-G aims to enable resource-efficient adaptation of MMTs to a wide range of previously unseen, radically resource-poor languages,[5] and contribute in this manner to more sustainable (Strubell et al., 2019; Moosavi et al., 2020) and more inclusive NLP (Joshi et al., 2020). We couple (i) the computational efficiency of the light-weight bottleneck adapters and (ii) knowledge sharing and zero-shot language transfer capabilities of CPG, with (iii) external linguistic (i.e., typological) knowledge (Ponti et al., 2019a) towards supporting arbitrary NLP tasks for (even radically) resource-poor languages.

MAD-G mitigates important limitations of prior work. Unlike Üstün et al. (2020), we generate *task-agnostic* LAs, (re)usable across NLP tasks. Unlike the MAD-X framework (Pfeiffer et al., 2020b), which trains LAs independently for each language (requiring sufficient monolingual corpora), MAD-G can support unseen and resource-poor languages in downstream tasks by generating LAs from typological vectors. Moreover, MAD-G leverages typological relations between languages. We also show that the two approaches can be successfully combined: monolingual MLM fine-tuning of a MAD-G-generated LA yields further benefits.

[5] With "radically resource-poor" languages we refer to languages for which even the acquisition of non-negligible amounts of text data is difficult.

3.3.1 Generating Language Adapters

Our input representation for each language is a sparse typological vector $\boldsymbol{t}^{(l)}$ encompassing 289 binary linguistic features (103 syntactic, 28 phonological and 158 phonetic features) from the URIEL language typology database (Littell et al., 2017). We obtain the language embedding $\boldsymbol{\lambda}^{(l)}$ from $\boldsymbol{t}^{(l)}$ using a single-layer linear down-projection: $\boldsymbol{\lambda}^{(l)} = V\boldsymbol{t}^{(l)}$, with the parameter matrix $V \in \mathbb{R}^{d_l \times 289}$. Down-projecting to a dimension $d_l \ll 289$ prevents W from being impractically large. By grounding language embeddings in external expert linguistic knowledge (i.e., URIEL vectors), we enable generalisation to all languages for which such typological vectors exist, regardless of the availability of monolingual text for those languages for generator training. In multilingual MLM training, we generate the bottleneck adapter parameters $\boldsymbol{\theta}^{(l)}$ for each instance from the embedding of the respective language, as specified in Equation 3.1.[6] Let n_b be the number of layers in the MMT (e.g., for mBERT (Devlin et al., 2019), $n_b = 12$). The MAD-G parameter matrix W then has $2hmn_b \times d_l$ parameters, where h is the hidden size of the Transformer layer and m the bottleneck size of the adapter layer (i.e., a single adapter module has $2hm$ parameters).

3.3.2 Factoring Out Layer Embeddings

By factoring out language-specific embeddings $\boldsymbol{\lambda}^{(l)}$, we force the MAD-G parameters W to share knowledge across languages. The generated language adapters in different Transformer layers are, however, still mutually independent. By additionally factoring out representations of each Transformer layer indices into *layer embeddings* $\boldsymbol{\lambda}^{(b)} \in \mathbb{R}^{d_b}$, $b \in \{1, 2, \ldots, n_b\}$, we can condition the adapter generation not only on languages but also on layers. This has two potential benefits: (i) it allows for information sharing between adapters of different layers, and, more importantly, (ii) it substantially reduces the size of the generator W. In this model variant, dubbed `MAD-G-LS`, the generator outputs adapters $\boldsymbol{\theta}^{(l,b)}$ for language-layer pairs:

$$\boldsymbol{\theta}^{(l,b)} = W(\boldsymbol{\lambda}^{(l)} \oplus \boldsymbol{\lambda}^{(\boldsymbol{b})}), \tag{3.2}$$

with the concatenation of the language embedding $\boldsymbol{\lambda}^{(l)}$ and layer embedding $\boldsymbol{\lambda}^{(\boldsymbol{b})}$ as input. The `MAD-G-LS` generator has $2hm \times (d_l + d_b)$ parameters, which is, assuming language and layer embeddings of equal size (i.e., $d_b = d_l$), a parameter reduction by a factor $\frac{n_b}{2}$ compared to the base MAD-G configuration from §3.3.1.

[6] An alternative option for adapter generator input would be randomly initialised language embeddings $\boldsymbol{\lambda}^{(l)}$; this would, however, not allow the opportunity for downstream generalisation to unseen languages.

group	definition	# w/ treebank	language examples
`mBERT-seen`	seen during mBERT pre-training	56	English, Japanese, Chinese
`MAD-G-seen`	seen only during MAD-G training	6	Buryat, Maltese, Erzya
`unseen`	completely unseen	33	Bhojpuri, Moksha, Warlpiri

Table 3.1 Definitions of three language groups. "# w/ treebank" is the number of languages belonging to each group included in the evaluation of the universal dependencies POS-tagging/dependency parsing tasks.

3.3.3 Multi-Source Task Adapters

Once the multilingual adapter generator has been trained via multilingual MLM, the generated LAs can be used to facilitate downstream cross-lingual transfer. Here, we follow the task-specific fine-tuning setup of MAD-X (Pfeiffer et al., 2020b): we insert and train the task-specific adapter (TA) on top of the language adapter of the source language—the parameters of the LA as well as parameters of the base MMT are kept frozen. In prior work, the TA is trained on data from a *single source* language l_s with the LA for l_s activated (with frozen parameters). At inference time, the LA for the *target language* l_t is plugged in instead of l_s's adapter, with the same TA (Pfeiffer et al., 2020b).

In downstream tasks with task data in multiple languages, we can resort to *multi-source* transfer, i.e., multilingual training of the task adapter. This is possible with per-language trained LAs (e.g., MAD-X adapters) as well as without any LAs. We hypothesised that multi-source training would be particularly beneficial with MAD-G because of the knowledge shared by LAs of different languages as a result of their generation with the MAD-G's multilingual generator. In other words, with MAD-G, the multi-source task adapter training is supported by a single LA generator model (see Figure 3.1), rather than a set of independently trained LAs. However, our experiments show that multi-source training is greatly beneficial regardless of language adapter type; the advantage does not seem larger for MAD-G in particular.

We employ a straightforward approach to TA training on the set of source languages L_s: in each step, we (1) randomly select a language l from L_s from which we sample a training batch and (2) in the forward pass – before the task adapter – activate the LA of the language l for that batch. To the best of our knowledge, we are the first to investigate multi-source adapter-based transfer in cross-lingual settings.

3.4 Experimental Setup

3.4.1 Tasks and Languages

We evaluate on three downstream tasks which provide sufficient evaluation data for low-resource languages: part-of-speech (POS) tagging, dependency parsing (DP), both on the Universal Dependencies (UD) 2.7 dataset (Zeman et al., 2020), and named entity recognition (NER) on the MasakhaNER dataset for African languages (Adelani et al., 2021). For POS and DP, we evaluate on a substantial subset of all UD languages with available treebanks.[7] We discern between three language groups in evaluation, with some examples in Table 3.1: (i) `mBERT-seen` languages are those included in mBERT's pre-training; (ii) `MAD-G-seen` languages were not part of mBERT's pre-training but are included in MAD-G training; and (iii) `unseen` languages are those not included in mBERT pre-training nor in MAD-G training.

The evaluation is confined to syntactic and word-level semantic tasks because there was a shortage of datasets for higher-level "natural language understanding" (NLU) tasks covering truly low-resource languages at the time this project was conducted - we introduce more such evaluation datasets in later chapters. However, we see no reason to expect MAD-G's behaviour to differ systematically on NLU tasks compared to the tasks in our evaluation.

3.4.2 Baselines and MAD-G Variants

`mBERT` is an MMT pre-trained on the Wikipedias of 104 languages. We use mBERT as the base MMT for MAD-G. `XLM-R` is a state-of-the-art MMT pre-trained on the CommonCrawl data of 100 languages (Conneau et al., 2020).[8] We evaluate them in the standard transfer setup with full-model fine-tuning (`-ft`).

`MAD-X` was the contemporary state-of-the-art modular bottleneck adapter-based framework for cross-lingual transfer (Pfeiffer et al., 2020b) based on independent MLM-training of a dedicated LA for each language. We train our own MAD-X LAs when no pre-trained ones are available, notably for the six `MAD-G-seen` UD languages. Training LAs for all

[7] For POS and DP, we omit only (i) languages with scripts unseen in mBERT's pre-training, where mBERT's tokenizer predominantly produces unknown (`UNK`) tokens (Pfeiffer et al., 2021b), (ii) languages lacking any information in URIEL, and (iii) languages whose treebanks have missing fields. For MasakhaNER, we evaluate on all dataset languages except Amharic, as Amharic also uses a script unseen by mBERT.

[8] Although it mostly outperforms mBERT in multilingual and cross-lingual transfer experiments, mBERT was used in prior work as a more robust choice for radically resource-poor languages in general (Pfeiffer et al., 2020b). Our NER experiments on African languages confirm this (Table 3.3 later). Note that MAD-G can be applied to XLM-R as well.

other low-resource languages, however, is prohibitively computationally expensive,[9] so during all MAD-X experiments, the pool of languages with available MAD-X adapters consists of the 20 high-resource source languages used in multi-source setups (see §3.4.3) and `MAD-G-seen` languages. When evaluating on a target language without an available MAD-X LA, we instead choose the available MAD-X LA for the language that is *closest* to the target language.[10]

`MAD-G` is the base setup of our method from §3.3.1. `MAD-G-LS` is the variant of MAD-G in which the adapter generation is additionally conditioned on layer embeddings, as described in §3.3.2. `MAD-G-en` uses the English adapter rather than that of the target language during inference on target language instances. The purpose of this baseline is to test if the parameters generated for different languages are actually meaningfully different and able to outperform the English LA.

`TA-only` trains the task adapter directly on top of the MMT, i.e., without any language adapter. With this baseline, we seek to quantify the contribution of dedicated LAs in general.

3.4.3 MAD-G Training Setup

MLM-training of MAD-G's adapter generator is run on Wikipedias of 95 languages. We considered only the languages with at least 1,000 Wikipedia articles and selected them following a greedy process that maximises typological diversity. At each step, we select the language with the largest number of articles belonging to the language family and its *genus* that are least represented in the current sample of languages (Ponti et al., 2020); see Appendix for a full list.

Following Pfeiffer et al. (2020b), the LA bottleneck size is $m = 384$. Both the language embedding dimension d_l and the layer embedding (if used) dimension d_b are set to 32. Training runs for 200,000 steps in total over all languages; batch size is 64 and the maximum sequence length is 256. We used a linearly decreasing learning rate, starting at 5×10^{-5}. In contrast, relying on the same batch size and max sequence length, MAD-X was trained for 100,000 steps *for each language*. This makes the average per-language duration of MAD-G training $\approx$*50 times shorter* than for MAD-X. Moreover, `MAD-G` and `MAD-G-LS` have 226M and 38M parameters respectively, compared to 728M for a hypothetical 95 MAD-X dedicated language adapters.

[9] Note that this efficiency and scalability shortcoming of MAD-X is precisely one of the main motivations for MAD-G, i.e., for language adapter generation for unseen languages.

[10] We quantify the linguistic proximity of languages as the cosine similarity between their respective URIEL-based language vectors (Lauscher et al., 2020).

source	method	Part-of-speech tagging			Dependency parsing		
		mBERT-seen	MAD-G-seen	unseen	mBERT-seen	MAD-G-seen	unseen
en	MAD-G	76.7	65.9	44.4	63.9/49.2	46.3/28.0	34.7/16.8
	MAD-G-LS	77.8	65.2	43.9	64.9/49.9	44.4/26.0	34.7/16.0
	MAD-G-en	76.5	40.5	44.9	66.4/**51.9**	27.6/11.0	35.4/**18.2**
	TA-only	78.4	40.8	**45.5**	**67.0**/51.8	29.6/11.4	**36.0**/18.1
	MAD-X	76.9	**68.8**	43.4	61.5/46.9	**48.6**/**30.8**	33.1/15.7
	mBERT-ft	76.6	38.7	43.9	66.3/51.3	27.8/10.0	34.0/16.4
	XLM-R-ft	**79.6**	46.8	43.6	55.4/42.0	30.0/13.4	31.9/15.5
multi	MAD-G	86.1	71.0	50.4	75.6/65.4	54.4/38.0	40.1/23.1
	MAD-G-LS	86.5	70.0	51.0	76.6/66.5	53.9/36.9	**41.6**/**23.7**
	MAD-G-en	85.8	45.8	50.5	75.8/65.6	33.1/15.2	40.3/23.6
	TA-only	86.8	48.8	51.2	76.9/66.8	35.7/17.0	41.3/**23.7**
	MAD-X	83.7	**73.8**	47.3	74.7/64.2	**58.1**/**42.9**	39.6/22.5
	mBERT-ft	87.4	45.4	51.2	**80.6**/**70.4**	35.5/15.6	41.3/23.4
	XLM-R-ft	**89.4**	53.9	**55.0**	65.5/55.4	36.8/19.4	36.3/21.4

Table 3.2 UD POS tagging accuracy scores and dependency parsing unlabelled/labelled attachment scores for various language adapter and fine-tuning settings. Values are shown as averages over each of the language groups `mBERT-seen`, `MAD-G-seen` and `unseen`, defined in Table 3.1. Task adapters are trained only on English data (*en*, upper part) and 20 diverse, high-resource languages (*multi*, lower part; note that each row corresponds to a *single* model trained on all 20 languages simultaneously). The highest score per column in each of the two setups is in **bold**, the second highest is underlined.

method	hau MAD-G-seen	ibo MAD-G-seen	kin MAD-G-seen	lug unseen	luo unseen	pcm unseen	swa mBERT-seen	wol unseen	yor mBERT-seen	avg.
MAD-G	**77.1**	**69.9**	**66.1**	54.2	32.5	**72.6**	72.6	32.1	**68.8**	**60.7**
MAD-G-LS	72.8	67.5	63.0	**55.7**	**33.3**	72.4	71.3	**36.7**	68.4	60.1
MAD-G-en	44.9	54.5	51.4	50.6	32.9	70.4	69.2	36.4	63.9	52.7
TA-only	43.4	55.7	52.8	47.9	32.8	72.3	68.6	32.1	65.3	52.3
mBERT-ft	43.2	45.5	49.9	49.3	31.6	70.5	65.8	28.1	54.3	48.7
XLM-R-ft†	66.4	45.5	36.1	34.8	31.9	68.4	**74.5**	21.6	33.4	45.8

Table 3.3 F_1 scores on the MasakhaNER dataset for African languages. Task adapter training/model fine-tuning is conducted on the CoNLL 2003 English NER dataset. †`XLM-R-ft` results are as reported by Adelani et al. (2021).

method	Part-of-speech tagging			Dependency parsing		
	mBERT-genus	MAD-G-genus	unseen-genus	mBERT-genus	MAD-G-genus	unseen-genus
MAD-G	49.1	40.6	34.0	38.2/19.7	**28.4**/**13.2**	28.5/11.1
MAD-G-LS	50.0	**40.8**	29.4	38.7/19.2	26.2/11.9	28.5/9.8
MAD-G-en	51.1	37.5	32.2	39.7/**21.4**	24.3/11.1	29.8/**13.2**
TA-only	**51.5**	37.9	33.4	**40.4**/21.3	26.9/11.9	29.0/12.7
MAD-X	49.3	38.3	30.3	37.3/18.8	23.8/9.0	26.5/10.7
mBERT-ft	48.7	37.3	**34.5**	37.6/19.4	23.5/8.6	**29.9**/12.4
XLM-R-ft	50.8	39.1	27.1	34.7/17.7	24.5/10.2	28.4/12.5

Table 3.4 UD POS tagging accuracy scores and dependency parsing unlabelled/labelled attachment scores for for various language adapter/fine-tuning settings. Values are shown as averages over each of the language groups `mBERT-genus`, `MAD-G-genus` and `unseen-genus`. The task adapter is trained only on English data.

The question of how often to sample training examples from each language during training is rather nuanced. The 95 languages we trained on varied in the amount of Wikipedia data available by several orders of magnitude. We first sampled 500,000 sequences from each language's Wikipedia corpus (or all available sequences for languages with fewer than 500,000), as this number was sufficient to ensure that no sequence would be seen more than once during pre-training. Bearing in mind that our aim was to improve performance for the lower-resource languages, we applied additional exponential smoothing with an exponent of 0.5 to the sampling distribution to up-sample lower-resource languages[11]. Thus, at each MLM training step, we randomly sample a batch in a language l with probability proportional to $\min(\text{n\_examples}^{(l)}, 500,000)^{0.5}$. We did not have sufficient computational resources to iterate on this sampling recipe or perform a hyperparameter search, so we simply chose values which yielded an intuitively sensible number of batches per language.

Single- and Multi-Source Transfer. We train task adapters on English data with the English MAD-G adapter. For comparability, we adopt the TA configuration of MAD-X (Pfeiffer et al., 2020b): the bottleneck size is $m = 48$. For POS-tagging and NER we use the standard token-level single-layer multi-class classifier. For DP, we use the shallow variant (Glavaš and Vulić, 2021) of the biaffine dependency parser of Dozat and Manning (2017). For POS tagging and DP, we train on the English EWT treebank. For consistency and comparability with multi-source experiments, we sample 12,000 sentences for training (out of the 12,543 available examples). For NER, we train on the CoNLL 2003 English dataset (Tjong Kim Sang and De Meulder, 2003).[12] For all tasks, we train for 15,000 steps with batch size 8 (roughly 10 epochs) and a linearly decreasing learning rate, starting at 5e-5.

For multi-source transfer experiments, we select 20 typologically diverse high-resource source languages for POS-tagging and DP using the following process: we iterate over the UD languages in the descending order of treebank size and select a language if it belongs to a genus not already represented in the sample.[13] We again sample a total of 12,000 examples (600 per language), training a single model per baseline/MAD-G variant on all examples.

[11] Devlin et al. (2019) applied a slightly less extreme exponent of 0.7 when sampling multilingual training data for mBERT.

[12] As MasakhaNER does not have the `MISC` category, we replace the `B-MISC` and `I-MISC` token tags with the `O` tag in the CoNLL training set. Similarly, we exclude the `DATE` class (i.e., `B-DATE` and `I-DATE` tags) from the MasakhaNER evaluation, because they do not exist in the CoNLL dataset.

[13] For comparability with single-source experiments, we selected English instead of German as the only exception.

3.5 Results and Discussion

In what follows, we focus on reporting and analysing the most important global trends in results with accompanying discussions and side experiments. For completeness, the full results per individual target language are provided in Appendix A.2.

3.5.1 Single-Source Transfer

Relative to all methods which do not employ language adaptation, we find that the use of MAD-G in the primary `MAD-G` and `MAD-G-LS` settings is greatly beneficial on all tasks for `MAD-G-seen` languages in both the single- and multi-source transfer scenarios (see Tables 3.2 and 3.3), with the very parameter-efficient `MAD-G-LS` being only slightly weaker than the base `MAD-G` variant in general, even slightly outperforming it for some languages and transfer setups. Despite having far less capacity per target language, MAD-G retains much of the performance gain of MAD-X on languages seen during language adapter training, showing that MAD-G achieves efficient yet effective language adaptation. The `MAD-G-en` variant does not achieve such gains on `MAD-G-seen` languages, demonstrating that MAD-G does generate meaningfully different adapter parameters for different languages.

The use of MAD-G is not in general beneficial for `mBERT-seen` languages; this is unsurprising since it is unrealistic to believe that mBERT's knowledge of languages observed during its own pre-training can be substantially improved through language adaptation on a much smaller amount of data. At first glance there also does not appear to be any benefit to using MAD-G for `unseen` target languages, except for NER, where gains are substantial. However, averaging the results over all languages in this group does not provide a full picture because it consists of languages whose relationships to those observed during training differ substantially. Therefore, we provide a finer-grained analysis below.

While the use of typological vectors for generating LAs allows MAD-G to learn features which could generalise well to unseen languages, this assumption should mostly hold for unseen languages whose "typological relatives" are available during training. To investigate the effect the degree of typological relatedness has on MAD-G's generalisation ability, we further divide the `unseen` languages into three subgroups: `mBERT-genus` (the 21 languages whose genus matches that of at least one language seen during mBERT pre-training); `MAD-G-genus` (the 4 languages whose genus was not seen during mBERT pre-training but was seen during MAD-G training); `unseen-genus` (the 8 languages whose genus is completely unseen). Table 3.4 shows the POS tagging and DP performance

for each of the three `unseen` subgroups. MAD-G is beneficial on the `MAD-G-genus` subgroup, while its benefits do not extend to the other two subgroups. The results for `mBERT-genus` versus `MAD-G-genus` languages mirror those for `mBERT-seen` versus `MAD-G-seen` languages; in general, mBERT's knowledge of a genus (or specific language) can be improved through language adaptation if and only if that genus/language was not observed during mBERT's pre-training. As expected, the scores on `unseen-genus` languages confirm the intuition that the performance on languages typologically unrelated to any language seen during mBERT and/or MAD-G training cannot be recovered solely on the basis of limited external typological information. For cross-lingual generalisation, the typological diversity of pre-training languages is therefore paramount.

Unfortunately, the power of these results is almost inevitably limited: the set of `MAD-G-genus` languages, where we see most evidence of zero-shot cross-lingual generalisation, consists of only four languages, as a language must satisfy a very specific set of conditions to fall into this category according to our selection strategy. First of all, mBERT is pre-trained on roughly the 100 highest-resource languages. To fall into the `MAD-G-genus` category, a language must be the *second*-highest resource member of a genus not covered by these 100 highest resource languages, and furthermore it must have a Universal Dependencies treebank. If we were able to train our own MMT from scratch, it would be possible to control the set of languages used during pre-training and MAD-G adaptation such that a larger set of languages with a more diverse range of evaluation datasets fell into the `MAD-G-genus` and `unseen` categories, and we could test our hypotheses more carefully.

3.5.2 Multi-Source Transfer

When training on 20 languages while maintaining the overall number of training examples, we observe large gains across all settings and language groups for both POS tagging and DP (see Table 3.2). This suggests that multi-source training yields a more general and language-agnostic representation of the task adapter, thus transferring better to unseen languages. We investigate the effect of multi-source training further in Figure 3.2, where we gradually add languages to the multi-source pool, while (again) maintaining the overall number of training examples. We find that the transition from one language to two languages in the source-pool results in the largest relative performance increase, but the performance still rises with the addition of more languages. In sum, in line with previous findings (Ponti et al., 2021b), our results indicate that the language diversity of training data has strong positive effects on zero-shot transfer across multiple methods and setups. However, it is important to note that, while increasing the number of source

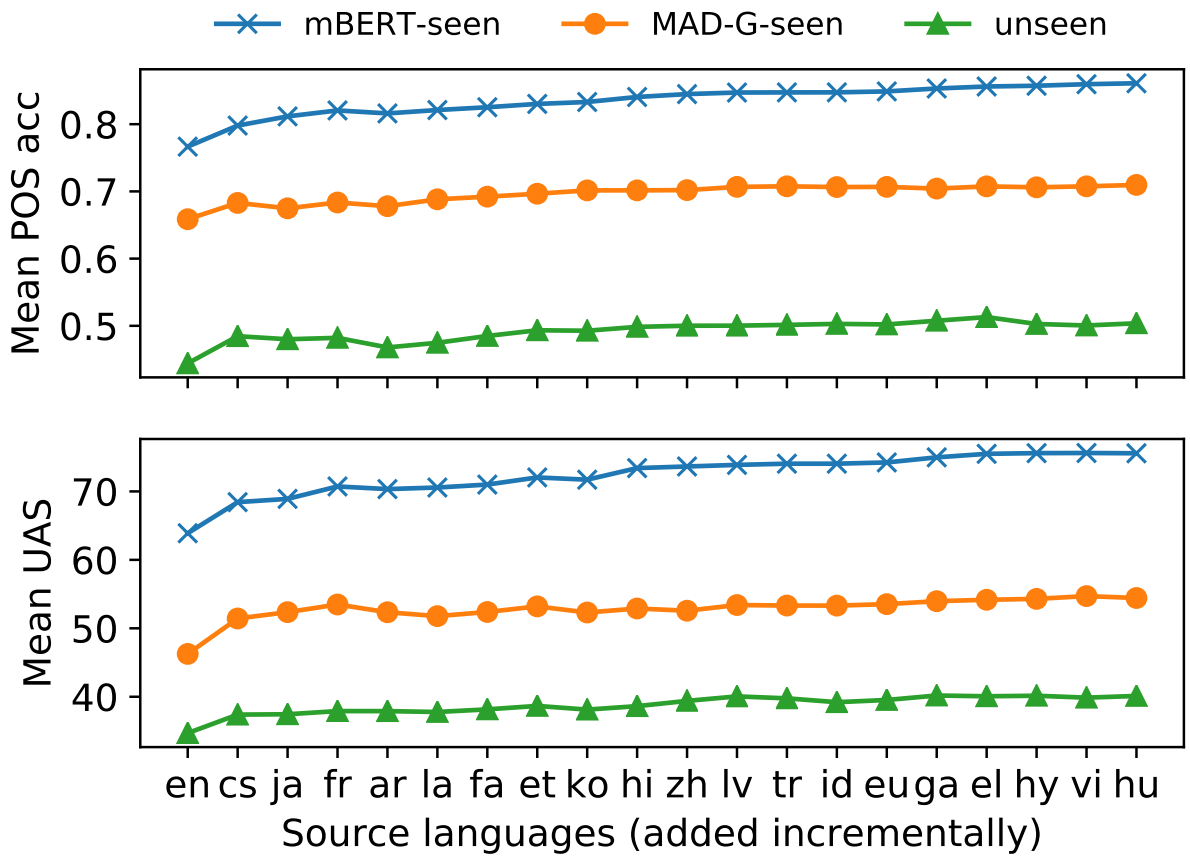


Fig. 3.2 Multi-source transfer with MAD-G. We increase the number of source languages left-to-right from 1 to 20 while keeping the total number of (multi-source) examples constant at each step.

languages improves the task adapter's ability to generalise to new languages, it may on the other hand weaken the performance on the source languages themselves: when the total number of training examples is kept fixed, the task adapter sees fewer examples per source language, and it has less capacity available for each source language. When English is both the target and the single source language, for instance, MAD-G's single source performance on POS tagging and DP respectively is 96.3 accuracy and 89.6/87.0 UAS/LAS, whereas its performance when English is one of 20 source languages is 92.2 accuracy and 82.6/77.5, a large decline. We therefore recommend that when the target language has a large task training set, it is best to train the task adapter primarily or entirely on this data, whereas when a language has little or no task training data, this should be supplemented by data from a diverse as possible range of other languages, perhaps prioritising languages closely related to the target language.

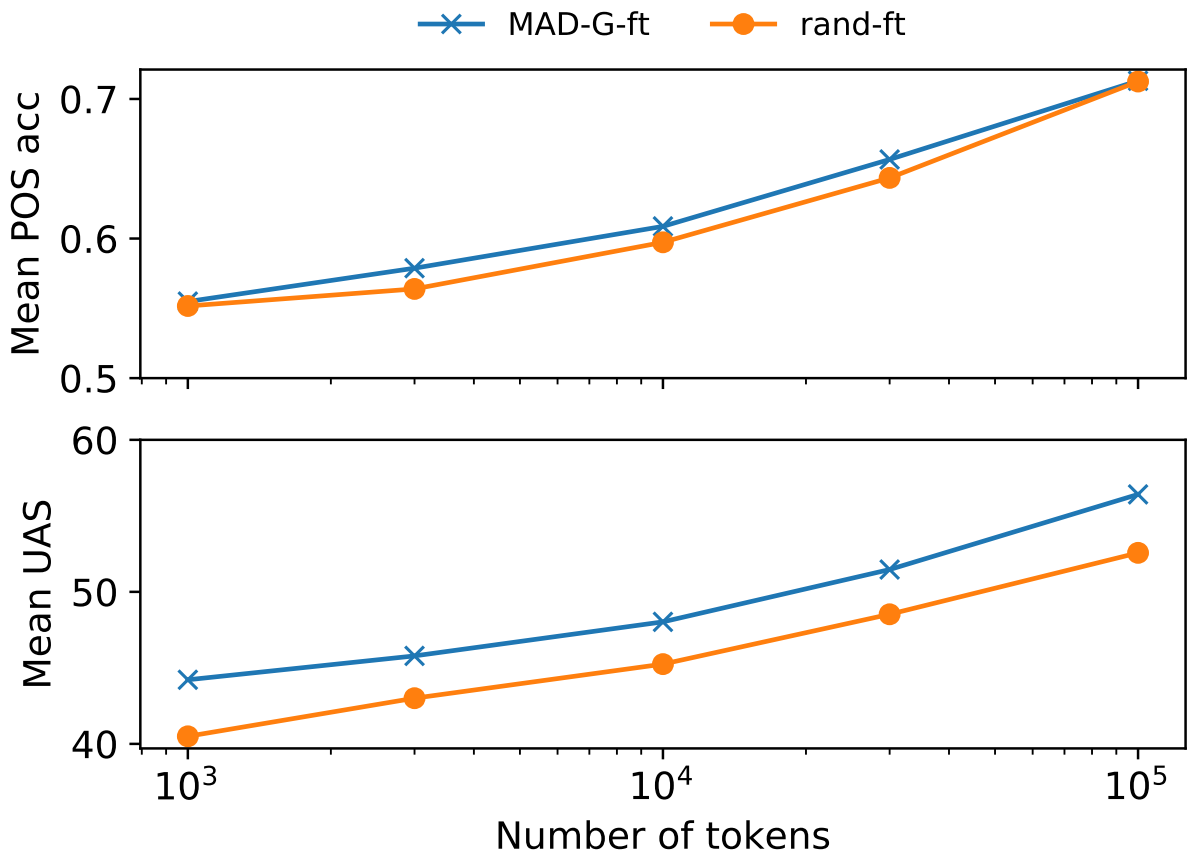


Fig. 3.3 Performance on POS tagging and DP on `unseen` languages when MAD-G-initialised (`MAD-G-ft`) or randomly initialised (`rand-ft`) language adapters are fine-tuned by MLM on varying amounts of unlabelled text.

3.5.3 Fine-tuning MAD-G-Initialised Adapters

Although interesting from a theoretical point of view, the scenario where there is no unannotated data whatsoever available for the target language might be unrealistic. We thus examine a setup where there is a small amount of unannotated data available. In this case, we can still exploit MAD-G by generating an initialisation of a language-specific adapter for a target language l_t, and then fine-tuning its parameters via MLM on the unannotated data.

We perform POS tagging and DP experiments when fine-tuning MAD-G-initialised language-specific adapters on the 14 `unseen` UD languages which have Wikipedias.[14] We simulate different degrees of resource-poverty by sampling training datasets with 1,000, 3,000, 10,000, 30,000 and 100,000 words from the full Wikipedia. We compare this

[14]We do not perform NER experiments because there are only two `unseen` MasakhaNER languages with Wikipedias.

`MAD-G-ft` setting with the results of fine-tuning randomly-initialised LAs on the same data (`rand-ft`).[15] Figure 3.3 shows that there is a large and consistent improvement on the 14 `unseen` evaluation languages as their language adapters are fine-tuned on increasingly large amounts of unannotated text. For both tasks, the performance is better when the language adapter is initialised with the weights generated by MAD-G than when the weights are randomly initialised. The difference between the two settings is modest for POS tagging, but it is larger for DP and is maintained even when 100,000 training tokens are available.

3.5.4 Suggestions for Further Analysis of Typological Features

We have limited ourselves to measuring the effect of employing typological features in terms of downstream performance on syntactic tasks. Here we suggest some potential methodologies for analysing how MAD-G uses these typological features in more detail.

Taking the product WV yields an $n_p \times 289$ matrix, where the i, jth element denotes the strength of the impact the jth typological feature has on the value of the ith generated parameter. Thus the norm of the jth column of this matrix gives an estimate of the overall importance that MAD-G has learned to ascribe to the jth parameter. It would also be possible to group parameters across adapter layers and investigate whether certain typological features are assigned higher and lower weights at different layers.

Another way of measuring the relevance of certain features, especially for cross-lingual transfer, would be to consider `MAD-G-genus` languages and their same-genus cousins introduced during MAD-G training - that is, the language pairs between which we have most expectation of positive transfer due to shared typological features. We could try perturbing each of these shared typological features one at a time by inverting its value in the target language's feature vector. Which features most harm task performance when changed? These are likely most heavily responsible for supporting cross-lingual transfer.

3.6 Conclusion

We have proposed MAD-G, a modular and efficient cross-lingual transfer framework for low-resource languages, that generates task-agnostic bottleneck adapters for multilingual Transformer encoders (e.g., mBERT) from typological language representations. MAD-G

[15] We fine-tune both variants for 200 epochs with batch size 4 and learning rate 5×10^{-5}. We evaluate the fine-tuned language adapters on POS tagging and DP using our better-performing multi-source task adapters trained on 20 languages.

performs competitively with a state-of-the-art adapter-based transfer approach MAD-X; yet its training is roughly 50 times more efficient per target language. MAD-G can also be applied to unseen languages, benefiting those belonging to a genus introduced during its training, and it can be used as a better initialisation for "radically low-resource languages"; there, its generated language adapters can be further refined on small amounts of text, improving downstream performance. We further show that cross-lingual performance with adapters can be greatly improved by training on multiple source languages.

Chapter 4

Composable Sparse Fine-Tuning for Cross-Lingual Transfer

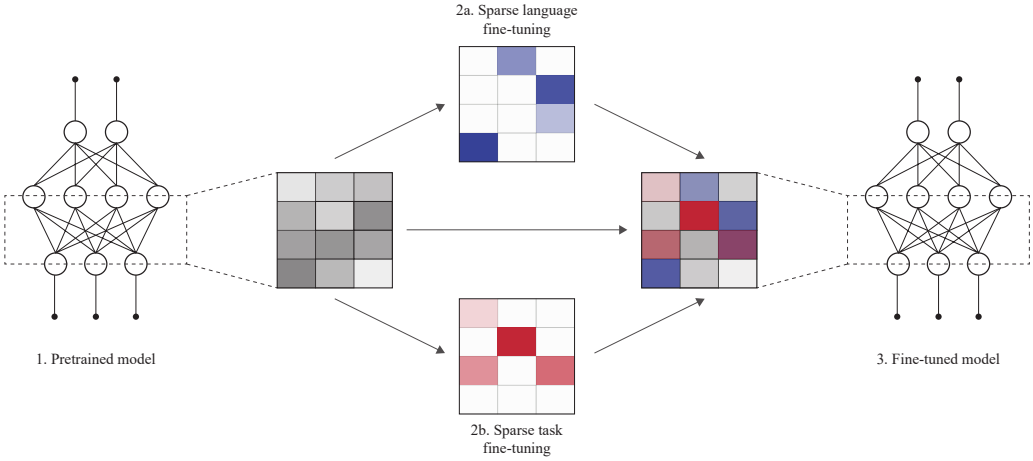


Fig. 4.1 A graphical representation of Lottery Ticket Sparse Fine-Tuning: from the parameters of a pre-trained model (gray, left), we generate sparse fine-tunings for task and language knowledge (blue and red, center). Finally, we sum these three components (right) to obtain the adapted/fine-tuned model. Best viewed in colour.

4.1 Introduction

Chapter 3 focused on *bottleneck adapters* (Rebuffi et al., 2017; Houlsby et al., 2019) as a means of parameter-efficient fine-tuning (PEFT; see §2.2.1) which have additional modularity properties making them ideal for cross-lingual transfer. The ability to disentangle and recombine orthogonal facets of knowledge in original ways (Ponti et al.,

2021a; Ponti, 2021) offers the opportunity to separately learn a task adapter from labelled data in a source language and dedicated language adapters from unlabelled data in the source language and target languages. By stacking these components, it is possible to perform zero-shot cross-lingual transfer. Compared to sequentially fine-tuning the full model on both the task and target language, this yields superior performance and efficiency (Pfeiffer et al., 2020b). Notably, achieving coverage over N_T tasks in N_L target languages with the sequential approach requires $N_T N_L$ models to be trained, whereas the modularity of adapters reduces this to $N_T + N_L$.

An alternative PEFT approach is *sparse* fine-tuning (SFT), where a small subset of the pre-trained model's existing parameters are fine-tuned (Guo et al., 2021; Ben Zaken et al., 2022; Xu et al., 2021, *inter alia*). A potential advantage of SFTs over bottleneck adapters is their *expressivity*: rather than a post-hoc transformation of the output of Transformer layers (e.g., using a shallow MLP as with adapters), they can operate directly on a pre-trained model's embedding and attention layers. In service of the aim of maximising cross-lingual transfer performance, it therefore seems natural to search for a PEFT method that is both modular and expressive.

To this end, we propose Lottery Ticket Sparse Fine-Tuning (LT-SFT), a simple and general-purpose adaptation technique inspired by the Lottery Ticket Hypothesis (LTH; Frankle and Carbin, 2019; Malach et al., 2020), which was originally conceived for pruning large neural networks. In particular, after fine-tuning a pre-trained model for a specific task or language, we select the subset of parameters that changed the most. Then, we rewind the model to its pre-trained initialisation (without setting any value to zero, contrary to the original LTH algorithm). By then tuning only the selected subset of parameters, we obtain a sparse fine-tuning in the form of a vector of differences with respect to the pre-trained model. Multiple SFTs can be *composed* by simply summing them with the pre-trained model. We provide a graphical representation of our method in Figure 4.1.

We benchmark LT-SFT on a series of multilingual datasets, including Universal Dependencies (Zeman et al., 2020) for part-of-speech tagging and dependency parsing, MasakhaNER (Adelani et al., 2021) for named entity recognition, and AmericasNLI (Ebrahimi et al., 2022) for natural language inference. We evaluate it in a zero-shot cross-lingual transfer setting on 35 typologically and geographically diverse languages that include both languages seen and unseen during the base model's pre-training. The results in all transfer tasks indicate that LT-SFT consistently achieves substantial gains over the previous state-of-the-art bottleneck adapter-based method for cross-lingual transfer, MAD-X (Pfeiffer et al., 2020b).

In addition to its superior performance, modularity, and expressivity, LT-SFT offers a number of advantages over bottleneck adapters: 1) the number of parameters remains constant, which prevents the decrease in inference speed observed when adapter layers are added; 2) the neural architecture remains identical to the pre-trained model, which makes code development model-independent rather than requiring special modifications for each possible architecture (Pfeiffer et al., 2020a). Finally, 3) we empirically demonstrate that the peak in performance for LT-SFT is consistently found with the same percentage of tunable parameters, whereas the best reduction factor for MAD-X is task-dependent. This makes our method more robust to the choice of hyper-parameters.

We also find that a high level of sparsity in language and task fine-tunings is beneficial to performance, as this makes overlaps less likely and poses a lower risk of creating interference between the knowledge they contain. Moreover, it makes fine-tunings less prone to overfitting due to their constrained capacity. Thus, sparsity is a fundamental ingredient for achieving modularity and composability. These properties in turn allow for systematic generalisation to new combinations of tasks and languages in a zero-shot fashion.

4.2 Background

Recall from §2.2.1 the use of bottleneck adapter modules for PEFT, and from §2.3.4 the MAD-X framework for training and composing independent task and language adapters for cross-lingual transfer. Also recall the definition of multi-source task training from §3.3.3, and §3.5.2, where we showed that training task adapters using data from multiple source languages can result in sizable improvements in downstream zero-shot transfer performance even when the total number of training examples is held constant. Here we additionally provide a succinct overview of sparse fine-tuning methods, and outline the Lottery Ticket Hypothesis, upon which our newly proposed method is built.

4.2.1 Sparse Fine-Tuning

We call $F' = F(\cdot\ ;\ \boldsymbol{\theta} + \boldsymbol{\phi})$ a *sparse fine-tuning* (SFT) of a pre-trained neural model $F(\cdot\ ;\ \boldsymbol{\theta})$ if $\boldsymbol{\phi}$ is sparse. We sometimes refer to $\boldsymbol{\phi}$ itself as an SFT, or as the SFT's *difference vector*. Previously proposed SFT methods include DiffPruning (Guo et al., 2021), BitFit (Ben Zaken et al., 2022) and ChildTuning (Xu et al., 2021). DiffPruning simulates sparsity of the difference vector during training by applying a continuous relaxation of a binary mask to it. BitFit on the other hand allows non-zero differences

only for bias parameters. ChildTuning selects a subset of fine-tunable parameters by using Fisher information to measure the relevance of each parameter to the task. These methods have proved competitive with full fine-tuning on GLUE (Wang et al., 2018), even when the difference vector $\boldsymbol{\phi}$ has fewer than 0.5% non-zero values.

4.2.2 Lottery Ticket Hypothesis

The Lottery Ticket Hypothesis (LTH; Frankle and Carbin, 2019; Malach et al., 2020) states that each neural model contains a sub-network (a "winning ticket") that, if trained again in isolation, can match or even exceed the performance of the original model. To achieve this, after a pruning stage where some parameters are zero-masked and frozen according to some criterion (e.g., weight magnitude), the remaining parameters are restored to their original values and then re-tuned. This process of pruning and re-training can be iterated multiple times.

The LTH has so far been used mostly for model *compression* through network pruning; to our knowledge, we are the first to use it for pre-trained model *adaptation*.

4.3 Methodology

4.3.1 Lottery Ticket Sparse Fine-Tuning

Training. In this work, we propose Lottery Ticket Sparse Fine-Tuning (LT-SFT). Similar to the Lottery Ticket algorithm of Frankle and Carbin (2019), our LT-SFT method consists of two phases:

1. Pre-trained model parameters $\boldsymbol{\theta}^{(0)}$ are fully fine-tuned on the target language or task data $\mathcal{D}$, yielding $\boldsymbol{\theta}^{(1)}$. Parameters are ranked according to some criterion, in our case greatest absolute difference $|\theta_i^{(1)} - \theta_i^{(0)}|$, and the top K are selected for tuning in the next phase: a binary mask $\boldsymbol{\mu}$ is set to have 1 in positions corresponding to these parameters, and 0 elsewhere.

2. After resetting the parameters to their original values $\boldsymbol{\theta}^{(0)}$, the model is again fine-tuned, but this time only the K selected parameters are trainable whereas the others are kept frozen. In practice, we implement this by passing the *masked* gradient $\boldsymbol{\mu} \odot \nabla_{\boldsymbol{\theta}} \mathcal{L}(F(\cdot\ ;\ \boldsymbol{\theta}), \mathcal{D})$ (where $\odot$ denotes element-wise multiplication and $\mathcal{L}$ a loss function) to the optimizer at each step. From the resulting fine-tuned parameters $\boldsymbol{\theta}^{(2)}$ we can obtain the sparse vector of differences $\boldsymbol{\phi} = \boldsymbol{\theta}^{(2)} - \boldsymbol{\theta}^{(0)}$.

We also apply a regularisation term which discourages parameters from deviating from their pre-trained values $\boldsymbol{\theta}^{(0)}$. Specifically, we use L1 regularisation of the form $J(\boldsymbol{\theta}) = \frac{\lambda}{N}\sum_i |\theta_i - \theta_i^{(0)}|$, where N is the total number of model parameters[1].

Composition. Although we often use the term "sparse fine-tuning" to refer to the difference vector $\boldsymbol{\phi}$ itself, an SFT is most accurately conceptualised as a functional which takes as its argument a parameterised function and returns a new function, where some sparse difference vector $\boldsymbol{\phi}$ has been added to the original parameter vector. Suppose we have a language SFT S_L and a task SFT S_T defined by

$$S_L(F(\cdot\ ;\ \boldsymbol{\theta})) = F(\cdot\ ;\ \boldsymbol{\theta} + \boldsymbol{\phi}_L)$$
$$S_T(F(\cdot\ ;\ \boldsymbol{\theta})) = F(\cdot\ ;\ \boldsymbol{\theta} + \boldsymbol{\phi}_T).$$

Then we have

$$S_L \circ S_T(F(\cdot\ ;\ \boldsymbol{\theta})) = F(\cdot\ ;\ \boldsymbol{\theta} + \boldsymbol{\phi}_T + \boldsymbol{\phi}_L).$$

Algorithm 1 Cross-Lingual Transfer with Lottery-Ticket Sparse Fine-Tuning

function LtSft($\mathcal{D}$, $\mathcal{L}$, $\boldsymbol{\theta}^{(0)}$, η, K)
  $\boldsymbol{\theta}^{(1)} \leftarrow \boldsymbol{\theta}^{(0)}$
  while not converged **do**
    $\boldsymbol{\theta}^{(1)} \leftarrow \boldsymbol{\theta}^{(1)} - \eta\nabla\mathcal{L}(\boldsymbol{\theta}^{(1)}, \mathcal{D})$
  $\mu_i \leftarrow \begin{cases} 1 & \text{if } \theta_i^{(1)} \in \operatorname{argmax}_{\theta_1,\ldots,\theta_K} |\boldsymbol{\theta}^{(1)} - \boldsymbol{\theta}^{(0)}| \\ 0 & \text{otherwise} \end{cases}$
  $\boldsymbol{\theta}^{(2)} \leftarrow \boldsymbol{\theta}^{(0)}$
  while not converged **do**
    $\boldsymbol{\theta}^{(2)} \leftarrow \boldsymbol{\theta}^{(2)} - \boldsymbol{\mu} \odot \eta\nabla\mathcal{L}(\boldsymbol{\theta}^{(2)}, \mathcal{D})$
  $\boldsymbol{\phi} \leftarrow \boldsymbol{\theta}^{(2)} - \boldsymbol{\theta}^{(0)}$
  return $\boldsymbol{\phi}$

function CrossLingualTransfer($\mathcal{D}_{\text{src}}$, $\mathcal{D}_{\text{tar}}$, $\mathcal{D}_{\text{task}}$, $\mathcal{L}_{\text{task}}$, $\boldsymbol{\theta}^{(0)}$, η, K)
  $\boldsymbol{\phi}_{\text{src}} \leftarrow \text{LtSft}(\mathcal{D}_{\text{src}}, \mathcal{L}_{\text{MLM}}, \boldsymbol{\theta}^{(0)}, \eta, K)$
  $\boldsymbol{\phi}_{\text{task}} \leftarrow \text{LtSft}(\mathcal{D}_{\text{task}}, \mathcal{L}_{\text{task}}, \boldsymbol{\theta}^{(0)} + \boldsymbol{\phi}_{\text{src}}, \eta, K)$
  $\boldsymbol{\phi}_{\text{tar}} \leftarrow \text{LtSft}(\mathcal{D}_{\text{tar}}, \mathcal{L}_{\text{MLM}}, \boldsymbol{\theta}^{(0)}, \eta, K)$
  return $\boldsymbol{\theta}^{(0)} + \boldsymbol{\phi}_{\text{task}} + \boldsymbol{\phi}_{\text{tar}}$

[1]It would be equally possible to use L2 regularisation here, but we did not test this formally as we considered ablations on other hyperparameters to be a higher priority. The main advantage of using the L1 loss was that it seemed easier to guess at a reasonable value for λ when the regularisation loss was on a similar scale to the MLM loss.

4.3.2 Zero-Shot Transfer with LT-SFT

We adopt a similar cross-lingual transfer setup to MAD-X (Pfeiffer et al., 2020b). We start with an MMT F with pre-trained parameters $\boldsymbol{\theta}$ learned through masked language modelling on many languages, such as mBERT (Devlin et al., 2019) or XLM-R (Conneau et al., 2020).

For each language of interest l, we learn a language SFT $\boldsymbol{\phi}_L^{(l)}$ through LT-SFT (also with an MLM objective) on text from language l.

For each task of interest t, we learn a task SFT $\boldsymbol{\phi}_T^{(t)}$ through LT-SFT on annotated data from some source language s. When learning the task SFT, we first adapt to the source language by applying the language SFT for s.[2] The language SFT is removed again after training. That is, we perform LT-SFT on $F(\cdot\ ;\ \boldsymbol{\theta}+\boldsymbol{\phi}_L^{(s)})$ to obtain fine-tuned parameter vector $\boldsymbol{\theta}'$. We then calculate $\boldsymbol{\phi}_T^{(t)} = \boldsymbol{\theta}' - (\boldsymbol{\theta}+\boldsymbol{\phi}_L^{(s)})$. Note that during task training, we also learn a classifier head, which is fully fine-tuned during both phases of LT-SFT adaptation, with the same random initialisation applied at the beginning of each phase.

We perform zero-shot adaptation of F to target language l for task t by composing language and task SFTs to obtain $F_{t,l} = F(\cdot\ ;\ \boldsymbol{\theta}+\boldsymbol{\phi}_T^{(t)}+\boldsymbol{\phi}_L^{(l)})$. On top of this, we stack the classifier head learned for t. We provide a formal algorithm of LT-SFT and the transfer procedure in Algorithm 1.

4.4 Experimental Setup

To evaluate our new method extensively, we benchmark its zero-shot cross-lingual performance on four distinct tasks: part-of-speech tagging (POS), dependency parsing (DP), named entity recognition (NER), and natural language inference (NLI). Table 4.1 summarises our experimental setup, including the datasets and languages considered in our experiments. We put emphasis on low-resource languages and languages unseen during MMT pre-training, although we also evaluate on a few high-resource languages. In total, we cover a set of 35 typologically and geographically diverse languages, which makes them representative of cross-lingual variation (Ponti et al., 2019a, 2020).

[2] Adapting to the source language yields substantial improvements in cross-lingual transfer performance with both MAD-X and LT-SFT, with gains of 2-3 points in our preliminary experiments. Paradoxically, our results (see Table 4.5) and results from previous work (Pfeiffer et al., 2020b; Ansell et al., 2021) suggest that adapting to high-resource *target* languages at inference time does not give similarly large benefits. We think this phenomenon warrants further investigation.

Task	Target Dataset	Source Dataset	MMT	Target Languages
Part-of-Speech Tagging (POS), Dependency Parsing (DP)	Universal Dependencies 2.7 (Zeman et al., 2020)	Universal Dependencies 2.7 (Zeman et al., 2020)	mBERT	Arabic†, Bambara, Buryat, Cantonese, Chinese†, Erzya, Faroese, Japanese†, Livvi, Maltese, Manx, North Sami, Komi Zyrian, Sanskrit, Upper Sorbian, Uyghur
Named Entity Recognition (NER)	MasakhaNER (Adelani et al., 2021)	CoNLL 2003 (Tjong Kim Sang and De Meulder, 2003)	mBERT	Hausa, Igbo, Kinyarwanda, Luganda, Luo, Nigerian-Pidgin, Swahili*, Wolof, Yorùbá*
Natural Language Inference (NLI)	AmericasNLI (Ebrahimi et al., 2022)	MultiNLI (Williams et al., 2018)	XLM-R	Aymara, Asháninka, Bribri, Guarani, Náhuatl, Otomí, Quechua, Rarámuri, Shipibo-Konibo, Wixarika

Table 4.1 Details of the tasks, datasets, MMTs and languages involved in our zero-shot cross-lingual transfer evaluation. * denotes low-resource languages seen during MMT pre-training; † denotes high-resource languages seen during MMT pre-training; all other languages are low-resource and unseen. The source language is always English. Further details of all the language and data sources used are provided in Appendix B.1.

4.4.1 Baselines and Model Variants

The main baseline is `MAD-X`, the state-of-the-art bottleneck adapter-based framework for cross-lingual transfer (Pfeiffer et al., 2020b). We use the "MAD-X 2.0" variant, where the last adapter layers are dropped. Pfeiffer et al. (2021b) found that this improved performance, which we could confirm in our preliminary experiments. Since adapters with the configuration used by Pfeiffer et al. (2020b) are unavailable for many languages in our evaluation, we train our own for all languages. In Appendix B.3 we also provide an evaluation with comparable language adapters from AdapterHub (Pfeiffer et al., 2020a) where available.

We also perform experiments with `BitFit` (Ben Zaken et al., 2022) to establish a baseline for an existing SFT technique. In addition to the main `LT-SFT` model variant, on POS and DP we test a `rand-SFT` variant as an ablation, where the K parameters to be fine-tuned are selected at random rather than based on an informed criterion. We did not consider MAD-G from Chapter 3 as a baseline because its main strengths are training efficiency and the ability to adapt to unseen languages, whereas the primary goal of LT-SFT is the maximisation of performance, for which MAD-X provides a stronger challenge.

For both LT-SFT and MAD-X, we also evaluate a task adaptation `(TA)-only` configuration, where only the task module is applied, without the target language module.

4.4.2 Language Module Training Setup

MLM Training Data. For all languages in our POS and DP evaluation, we perform MLM language module (i.e. SFT/bottleneck adapter) training on Wikipedia corpora. We also use Wikipedia for all languages in our NER evaluation if available. Where this is not the case, we use the Luo News Dataset (Adelani et al., 2021) for Luo and the JW300 corpus (Agić and Vulić, 2019) for Nigerian Pidgin. The main corpora for the languages in our NLI evaluation are those used by the dataset creators to train their baseline models (Ebrahimi et al., 2022); however, since the sizes of these parallel corpora are small, we further augment them with data from Wikipedia and the corpora of indigenous Peruvian languages of Bustamante et al. (2020) where available. More details on data sources are provided in Appendix B.1.

Training Setup and Hyper-parameters. For both SFTs and adapters, we train for the lesser of 100 epochs or 100,000 steps of batch size 8 and maximum sequence length 256, subject to an absolute minimum of 30,000 steps since 100 epochs seemed insufficient for some languages with very small corpora. Model checkpoints are evaluated every 1,000 steps (5,000 for high-resource languages) on a held-out set of 5% of the corpus (1% for high-resource languages), and the one with the smallest loss is selected at the end of training. We use the Adam optimizer (Kingma and Ba, 2015) with an initial learning rate of 5×10^{-5} which is linearly reduced to 0 over the course of training.

Following Pfeiffer et al. (2020b), the reduction factor (i.e., the ratio between model hidden size and adapter bottleneck size) for the adapter baseline was set to 2 for a total of ∼7.6M trainable parameters. For comparability, we set the same number of trainable parameters K for our language LT-SFTs. This results in language SFTs with a density of 4.3% for mBERT and 2.8% for XLM-R. Since `BitFit` exclusively tunes the bias parameters, its language SFTs have a fixed density of 0.047% for mBERT and 0.030% for XLM-R.

Importantly, during language sparse fine-tuning, we decouple the input and output embedding matrices and fix the parameters of the output matrix; otherwise, we find that the vast majority of the K most changed parameters during full fine-tuning belong to the embedding matrix, seemingly due to its proximity to the model output, which damages downstream performance. We also fix the layer normalisation parameters; all other parameters are trainable. For language adaptation, we apply L1 regularisation as

described in §4.3.1 with $\lambda = 0.1$. Note that the specified training regime is applied in the same way during both phases of LT-SFT.

For language adapter training in the MAD-X baseline, we use the Pfeiffer configuration (Pfeiffer et al., 2021a) with invertible adapters, special additional sub-components designed for adapting to the vocabulary of the target language.

For simplicity, we use the same hyperparameters during both phases of LT-SFT training. However, subsequent practical experience has suggested that the sparse phase remains stable at a higher learning rate than the dense phase, so we recommend that future work consider using a different learning rate for the two phases. Relatedly, in our preliminary experiments we considered shortening the dense phase of language SFT training (training for a maximum of 30,000 steps instead of 100,000) in the hope that this would not compromise the selection of parameters. We found a small but seemingly significant reduction in downstream performance - however, the trade-off between training efficiency and test-time performance might be acceptable in some cases. In subsequent work, we also considered a class of more general schemes with S parameter selection and sparsification steps, where (for instance) after each step the $\frac{|\boldsymbol{\theta}|-K}{S}$ active parameters least changed from their pre-trained values are reset and then frozen. We hoped that by resetting and freezing parameters gradually instead of "losing progress" by completely resetting them in the middle of training, it would be possible to greatly reduce the length of training. However, we found that it was difficult to replicate the performance of LT-SFT like this when the number of training steps was significantly reduced, as many training steps are required to "recover" after resetting a group of parameters.

4.4.3 Task Module Training Setup

For POS tagging, DP, and NER,[3] we train task SFTs/adapters on the datasets indicated in Table 4.1 for 10 epochs with batch size 8, except during the first phase of LT-SFT training where we train for only 3 epochs.[4] Model checkpoints are evaluated on the validation set every 250 steps, and the best checkpoint is taken at the end of training, with the selection metric being accuracy for POS, labelled attachment score for DP, and F1-score for NER. Similarly to language fine-tuning, we use an initial learning rate of

[3]MasakhaNER and CoNLL 2003 datasets respectively use the `DATE` and `MISC` tags which are not used by the other; we replace these with the `O` tag at both train and test time.

[4]This is because full fine-tuning is more prone to overfitting than parameter-efficient fine-tuning. Early stopping somewhat addresses overfitting, but it is insufficient in a cross-lingual setting because the target language performance generally starts to deteriorate faster than the source language performance. As per our earlier comment, it may also be possible to address this by applying a lower learning rate during the dense phase of training.

5×10^{-5} which is linearly reduced to 0 over the course of training. For POS and NER we use the standard token-level single-layer multi-class model head. For DP, we use the shallow variant (Glavaš and Vulić, 2021) of the biaffine dependency parser of Dozat and Manning (2017).

For NLI, we employ the same fine-tuning hyper-parameters as Ebrahimi et al. (2022): 5 epochs with batch size 32, with checkpoint evaluation on the validation set every 625 steps, and an initial learning rate of 2×10^{-5}. We apply a two-layer multi-class classification head atop the MMT output corresponding to the `[CLS]` token.

We found that the number of trainable parameters during task adaptation (governed by K for SFTs and reduction factor for adapters) has a large effect on performance: we thus experiment with a range of values. Specifically, we test adapter reduction factors of 32, 16, 8, 4, 2, and 1, and equivalent values of K[5] for SFT.

During task adaptation, we always apply the source language adapter following Pfeiffer et al. (2020b), or source language SFT (see §4.3.2).

4.4.4 Multi-Source Training

To validate that task LT-SFT training, like task adapter training in prior work (Ansell et al., 2021), benefits from the presence of multiple source languages in the training data, and to push the boundaries of zero-shot cross lingual transfer, we perform multi-source training experiments on DP and NLI.

We adopt a similar setup as in §3.4.3: we obtain the training set by concatenating the training data for all source languages. We randomly shuffle the training set and train as in the single-source case, except that each batch is composed of examples from a single source language, whose language SFT is applied during the training step.

We prioritise maximising performance rather than providing a fair comparison against the single-source case, so unlike Ansell et al. (2021), we use the entirety of the training sets, other than that we set a maximum of 15K examples per language for DP to better balance our sample.

For DP, we train our models on the UD treebanks of 11 diverse high-resource languages. For NLI, we train on MultiNLI (Williams et al., 2018) plus the data for all 14 non-English languages in the XNLI dataset (Conneau et al., 2018).

We also evaluate multi-source task SFT training on extractive question answering (QA), as a comparatively generous amount of multilingual data is available for this task.

[5]Approximately 442K, 884K, 1.7M, 3.5M, 7.1M, and 14.2M respectively, amounting to sparsity levels of 0.25%, 0.50%, 1.0%, 2.0%, 4.0% and 8.0% for mBERT and 0.16%, 0.32%, 0.63%, 1.3%, 2.6% and 5.1% for XLM-R.

	POS	DP		NER	NLI
	Accuracy	UAS	LAS	F1 score	Accuracy
`LT-SFT`	**71.1** (1)	**57.1** (1)	**37.8** (1)	**71.7** (1)	**51.4** (1)
`rand-SFT`	69.2 (1)	54.3 (1)	33.9 (1)	-	-
`MAD-X`	68.6 (16)	54.6 (2)	34.1 (1)	69.9 (8)	49.5 (2)
`BitFit`	58.1	45.7	23.9	54.9	38.3
`LT-SFT TA-only`	51.3 (32)	39.1 (1)	19.9 (1)	55.3 (8)	39.9 (4)
`MAD-X TA-only`	52.1 (32)	38.9 (1)	19.5 (1)	52.4 (32)	41.7 (4)

Table 4.2 Results of zero-shot cross-lingual transfer evaluation averaged over all languages when best equivalent reduction factor (shown in parentheses after each result) is chosen.

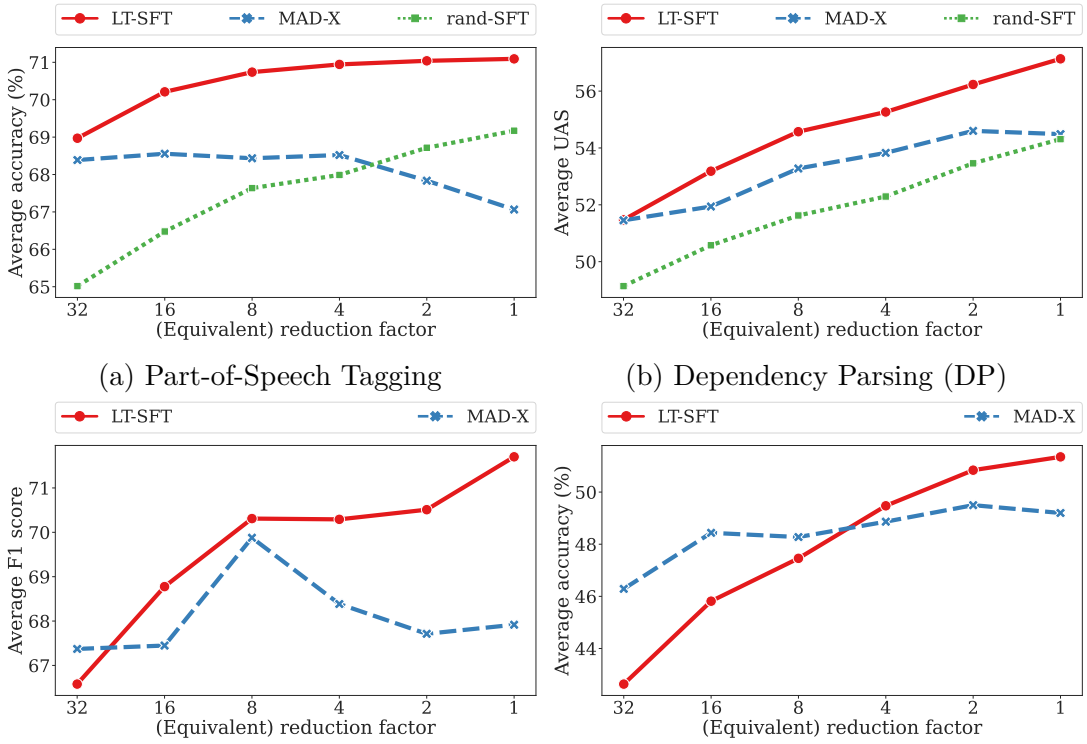
Specifically, we train on English data from SQuAD version 1 (Rajpurkar et al., 2016), all languages from MLQA (Lewis et al., 2020b), and those languages from XQuAD (Artetxe et al., 2020b) which also appear in MLQA. We evaluate on the languages present in XQuAD but not in MLQA. For QA, we train for 5 epochs with batch size 12 and initial learning rate 3×10^{-5}. Full details of the source languages can be found in Appendix B.1.

We use an equivalent reduction factor of 1 for all tasks, following the strongest setting from our single-source experiments. Except as stated above, the training configuration and hyper-parameters are the same as for single-source training.

4.5 Results and Discussion

We report the average test performance of zero-shot cross-lingual transfer for the best reduction factor (or equivalent K) in Table 4.2. Some patterns emerge across all four tasks: first, `LT-SFT` consistently outperforms all the baselines. In particular, it surpasses the state-of-the-art `MAD-X` across all tasks, with gains of 2.5 accuracy in part-of-speech tagging, 2.5 UAS and 3.7 LAS in dependency parsing, 1.8 F1 score in named entity recognition, and 1.9 accuracy in natural language inference. Compared to `rand-SFT`, its superior performance demonstrates the importance of selecting "winning tickets" rather than a random subset of parameters. Secondly, the results demonstrate the importance of language modules for specialising pre-trained models to unseen languages, as they bring about a large increase in performance across the 4 tasks compared to the corresponding settings with task adaptation only (`TA-only`).

We remark that LT-SFT's zero-shot performance also exceeds translation-based baselines on the AmericasNLI task, achieving an average accuracy of 51.4%, compared with the 48.7% of the "translate-train" baseline of Ebrahimi et al. (2022).



(a) Part-of-Speech Tagging

(b) Dependency Parsing (DP)

(c) Named Entity Recognition (NER)

(d) Natural Language Inference (NLI)

Fig. 4.2 Zero-shot cross-lingual transfer evaluation of Lottery-Ticket Sparse Fine-Tuning (`LT-SFT`), Random Sparse Fine-Tuning (`rand-SFT`), and bottleneck adapter-based `MAD-X` over four tasks with varying numbers of trainable parameters during task adaptation. Results are averages over all target languages.

In Figure 4.2, we provide a more detailed overview of average cross-lingual model performance across a range of different reduction factors. The results for the `LT-SFT` and `rand-SFT` methods generally improve or stay steady as the number of trainable task parameters increases. On the contrary, there is not such a trend for `MAD-X`, as lower reduction factors may degrade its results. This makes it easier to choose a good setting for this hyper-parameter when using SFT. Moreover, it is worth stressing again that, contrary to `MAD-X`, this hyper-parameter does not affect inference time.

`BitFit` performs much worse than the other methods which perform language adaptation across all tasks. Bearing in mind the strong trend towards increasing performance with increasing K for the other SFT methods, it seems likely that `BitFit`, with two orders of magnitude fewer trainable parameters, lacks the capacity to learn effective task and language SFTs.

	el	ro	ru	th	tr
XLM-R Base, full FT	71.1/54.3	78.3/63.7	74.1/57.8	67.1/55.7	67.5/51.1
XLM-R Large, full FT	79.8/61.7	83.6/69.7	80.1/64.3	74.2/62.8	**75.9/59.3**
XLM-R Base MS, LT-SFT	**81.9/65.5**	**86.3/73.3**	**81.4/64.6**	**82.4/75.2**	75.2/58.6

Table 4.3 Results of zero-shot cross-lingual transfer evaluation on XQuAD (Artetxe et al., 2020b), restricted to languages which do not appear in MLQA (Lewis et al., 2020b) (see §4.4.4) in the format F1/exact match score. "Full FT" denotes full fine-tuning, MS denotes multi-source training, where additional data from MLQA and XQuAD is utilised, LT-SFT denotes Lottery-Ticket Sparse Fine-Tuning. The full FT results are due to Artetxe et al. (2020b) for XLM-R Large, and our own experiments for XLM-R Base.

	DP UAS	DP LAS	NLI Accuracy
`single source`	57.1	37.8	51.4
`multi-source`	**64.3**	**47.6**	**53.1**

Table 4.4 Results of zero-shot cross-lingual transfer evaluation of single- vs. multi-source LT-SFT task training averaged over all target languages.

For additional results at the level of individual languages and an analysis of the efficacy of language adaptation for high- versus low- resource target languages, I refer the reader to Appendix B.2.

4.5.1 Multi-Source Training

As shown in Table 4.4, multi-source LT-SFT training brings about a large improvement in zero-shot cross-lingual transfer performance on DP, and a modest improvement for NLI. This may be a result of the fact that the training set for NLI contains a relatively small number of non-English examples compared to the DP training set. Also, the AmericasNLI target languages generally have a lower degree of genealogical relatedness to the source languages compared to the DP target languages.

Table 4.3 demonstrates that multi-source training is also beneficial to zero-shot cross-lingual transfer for QA on a series of relatively high-resource languages. In particular, LT-SFT multi-source training of XLM-R Base outperforms single-source full fine-tuning of XLM-R Large (a larger model) comfortably, and outperforms XLM-R Base single-source full fine-tuning by a significant margin. The fact that such an improvement occurs despite each of the 6 non-English source languages having more than an order of magnitude less training data than the English data from SQuAD illustrates the disproportionate advantage of multilingual source data.

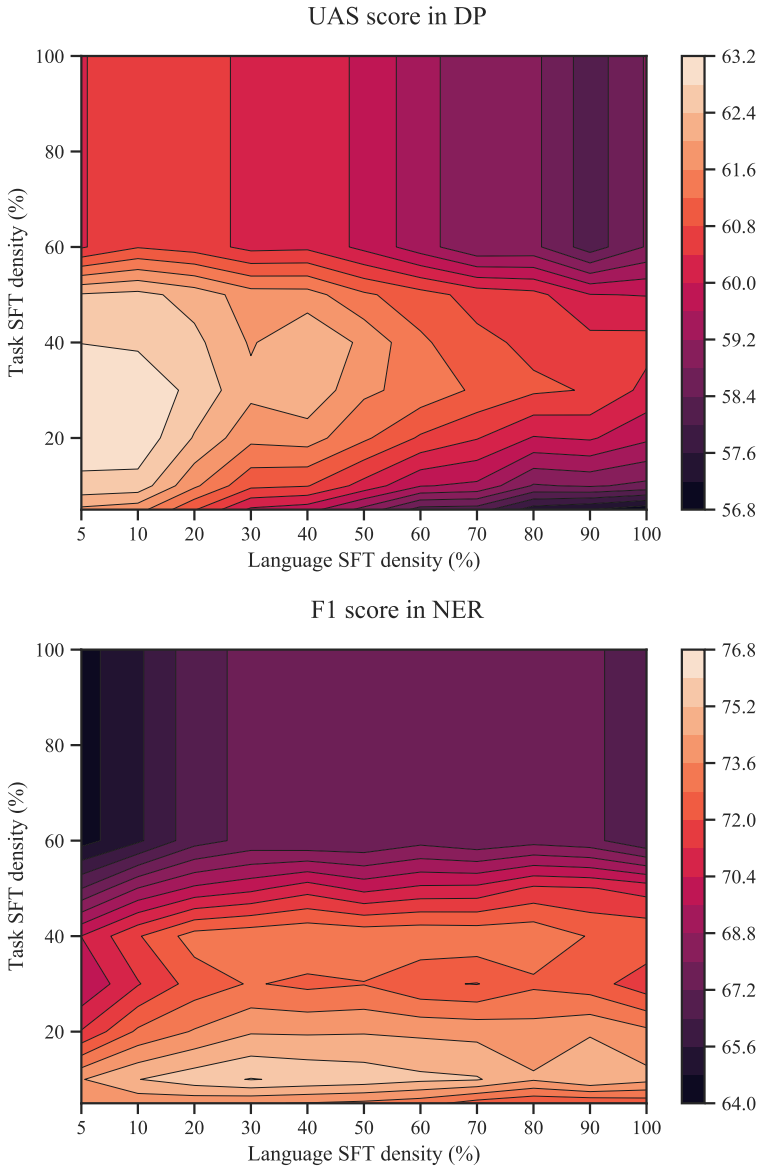


Fig. 4.3 Performance of LT-SFT on DP and NER controlling for the sparsity of task and language fine-tuning. Results are averaged over several selected languages. Denser fine-tunings may interfere with each other and consequently degrade the model performance.

4.5.2 Benefits of Sparsity

We now address the following question: is sparsity responsible for preventing the interference of separate fine-tunings when they are composed? To support this hypothesis with empirical evidence, we use LT-SFT to train language[6] and task fine-tunings with different

[6]To reduce computational cost, we train language fine-tunings for a maximum of 30K steps rather than the 100K of our main experiments.

levels of density, i.e. the percentage of non-zero values (from 5% to 100%). We then evaluate all possible combinations of density levels. The results are visualised in the form of a contour plot in Figure 4.3 for selected combinations of tasks and languages: Buryat, Cantonese, Erzya, Maltese, and Upper Sorbian for DP, and Hausa, Igbo, Luganda, Swahili and Wolof for NER.

From Figure 4.3, it emerges that the performance decreases markedly for SFTs with a density level greater than ~30% of fine-tuned parameters.[7] We speculate that this is due to the fact that sparser fine-tunings have a lower risk of overlapping with each other, thus creating interference between the different facets of knowledge they encapsulate. It must be noted, however, that alternative hypotheses could explain the performance degradation in addition to parameter overlap, such as overfitting as a result of excessive capacity. While we leave the search for conclusive evidence to future work, both of these hypotheses illustrate why enforcing sparsity in adaptation, as we propose in our method, is crucial to achieving modularity.

An experimental method to disentangle the effects of overfitting and parameter interference would need to hold the capacity of the language and task SFTs constant while varying the degree of overlap between masks (or vice versa). One possible algorithm for selecting the active parameters for language and task SFTs of size K_l and K_t respectively with K_s shared parameters would be to first select the K_s parameters with greatest total magnitude change $\boldsymbol{\delta}_{\text{total}}$ during the first phase of the respective fine-tunings, i.e. $\boldsymbol{\delta}_{\text{total}} = |\boldsymbol{\theta}_l^{(1)} - \boldsymbol{\theta}^{(0)}| + |\boldsymbol{\theta}_t^{(1)} - \boldsymbol{\theta}^{(0)}|$, where $\boldsymbol{\theta}_l^{(1)}$ and $\boldsymbol{\theta}_l^{(1)}$ are the model parameters after full fine-tuning on the language and task respectively[8]. The remaining active parameters for each SFT would be made up from the top most changed parameters for its individual full fine-tuning run, avoiding already selected parameters and further overlaps. The resulting SFTs could be trained for a range of densities and overlap levels. A strong trend of decreasing performance with increasing overlap at the same density level would suggest that interference is a primary contributor to the findings from Figure 4.3; a trend of decreasing performance at higher densities for a fixed level of overlap would point to overfitting as a primary contributor. It is entirely possible that both trends would be observed.

[7] Note, furthermore, that levels of task fine-tuning density greater than ~60% do not vary in performance. This is because their subsets of parameters include embeddings of tokens never encountered during task training, which are therefore never updated even if trainable.

[8] Optionally, the parameter changes could be normalised to account for the fact that language fine-tuning likely changes the parameters much more on average than task fine-tuning, i.e. $\boldsymbol{\delta}_{\text{total}} = \frac{|\boldsymbol{\theta}_l^{(1)} - \boldsymbol{\theta}^{(0)}|}{\sum_i |\boldsymbol{\theta}_{l,i}^{(1)} - \boldsymbol{\theta}_i^{(0)}|} + \frac{|\boldsymbol{\theta}_t^{(1)} - \boldsymbol{\theta}^{(0)}|}{\sum_i |\boldsymbol{\theta}_{t,i}^{(1)} - \boldsymbol{\theta}_i^{(0)}|}$.

Assuming that the degradation in performance at high densities is indeed largely a result of interference due to parameters overlaps, we could address this problem by choosing the active parameters for a pair of language and task SFTs such that there is no overlap. However, this would come at the cost of modularity, since it makes the language SFT specific to a particular task and vice versa when we want it to be possible to compose any pair of language and task SFTs. A more sophisticated approach would be to retain modularity while eliminating overlaps by somehow partitioning all model parameters *a priori* into two groups, those which are allowed to be active for language SFTs, and those which can be active for task SFTs. For instance, recent evidence suggests that the first and last layers of large language models are more important for language adaptation while the middle layers are more important for task adaptation (Bandarkar et al., 2025).

It is also interesting to consider the possibility of composing more than two SFTs. While it is hard to think of a natural application for such a composition in cross-lingual transfer, combining several models with the same architecture into a single model in a way that maintains the attributes of each is the object of "model merging," a field which has attracted significant attention recently (Yang et al., 2024). Ilharco et al. (2023), for instance, compose "task vectors" (equivalent to fully-dense SFTs) for several image classification tasks through addition, finding a model's general image classification performance improves as vectors for more image classification tasks are added. Under the interference hypothesis, it seems reasonable to imagine that sparsity could be beneficial for task vectors, and a strong prior for model merging in general.

4.5.3 Adaptation for Medium-to-High Resource Languages

	POS (accuracy)				**DP (UAS)**				**NER (F1)**		
	ar	ja	zh	avg.	ar	ja	zh	avg.	swa	yor	avg.
`LT-SFT`	68.7	53.9	67.5	63.4	**70.8**	**36.9**	**59.8**	**55.9**	**79.4**	**74.8**	**77.1**
`rand-SFT`	69.3	**54.3**	68.0	63.9	68.7	34.8	58.2	53.9	-	-	-
`MAD-X`	70.1	51.1	67.6	62.9	69.5	33.0	58.5	53.7	77.6	74.0	75.8
`BitFit`	69.8	53.9	**69.2**	**64.3**	64.0	34.3	55.9	51.4	67.4	64.7	66.0
`LT-SFT TA-only`	70.6	54.1	65.9	63.5	68.7	36.0	58.4	54.4	69.5	69.3	69.4
`MAD-X TA-only`	**70.8**	51.2	67.6	63.2	68.6	33.8	59.1	53.8	69.6	66.6	68.1

Table 4.5 Results for zero-shot cross-lingual transfer evaluation of the seen languages included in the POS, DP and NER evaluations. For each method/metric pair, the best equivalent reduction factor from Table 4.2 is used.

Table 4.5 contains per-language results for the languages in our evaluation which were seen during mBERT's pre-training. Arabic, Japanese and Chinese, which were included in the POS/DP evaluation, can be considered high-resource languages; Swahili and Yorùbá, on the other hand, were included in the NER evaluation and are arguably medium-to-low resource. In keeping with previous work, we find that language adaptation benefits seen languages less than unseen languages and—among the former—resource-rich languages less than resource-poor languages. This agrees with the intuition that lower-resource languages have greater scope for improvement through language adaptation due to the fact that they receive less signal during MMT pre-training. Interestingly, `BitFit` performs much more competitively on the high-resource languages than low-resource and unseen languages, suggesting that its lack of capacity is more problematic for language adaptation rather than for task fine-tuning.

4.5.4 Parameter Overlap between Languages

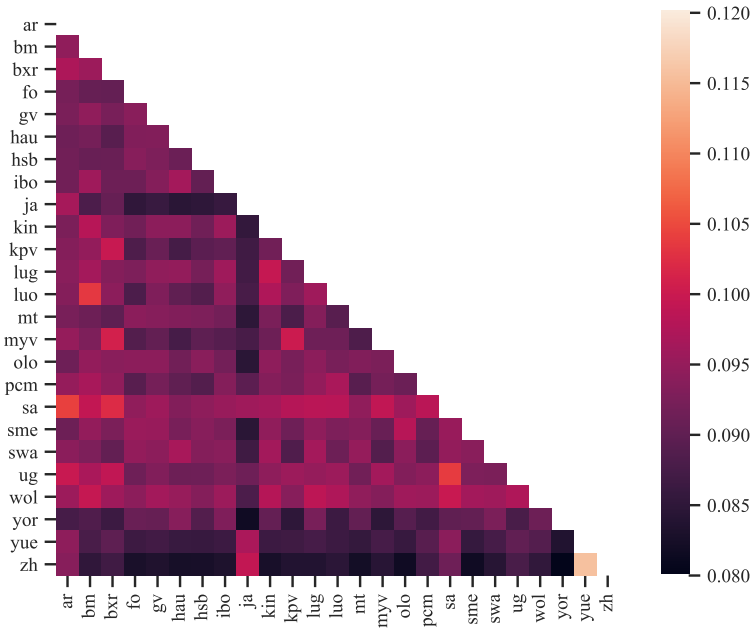


Fig. 4.4 Percentage of parameters selected for the sparse fine-tuning of both languages in a pair.

In order to understand whether similar languages also share similar sub-networks, we plot the pairwise overlap (in percentage) between parameter subsets of language SFTs in Figure 4.4. Except for a single instance (Mandarin Chinese and Cantonese) where the high overlap reflects the fact that both languages are genealogically related, we find that

the overlap is small for most language pairs. The explanation, we believe, is two-fold. Firstly, most of the languages in the multilingual datasets considered in our experiments belong to separate genera and families. Therefore, a lack of correlation in parameter subsets is expected. Secondly, for a pre-trained model, there exist multiple parameter subsets ("winning tickets") with comparable performance (Prasanna et al., 2020). The Lottery Ticket algorithm selects randomly among these equally valid subsets. Hence, a lack of overlap does not necessarily imply reliance on disjoint sub-networks.

4.6 Conclusion

We have presented a new method to fine-tune pre-trained models that is both modular (like bottleneck adapters) and expressive (like sparse fine-tuning). This method is based on a variant of the algorithm to find winning tickets under the framework of the Lottery Ticket Hypothesis. We infer a sparse vector of differences with respect to the original model for each individual language (by modelling unlabelled text) and each individual task (with supervised learning). The adaptations for a language and a task can then be composed with the pre-trained model to enable zero-shot cross-lingual transfer. Comparing our method with the state-of-the-art baseline in several multilingual tasks, the results have indicated substantial gains across the board in both languages seen and unseen during pre-training (which includes many truly low-resource languages). Our code and models are available at https://github.com/cambridgeltl/composable-sft.

Chapter 5

Distilling Efficient Language-Specific Models for Cross-Lingual Transfer

5.1 Introduction

Massively multilingual Transformers (MMTs) suffer from several limitations as a result of their ample language coverage. Firstly, aiming to represent many languages within their parameter budget and dealing with the training signals from different languages might result in negative interference. This is known as the "curse of multilinguality" (Conneau et al., 2020), which impairs the MMT's transfer capabilities (Pfeiffer et al., 2022). Secondly, in practice people are often interested in using or researching NLP systems in just a *single* language. This makes MMTs *unnecessarily costly* in terms of storage, memory, and compute and thus hard to deploy. This especially impacts communities which speak low-resource languages, which are more likely to have limited access to computational resources (Alabi et al., 2022).

In this chapter, we address the question: *can we increase the time-efficiency and memory-efficiency of MMTs while retaining their performance in cross-lingual transfer?* Knowledge distillation (Hinton et al., 2015) is a family of general methods to achieve the first goal by producing smaller, faster models (Sanh et al., 2019; Jiao et al., 2020, *inter alia*) and has also been applied to MMTs specifically. However, when the distilled MMT is required to cover the same number of languages as the original model, whose capacity is already thinly stretched over (say) a hundred languages, the curse of multilinguality asserts itself, resulting in a significant loss in performance (Sanh et al., 2019).

As a consequence, to achieve the best possible performance with reduced capacity, we depart from the practice of retaining all the languages from the original MMT in the distilled model. Instead, we argue, we should cover only *two* languages, namely the

source and target language of interest. In fact, distilling just *one* language would fall short of the second goal stated above, namely facilitating cross-lingual transfer, as a monolingually distilled model would be unable to learn from a distinct source language during task-specific fine-tuning. Maintaining cross-lingual transfer capabilities, however, is crucial due to the paucity of labelled task data in many of the world's languages in most tasks (Ponti et al., 2019a; Joshi et al., 2020).

In particular, we propose a method for *bilingual distillation* of MMTs, termed BiStillation, inspired by the two-phase recipe of Jiao et al. (2020). We start from a *"student"* model, initialised by discarding a subset of layers of the original *"teacher"* MMT, as well as the irrelevant part of its vocabulary. In the first, *"general"* phase of distillation, unlabelled data is used to align the hidden representations and attention distributions of the student with those of the teacher. In the second, *task-specific* phase, the student is fine-tuned for the task of interest through guidance from a task-adapted variant of the teacher. Rather than fully fine-tuning the student during this second phase, we instead use the parameter-efficient Lottery-Ticket Sparse Fine-Tuning (LT-SFT) method proposed in §4.3. Parameter-efficient task fine-tuning enables a system to support multiple tasks with the same distilled compact model, without unnecessarily creating full model copies per each task.

We evaluate our efficient *"bistilled"* models on a range of downstream tasks from several benchmarks for multilingual NLP, including dependency parsing from Universal Dependencies (UD; Zeman et al., 2020), named entity recognition from MasakhaNER (Adelani et al., 2021), natural language inference from AmericasNLI (Ebrahimi et al., 2022), and QA from XQuAD (Artetxe et al., 2020b). We evaluate the model performance as well as its space efficiency (measured in terms of parameter count) and time efficiency (measured in terms of FLOPs and inference time). We compare it against highly relevant baselines: bilingual models pre-trained from scratch and two existing multilingual distilled models, DistilmBERT (Sanh et al., 2019) and MiniLMv2 (Wang et al., 2021).

We find that while our bilingually distilled models are twice or thrice smaller and faster than the original MMT, their performance is only slightly degraded, as illustrated in Figure 5.1. Our method outperforms the baselines by sizable margins, showing the advantages of (i) bilingual as opposed to multilingual distillation, and (ii) distilling models from MMTs rather than training them from scratch. We hope that our endeavour will benefit end-users of multilingual models, and potential users under-served by currently available technologies, by making NLP systems more accessible.

5.2 Background

In this chapter we will make use of the aforementioned MMTs mBERT (Devlin et al., 2019), XLM-R (Conneau et al., 2020) and mDeBERTa (He et al., 2021). We will also employ modular language- and task-specific adaptation of MMTs through LT-SFT as proposed in §4.3 and make use of multi-source training as described in §3.3.3.

5.2.1 Distilling pre-trained Language Models

Knowledge distillation (Buciluǎ et al., 2006; Hinton et al., 2015) is a technique for compressing a pre-trained large "teacher" model into a smaller "student" model by training the student to copy the behaviour of the teacher. Whereas during standard pre-training, the model receives a single "hard" label per training example, during distillation the student benefits from the enriched signal provided by the full label distribution predicted by the teacher model. Sanh et al. (2019) use this technique to produce *DistilBERT*, a distilled version of $\text{BERT}_{\text{base}}$ (Devlin et al., 2019) with 6 instead of the original 12 layers, and *DistilmBERT*, a corresponding distilled version of multilingual BERT. There has been extensive subsequent work on distillation of pre-trained language models, though with less focus on distilling MMTs in particular. More sophisticated approaches try to align the hidden states and self-attention distributions of the student and teacher (Sun et al., 2020; Jiao et al., 2020) and/or finer-grained aspects of the self-attention mechanism (Wang et al., 2020, 2021). Mukherjee et al. (2021) initialise the student's embedding matrix with a factorisation of the teacher's for better performance when their hidden dimensions differ. Of these, Wang et al. (2020, 2021); Mukherjee et al. (2021) apply their methods to produce distilled versions of MMTs. In this work, we employ the "TinyBERT" distillation recipe of Jiao et al. (2020), which is described in further detail in §5.3.

5.3 BiStillation: Methodology

We are interested in providing NLP capabilities with limited computational resources in a specific target language T which lacks training data in the tasks of interest. A common paradigm in previous work (Pfeiffer et al., 2020b; Ansell et al., 2022) is to use cross-lingual transfer with an MMT in conjunction with parameter-efficient task and language adaptation to support multiple tasks without adding a large number of additional parameters per task (§2.3.4). Our goal in this work is to replace the highly

general MMT, plus optional language adaptation, with a target language-specific model which maintains the benefits of cross-lingual transfer.

An obvious first attempt would be to simply distil the MMT into a smaller model using only text in the target language. However, this monolingual distillation approach is insufficient, as during task fine-tuning, the monolingually distilled student model no longer "understands" the source language. Not only would the task performance of the student therefore be poor on the source language, but the target language performance would likely be even worse because it is optimistic to assume that the source and target language representations would remain aligned in the student model without any training on the source language. Indeed, our preliminary experiments confirmed the intuition that this approach is inadequate. This problem can be overcome through *bilingual* distillation, where text from both the source and target language is used to train the student model.[1]

Therefore, our aim is to devise a method for deriving from an MMT M a smaller model $M'_{S,T,\tau}$ to perform a given task τ in the target language T given only training data in the source language S. Our approach is inspired by the two-stage distillation paradigm of Jiao et al. (2020). In the first, "general" phase, a bilingual student model $M'_{S,T}$ is distilled from M using the same unsupervised task (e.g., masked language modelling) that was used for M's pre-training. In the second, "task-specific" phase, $M'_{S,T,\tau}$ is produced by fine-tuning $M'_{S,T}$ using M_τ as its teacher, where M_τ is derived from M by fine-tuning it for task τ. The following sections explain the details of these phases.

5.3.1 Distillation Method

Let L_T be the number of Transformer layers in the teacher model, indexed from 1 to L_T. The number of student model layers L_S is required to evenly divide L_T. We define the downscaling stride as $s = \frac{L_T}{L_S}$.

Following Jiao et al. (2020), the loss functions of the two distillation phases make use of three components, (i) *attention-based*, (ii) *hidden state-based*, and (iii) *prediction-based*. Attention-based loss is defined as follows:

$$\mathcal{L}_{\text{attn}}(x\ ;\ \boldsymbol{\theta}) = \frac{1}{L_S}\sum_{i=1}^{L_S} \texttt{MSE}(A_i^S(x\ ;\ \boldsymbol{\theta}),\ A_{i\cdot s}^T(x\ ;\ \boldsymbol{\theta})). \tag{5.1}$$

[1]This is similar to the idea of bilingual language adapters proposed by Parović et al. (2022), which obtain superior transfer performance by adapting the MMT to both source and target language simultaneously, removing the need to use different and possibly incompatible language adapters during training and inference.

Here, $A_i^S(x\ ;\ \boldsymbol{\theta})$ and $A_i^T(x\ ;\ \boldsymbol{\theta}) \in \mathbb{R}^{l\times l}$ refer to the attention distribution[2] of Transformer layer i of the student and teacher model, respectively, given input sequence x and model parameters $\boldsymbol{\theta}$; l refers to the input sequence length; $\texttt{MSE}()$ denotes mean squared error loss.

Hidden state-base loss is defined as follows:

$$\mathcal{L}_{\text{hidden}}(x\ ;\ \boldsymbol{\theta}) = \frac{1}{L_S+1}\sum_{i=0}^{L_S}\texttt{MSE}(H_i^S(x\ ;\ \boldsymbol{\theta}),\ H_{i\cdot s}^T(x\ ;\ \boldsymbol{\theta})), \tag{5.2}$$

where $H_i^S(x\ ;\ \boldsymbol{\theta})$ and $H_i^T(x\ ;\ \boldsymbol{\theta}) \in \mathbb{R}^{l\times d}$ refer to the hidden representations output by Transformer layer i of the student and teacher model, respectively, or the output of the embedding layer when $i = 0$. Note that we assume that the student and teacher share the same hidden dimensionality d.

Finally, the prediction-based loss is defined as

$$\mathcal{L}_{\text{pred}}(x\ ;\ \boldsymbol{\theta}) = \texttt{CE}(\boldsymbol{z}^S(x\ ;\ \boldsymbol{\theta}),\ \boldsymbol{z}^T(x\ ;\ \boldsymbol{\theta})), \tag{5.3}$$

where $\boldsymbol{z}^S(x\ ;\ \boldsymbol{\theta})$ and $\boldsymbol{z}^T(x\ ;\ \boldsymbol{\theta})$ are the label distributions predicted by the student and teacher model, respectively, and $\texttt{CE}$ denotes cross-entropy loss.

The intuition behind using attention-based and hidden state-based loss for our purposes is as follows. We (i) require good monolingual performance in the source and target language, but we also (ii) must preserve the existing alignment between these languages in the MMT which would consequently facilitate transfer between them. The intuition is that encouraging the student's intermediate representations to match those of the teacher will help to preserve this alignment.

We next describe how these loss components are employed in each phase of BiStillation.

5.3.2 Stage 1: General Bilingual Distillation

Initialisation. We initialise all parameters of the student model by copying those of the teacher model, but retaining only the Transformer layers whose indices are multiples of s. We found in our preliminary experiments that this greatly accelerates convergence compared to random initialisation.

Vocabulary Reduction. Our distilled models can dispose of the many irrelevant tokens in the base MMT's vocabulary, i.e. those which are not frequently used in either the

[2]Here, for ease of implementation within the Huggingface Transformers library (Wolf et al., 2020), we differ from Jiao et al. (2020), who use raw attention scores.

Task	Target Dataset	Source Dataset	MMT	Target Languages
Dependency Parsing (DP)	Universal Dependencies 2.7 (Zeman et al., 2020)	Universal Dependencies 2.7 (Zeman et al., 2020)	mBERT	Arabic†, Bambara, Buryat, Cantonese, Chinese†, Erzya, Faroese, Japanese†, Livvi, Maltese, Manx, North Sami, Komi Zyrian, Sanskrit, Upper Sorbian, Uyghur
Named Entity Recognition (NER)	MasakhaNER (Adelani et al., 2021)	CoNLL 2003 (Tjong Kim Sang and De Meulder, 2003)	mBERT	Hausa, Igbo, Kinyarwanda, Luganda, Luo, Nigerian-Pidgin, Swahili*, Wolof, Yorùbá*
Natural Language Inference (NLI)	AmericasNLI (Ebrahimi et al., 2022)	MultiNLI (Williams et al., 2018)	XLM-R	Aymara, Asháninka, Bribri, Guarani, Náhuatl, Otomí, Quechua, Rarámuri, Shipibo-Konibo, Wixarika
Question Answering (QA)	XQuAD (Artetxe et al., 2020b)	SQuAD v1.1 (Rajpurkar et al., 2016)	mDeBERTa	Arabic†, Chinese†, German†, Greek†, Hindi†, Romanian†, Russian†, Spanish†, Thai†, Turkish†, Vietnamese†

Table 5.1 Details of the tasks, datasets, MMTs and languages involved in our zero-shot cross-lingual transfer evaluation. * denotes low-resource languages and † high-resource languages seen during MMT pre-training; all other languages are low-resource and unseen. The source language is always English. We *"bistil"* the MMT listed per each task and target language. Further details of all the language and data sources used are provided in Appendix C.1.

source or target language of interest, an idea previously proposed by Abdaoui et al. (2020). During initialisation, the vocabulary of the student model is selected by retaining only the tokens of the teacher's vocabulary whose unigram probability in either the source or target language corpus is $\geq 10^{-6}$.

Teacher Language Adaptation. As we wish to be able to produce distilled models for languages not covered in the base MMT, and to obtain the best possible performance for languages which are covered, we employ language adaptation of the teacher MMT with language-specific SFTs (Ansell et al., 2022) applied on top of the original MMT during distillation. Since it draws examples from two languages, each with its own language SFT, bilingual distillation becomes a special case of multi-source training as described in §3.3.3. At each training step, either the source or target language is selected at random with equal probability; the batch is composed of sequences drawn from the training corpus of the chosen language, and a pre-trained SFT for that language is applied to the teacher MMT.

Objective. The overall loss function for this phase is given by the sum of the attention-based and hidden state-based loss. Omitting the prediction-based loss here has the advantage of avoiding the need to evaluate the distribution of tokens predicted by the MLM head, which is costly because of the considerable size of MMTs' embedding matrices.

5.3.3 Stage 2: Task-Specific Distillation

After a general bilingual model has been distilled from the teacher MMT in Stage 1, it can be fine-tuned for a specific task. We first obtain the teacher for task-specific distillation by applying task-specific LT-SFT to fine-tune the base MMT (i.e., the teacher in the general distillation phase) for the task in question. This teacher's outputs and representations are then used to fine-tune the bilingual student model, again using task LT-SFT at the student's end. The use of parameter-efficient task adaptation here avoids adding a large number of parameters to the system for each task. The objective during this task-specific fine-tuning consists of the sum of all three losses from §5.3.1: $\mathcal{L}_{\text{attn}}$, $\mathcal{L}_{\text{hidden}}$, and $\mathcal{L}_{\text{pred}}$.

5.4 Experimental Setup

We largely adopt the same evaluation framework as §4.4 for direct comparability with LT-SFT on undistilled MMTs. Specifically, we evaluate zero-shot cross-lingual transfer performance on four representative tasks: dependency parsing, named entity recognition, natural language inference and QA. While the prior work focused only on low-resource languages, our method is also highly relevant to high-resource languages: the XQuAD QA task (Artetxe et al., 2020b) provides additional insight into high-resource target language performance. Table 5.1 summarises the experimental setup, including the datasets and languages considered in our experiments. In total, we cover a set of 44 languages which are typologically and geographically diverse, making them representative of cross-lingual variation (Ponti et al., 2020).

We experiment with three different MMTs as shown in Table 5.1: mBERT (Devlin et al., 2019), XLM-R_{base} (Conneau et al., 2020), and $\text{mDeBERTa}_{\text{base}}$ (He et al., 2021).

5.4.1 Baselines and Model Variants

We refer to our main method as `BiStil`. We compare it with several relevant approaches. First, the `LtSft` method uses LT-SFT with language adaptation on the base MMT. `LtSft` can be seen as an upper bound for `BiStil`, allowing us to measure how much the performance suffers as a result of replacing the MMT with its bilingually distilled variant.

MMT	Distillation	LRF	DRF	#L	D	#V	#P
mBERT	none	-	-	12	768	120K	178M
	D'mBERT	2	-	6	768	120K	135M
	BiStil*	2	-	6	768	31K	67M
		3	-	4	768	31K	53M
XLM-R_{base}	none	-	-	12	768	250K	278M
	BiStil*	2	-	6	768	28K	65M
		3	-	4	768	28K	51M
XLM-R_{large}	none	-	-	24	1024	250K	560M
	MiniLMv2	2	2.67	12	384	250K	118M
mDeBERTa	none	-	-	12	768	250K	278M
	BiStil*	2	-	6	768	41K	75M
		3	-	4	768	41K	60M

Table 5.2 Dimensions of models *before* and *after* distillation. LRF = Layer Reduction Factor; DRF = hidden Dimension Reduction Factor; #L = number of Transformer Layers; D = hidden Dimension; #V = Vocabulary size; #P = total number of model Parameters; D'mBERT = DistilmBERT. * - because of its vocabulary reduction procedure, BiStil can produce models of slightly different sizes for different languages; the vocabulary sizes and numbers of parameters shown are averages over all BiStil models trained.

For each task except NLI,[3] we also compare against a multilingually distilled MMT, i.e. with all pre-training languages used for distillation as well. For DP and NER, where mBERT is the base MMT, the distilled MMT is DistilmBERT (Sanh et al., 2019), which is similarly based on mBERT. For QA, where BiStil uses mDeBERTa as the base MMT, no directly comparable multilingually distilled MMT is available, so we opt for a loose comparison with MiniLMv2 (Wang et al., 2021), distilled from XLM-R_{large}, which has achieved strong results on cross-lingual transfer in high-resource languages. We perform task-specific fine-tuning with LT-SFT on DistilmBERT and MiniLMv2 in the same way as for the the undistilled MMTs in the LtSft setting. For DP and NER we also perform language adaptation of DistilmBERT.[4]

We also consider Scratch, a setting where we train bilingual models from scratch instead of distilling them from a pre-trained MMT. We then apply the same LT-SFT task fine-tuning method as for the other baselines. This comparison allows us to evaluate

[3]There does not seem to be a multilingually distilled MMT that would provide a suitable comparison on the AmericasNLI task, as MiniLMv2 and other models distilled without an MLM head do not support adaptation to unseen languages through continued pre-training. DistilmBERT on the other hand is based on a generally weaker MMT than XLM-R, which is used as BiStil's base MMT for NLI.

[4]MiniLMv2 does not support language adaptation (see Footnote 3), but this is not as important for the high-resource languages used in XQuAD (Ansell et al., 2022).

the benefit of distilling efficient bilingual models from the MMT rather than pre-training the same-sized bilingual models from scratch.

We refer to our main method, with the task-specific distillation stage as described in §5.3.3, as `BiStil-TF` (TF = *teacher forcing*). We also carry out an ablation focused on the second phase of BiStillation: here, we consider performing task-specific fine-tuning without the assistance of a teacher, i.e. in the same manner as `LtSft`. We refer to this variant as `BiStil-ST` (ST = *self-taught*).

Table 5.2 provides details of the model sizes, before and after distillation using the above methods, demonstrating the benefits of BiStillation with respect to model compactness.

5.4.2 Language Distillation/Adaptation

We always perform language adaptation of the teacher model during both phases of BiStillation and during `LtSft` except for mDeBERTa and MiniLMv2[5]. For language adaptation of MMTs we use the pre-trained language SFTs of Ansell et al. (2022), and we train our own for DistilmBERT. Similarly, for the `LtSft` baseline, and for task adaptation of the teacher in the `BiStil-TF` configuration, we use their pre-trained single-source task SFTs or train our own when necessary. See Appendix C.1 for details of the training corpora.

When training/distilling our own models or SFTs, we generally choose hyperparameters which match those used in the original work. The following are constant across all language distillation/SFT training: we use a batch size of 8 and a maximum sequence length of 256; model checkpoints are evaluated every 1,000 steps (5,000 for high-resource languages) on a held-out set of 5% of the corpus (1% for high-resource languages), and the one with the smallest loss is selected at the end of training; we use the Adam optimiser (Kingma and Ba, 2015) with linear decay and without any warm-up.

During LT-SFT training of DistilmBERT's language SFTs, the dense and sparse fine-tuning phases each last the lesser of 100,000 steps or 200 epochs, but at least 30,000 steps if 200 epochs is less. The initial learning rate is 5×10^{-5}. The SFT density is set to 4%.[6]

When distilling bilingual models or learning them from scratch, training lasts 200,000 steps (to equal the total length of the two phases of LT-SFT training). The initial

[5]See Footnote 3 for MiniLMv2; mDeBERTa could in theory support language adaptation but its pre-training code was not made publicly available in time to be used in this work.

[6]This is similar but not identical to the density used in §4.4.2, which was a very specific number of trainable parameters for comparability to the MAD-X baseline; here, we prefer to use a round number.

	ar	bm	bxr	fo	gv	hsb	ja	kpv	mt	myv	olo	sa	sme	ug	yue	zh	avg	avgΔ
LtSft	53.6	16.5	25.9	55.5	42.4	60.5	19.7	27.2	55.4	45.3	47.8	25.2	42.1	16.7	34.0	37.0	37.8	-
DistilmBert	47.7	9.9	19.5	49.1	31.7	53.2	16.2	20.0	43.0	34.9	37.6	17.7	31.4	11.4	28.9	33.9	30.4	-7.4
Scratch, lrf = 2	16.9	4.9	6.7	27.8	9.1	15.2	6.7	5.6	16.1	12.7	11.1	3.5	9.3	3.9	11.5	14.6	11.0	-26.8
BiStil-ST, lrf = 2	50.9	15.8	24.1	53.7	38.3	57.1	18.7	**23.9**	52.2	**43.7**	**46.5**	25.2	39.8	13.3	31.8	34.8	35.6	-2.2
BiStil-ST, lrf = 3	48.2	16.1	23.4	52.1	35.0	55.1	18.1	22.2	49.9	40.3	41.3	22.2	37.6	13.3	30.7	33.4	33.7	-4.1
BiStil-TF, lrf = 2	**53.2**	**16.4**	**24.6**	**54.8**	**39.1**	**59.0**	**19.0**	23.8	**54.1**	43.5	46.0	**26.9**	**40.7**	13.1	**32.7**	**36.4**	**36.5**	**-1.3**
BiStil-TF, lrf = 3	49.7	**16.4**	24.4	52.7	36.8	57.1	18.2	21.0	52.2	41.0	43.3	25.1	38.1	**14.5**	31.3	34.9	34.8	-3.0

Table 5.3 DP; LAS scores. The results with the smallest gap to the upper-bound `LtSft` model are in **bold**.

learning rate is 1×10^{-4}. The model architecture and hyperparameters are identical to the teacher MMT's other than a reduction in the number of layers and the use of vocabulary reduction as described in §5.3.2.

We experiment with two layer reduction factors (LRF) for BiStillation, 2 (a reduction from 12 to 6 layers) and 3 (12 to 4 layers). Whereas the `BiStil` setting initialises the model from the teacher (see §5.3.2), the `Scratch` setting initialises it randomly.

5.4.3 Task Distillation/Adaptation

For DP and NER, we train task SFTs for 3 epochs in the dense phase of LT-SFT and 10 epochs in the sparse phase, evaluating the model checkpoint on the validation set at the end of each epoch, and taking the best checkpoint at the end of training. The selection metric is labelled attachment score for DP and F1-score for NER. The initial learning rate is 5×10^{-5} with linear decay. For NER, we use the standard token-level single-layer multi-class model head. For DP, we use the shallow variant (Glavaš and Vulić, 2021) of the biaffine dependency parser of Dozat and Manning (2017). For NLI, we train for 5 epochs with batch size 32, with checkpoint evaluation on the validation set every 625 steps, and an initial learning rate of 2×10^{-5}. We apply a two-layer multi-class classification head atop the model output corresponding to the `[CLS]` token. For QA, we train for 5 epochs with a batch size of 12, with checkpoint evaluation every 2000 steps and an initial learning rate of 3×10^{-5}. The single-layer model head independently predicts the start and end positions of the answer span, and at inference time the span whose endpoints have the largest sum of logits is selected.

We set the density of our task SFTs to 8%, which we found in §4.5 to offer the best task performance in all our experiments.

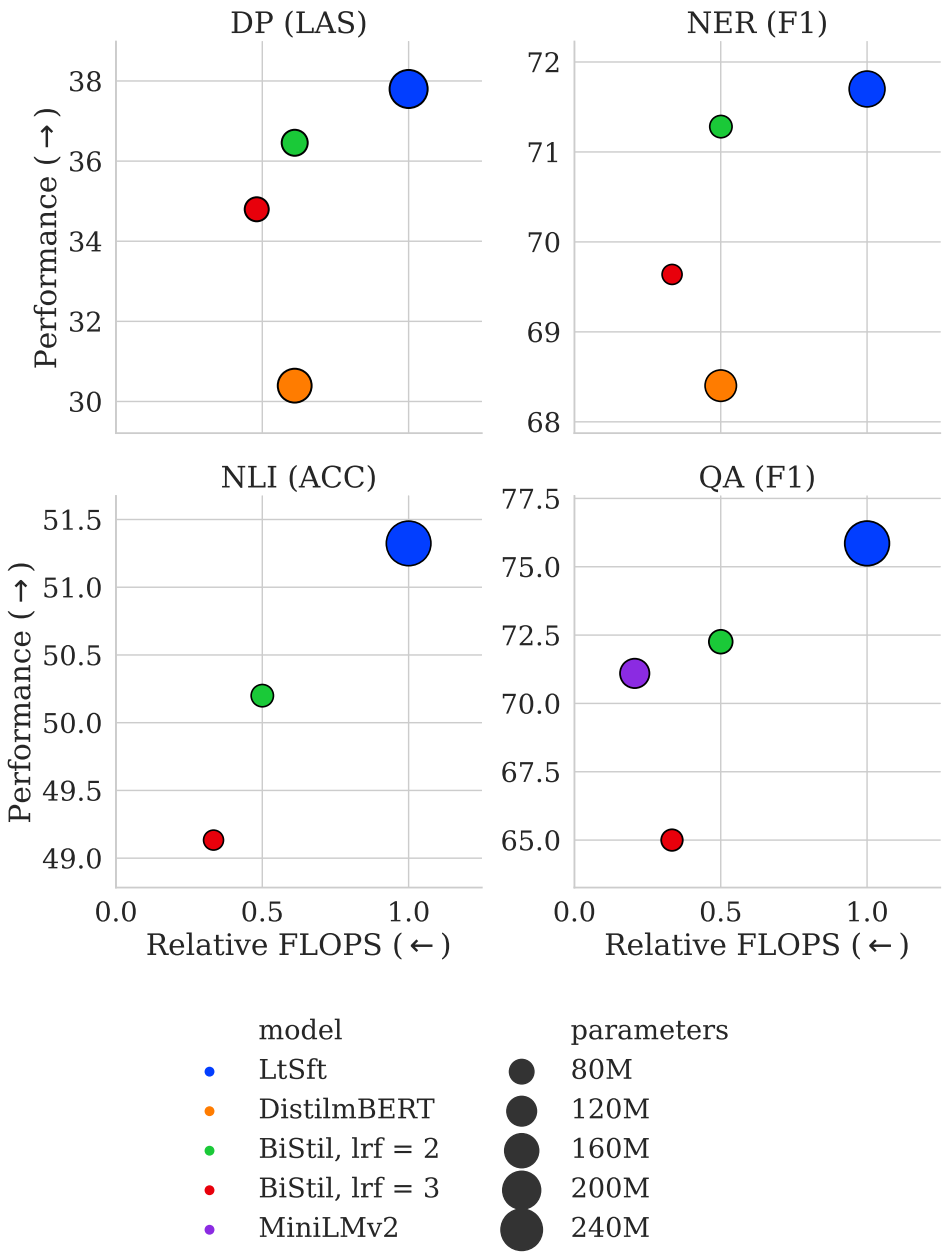


Fig. 5.1 Trade-off between parameter count, inference FLOPs and averaged performance for our `BiStil` models for cross-lingual transfer and several baselines.

5.5 Results and Discussion

The results in terms of task performance are summarised in Figure 5.1 and Tables 5.3-5.6. As expected, `LtSft` on the undistilled MMTs performs best across all tasks. However, `BiStil`-TF with reduction factor 2 is not much worse, with a degradation in performance not exceeding 1.3 points relative to `LtSft` on DP, NER and NLI. The larger gap of

	hau	ibo	kin	lug	luo	pcm	swa	wol	yor	avg	avgΔ
LtSft	83.5	76.7	67.4	67.9	54.7	74.6	79.4	66.3	74.8	71.7	-
DistilmBert	81.1	73.2	65.3	63.4	50.0	69.2	77.7	64.4	71.2	68.4	-3.3
BiStil-ST, lrf = 2	**81.3**	74.1	65.9	66.7	53.5	72.1	77.1	64.6	72.8	69.8	-1.9
BiStil-ST, lrf = 3	80.3	74.0	63.1	64.6	54.7	69.6	76.9	68.0	70.5	69.1	-2.6
BiStil-TF, lrf = 2	81.0	**74.8**	**67.5**	**67.3**	55.0	**72.9**	**78.4**	**69.0**	**75.7**	**71.3**	**-0.4**
BiStil-TF, lrf = 3	79.6	**74.8**	64.6	64.5	**56.7**	70.6	77.2	66.1	72.8	69.6	-2.1

Table 5.4 NER; F1 scores.

	aym	bzd	cni	gn	hch	nah	oto	quy	shp	tar	avg	avgΔ
LtSft	58.1	44.4	47.9	63.5	42.8	52.4	48.5	62.0	50.3	43.3	51.3	-
BiStil-TF, lrf = 2	**58.9**	**45.7**	46.4	**62.9**	**44.3**	50.8	**44.0**	58.7	**47.2**	**43.1**	**50.2**	**-1.1**
BiStil-TF, lrf = 3	57.7	43.6	**48.1**	60.9	41.3	**51.4**	42.6	**59.9**	45.5	40.3	49.1	-2.2

Table 5.5 NLI accuracy (%).

	ar	de	el	es	hi	ro	ru	th	tr	vi	zh	avg	avgΔ
LtSft	56.5	64.7	61.2	62.4	57.8	69.0	61.8	56.0	56.4	57.1	60.8	60.3	-
MiniLMv2	50.4	59.4	54.4	57.9	52.9	64.5	57.6	50.3	51.3	**53.8**	55.0	55.2	-5.1
BiStil-TF, lrf = 2	**53.5**	**62.2**	**55.4**	**59.8**	**54.5**	**66.2**	**58.3**	**54.4**	**53.1**	53.4	**55.7**	**57.0**	**-3.4**
BiStil-TF, lrf = 3	44.3	55.0	44.1	55.2	46.1	59.5	51.3	42.4	48.3	44.6	50.9	49.3	-11.1

Table 5.6 XQuAD; Exact Match scores.

3.4 EM points on QA is likely a result of the fact that the base MMT is much more thoroughly pre-trained on the high-resource languages found in XQuAD than on the lower-resource languages found in the datasets for the other tasks. It is therefore harder for `BiDistil` to achieve the base MMT's depth of knowledge of the target language during its relatively short distillation training time. `BiStil-TF, lrf = 2` nevertheless outperforms MiniLMv2 on QA by 1.7 EM points, despite MiniLMv2 receiving 320 times more training than each `BiDistil` model, or roughly 6 times more per language[7].

Furthermore, `BiStil-TF, lrf = 2` significantly outperforms `DistilmBERT`, with a 6.1 LAS gap on DP and 2.9 F1 gap on NER. `BiStil, lrf = 2` produces models roughly half the size of `DistilmBERT` and that, once again, are trained for vastly less time[8].

Training bilingual models from `Scratch` performs poorly, lagging behind the other methods by more than 20 points on DP.[9] This is likely at least partly because, unlike for

[7]MiniLMv2 is trained for 1M steps with a batch size of 256 and max sequence length of 512; `BiDistil` for 200K steps with a batch size of 8 and max sequence length of 256.

[8]Sanh et al. (2019) note that their monolingual DistilBERT model was trained on 8 16GB V100 GPUs for approximately 90 hours. Our `BiStil` models are trained on a single 10GB RTX 3080 GPU for approximately 9 hours.

[9]As this method is clearly inferior, we opted to reduce computational expense by not repeating it for other tasks.

	cpu ↑	gpu ↑	flops ↓
DistilmBERT	1.41x	1.03x	0.61x
BiStil, lrf = 2	1.44x	1.25x	0.61x
BiStil, lrf = 3	1.71x	1.36x	0.48x

(a) DP efficiency

	cpu ↑	gpu ↑	flops ↓
DistilmBERT	1.93x	1.94x	0.50x
BiStil, lrf = 2	1.97x	1.98x	0.50x
BiStil, lrf = 3	2.97x	2.78x	0.33x

(b) NER efficiency

	cpu ↑	gpu ↑	flops ↓
BiStil, lrf = 2	2.02x	1.97x	0.50x
BiStil, lrf = 3	2.89x	2.85x	0.33x

(c) NLI efficiency

	cpu ↑	gpu ↑	flops ↓
MiniLMv2	4.25x	3.44x	0.21x
BiStil, lrf = 2	1.99x	1.85x	0.50x
BiStil, lrf = 3	2.85x	2.42x	0.33x

(d) QA efficiency

Table 5.7 Relative inference speed and FLOP cost. Values are given relative to LtSft without distillation, i.e. a speed reading of "2.00x" means the distilled model can on average process twice as many inference examples per second as the undistilled MMT. Likewise a FLOPs reading of "0.50x" would mean that the distilled model on average requires half as many FLOPs to process an inference example as the undistilled MMT does.

BiStil, 200,000 training steps are insufficient for Scratch to fully converge, suggesting (unsurprisingly) that it is less sample-efficient - reaching full convergence would likely have taken many more steps and exceeded our computational budget. Interestingly, when we evaluate the Scratch models on their *English* DP performance, we obtain an average UAS/LAS score of 81.8/77.1, which is much more competitive in relative terms with the BiStil-TF, lrf = 2 English DP score of 91.0/88.2 than the corresponding comparison in average target language DP scores of 29.9/11.0 to 55.5/36.5. This suggests that an even larger factor in Scratch's weakness than its poor monolingual performance is a lack of alignment between its representations of the source and target languages, severely impairing cross-lingual transfer. This highlights the advantage of distilling a bilingual model from an MMT within which cross-lingual alignment is already present.

As expected, the performance of BiStil is somewhat weaker with a larger layer reduction factor of 3, though this is heavily task-dependent. With an LRF of 3, BiStil-TF still comfortably outperforms DistilmBERT on DP and NER, and does not fall much behind LRF = 2 for NLI. However, we observe a considerable degradation in performance for LRF = 3 for QA; this may indicate that a 4-layer Transformer struggles to adapt to this particular task, or that for this architecture the modest training time is not sufficient to approach the base MMT's understanding of the source and target languages.

Table 5.7 presents an analysis of the inference time efficiency. We measure the inference speed both on CPU with batch size 1 and GPU with the same batch size as

during task-specific training. We also calculate the number of floating-point operations (FLOPs) per example using fvcore[10], measured during an inference pass over the test set of the first language in each task.

For NER, NLI and QA, the efficiency results conform quite closely to the intuitive expectation that a model's inference time should scale linearly with its number of layers; that is, `BiDistil` with LRF = 2 is generally around twice as fast as the base MMT. For DP, we observe a seemingly sub-linear scaling which is caused by the very large biaffine parsing head, consisting of ∼23M parameters. The significant cost of applying the model head contributes equally to all models regardless of their degree of distillation. Despite having a moderate LRF of 2, `MiniLMv2` exhibits impressive speed as a result of the fact that it additionally has a smaller hidden dimension than its teacher (see Table 5.2), a technique which we do not consider for `BiDistil`, but may be a promising avenue for future work.

We argue that `BiDistil` accomplishes its aim by achieving two- to three-fold reductions in inference time and model size without sacrificing much in the way of raw performance. Its superior performance relative to multilingually distilled models despite its comparatively very modest training budget supports the assertion that specialising multilingual models for a specific transfer pair during distillation helps to avoid performance degradation resulting from the curse of multilinguality.

5.6 Conclusions

While MMTs are an effective tool for cross-lingual transfer, their broad language coverage makes them unnecessarily costly to deploy in the frequently-encountered situation where capability is required in only a single, often low-resource, language. We have proposed BiStillation, a method of training more efficient models suited to this scenario which works by distilling an MMT using only the source-target language pair of interest. We show that this approach produces models that offer an excellent trade off between target language performance, efficiency, and model compactness. The "bistilled" models exhibit only a slight decrease in performance relative to their base MMTs whilst achieving considerable reduction in both model size and inference time. Their results also compare favorably to those of multilingually distilled MMTs despite receiving substantially less training even on a per-language basis.

Our code is available at https://github.com/AlanAnsell/bistil.

[10] https://github.com/facebookresearch/fvcore

Chapter 6

A Holistic View of Cross-Lingual Transfer

6.1 Introduction

As we have seen, cross-lingual transfer is a means of compensating for a lack of resources in one or more *target* languages by employing the more plentiful resources available in one or more *source* languages. A range of techniques have been developed to deal with the various dimensions of resource scarcity, which encompass not just data availability, but also the degree of support by pre-trained models. These research threads have generally been investigated somewhat independently. In this chapter, we attempt to unify several of the most prominent threads of cross-lingual transfer research into a single framework. Specifically, we synthesise findings from zero-shot cross-lingual transfer (ZS-XLT) with multilingual Transformers encoders (MMTs), few-shot cross-lingual transfer (FS-XLT), cross-lingual transfer for low-resource languages and cross-lingual transfer through machine translation (MT) to formulate a practical, general-purpose approach to cross-lingual transfer, with a focus on low-resource scenarios.

So far, we have focused on ZS-XLT, where an MMT pre-trained on many languages is fine-tuned for a certain task using training data from some source language, and is then used for inference on data from the target language without seeing a single task-specific training example from that language. We have closely examined the practice of language adaptation, where a module for a specific language is trained to enhance the performance of an MMT when it has received little or not pre-training for that language.

In contrast to ZS-XLT, often more realistic is the few-shot case (FS-XLT), where a small number of gold-standard target language examples are available during training. Though it may be expensive to annotate target language data, especially for low-resource

languages where native speakers are hard to access, prior work has shown that using even a small amount during training can yield significant gains in performance (Zhao et al., 2021). While early approaches to FS-XLT involved fine-tuning first on the source language data, then separately on the few target language shots (Lauscher et al., 2020), recent work has shown that it is more effective to jointly train on both at once (Xu and Murray, 2022; Schmidt et al., 2022).

Another tool often employed for cross-lingual transfer is machine translation (MT). MT approaches can generally be categorised as *translate-train* or *translate-test* (Hu et al., 2020). We focus on the translate-train approach, which has so far been predominant and seems more amenable to combination with other cross-lingual transfer techniques. We consider two translate-train variants: TTRAIN-SINGLE, where a model is trained for each target language using only its own translated data; and TTRAIN-ALL, where one model covering all target languages is trained on their translated data and the source language data simultaneously. To constrain the scope of this work, we settle for providing a simple translate-test configuration as a point of comparison rather than pushing the envelope of translate-test performance. However, Artetxe et al. (2023) have made a strong case for considering translate-test further in future work, proposing a set of techniques for optimising its performance.

In this work, we consider how best to employ the above techniques in response to cross-lingual transfer scenarios with varying levels of data scarcity. We explore several promising directions toward integrating zero-shot, few-shot and translate-train techniques across a range of resource levels in order to obtain a deeper insight into: (i) to what extent these techniques and the different data sources they exploit are complementary, (ii) what is the most effective way of combining different data sources in order to maximise the performance, and (iii) how much each of the available sources of data contributes to the overall performance. We aim to equip practitioners with a recipe for how to use the available data resources in the most effective way.

We experiment on the AmericasNLI natural language inference (NLI) dataset for American languages (Ebrahimi et al., 2022), and the NusaX sentiment analysis (SA) dataset for Indonesian languages (Winata et al., 2023). We find that combining language adaptation, few-shot learning and translation can be highly effective, yielding average performance gains of 14-24 points over the zero-shot baseline without language adaptation.

6.2 Background and Related Work

In this section, I briefly recap the most relevant material on language adaptation from earlier chapters. I then expand on the prior work on few-shot cross-lingual transfer and machine translation for cross-lingual transfer discussed in §2.3.

6.2.1 Language Adaptation

Because MMTs divide their capacity among many languages, they may often perform sub-optimally with respect to a single source or target language. Furthermore, we are sometimes interested in a target language not covered by the MMT. Recall the MAD-X approach for training bottleneck adapters to specialise MMTs to specific target languages from §2.3.4 and our Lottery Ticket Sparse Fine-Tuning (LT-SFT) approach from §4.3.

While SFT composition generally exhibits somewhat better zero-shot cross-lingual transfer performance across a range of tasks than bottleneck adapter composition (Ansell et al., 2022; Alabi et al., 2022), and avoids the overhead incurred during inference by adapters, adapters are more efficient to train and leave the base MMT fully unmodified. In this work, we consider both methods. However, we note that other modular, parameter-efficient methods exist and could be used with our tools in future work (Pfeiffer et al., 2023b).

Also recall from §3.3.3 and §4.4.4 how *multi-source* training extends language adaptation to scenarios with multiple source languages by training on batches consisting of examples from a single source language (selected randomly at each step), and applying the language module for that language during the training step.

6.2.2 Few-Shot Cross-Lingual Transfer

Recall from §2.3.3 that few-shot cross-lingual transfer (FS-XLT) describes the scenario where, in addition to the source language task training data available in the zero-shot case, a small number of target language task examples ("shots") are available. Practical experience currently suggests that it is best to mix the few shots into the source language data and train on both simultaneously (Xu and Murray, 2022; Schmidt et al., 2022) rather than sequentially training on source followed by target language data (Lauscher et al., 2020).

Jundi and Lapesa (2022) compare few-shot and translation-based approaches, trying to gain an insight into which approach is better and under which circumstances. We consider these approaches in combination rather than in competition and find that the

use of few-shot data can enhance performance even when a machine translation of the full source language dataset is available. However, their work complements ours by proposing a way to identify the examples which may be most profitable for humans to translate into target language "shots."

Winata et al. (2022) study few-shot cross-lingual transfer on languages unseen by MMTs using the NusaX dataset. They analyse the effectiveness of several few-shot strategies focusing on selecting languages for transfer and different learning dynamics exhibited by different types of MMTs.

6.2.3 Machine Translation for Cross-Lingual Transfer

Recall the definition of the translate-train and translate-test approaches to cross-lingual transfer from §2.3.1, to which a number of enhancements have been proposed. Artetxe et al. (2020a) showed that translate-test performance could be improved by training on back-translated rather than the original source language data to better model translation artifacts encountered at inference time. Ponti et al. (2021c) note that translation-based approaches suffer from an error accumulation over the phases of the pipeline. They re-interpret this pipeline as a single model with an intermediate "latent translation" between the target text and its classification label, permitting the translation model to be fine-tuned according to a feedback signal from the task loss. Oh et al. (2022) show that the translate-train and translate-test approaches can be combined synergistically. Artetxe et al. (2023) show that translate-test is more favourable relative to translate-train than previously thought when better translation and monolingual models are used, and when measures are taken to correct the MT-induced mismatch between the data encountered at train and inference time. While we only consider applying translation-based cross-lingual transfer to classification tasks, prior work has considered its application to sequence labelling tasks as well (Jain et al., 2019; Fei et al., 2020; García-Ferrero et al., 2022, 2023). In this work, we counteract the lack of support for certain languages by the multilingual NMT model NLLB (NLLB Team et al., 2022) by adapting it to these languages through further training. For simplicity, we employ only continued pre-training on the standard MT task for this purpose. However, Ko et al. (2021) enhance NMT model adaptation with additional tasks: denoising autoencoding, which exploits monolingual target language data; back-translation; and adversarial training which encourages the encoder to output language-agnostic features.

6.3 Methodology

6.3.1 Recipe

We propose a recipe for cross-lingual transfer which is flexible and effective across scenarios of resource scarcity. It can be summarised as:

1. Select a base MMT and train language modules for the source language and all target languages for which monolingual data is available.

2. Using a multilingual NMT model, translate the task data into every target language it supports; if there are target languages the MT model does not support but for which parallel data is available, adapt it using this parallel data. This process is described further in §6.3.3.

3. Learn a task module through joint multi-source training on all available data (i.e. source data, translated data and any gold-standard target language ("few-shot") data available).

We propose two methodological novelties to enhance this recipe.

6.3.2 Few-shot Upsampling

When gold-standard target language data is available during training, it is generally present in much smaller quantity than the source language data (in our case, the difference is in orders of magnitude, but it can vary). Furthermore, it is typically higher in quality than machine-translated target language data and may be more representative of the distribution that would be observed in practice due to the "translationese" effect of translated text (see §2.3.1). For this reason, we suggest upsampling this few-shot data relative to the source and machine-translated data during multi-source training. We show in our experiments that this can improve downstream performance.

6.3.3 NMT Model Adaptation

While recent multilingual NMT models such as NLLB (NLLB Team et al., 2022) provide impressive language coverage, there are still many languages they do not support. We therefore adapt NLLB to unseen target languages by initialising a new language token and embedding for the target language and then performing continued pre-training with parallel data for the relevant language pair. Conveniently, it is not necessary for the source language in the parallel corpus to match the intended source language for

Task	Target Dataset(s)	Source Dataset(s)	MMT	Target Languages
Natural Language Inference (NLI)	AmericasNLI (Ebrahimi et al., 2022) (sh: 743 / tst: 750)	MultiNLI (tr: 393K / dev: 10K) (Williams et al., 2018)	XLM-R Base	Aymara*, Asháninka*†, Bribri*†, Guarani*, Náhuatl*†, Otomí*†, Quechua*, Rarámuri*†, Shipibo-Konibo*†, Wixarika*†
Sentiment Analysis (SA)	NusaX (Winata et al., 2023) (sh: 600, tst: 400)	SMSA (tr: 11K, dev: 1.3K) (Purwarianti and Crisdayanti, 2019; Wilie et al., 2020)	XLM-R Base	Acehnese*, Balinese*, Banjarese*, Buginese*, Javanese, Madurese*†, Minangkabau*, Ngaju*†‡, Sundanese, Toba Batak*†

Table 6.1 Details of tasks, datasets, MMTs and languages involved in our experiments. sh = # of few-shot examples available per target language; tst = # of test set examples; tr = # of train set examples; dev = # of development set examples; * denotes languages unseen during MMT pre-training; † denotes languages not supported by the NLLB MT model; ‡ denotes languages for which no satisfactory monolingual corpus was available and hence no language module was trained. Further details of all the language and data sources used are provided in Appendix D.1. Note that since the NusaX dataset is created through human translation of a subset of the SMSA dataset, we carefully remove every example from SMSA which appears in its original or modified form in the NusaX test set to avoid a data leak.

cross-lingual transfer, since multilingual NMT models can, in theory, support unseen transfer directions provided the source and target languages have been seen as part of other pairs during training; this is the case in NMT adaptation for AmericasNLI, where the parallel data is Spanish-to-X but the transfer direction is English-to-X.

6.4 Experimental Setup

6.4.1 Evaluation Tasks and Languages

There are few datasets which meet all the ideal conditions for our evaluation: namely (1) many of the languages covered must not be well-represented in pre-trained MMTs so that we can test the effect of language adaptation; (2) the dataset must have at least a small training/validation set for the target languages which we can use as few-shot examples; and (3) the languages covered must either be supported by contemporary NMT models or have parallel data available data which can be used for NMT adaptation; (4) the task should ideally be classification-based so that it is amenable to the translate-train approach (since in classification tasks, the labels are preserved even after data has been translated into another language).

The datasets that best met these conditions were AmericasNLI (Ebrahimi et al., 2022), a natural language inference (NLI) dataset covering 10 low-resource languages from the Americas, and NusaX-senti (Winata et al., 2023), a sentiment analysis (SA) dataset spanning 10 low-resource Indonesian languages. In the NLI task, the source language is English, while for SA it is Indonesian. We provide the list of all datasets and languages used in Table 6.1.

In a less constrained environment, it would be possible to expand the set of evaluation tasks by training MMTs and NMT models on corpora where the amount of data for certain languages was artificially restricted to simulate low-resourceness. This would enable us to study the effectiveness of our techniques across an arbitrary combination of resource levels; as it is, there tends to be a strong correlation between the availability of different kinds of resource, e.g. monolingual and parallel data.

6.4.2 Models and Training Details

MMT. In this work, we use the base version of XLM-R (Conneau et al., 2020), an MMT with 270M parameters pre-trained on 100 languages.[1]

NMT model. As our primary MT model for obtaining translated data, we choose the NLLB model with 3.3B parameters (NLLB Team et al., 2022), trained to translate between any pair of 200+ languages, including many low-resource languages. We also experiment with two additional NLLB variants: distilled models with 600M and 1.3B parameters, enabling us to understand the effect of model size on the "quality" of the obtained data. Despite the broad language coverage, half of our target languages are unsupported by the NLLB models (7 languages from the AmericasNLI and 3 languages from the NusaX dataset).

We adapt the 3.3B parameter NLLB model to unseen languages through continued pre-training on the parallel corpora listed in Appendix D.1. We perform full fine-tuning for 5 epochs with a batch size of 8 and an initial learning rate of 2×10^{-5} which is linearly decreased to zero during training.

Language Modules. In general, we use the same algorithms and hyperparameters as the original papers (Pfeiffer et al., 2020b; Ansell et al., 2022) when training language modules. However, we use the variant of MAD-X proposed by Pfeiffer et al. (2021b), where the last adapter layers are dropped for an increase in cross-lingual transfer performance.

[1]While more powerful MMTs are available, such as XLM-R-large or mDeBERTa (He et al., 2023), our primary purpose is not the maximisation of raw performance, nor a comparison of different MMTs, so we opt for a smaller model to stretch our computational budget over a broad range of scenarios and languages.

We provide a list of resources for the monolingual corpora in Appendix D.1. Language modules are trained for a minimum of 100 epochs and 100,000 steps with a batch size of 8, a learning rate of 5×10^{-5} and a maximum sequence length of 256. We evaluate the language modules every 1,000 steps with low-resource languages, and every 5,000 steps with high-resource languages. Finally, we choose the module that has obtained the lowest perplexity on the validation set, which is created by taking 5% of the unlabelled data for low-resource languages or 1% for high-resource languages.

Task Modules. We again follow Pfeiffer et al. (2020b) and Ansell et al. (2022) except where stated otherwise. We train task adapters with a reduction factor of 16 (i.e. the ratio between the dimension of the MMT hidden state and the "bottleneck" dimension of the adapter hidden state is 16) and task SFTs with 8% density. When jointly training on data from more than one language, the training examples are batched such that each batch consists of examples from a single language, and the batches are ordered randomly. For the configurations which employ language adaptation, the language module for the relevant language is activated at the beginning of the training step and deactivated at the end of the step, following Ansell et al. (2021).

AmericasNLI task modules are trained for 5 epochs with a batch size of 32 and an initial learning rate of 2×10^{-5}. Evaluation is carried out every 625 steps and the checkpoint with the best evaluation accuracy is selected at the end of training. NusaX task modules are trained for 10 epochs (or 3 during the full fine-tuning phase of LT-SFT), with a batch size of 16 and an initial learning rate of 2×10^{-5}. They are evaluated after every 250 steps and the final module is the one with the best evaluation F1 score. For both tasks, the learning rate is linearly decreased to zero over the course of training.

6.4.3 Configurations and Ablations

ZS-XLT. We include zero-shot transfer results with language adaptation, equivalent to MAD-X (Pfeiffer et al., 2020b) in the case of adapters. We also have a variant where language adaptation is not employed, thus only the task module is used for training and inference. These variants are denoted by ZS and ZS − LA, respectively.

FS-XLT. In our default FS-XLT setup, "FS-SINGLE", we add $K = 100$ target shots to the source language task data, training a separate task module for each target language. We also consider "FS-ALL", where a single task module is trained on the source language data plus $K = 100$ shots from each target language. We investigate the effect of different

numbers of shots by also carrying out FS-SINGLE experiments with $K \in \{20, 500\}$.[2] We employ language adaptation in all these setups, but as an ablation, we also test $K = 100$ without language adaptation (denoted as FS − LA).

In all FS experiments, the model is jointly trained on source and target data, as per Xu and Murray (2022). We upsample the data in the target language(s) by a factor of 10 to increase its presence during training, since K is still rather small compared to the number of examples available in the source language. When evaluating during training for checkpoint selection purposes, we only use the source language validation data following Xu and Murray (2022), who point out that the presence of large validation sets in truly low-resource languages is unrealistic[3], and show that while validating on the target language is still beneficial for the joint training procedure, the gap becomes much smaller. They stress that such data would be better used for training, in line with Kann et al. (2019).

Translate-Train. In our main translate-train variant, named TTRAIN-ALL, we create a single task module covering all the target languages, which is trained and evaluated on the translated data of all target languages together with the source language data. We also consider TTRAIN-SINGLE, where a separate task module is trained on the data of each target language alone.

FS-XLT meets Translate-Train. Here, we investigate to what extent the benefits gained from the few-shot and translate-train methods add up when they are combined. To test this, we introduce the FS + TTRAIN-ALL configuration, where we train a single task module on the union of the source language data and translated and few-shot data (with $K = 100$) for every target language. This module is evaluated on the source language data and the translated data in all target languages.

Translate-Test. As a point of comparison, we provide a simple translate-test baseline (denoted TTEST). We use the NLLB 3.3B model (with adaptation in the target-to-source direction for unsupported languages) to translate the test set for each target language into the source language. We then evaluate the model checkpoint trained for the ZS configuration (with SFT as the fine-tuning method) on the translated test set for each target language.

[2]For the AmericasNLI target languages NAH and OTO we actually use 223 and 377 shots respectively under the $K = 500$ setting since this is the maximum available.

[3]The requirement for validation data in the target language originated from two-step methods, where such data is needed to prevent the model from overfitting to the small number of examples in the target language used during the second stage.

6.5 Results and Discussion

(a) AmericasNLI: accuracy

	Method	AYM	BZD*	CNI*	GN	HCH*	NAH*	OTO*	QUY	SHP*	TAR*	avg
ADAPTER	ZS	53.0	42.8	44.8	59.6	40.3	50.8	41.2	55.2	50.0	40.0	47.77
	FS-SINGLE	55.2	47.1	47.7	58.3	40.7	55.8	45.9	59.7	52.9	47.5	51.08
	FS-ALL	57.1	47.9	49.3	59.1	44.1	54.5	**48.4**	59.6	52.3	47.5	51.98
	TTRAIN-SINGLE	58.7	56.5	53.9	64.4	48.7	56.4	40.1	61.5	57.5	51.5	54.92
	TTRAIN-ALL	63.3	**60.5**	56.1	66.1	**52.1**	**60.4**	42.4	**64.9**	63.3	55.3	58.44
	FS + TTRAIN-ALL	**63.6**	59.1	**57.7**	**67.3**	**52.1**	59.6	48.3	64.5	**64.4**	**56.7**	**59.33**
SFT	ZS	58.4	44.7	47.6	62.2	44.4	50.8	46.4	60.4	49.5	43.1	50.75
	FS-SINGLE	59.2	58.7	53.2	63.9	45.9	54.6	49.1	61.2	53.1	51.2	55.01
	FS-ALL	60.8	58.1	52.3	63.3	47.3	56.5	**53.3**	61.3	54.9	51.3	55.91
	TTRAIN-SINGLE	64.5	58.7	54.9	69.2	53.2	61.5	43.4	65.1	59.5	56.1	58.61
	TTRAIN-ALL	**65.5**	61.9	56.7	**70.5**	**55.5**	**62.2**	42.6	68.5	66.0	58.7	60.81
	FS + TTRAIN-ALL	65.1	**62.1**	**58.5**	70.0	54.8	61.8	51.2	**68.8**	**67.6**	**60.3**	**62.02**
	TTEST	47.1	40.7	35.1	60.3	37.2	40.2	38.0	54.5	42.1	37.5	43.26

(b) NusaX: F1

	Method	ACE	BAN	BBC*	BJN	BUG	JAV	MAD*	MIN	SUN	avg
ADAPTER	ZS	74.9	78.0	72.3	77.6	57.6	82.9	68.5	79.9	80.5	74.69
	FS-SINGLE	79.1	80.5	77.2	86.0	72.0	85.0	77.5	84.8	83.7	80.64
	FS-ALL	79.5	80.0	75.3	**86.2**	70.4	**86.2**	76.8	84.4	82.4	80.13
	TTRAIN-SINGLE	74.0	77.7	73.8	82.0	66.6	83.6	70.8	78.0	79.7	76.24
	TTRAIN-ALL	79.6	82.3	81.0	85.0	68.1	**86.2**	78.8	83.3	**85.7**	81.11
	FS + TTRAIN-ALL	**82.3**	**82.9**	**82.8**	84.9	**72.2**	85.3	**80.5**	**85.7**	85.0	**82.40**
SFT	ZS	80.0	81.3	65.8	82.0	63.8	84.3	73.5	86.6	84.4	77.97
	FS-SINGLE	82.2	84.1	80.6	**88.3**	77.7	**88.0**	78.6	**89.2**	85.1	83.76
	FS-ALL	83.7	**87.2**	79.6	87.9	75.9	87.3	77.2	86.7	84.0	83.30
	TTRAIN-SINGLE	82.4	82.0	**83.5**	85.4	68.1	85.6	80.9	87.0	83.0	81.93
	TTRAIN-ALL	83.4	82.4	82.7	84.6	76.8	86.4	**81.6**	87.4	85.2	83.39
	FS + TTRAIN-ALL	**85.0**	85.8	82.7	87.4	**78.7**	87.4	81.0	88.8	**86.5**	**84.81**
	TTEST	81.2	80.4	55.8	81.8	71.1	84.5	70.1	86.5	84.9	77.36

Table 6.2 Results of ZS, FS, TTRAIN and TTEST methods on AmericasNLI and NusaX with bottleneck adapters and SFTs. For the FS methods, K = 100, and for the TTRAIN and TTEST methods, the MT model is NLLB with 3.3B parameters. The last column is the average score over all languages. **Bold**: the best approach within adapter/SFT. Underline: overall best score. *: NLLB MT model adaptation required to support this language.

Main Results. The results of our primary configurations on NLI and SA are presented in Table 6.2, with ablations shown in Table 6.3.[4] We find that the various cross-lingual

[4]For Nusa-X, our main results exclude NIJ for which no monolingual data is available and thus no language module was trained; results without language adaptation are available in Appendix D.2.

transfer techniques we consider can be combined very effectively to improve performance. For instance, the average SA performance can be improved from the most basic ZS – LA setting by 14-17 points (depending on the PEFT method) through the use of language adaptation, translate-train and few-shot techniques with $K = 100$ shots (FS + TTRAIN-ALL). In the case of the NLI task, the gains under the same conditions are 19-24 points. Each of these components individually adds several points of performance, and although the gain from using FS and TTRAIN together is much smaller than the sum of their individual gains, it is still 1-2 points better than using either technique on its own. Although we did not consider FS + TTRAIN-ALL with $K = 500$ shots, the strength of FS-SINGLE with $K = 500$ as shown in Figure 6.1 suggests that this gap would be larger with larger K. The finding that high-quality machine translation of the entire source dataset cannot eliminate the utility of human-crafted examples contains a potentially useful lesson – we would encourage designers of datasets for cross-lingual transfer to provide at least two splits for target languages, even if the training/validation split contains only 100 examples. The relative value of few-shot and machine-translated data appears to be task-dependent. Whereas for AmericasNLI we see TTRAIN outperforming FS by 4-6 points, neither approach has a clear advantage on the NusaX task.

While TTEST is competitive with ZS (and to some extent with TTRAIN-SINGLE) on NusaX, it falls well short on AmericasNLI, even for languages where NMT model adaptation is not required. This discrepancy likely results from the translation quality for the AmericasNLI languages being lower than for the NusaX languages, since the AmericasNLI are not closely related to any high-resource language seen during NLLB or XLM-R pre-training. While translate-train can to some extent learn to adjust for the poor quality of the translated data it sees during training, in translate-test, the fine-tuned task model is presented zero-shot at inference time with translationese, which is likely to be of a particularly perverse kind for the AmericasNLI languages (hence Artetxe et al. (2020a)'s suggestion to fine-tune the classification model for translate-test on back-translated data instead of "natural" data from the source language).

MT Model Size. In Table 6.4, we see the effect of translation model quality on TTRAIN-ALL performance, with gains of 0.5-2 points from upscaling the NLLB model from 600 million to 3.3 billion parameters. This upscaling comes at a relatively small cost in the translate-train setup, since the training data only needs to be translated once for each target language. Translate-test setups, on the other hand, incur the cost of translating each example encountered at inference time, which is potentially much more costly for large-scale deployments.

	Adapter		SFT	
Method	NLI	SA	NLI	SA
ZS	**47.77**	**74.69**	**50.75**	**77.97**
ZS – LA	40.57	65.38	38.21	70.43
FS	**51.08**	**80.64**	**55.01**	**83.76**
FS – LA	44.82	74.03	49.51	78.32
FS – UPSAMPLE	48.27	76.78	52.60	83.18
TTRAIN-ALL	**58.44**	**81.11**	**60.81**	**83.39**
TTRAIN-ALL – LA	55.50	77.83	57.88	79.39

Table 6.3 Ablation experiments: – LA denotes the corresponding method without language adaptation (i.e. only task module is present); – UPSAMPLE indicates no upsampling of the gold-standard target shots is performed. FS refers to the -SINGLE variant. The scores are averages over all target languages.

		NLI				SA							
	Model	AYM	GN	QUY	**avg**	ACE	BAN	BJN	BUG	JAV	MIN	SUN	**avg**
Adapter	NLLB 600M	**62.9**	65.1	63.2	63.73	76.6	79.0	83.7	66.0	80.9	81.5	81.1	78.40
	NLLB 1.3B	62.3	**67.9**	63.2	64.47	73.4	80.2	**85.3**	64.8	83.4	80.6	83.0	78.67
	NLLB 3.3B	62.4	67.6	**65.2**	**65.07**	**79.6**	**82.4**	84.6	**66.4**	**84.8**	**81.9**	**84.6**	**80.61**
SFT	NLLB 600M	**67.3**	69.7	68.7	68.57	**83.8**	81.2	84.6	72.8	85.7	84.2	82.1	82.06
	NLLB 1.3B	66.4	**72.0**	69.2	**69.20**	81.3	81.9	84.2	68.5	84.5	84.2	83.1	81.10
	NLLB 3.3B	65.7	70.9	**70.5**	69.03	83.2	**83.7**	**85.1**	**76.8**	**88.1**	**87.6**	**83.7**	**84.03**

Table 6.4 Results of TTRAIN-ALL method on the target languages supported by the NLLB model(s) on AmericasNLI (accuracy) and NusaX (F1 score) with bottleneck adapters and SFTs. The models are labelled by their number of parameters.

MT adaptation. The adaptation of the NMT model to new languages appears to be highly effective, with these languages generally enjoying large gains from the use of TTRAIN despite the small size of the parallel corpora available: the AmericasNLI languages have parallel corpora containing 5,000-17,000 sentences, while the NusaX parallel corpora have fewer than 1,000 sentences. It is interesting to note that for AmericasNLI, the language pairs used in MT model adaptation are different from those used during cross-lingual transfer: the source language in the parallel corpora is Spanish, so the English-to-X direction required during translation of the MultiNLI dataset is completely unseen. The success of the TTRAIN configurations on this task is thus a testament to the strength and flexibility of multilingual NMT.

Number of Shots. We observe a rather large impact on performance from increasing the amount of few-shot data. While even 20 shots are enough to bring about a 3-5 point

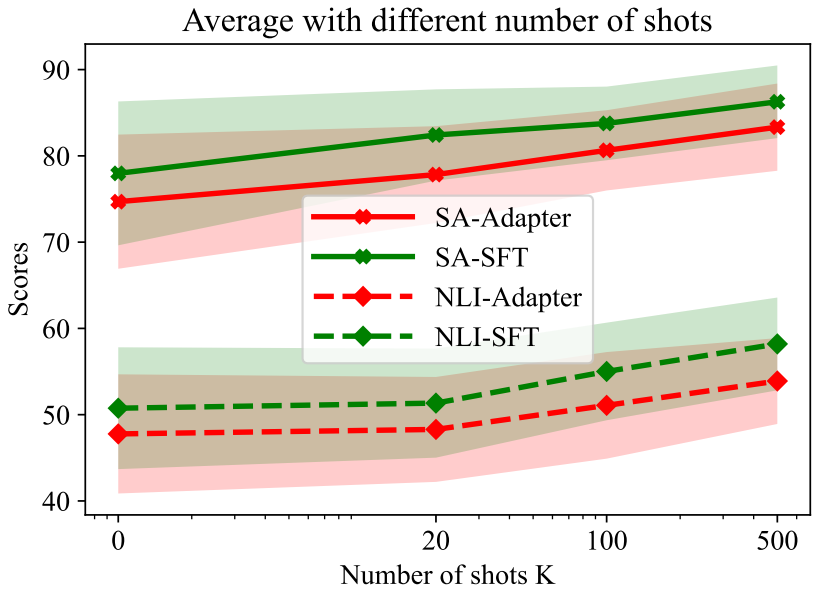


Fig. 6.1 Performance when the number of gold-standard target shots K varies, taking values 0, 20, 100, and 500. We show the average across all target languages; the shaded area is the standard deviation (full results available in Appendix D.3).

average gain on the NusaX task, we do not see a plateau in performance on either task even with the increase from 100 to 500 shots. Upsampling the few shots seems beneficial, yielding a 0.5-4 point gain in performance when $K = 100$. A finer-grained and wider exploration of this finding is warranted in future work.

Language Resourcefulness. As suggested by prior work (Pfeiffer et al., 2020b; Ansell et al., 2021) and by Table 6.3, language adaptation has a very large impact on all configurations. In the case of the TTRAIN-ALL configuration, language adaptation yields gains of 3-4 points, while for ZS and FS they vary between 6-13 points. Figure 6.2 further illustrates this effect on all three methods with bottleneck adapters, showing the gains from language adaptation on NLI languages against the size of their monolingual corpora: languages with larger corpora generally exhibit larger gains. The effect is visibly less pronounced for the TTRAIN-ALL where a relatively large amount of the target data (albeit translated) lessens the significance of language modules and monolingual corpora size. Finally, Figure 6.3 illustrates that the performance of our strongest configuration (FS + TTRAIN-ALL) with SFTs is the highest for languages that are seen by either the NLLB or MMT model – these are (incidentally) the languages with the largest corpora sizes too. Conversely, this pattern is absent with the ZS – LA configuration.

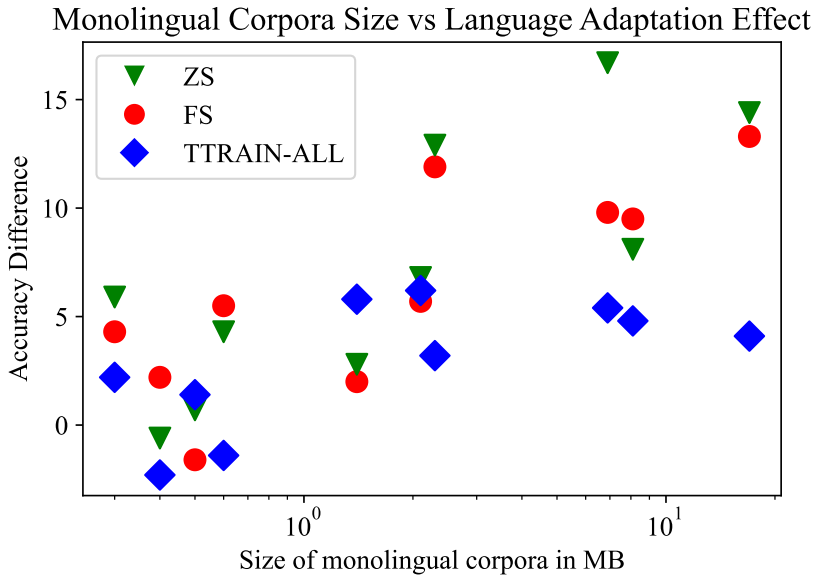


Fig. 6.2 Gains from language adaptation against the size of monolingual corpora for NLI target languages. We show the difference in the accuracy of ZS (i.e. ZS - ZS − LA), FS and TTRAIN-ALL with bottleneck adapters. Larger monolingual corpus size is correlated with larger gains from language adaptation; the effect is more pronounced with ZS and FS than with TTRAIN-ALL.

Language families for our target languages are given in Appendix D.1. While they could have an impact on the performance, it is difficult to disentangle the effect of the language family from the amount of data available without having a much larger set of evaluation languages.

PEFT Method. The relative performances of the various configurations are very similar regardless of PEFT method, and in accordance with previous work (Ansell et al., 2022; Alabi et al., 2022), SFTs consistently outperform bottleneck adapters by 2-3 points. However, we estimate that training adapters is generally around 3 times faster than training SFTs.

6.6 Conclusions

We have investigated how to combine several cross-lingual transfer techniques which are applicable across several dimensions of resource scarcity into a single framework. We find that parameter-efficient language adaptation, few-shot learning and translate-train are complementary when employed in a multi-source training setup with few-shot upsampling. However, our training setup supports the use of any subset of these techniques depending

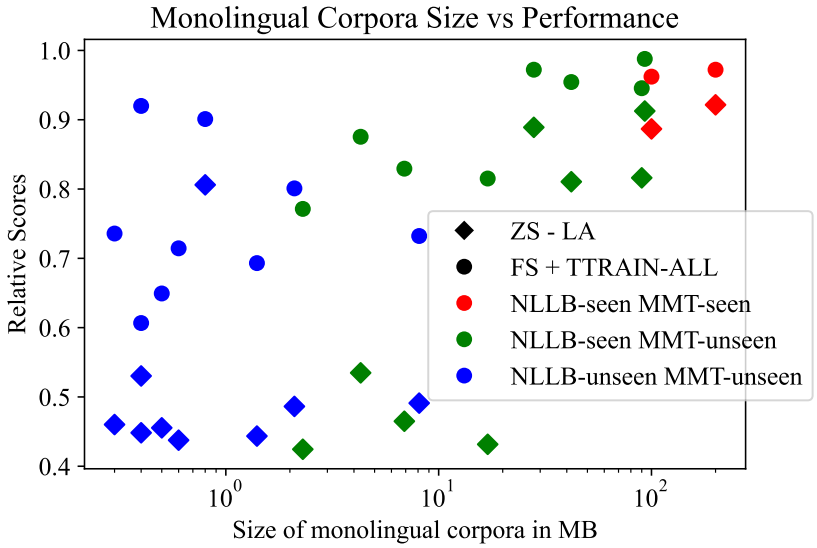


Fig. 6.3 The scores of ZS - LA and FS + TTRAIN-ALL methods with SFTs against the monolingual corpus sizes. All target languages (from NLI and SA) are shown and they are grouped based on the coverage by the NLLB and MMT models. Relative scores are displayed (a fraction of the source language performance).

on the availability of the necessary data and models. We remark on the significance of the finding that gold-standard few-shot target data can improve performance even when the entirety of the training data is translated into the target language by a high-quality NMT model. We also observe that languages not natively supported by an NMT model can benefit from translate-train through a simple adaptation procedure even with a small amount of parallel data.

Our code is available at https://github.com/parovicm/unified-xlt.

Chapter 7

Scaling Sparse Fine-Tuning to Large Language Models

7.1 Introduction

The scale of Large Language Models (LLMs), such as Falcon (Almazrouei et al., 2023), Llama 2 (Touvron et al., 2023b), and Mistral (Jiang et al., 2023), is one of the keys to their state-of-the-art performance (Kaplan et al., 2020; Hoffmann et al., 2022). However, this scale is both a blessing and a curse as tailoring LLMs to specific applications via fine-tuning presents a formidable challenge: if performed naively, this incurs the cost of updating an incredibly large set of parameters. The reader should by now be well acquainted with the family of lightweight methods for LLM adaptation that have been proposed to mitigate this issue, known collectively as Parameter-Efficient Fine-Tuning (PEFT). PEFT methods learn a small number of new parameters, denoted as $\boldsymbol{\phi}$, which augment the frozen LLM weights $\boldsymbol{\theta}$ (Pfeiffer et al., 2023b; Lialin et al., 2023). For instance, Low-Rank Adapters (LoRA; Hu et al., 2022) learn additional low-rank matrices to modify the linear layers in Transformer blocks.

PEFT methods based on unstructured sparse fine-tuning (SFT), where a sparse vector $\boldsymbol{\phi}$ is added to $\boldsymbol{\theta}$, have shown promise in our work and elsewhere (Sung et al., 2021; Guo et al., 2021). These offer a strong trade-off between low count of updated parameters and high model performance without inserting additional layers into the LLM's neural architecture, which would reduce inference efficiency. In addition, multiple SFTs are *composable* while avoiding interference (Ansell et al., 2022), which facilitates the integration of multiple sources of knowledge into LLMs. Formally, SFT can be conceived of as performing joint optimisation over the fixed-size set of non-zero indices of $\boldsymbol{\phi}$ and their deltas with respect to the LLM weights. Due to the intricacies of this optimisation,

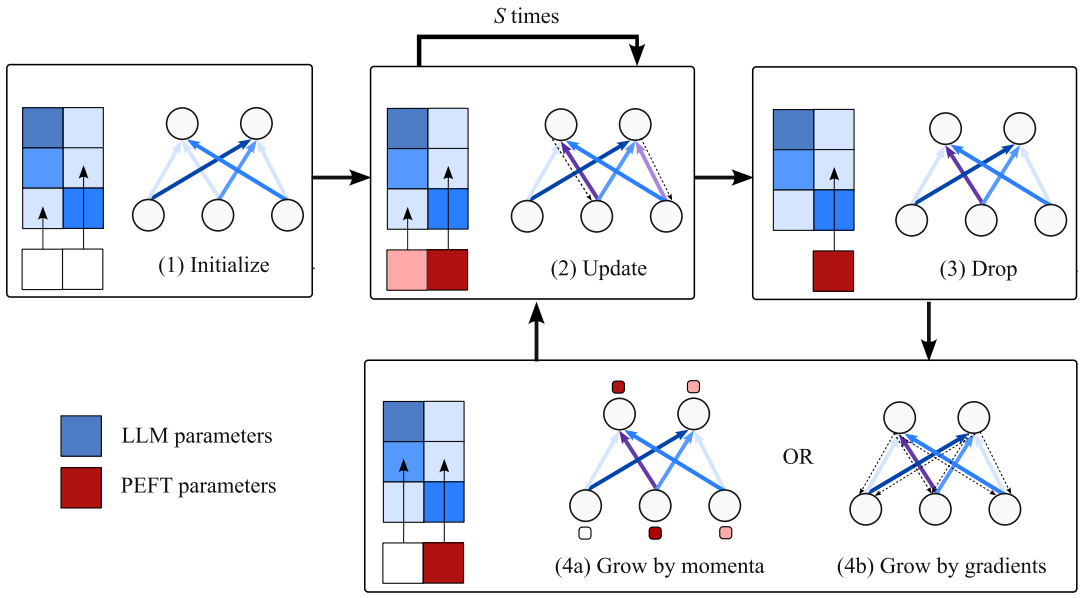


Fig. 7.1 A visualisation of the proposed Sparse Fine-Tuning (SFT) method scaled to a Large Language Model (LLM). PEFT parameters consist of indices (arrows) and corresponding deltas (red squares) with respect to LLM parameters (blue squares). After initialisation (1), PEFT deltas are updated for S steps (2). Next, obsolete indices are dropped (3) and new indices are grown (4) according to either accumulated gradients or approximate momenta. The algorithm then returns to the update step (2) and is repeated iteratively.

however, SFT has so far been severely limited by a major drawback, namely, its high memory requirements: existing methods for selecting non-zero indices include learning a mask (Guo et al., 2021), estimating the Fisher information (Sung et al., 2021), or calculating the difference between initialisation and convergence (Ansell et al., 2022) for *all LLM parameters*. Hence, SFT is not currently suitable for adapting LLMs at large scales.

The main goal of this work is to overcome these challenges by devising memory-efficient methods to update Large Language Models (LLMs) sparsely, while maintaining performance benefits, that is, retaining the same performance of full-model fine-tuning or even surpassing it. Specifically, we wish for the memory use during training (beyond that required to store the pre-trained model weights) to scale linearly with the number of SFT parameters $\mathcal{O}(d_{\boldsymbol{\phi}})$ rather than LLM parameters $\mathcal{O}(d_{\boldsymbol{\theta}})$. To achieve this, we introduce SpIEL (Sparse Iterative Efficient Learning), an iterative paradigm for SFT that alternates between updating deltas of active indices, discarding obsolete indices, and growing new ones (Evci et al., 2020). Discarding is determined by change of magnitude between training steps whereas growth is determined by largest (transiently calculated) gradient

magnitudes. The growth criterion is inspired by Evci et al. (2020), which we further improve upon by *efficiently accumulating* gradients to reduce their variance. Moreover, while our algorithm (SpIEL-AG) is reminiscent of sparse training (Evci et al., 2020), the scattering of the SFT onto the base LLM effectively yields a dense model. This side-steps the problem of the "hardware lottery" (Hooker, 2021), which disadvantages algorithms which rely on sparse tensor operations given that these are currently not efficiently implemented by many software packages and GPU architectures. We provide a visual overview of our algorithms in Figure 7.1.

When extreme memory efficiency is required, we show how SpIEL can be combined with efficient optimisers such as SM3, where the momenta of parameter matrices are approximated by row-wise and column-wise summary metrics (Anil et al., 2019). In these settings, gradients become unreliable, as their variance increases in single-example mini-batches and it is costly memory-wise to accumulate even a subset of them. Hence, we propose that approximate momenta can additionally substitute gradients as a criterion for growth, yielding the SpIEL-MA model variant.

We compare our SpIEL variants with the state-of-the-art PEFT methods LoRA (Hu et al., 2022) and $(IA)^3$ (Liu et al., 2022), as well as with full fine-tuning, starting from Llama 2 (Touvron et al., 2023b) as a base model. We instruction-tune them on multi-task data such as Flan v2 (Longpre et al., 2023), data generated by proprietary models such as GPT4-Alpaca (Peng et al., 2023), or a mixture of both with Tülu v2 (Ivison et al., 2023). We extensively evaluate the resulting models on standard benchmarks for factuality (MMLU; Hendrycks et al., 2021), reasoning (GSM and BBH; Cobbe et al., 2021; Suzgun et al., 2023), multilinguality (TyDiQA; Clark et al., 2020a), and coding (HumanEval; Chen et al., 2021).

The main results reveal that SpIEL outperforms the PEFT and full fine-tuning baselines on most tasks and configurations we test, both with and without 4-bit LLM quantisation during fine-tuning (Dettmers et al., 2023). In combination with the SpIEL-MA variant, this allows for scaling fine-tuning to very large LLMs with a modest memory footprint.

7.2 Background and Related Work

7.2.1 Parameter-Efficient and Memory-Efficient Fine-Tuning

Recall from §2.2 the definition of parameter-efficient fine-tuning (PEFT) methods as those which fine-tune a small number of parameters relative to the total size of the pre-

trained model. The possible benefits of using a PEFT method as opposed to full model fine-tuning include: (1) reduced GPU memory usage during training; (2) faster training; (3) faster saving and loading of the fine-tuning with less permanent storage required, as the "frozen" original weights of the large underlying model are shared across multiple tasks and applications[1]; (4) the "composability" property of some PEFT methods, which allows modules to be combined with less interference than with full fine-tuning (Pfeiffer et al., 2020b; Ansell et al., 2022); and (5) less tendency to overfit due to the reduced capacity of PEFT with respect to the full model.

Of these, (1) is perhaps the most critical in the era of Large Language Models whose GPU memory requirement for full fine-tuning is beyond the reach of researchers and developers without multiple high-end GPUs. Nonetheless, parameter-efficiency alone does not guarantee a reduction in GPU memory usage, though it almost certainly implies that less space is required to save the fine-tuning in permanent memory. It is thus important to draw a distinction between (i) efficiency in the number of fine-tuned parameters versus (ii) the peak GPU memory usage during fine-tuning: we refer to the former as *parameter efficiency* and the latter as *memory efficiency*.

7.2.2 Sparse Fine-Tuning

Recall from §4.2.1 that a sparse fine-tuning F' of a neural function F entails the addition of a sparse "difference" or "delta" vector $\boldsymbol{\delta} \in \mathbb{R}^{d_{\boldsymbol{\theta}}}$ to its parameters $\boldsymbol{\theta} \in \mathbb{R}^{d_{\boldsymbol{\theta}}}$, i.e. $F'(\ ;\ \boldsymbol{\theta}) = F(\ ;\ \boldsymbol{\theta} + \boldsymbol{\delta})$. A delta vector $\boldsymbol{\delta}$ with $d_{\boldsymbol{\phi}}$ non-zero values can be expressed in terms of a vector of unique indices $\boldsymbol{\eta} \in \{1, 2, ..., d_{\boldsymbol{\theta}}\}^{d_{\boldsymbol{\phi}}}$ and their corresponding values $\boldsymbol{\phi} \in \mathbb{R}^{d_{\boldsymbol{\phi}}}$. Typically, the SFT density $\frac{d_{\boldsymbol{\phi}}}{d_{\boldsymbol{\theta}}}$ is a user-definable hyperparameter. We provide an illustration of SFT adapting a Transformer block in Figure 7.2, alongside LoRA, an alternative PEFT method which is the main baseline in this chapter.

In its most general form, sparse fine-tuning can be interpreted as performing joint optimisation over $\boldsymbol{\eta}$ and $\boldsymbol{\phi}$:

$$\boldsymbol{\eta}^{\star}, \boldsymbol{\phi}^{\star} = \arg\max_{\boldsymbol{\eta}, \boldsymbol{\phi}} \log \mathbb{P}(\mathcal{D} \mid \boldsymbol{\theta}, \boldsymbol{\eta}, \boldsymbol{\phi}). \tag{7.1}$$

A specialised optimisation approach is required for $\boldsymbol{\eta}$ since it is a discrete latent variable and cannot be directly optimised through stochastic gradient descent. Approaches

[1]Although loading and saving checkpoints are usually one-off costs, the benefit may be significant when training with frequent checkpoint saving, or when there are many task-specific model variants, not all of which can be kept in RAM/GPU memory simultaneously; this may occur on mobile devices for instance.

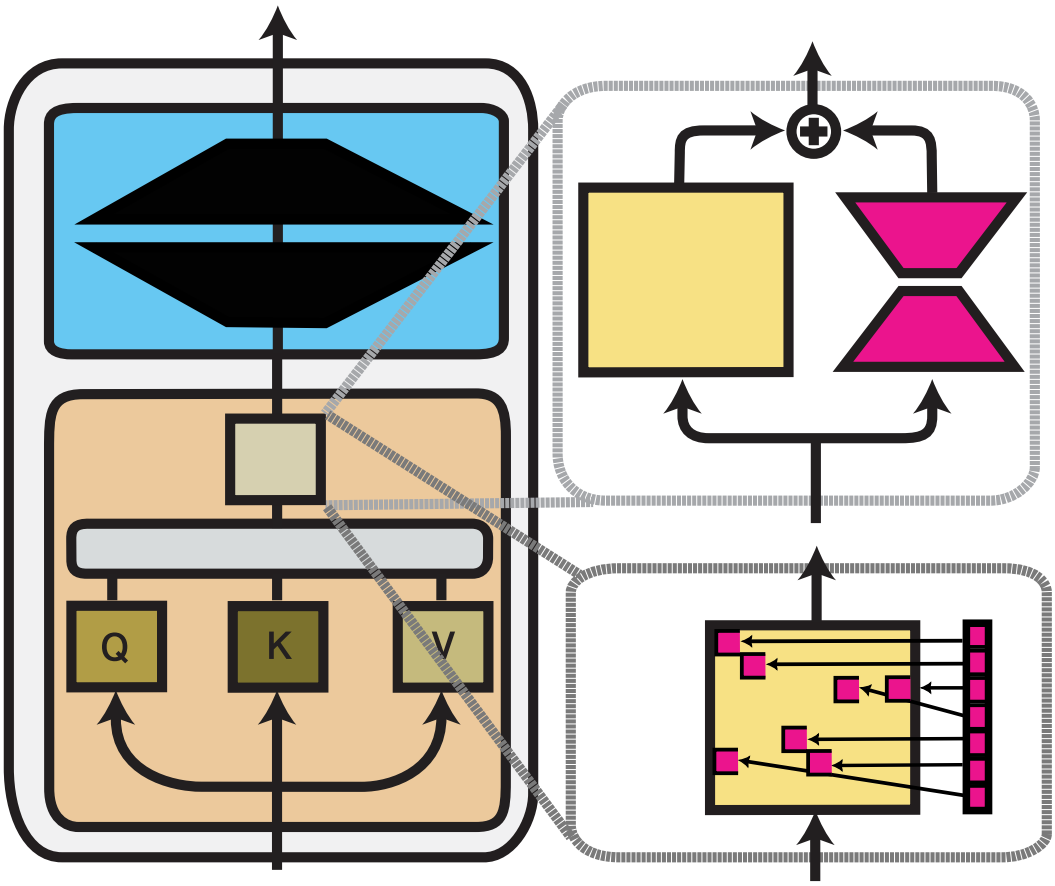


Fig. 7.2 SFT (lower right) versus LoRA (upper right) applied to a linear layer (the output projection of self-attention) of a Transformer block.

proposed in previous works include: *DiffPruning* (Guo et al., 2021), which applies a continuous relaxation of a binary mask to $\boldsymbol{\delta}$ during fine-tuning which is sparsified with a regularisation term, and takes $\boldsymbol{\eta}$ to be the $d_{\boldsymbol{\phi}}$ indices with mask values closest to 1 at the end; *FISH Mask* (Sung et al., 2021), where $\boldsymbol{\eta}$ is fixed at the beginning of training to be the indices of the $d_{\boldsymbol{\phi}}$ weights with highest observed Fisher information; and *Lottery-Ticket SFT* (LT-SFT), as proposed in §4.3.1, where $\boldsymbol{\eta}$ is fixed to be the indices of the $d_{\boldsymbol{\phi}}$ weights which change the most during an initial round of full fine-tuning. These methods share a common drawback, namely that the amount of memory they use during training in addition to that required to store the pre-trained model weights is proportional to $d_{\boldsymbol{\theta}}$, the total number of model parameters. As discussed above, this makes these methods prohibitively expensive in many LLM fine-tuning scenarios, especially with very large models; we thus seek a method whose memory overhead is instead proportional to $d_{\boldsymbol{\phi}}$, the number of parameters which are actually modified during fine-tuning.

Finally, there exists a separate but related literature on *pre-training* sparse neural networks, for which we refer the reader to Hoefler et al. (2021) for a detailed overview. This work owes most to Evci et al. (2020), whose dropping and growth paradigm we extend to sparse fine-tuning in §7.3.1. Contrary to their RigL method, however, SpIEL results in a *dense* model, which side-steps the problem that available hardware is not well suited to sparse tensor operations (Hooker, 2021). Moreover, we introduce novel and enhanced growth criteria in Section 7.3.1 and Section 7.3.2.

7.2.3 Quantised PEFT

A significant recent advance in efficient methods for NLP has been the development of quantisation methods which are suitable for LLMs and incur minimal degradation in performance. Dettmers et al. (2022) and later Dettmers et al. (2023) proposed 8-bit and 4-bit quantisation techniques for LLM parameter tensors that yield close to full-precision performance during inference or parameter-efficient fine-tuning. The qLoRA fine-tuning method of Dettmers et al. (2023) reduces memory usage by applying 4-bit quantisation to most of the pre-trained LLM parameters while storing the small number of additional trainable LoRA parameters in full precision and dequantising the pre-trained weights only when required; see §2.2.1 for further details. We show that sparse fine-tuning is also amenable to quantisation of the pre-trained weights. We use $\textsc{quant}(\cdot)$ and $\textsc{dequant}(\cdot)$ to denote 4-bit NormalFloat quantisation and dequantisation respectively with double quantisation (Dettmers et al., 2023).

7.3 Method

In practice, the weights $\boldsymbol{\theta}$ of a neural function are partitioned into a sequence of n_p "parameter tensors." To simplify indexing, we think in terms of the flattened versions of these tensors and refer to them as "parameter subvectors", denoted as $\{\boldsymbol{\theta}^{(1)} \in \mathbb{R}^{d_{\boldsymbol{\theta}^{(1)}}}, \boldsymbol{\theta}^{(2)} \in \mathbb{R}^{d_{\boldsymbol{\theta}^{(2)}}}, ..., \boldsymbol{\theta}^{(n_p)} \in \mathbb{R}^{d_{\boldsymbol{\theta}^{(n_p)}}}\}$. Similarly, we denote the sections of $\boldsymbol{\eta}$ and $\boldsymbol{\phi}$ corresponding to parameter subvector $\boldsymbol{\theta}^{(i)}$ as $\boldsymbol{\eta}^{(i)}$ and $\boldsymbol{\phi}^{(i)}$ respectively. We observe that, for a fixed $\boldsymbol{\eta}$ and assuming $\max_i d_{\boldsymbol{\theta}^{(i)}} < d_{\boldsymbol{\phi}}$, it is possible to perform sparse fine-tuning with the desired $\mathcal{O}(d_{\boldsymbol{\phi}})$ memory overhead by scatter-adding each $\boldsymbol{\phi}^{(i)}$ into its corresponding $\boldsymbol{\theta}^{(i)}$ (which stays frozen) before it is used:

$$\boldsymbol{\theta}'^{(i)} = \boldsymbol{\theta}^{(i)} + \text{scatter}(\boldsymbol{\eta}^{(i)}, \boldsymbol{\phi}^{(i)}, d_{\boldsymbol{\theta}^{(i)}}), \tag{7.2}$$

where $\text{scatter}(\boldsymbol{\eta}, \boldsymbol{\phi}, d) \in \mathbb{R}^d$ such that

$$[\text{scatter}(\boldsymbol{\eta}, \boldsymbol{\phi}, d)]_i = \sum_{j=1}^{d_{\boldsymbol{\phi}}} \mathbb{I}_{\boldsymbol{\eta}_j = i} \; \boldsymbol{\phi}_j. \tag{7.3}$$

The simplest way of calculating the gradient of $\boldsymbol{\phi}^{(i)}$ during the backward pass of backpropagation is by gathering the relevant indices of the gradient of $\boldsymbol{\theta}'^{(i)}$:

$$\frac{\partial \mathcal{L}}{\partial \phi_j^{(i)}} = \frac{\partial \mathcal{L}}{\partial \theta'^{(i)}_{\eta_j}}. \tag{7.4}$$

While Equation 7.4 requires the computation of the dense gradient of each $\boldsymbol{\theta}^{(i)}$, this can be disposed of as soon as the required values are gathered from it. Because $d_{\boldsymbol{\theta}^{(i)}} \ll d_{\boldsymbol{\theta}}$, this does not add significantly to the peak memory usage. Furthermore, we show in Appendix E.4 that for a linear layer, it is possible in principle to calculate the gradient of $\boldsymbol{\phi}^{(i)}$ without needing to compute the full gradient of $\boldsymbol{\theta}^{(i)}$, which we will explore in future work.

This implementation of sparse fine-tuning stands in contrast to previous approaches (e.g. Sung et al., 2021; Ansell et al., 2022) which, instead of vectors $\boldsymbol{\eta}$ and $\boldsymbol{\phi}$, maintain a binary mask $\boldsymbol{b}^{(i)} \in \{0, 1\}^{d_{\boldsymbol{\theta}}^{(i)}}$ which is applied to the gradient of $\boldsymbol{\theta}^{(i)}$ after it is calculated. Since $\boldsymbol{b}$ and the gradient of $\boldsymbol{\theta}$ both have size proportional to $d_{\boldsymbol{\theta}}$, the memory overhead of this implementation is $\mathcal{O}(d_{\boldsymbol{\theta}})$. Although the above implementation would enable the fixed-$\boldsymbol{\eta}$ stage of FISH Mask or LT-SFT to be performed with acceptable memory overhead, both methods incur $\mathcal{O}(d_{\boldsymbol{\theta}})$ memory cost when selecting $\boldsymbol{\eta}$.

7.3.1 SpIEL-AG: Accumulated Gradient SpIEL

Building on Evci et al. (2020)'s "Rigging the Lottery" (RigL) method for pre-training sparse neural networks, we propose SpIEL-AG. SpIEL-AG maintains a fixed-size $\boldsymbol{\eta}^{(i)}$ and $\boldsymbol{\phi}^{(i)}$ for each parameter subvector $\boldsymbol{\theta}^{(i)}$, but unlike the methods discussed in the previous section, it allows $\boldsymbol{\eta}^{(i)}$ to change dynamically during training. $\boldsymbol{\phi}^{(i)}$ is initialised[2] as $[0]^{d_{\boldsymbol{\phi}^{(i)}}}$ and $\boldsymbol{\eta}^{(i)}$ is initialised as a random subset of $\{1, 2, ..., d_{\boldsymbol{\theta}^{(i)}}\}$. Every S training steps, $\boldsymbol{\eta}^{(i)}$ is updated by freezing some of the currently trainable weights after resetting them to their pre-trained values (*"dropping"*), while unfreezing some of the currently frozen weights

[2]The total number of tunable parameters $d_{\boldsymbol{\phi}}$ is a hyper-parameter, and $d_{\boldsymbol{\phi}^{(i)}}$ is set such that the proportion of tunable parameters $\frac{d_{\boldsymbol{\phi}^{(i)}}}{d_{\boldsymbol{\theta}^{(i)}}}$ is the same for each i-th parameter subvector.

(*"growth"*). A family of possible SpIEL methods arises from the choice of criteria for dropping and growth. For SpIEL-AG, we use the following criteria:

- DROP: the $k(i,t)$ weights in $\boldsymbol{\eta}^{(i)}$ which have changed the least from their pre-trained values, i.e. $\boldsymbol{\eta}^{(i)}_{\text{drop}} = \text{argtopk}(i,t)\ -|\boldsymbol{\phi}^{(i)}|$, where $k(i,t)$ is a schedule defining the number of weights in parameter subvector i to replace at step t.
- GROW: the $k(i,t)$ weights in $\boldsymbol{\theta}^{(i)}$ with largest estimated "long-run" gradient magnitudes, i.e. $\boldsymbol{\eta}^{(i)}_{\text{grow}} = \text{argtopk}(i,t)\ \left|\hat{\mathbb{E}}_{\boldsymbol{x}\sim\mathcal{D}}[\nabla_{\boldsymbol{\theta}^{(i)}}\mathcal{L}(\boldsymbol{x};\ \boldsymbol{\theta}')]\right|$.

The gradients required for growth selection are estimated over the γ training steps before $\boldsymbol{\eta}$ is updated, which we refer to as the *gradient estimation phase*. Here we diverge from Evci et al. (2020), who select the weights to grow on the basis of gradients from a single instantaneous minibatch.[3] It is not possible to maintain an estimate of the full gradient of dimension $d_{\boldsymbol{\theta}^{(i)}}$ without exceeding our memory budget. Therefore, we restrict ourselves to maintaining such an estimate for just K_i "growth candidates." These are selected as the weights in $\boldsymbol{\theta}^{(i)}$ with top K_i gradient magnitudes during the first batch of the gradient estimation phase. The long-run gradients of the growth candidates are estimated by averaging their gradients over all batches in the gradient estimation phase:

$$\hat{g}^{(i)}_j(t) = \frac{1}{\gamma}\sum_{s=t-\gamma+1}^{t}\frac{\partial}{\partial\theta^{(i)}_j}\mathcal{L}(\boldsymbol{x}_s;\ \boldsymbol{\theta}'). \tag{7.5}$$

Note that although it is necessary to calculate the dense gradient $\nabla_{\boldsymbol{\theta}^{(i)}}\mathcal{L}(\boldsymbol{x};\ \boldsymbol{\theta}')$ at each step of the gradient estimation phase, we never need to *store* it since we can immediately gather the K_i values we need from it. This can be implemented with a backward hook in PyTorch, for instance. In our experiments, we set $K_i = d_{\boldsymbol{\phi}^{(i)}}$.

There is some additional housekeeping to do after updating $\boldsymbol{\eta}$. We must reset $\boldsymbol{\phi}^{(i)}$ to zero at indices $\boldsymbol{\eta}^{(i)}_{\text{drop}}$. We also need to set the optimiser buffers for the newly grown weights $\boldsymbol{\eta}^{(i)}_{\text{grow}}$ to appropriate values. In our experiments, we use the Adam optimiser (Kingma and Ba, 2015), which tracks the exponentially smoothed means of the first and second momenta of the parameter gradients. Conveniently, the gradient estimation phase already produces an estimate of the first momentum of the newly grown weights, and we extend it to estimate the second as well so that we can use these values to seed the optimiser momenta. A minor complication here is that Adam multiplies the i-th momentum by a factor of $\frac{1}{1-\beta_i^t}$ before use to correct for its bias toward zero. Since in

[3] In a memory-constrained setting where it may be possible to process just a single training example at a time, the high level of variance in the gradients from one minibatch may harm the selection of weights to grow.

Algorithm 2 Accumulated Gradient SpIEL

procedure SpIEL-AG($\boldsymbol{\theta}, \mathcal{D}, \gamma, k$)
  $\boldsymbol{\phi} \leftarrow [0]^{d_{\boldsymbol{\phi}}}$ ▷ Initialise SFT values
  $\boldsymbol{\phi}_0 \leftarrow \boldsymbol{\phi}$
  $\boldsymbol{\eta} \sim_{d_{\boldsymbol{\phi}}} [1..d_{\boldsymbol{\theta}}]$ ▷ Initialise SFT indices
  for t in $1..T$ **do**
    $\mathbf{x}_t \sim \mathcal{D}$
    $\boldsymbol{\theta}' \leftarrow$ scatter-add$(\boldsymbol{\phi}, \boldsymbol{\eta}, \boldsymbol{\theta})$
    UPDATE$(\boldsymbol{\phi}, \nabla_{\boldsymbol{\phi}} \mathcal{L}(f_{\boldsymbol{\theta}'}(\mathbf{x}_t)))$
    if $\gamma \mid t$ **then**
      $\boldsymbol{d} \leftarrow$ top-$k(t)(-|\boldsymbol{\phi} - \boldsymbol{\phi}_0|)$
      $\boldsymbol{\phi}, \boldsymbol{\eta} =$ DROP$(\boldsymbol{\phi}, \boldsymbol{\eta}, \boldsymbol{d})$
      $\boldsymbol{g} \leftarrow$ top-$k(t)[\sum_{i=t-\gamma+1}^{t} \nabla_{\boldsymbol{\theta}} \mathcal{L}(f_{\boldsymbol{\theta}'}(\mathbf{x}_i))]$
      $\boldsymbol{\phi}, \boldsymbol{\eta} =$ GROW$(\boldsymbol{\phi}, \boldsymbol{\eta}, \boldsymbol{g})$
      $\boldsymbol{\phi}_0 \leftarrow \boldsymbol{\phi}$
  return $\boldsymbol{\phi}$, $\boldsymbol{\eta}$

SpIEL, different weights are "initialised" at different times, we track the age of each weight (i.e. the number of steps since it was last grown) individually, and use it to calculate the appropriate bias correction.

For simplicity, we set the update rate schedule $k(i, t)$ to decrease linearly to 0 over the course of training, as follows:

$$k(i,t) = \begin{cases} d_{\boldsymbol{\phi}^{(i)}} & \text{if } t = \gamma, \\ \frac{\xi(T-t)}{T} d_{\boldsymbol{\phi}^{(i)}} & \text{otherwise,} \end{cases} \tag{7.6}$$

where T is the total number of training steps and ξ is a hyperparameter denoting the peak replacement rate. Note that during the first $\boldsymbol{\eta}$ update at step γ, we replace all the weights, since the indices in $\boldsymbol{\eta}$ are randomly initialised; there is no reason to believe that they are relevant to the task.

We provide high-level pseudocode for SpIEL-AG in Algorithm 2. Note that some details such as the selection of candidates and optimiser momenta seeding are omitted for conciseness.

7.3.2 SpIEL-MA: Momentum-Approximation SpIEL

The SpIEL-AG method prioritises making a high-quality selection of weights to grow when updating $\boldsymbol{\eta}$, but this comes at the cost of the extra memory required during the gradient estimation phase to store the indices of the growth candidates and their estimated

momenta. We propose an alternative algorithm for the most memory-constrained scenarios, which we call SpIEL-MA, employing the SM3 memory-efficient adaptive optimiser of Anil et al. (2019). For a two-dimensional parameter tensor Θ of size $r \times c$, the SM3 optimiser maintains buffers $\boldsymbol{r} \in \mathbb{R}^r$ and $\boldsymbol{c} \in \mathbb{R}^c$, which contain running sums of the maximum squared gradients over the columns and rows of Θ, respectively. We can obtain a (low-quality but cheap) estimate of the absolute value of the momentum m_{ij} for each element θ_{ij} of Θ by taking the elementwise fourth root of the outer product of $\boldsymbol{r}$ and $\boldsymbol{c}$:

$$|\hat{m}|_{ij} = \sqrt[4]{r_i c_j}. \tag{7.7}$$

SpIEL-MA uses this estimate to rank weights for growth, and otherwise it is the same as SpIEL-AG. Since the SM3 optimiser is very memory efficient, storing only $r + c$ values in its buffers for an $r \times c$ parameter tensor compared to Adam's $2rc$, and SpIEL-MA uses no additional persistent memory to track statistics to inform $\boldsymbol{\eta}$ updates, significantly less memory in total is required than for SpIEL-AG. Incidentally, we remark that the growth criterion of SpIEL-MA assumes *locality*, that is, that the importances of parameters appearing in the same row and column are correlated. We provide high-level pseudocode for SpIEL-MA in Algorithm 3.

Regularising SpIEL Similar to LoRA, which is regularised by its dropout, SpIEL is also likely to overfit as the model diverges from its pre-trained state. We therefore regularise SpIEL by applying L2 regularisation to the $\boldsymbol{\phi}$ parameters in the form of weight decay (Loshchilov and Hutter, 2019) with strength λ.

Quantised SpIEL As another contribution, we extend the proposed SpIEL techniques to quantised LLMs ("qSpIEL"). Consider parameter matrix $W^{(i)}_{\text{PT}} \in \mathbb{R}^{h \times o}$ in its pre-trained data type (e.g., FP32). Instead of storing $W^{(i)}_{\text{PT}}$ itself on GPU, we store its quantised version $W^{(i)}_{\text{NF4}} = \textsc{quant}(W^{(i)}_{\text{PT}})$. During the forward pass of the linear module, qSpIEL computes the following:

$$Y = X(\textsc{dequant}(W^{(i)}_{\text{PT}}) + \Delta W^{(i)}), \tag{7.8}$$

where $X \in \mathbb{R}^{b \times h}$ is the input to the linear module, $Y \in \mathbb{R}^{b \times o}$ is the output, and

$$\Delta W^{(i)} = \text{reshape}(\text{scatter}(\boldsymbol{\eta}^{(i)}, \boldsymbol{\phi}^{(i)}, ho), [h, o]).$$

Algorithm 3 Momentum-Approximation SpIEL

procedure SPIEL-MA($\boldsymbol{\theta}^{(1)}, .., \boldsymbol{\theta}^{(n_p)}, \mathcal{D}, \gamma, k$)
  for i in 1..P **do**
    $\boldsymbol{\phi}^{(i)} \leftarrow [0]^{d_{\boldsymbol{\phi}^{(i)}}}$
    $\boldsymbol{\phi}_0^{(i)} \leftarrow \boldsymbol{\phi}^{(i)}$
    $\boldsymbol{\eta}^{(i)} \sim_{d_{\boldsymbol{\phi}^{(i)}}} [1..d_{\boldsymbol{\theta}^{(i)}}]$
    $\mathbf{r}^{(i)} = [0]^{h^{(i)}}$ ▷ Initialise row accumulator
    $\mathbf{c}^{(i)} = [0]^{w^{(i)}}$ ▷ Initialise column accumulator
  for t in 1..T **do**
    $\mathbf{x} \sim \mathcal{D}$
    $\boldsymbol{\theta}' \leftarrow \text{concat}_i \text{ scatter-add}(\boldsymbol{\phi}^{(i)}, \boldsymbol{\eta}^{(i)}, \boldsymbol{\theta}^{(i)})$
    for i in 1..n_p **do**
      UPDATE-SM3$[\boldsymbol{\phi}^{(i)}, \boldsymbol{\eta}^{(i)}, \mathbf{r}^{(i)}, \mathbf{c}^{(i)},$
        $\nabla_{\boldsymbol{\phi}^{(i)}} \mathcal{L}(f_{\boldsymbol{\theta}'}(\mathbf{x}))]$
      if $\gamma \,|\, t$ **then**
        $\boldsymbol{d}^{(i)} \leftarrow \text{top-}k(i,t)(-|\boldsymbol{\phi}^{(i)} - \boldsymbol{\phi}_0^{(i)}|)$
        $\boldsymbol{\phi}^{(i)}, \boldsymbol{\eta}^{(i)} = \text{DROP}(\boldsymbol{\phi}^{(i)}, \boldsymbol{\eta}^{(i)}, \boldsymbol{d}^{(i)})$
        $\boldsymbol{g}^{(i)} \leftarrow \text{top-}k(i,t)[\mathbf{r}^{(i)} \otimes \mathbf{c}^{(i)}]$
        $\boldsymbol{\phi}^{(i)}, \boldsymbol{\eta}^{(i)} = \text{GROW}(\boldsymbol{\phi}^{(i)}, \boldsymbol{\eta}^{(i)}, \boldsymbol{g}^{(i)})$
        $\boldsymbol{\phi}_0^{(i)} \leftarrow \boldsymbol{\phi}^{(i)}$
  return $\boldsymbol{\phi}^{(1)}, .., \boldsymbol{\phi}^{(n_p)}, \boldsymbol{\eta}^{(1)}, .., \boldsymbol{\eta}^{(n_p)}$

This is equivalent to the behavior of a linear module in ordinary, non-quantised SpIEL except that the pre-trained parameter matrix gets quantised at the beginning of training and temporarily dequantised each time it is used.

7.4 Experimental Setup

7.4.1 Training and Evaluation Data

To demonstrate the effectiveness of SpIEL, we loosely base our experimental setup on that of Wang et al. (2023), who compare different data mixtures. In particular, we instruction-tune LLMs on (i) Wang et al. (2023)'s 50K sub-sample of Flan v2 (Longpre et al., 2023), a dataset collecting manually annotated examples for multiple tasks augmented with instructions; (ii) GPT4-Alpaca (Peng et al., 2023), a dataset of 50K outputs generated by `davinci-003` and GPT-4 (OpenAI, 2023) prompted with inputs from Alpaca (Taori et al., 2023); or (iii) the Tülu v2 mixture (Ivison et al., 2023), consisting of 326K examples from multiple instruction following datasets.

Following Wang et al. (2023) and Ivison et al. (2023), we evaluate instruction-tuned LLMs on the following benchmarks to capture a range of abilities: Massively Multitask Language Understanding (MMLU; Hendrycks et al., 2021), Grade School Math (GSM; Cobbe et al., 2021), BIG-Bench Hard (BBH; Suzgun et al., 2023), Typologically Diverse Question Answering (TyDiQA; Clark et al., 2020a) and HumanEval (Chen et al., 2021). For each training data mixture, we evaluate on the most relevant downstream benchmarks, that is, those which experienced the largest performance gains from fine-tuning on that data mixture as reported by Wang et al. (2023). See Appendix E.1 for full details of our evaluation setup.

7.4.2 Models and Baselines

As LLMs to fine-tune, we choose state-of-the-art Llama 2 (Touvron et al., 2023b) at both 7b and 13b parameter scales. We report the performance of the unmodified "vanilla" models as a baseline for a series of PEFT methods. Specifically, we compare SpIEL with LoRA (Hu et al., 2022, see §2.2.1), as it offers the best performance–efficiency trade-off (Pfeiffer et al., 2023b) and is arguably the most widespread pre-existing PEFT method (Dettmers et al., 2023), as well as $(IA)^3$ (Liu et al., 2022). We use the LoRA and $(IA)^3$ implementations in the `peft` library (Mangrulkar et al., 2022). For the Tülu v2 fine-tuning mixture, we include the fully fine-tuned Llama 2 models of Ivison et al. (2023) as a baseline ("FullFT"), which we would expect to provide a soft upper bound on the performance of the PEFT methods.[4] At 7b scale, we also perform our own full fine-tuning on the Flan v2 and GPT4-Alpaca splits. We did not perform these experiments at 13b scale due to the computational expense.

Note that the pre-existing SFT methods (see §4.2.1) such as DiffPruning (Sung et al., 2021), FISH Mask (Guo et al., 2021), and LT-SFT (Ansell et al., 2022) are not viable as baselines for LLM fine-tuning as their memory complexity scales as $\mathcal{O}(d_{\boldsymbol{\theta}})$, similar to full fine-tuning.

7.4.3 Training Details and Hyperparameters

To select the most important hyperparameters of the PEFT methods, we perform a hyperparameter search as detailed in §7.4.4 with respect to (i) the number of trainable parameters (determined by rank r in LoRA or corresponding density in SpIEL)[5]; (ii)

[4]These follow a similar hyper-parameter setup except for a much higher maximum context length of 8,192.

[5]The number of trainable parameters for $(IA)^3$ is not tunable.

the learning rate; (iii) weight decay strength λ for SpIEL methods. We use the found hyperparameters of each PEFT method for all fine-tuning experiments.

For SpIEL, we update the set of trainable parameters $\boldsymbol{\eta}$ every $S = 20$ steps, fix the initial update rate ξ to 0.2, and for SpIEL-AG, we set the length γ of the gradient estimation phase to 5 steps. Note that we have performed only minimal manual tuning on the values of S, ξ and γ due to a large number of large-scale experiments coupled with constraints on our computational budget; it is possible that other values would yield better results.

We follow Wang et al. (2023)'s choice for the remaining hyperparameters: we always train for two epochs, and apply linear learning rate decay following warmup over the first 3% of training steps. The batch size is fixed at 128 and the maximum sequence length at 2,048, with longer sequences truncated. The LoRA dropout rate is set to 0.1 and LoRA α to 16. LoRA and SpIEL are applied to all linear Transformer block layers. The pre-trained parameters are stored in `bfloat16` data type, while the PEFT parameters are stored in `float32`, except for quantised training, where the pre-trained parameters are stored in `NormalFloat4` and dequantised into `bfloat16`, and the PEFT parameters are also stored in `bfloat16`.

For full fine-tuning, we use the learning rate of 2×10^{-5} from Wang et al. (2023). To fit the model into a single A100, we additionally resort to activation checkpointing and paged optimisation.

7.4.4 Hyperparameter Search

We first performed a grid search over (i) learning rate and (ii) number of PEFT parameters (determined by rank r for LoRA and d_ϕ for SpIEL), except for $(IA)^3$ where this number is fixed. For the SpIEL methods, we also wished to establish a good value for (iii) weight decay strength λ. Since we generally obtained a good performance with the highest tested equivalent rank of $r = 64$, and being the most highly parameterised setting this was the most likely to benefit from regularisation, we searched over a range of λ values with r set to 64 and the learning rate to the best value found in the initial search.

We searched for the optimal learning rate in the range of $\{3 \times 10^{-6}, 1 \times 10^{-5}, 3 \times 10^{-5}, 1 \times 10^{-4}\}$ for LoRA and SpIEL-AG, $\{4 \times 10^{-4}, 7 \times 10^{-4}, 1 \times 10^{-3}, 2 \times 10^{-3}\}$ for SpIEL-MA (the SM3 optimiser generally benefits from higher learning rates than Adam), and $\{1 \times 10^{-5}, 3 \times 10^{-5}, 1 \times 10^{-4}, 3 \times 10^{-4}\}$ for $(IA)^3$.

As for the number of PEFT parameters, we searched over the range $r = \{8, 16, 32, 64\}$ for LoRA and the equivalent d_ϕ for the SpIEL methods. For Llama2-7b, these values

correspond to 0.30%, 0.59%, 1.2% and 2.3% of the total parameter count respectively, and for Llama2-13b, they correspond to 0.24%, 0.48%, 0.95% and 1.9%.

We searched over λ values in the range $\{0, 1, 3, 10, 30\}$ for SpIEL-AG and $\{0, 0.1, 0.3, 1, 3\}$ for SpIEL-MA (since we use higher learning rates for SpIEL-MA, a lower weight decay strength is required to have the same effect).

Each hyperparameter setting was evaluated by training on the Flan v2 subset and taking the 5-shot performance on the MMLU development set.

Table 7.1 summarises the best hyperparameter values found for each configuration. We present the full results of the hyperparameter search in Appendix E.3.

	Llama2-7b			Llama2-13b		
Method	lr	r	λ	lr	r	λ
$(IA)^3$	3×10^{-4}	-	-	1×10^{-4}	-	-
LoRA	1×10^{-4}	64	-	3×10^{-5}	64	-
SpIEL-AG	1×10^{-4}	64	30	1×10^{-5}	64	30
SpIEL-MA	1×10^{-3}	64	0.3	4×10^{-4}	64	0
Full FT	2×10^{-5}	-	-	2×10^{-5}	-	-

Table 7.1 Optimal settings yielded by hyperparameter search for each PEFT method and model size.

7.5 Results

7.5.1 Main Results

We present the main results of our experiments in Table 7.2. For Llama2-7b, we find that SpIEL-AG outperforms LoRA and $(IA)^3$ consistently across evaluation benchmarks and instruction tuning datasets, including Flan v2, GPT4-Alpaca, and Tülu v2. For Llama2-13b, SpIEL-AG similarly outperforms all baselines when fine-tuned on Flan v2 and GPT4-Alpaca; however, we report more mixed results for Tülu v2[6]. Nonetheless, SpIEL-AG is superior in 5 out of the 6 combinations of scales and instruction tuning datasets. Hence, SpIEL-AG appears to be the strongest method overall.

AG vs MA Comparing the two SpIEL growth criteria, there appears to be a trade-off between performance and memory usage, with the more memory-efficient

[6]It is possible that the hyperparameters chosen for SpIEL during the hyperparameter search on Flan v2 do not always transfer well to Tülu v2, which is a much larger dataset with a different distribution of task types.

		Flan v2		GPT4-Alpaca	Tülu v2					Tülu v2 ⊕ GPT4-Alpaca
	Model / Method	**MMLU**	**TyDiQA**	**HumanEval**	**MMLU**	**GSM**	**BBH**	**TyDiQA**	**HumanEval**	**HumanEval**
Llama2-7b	Original	45.8	50.9	13.5	45.8	13.0	40.6	50.9	13.5	-
	FullFT	50.5	55.5	15.2	51.3†	34.5†	45.3†	56.5†	22.6†	-
	$(IA)^3$	46.7	51.6	14.7	47.8	17.0	40.6	52.2	15.5	16.6
	LoRA	49.3	55.3	15.7	51.3	22.0	43.8	58.0	15.5	16.1
	SpIEL-AG	**50.7**	**56.2**	15.6	**51.5**	**23.0**	**44.8**	**59.4**	**17.1**	**18.8**
	SpIEL-MA	48.8	55.8	**16.2**	49.5	16.5	44.7	56.7	15.3	17.0
Llama2-13b	Original	55.3	60.3	17.8	55.3	23.0	47.8	60.3	17.8	-
	FullFT	-	-	-	56.7†	50.0†	51.9†	62.9†	26.7†	-
	$(IA)^3$	55.1	60.1	18.5	55.6	27.5	50.6	62.8	18.1	18.4
	LoRA	**55.8**	61.4	19.8	**56.2**	29.0	**54.6**	**63.9**	19.5	22.4
	SpIEL-AG	**55.8**	**62.5**	**20.0**	55.9	**31.5**	52.8	63.5	**20.3**	**22.7**
	SpIEL-MA	55.5	**62.5**	19.9	55.9	29.0	51.2	62.8	20.2	20.8

Table 7.2 Performance of PEFT methods on a range of evaluation tasks for various instruction tuning datasets. † indicates results obtained using the models of Ivison et al. (2023). Tülu v2 ⊕ GPT4-Alpaca refers to a setting where we compose the respective PEFTs.

		Flan v2		GPT4-Alpaca
	Model / Method	**MMLU**	**TyDiQA**	**HumanEval**
Ll2-7b	Original (4-bit)	44.8	50.2	11.0
	qLoRA	47.7	54.3	13.3
	qSpIEL-AG	**48.8**	**54.9**	**15.3**
	qSpIEL-MA	48.3	52.7	**15.3**
Ll2-13b	Original (4-bit)	54.7	59.6	15.5
	qLoRA	55.0	60.7	18.2
	qSpIEL-AG	**55.5**	61.5	**18.8**
	qSpIEL-MA	55.4	**61.7**	18.4

Table 7.3 Performance of quantised PEFT methods on a range of evaluation tasks for various instruction tuning datasets.

SpIEL-MA generally performing a little worse than SpIEL-AG, except for a few cases, such as HumanEval evaluation for Llama2-7b. Since GPT4-Alpaca (used for HumanEval evaluation) has the same amount of training examples as Flan v2, we rule out that this difference is due to different levels of sample efficiency between SpIEL-AG and SpIEL-MA.

Ceiling We find that SpIEL-AG generally matches the performance of full fine-tuning on most tasks, except GSM and HumanEval when fine-tuned on Tülu v2, where FullFT vastly outperforms all PEFT methods. This effect was also observed by Ivison et al. (2023) in their qLoRA evaluation. We note that the maximum sequence length we use during fine-tuning is 2,048, whereas Ivison et al. (2023) use 8,192 for full fine-tuning and 4,096 for qLoRA. It is possible that the shorter sequence length for PEFT has a significant effect on downstream performance for open-ended generation tasks, or that

PEFT is weaker on these tasks in general, perhaps due to its inability to modify the input and output embedding layers. This warrants further investigation as part of future work.

Quantisation According to the scores in Table 7.3, we find that 4-bit quantisation results in only a modest reduction in performance across PEFT methods, and their relative performance remains similar. In fact, SpIEL-AG is superior again to both LoRA and $(IA)^3$ by an even higher margin compared to Table 7.2. These results demonstrate that SpIEL is a competitive PEFT method even in extremely resource-constrained scenarios.

Compositionality Finally, in Table 7.2 we also study whether *multiple* PEFT modules, trained on different data mixtures, can be composed with the LLM. We find that composing GPT4-Alpaca and Tülu v2 modules by merging both their weights into the base model parameters increases the performance on HumanEval[7] for PEFT, compared to the individual modules. This is the same composition method as for LT-SFT in §4.3:

$$\boldsymbol{\theta}_{\mathrm{T}+\mathrm{G}} = \boldsymbol{\theta} + \mathrm{scatter}(\boldsymbol{\eta}_{\mathrm{T}}, \boldsymbol{\phi}_{\mathrm{T}}, d_{\boldsymbol{\theta}}) + \mathrm{scatter}(\boldsymbol{\eta}_{\mathrm{G}}, \boldsymbol{\phi}_{\mathrm{G}}, d_{\boldsymbol{\theta}}), \tag{7.9}$$

where $\boldsymbol{\eta}_T$ and $\boldsymbol{\phi}_T$, and $\boldsymbol{\eta}_G$ and $\boldsymbol{\phi}_G$, refer to the SFT indices and updates on Tülu v2 and GPT4-Alpaca respectively. We can perform a similar composition for LoRA by adding the combined update $\frac{\alpha}{r}B_T A_T + \frac{\alpha}{r}B_G A_G$ to each adapted parameter matrix. SpIEL-AG performs favourably in composition compared to LoRA as it yields a larger boost for 7b (+1.7 vs +0.4) and an overall better performance on 13b.

7.5.2 Training Speed and Memory Efficiency

In Table 7.4, we compare the memory requirements and training time of LoRA, SpIEL and their quantised variants. We consider two settings, with and without activation checkpointing. We define the memory requirements to be the minimum amount of GPU memory needed to complete training successfully. We provide more details on our measurements in Section E.5.

We find that LoRA, SpIEL-AG and SpIEL-MA are broadly similar in terms of speed and memory requirements. SpIEL-MA is consistently somewhat faster and more memory

[7]We consider the composition of these two modules on HumanEval and not other combinations because GPT4-Alpaca and Tülu v2 each contain a large number of non-overlapping training examples targeted at coding, so there is reason to believe that composing them could improve performance on a coding task; there are no other such synergies between the datasets in our evaluation. For instance, Tülu v2 already contains all 50K examples from the Flan v2 dataset.

Method	LlaMA 2 7b		LlaMA 2 13b	
	Mem. ↓	Time ↓	Mem. ↓	Time ↓
LoRA	40 / 18	**30.5** / 42.5	66 / **31**	**45.9** / **64.0**
SpIEL-AG	34 / 20	33.4 / 44.8	56 / 36	56.2 / 76.3
SpIEL-MA	**30** / **17**	30.6 / **41.7**	**51** / **31**	50.6 / 70.9
qLoRA	30 / **8**	**38.5** / **55.2**	46 / **12**	**63.6** / **95.3**
qSpIEL-AG	26 / 13	42.8 / 58.4	40 / 19	73.4 / 101.1
qSpIEL-MA	**22** / 10	39.6 / 55.5	**35** / 14	70.1 / 97.3

Table 7.4 GPU memory requirements (in GB) and average time per training step (in seconds) for fine-tuning SFT and LoRA on Flan v2 on an A100 GPU. We report values either without (left) or with (right) activation checkpointing.

efficient than SpIEL-AG. Without activation checkpointing, we find that the SpIEL methods are more memory efficient than LoRA; however, activation checkpointing is especially effective in reducing LoRA's memory consumption, likely because LoRA stores more intermediate activations due to the parallel computation of the LoRA update. When both quantisation and activation checkpointing are applied, LoRA is the most memory-efficient technique by a small margin.[8] We also find that LoRA is generally slightly faster than the SpIEL methods, despite being generally outperformed by SpIEL-AG, especially with quantisation.

As expected, quantisation and activation checkpointing both trade a slowdown in training speed for a reduction in memory usage. However, as a more general finding, these results would suggest that activation checkpointing should be prioritised ahead of quantisation in most cases when performing PEFT on LLMs of this size ($\geq$ 7B), as a larger memory saving is achieved for an only marginally larger slowdown, and without incurring any cost in performance.

We additionally plot the efficiency of LoRA and SpIEL against their performance for MMLU, TyDiQA and HumanEval in Figure 7.3. The plot suggests that the main distinction between SpIEL-AG and LoRA is that SpIEL-AG trains slightly slower while delivering slightly better performance. However, we note that this efficiency penalty only needs to be paid during training and not at inference time.

[8]We note that, unlike our qSpIEL implementation, qLoRA employs a paged optimiser (Dettmers et al., 2023), which might explain why there is a larger memory reduction for qLoRA. While qSpIEL is also compatible with the use of paged optimisers, we leave the actual implementation for future work.

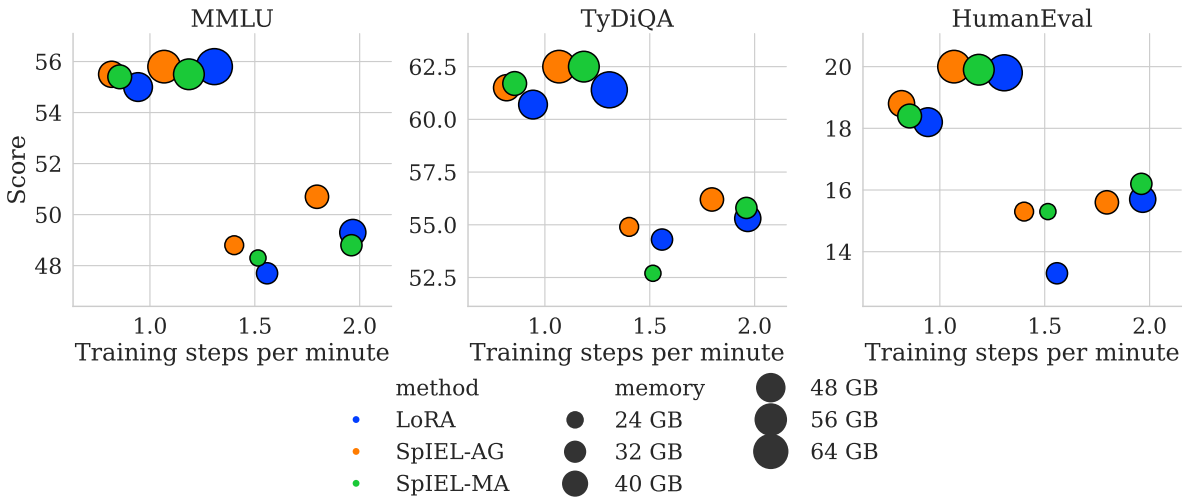


Fig. 7.3 Performance of LoRA and SpIEL on the MMLU, TyDiQA and HumanEval tasks plotted against training speed and memory usage. For each method, we plot both the quantised and unquantised variants for Llama2-7b and Llama2-13b without activation checkpointing. For MMLU and TyDiQA, pre-training is on the Flan v2 subset, and for HumanEval it is on GPT4-Alpaca.

7.5.3 Parameter Ages

To shed light on the training dynamics of SpIEL, we study the age of each index (i.e., the iteration when it was last grown) of the converged parameters. We plot the proportion of indices grown at a certain iteration in Figure 7.4, based on models trained on Flan v2. In general, SpIEL-MA introduces fewer new parameter indices later in training compared to SpIEL-AG. We also find that 13b models tend to retain parameter indices found earlier in training compared to their 7b counterparts. This preference for earlier index sets is not necessarily desirable as it might reflect the fact that SpIEL-MA and 13b models tend to get stuck in early local minima. We speculate that better schedules for dropping and growth might alleviate this issue in the future. In Appendix E.2, we present some experiments which suggest that dropping/growing parameters more frequently might be beneficial.

7.6 Conclusions and Future Work

We have proposed SpIEL, the first method (to our knowledge) for Sparse Fine-Tuning (SFT) that is not only parameter-efficient but also memory-efficient. This allows it to scale to Large Language Models. Taking inspiration from iterative methods for sparse pre-training, we alternate among phases where we update, drop, and then grow parameters. In

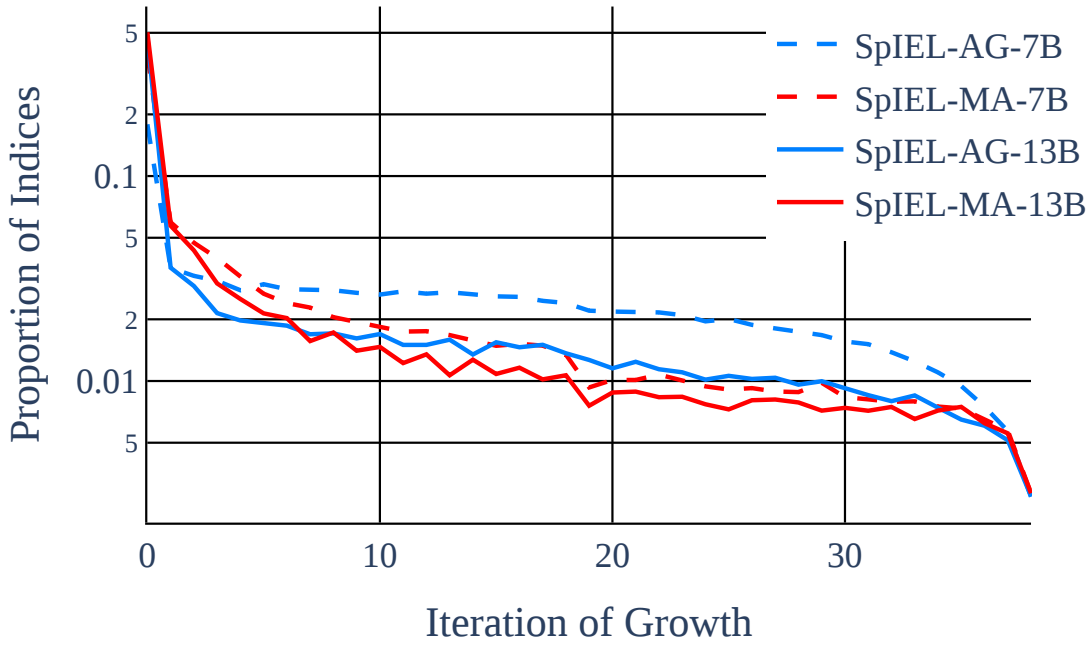


Fig. 7.4 Proportion of indices with a certain age (i.e., the iteration when they were last grown) of the converged $\boldsymbol{\eta}$ after training on the Flan v2 dataset.

particular, we introduce two new criteria for parameter growth: Accumulated Gradients across multiple iterations (SpIEL-AG) and Momentum Approximation (SpIEL-MA), where we reuse information tracked by optimisers like SM3. We find that SpIEL often surpasses alternative PEFT methods such as $(IA)^3$ and LoRA in terms of performance and is comparable to them in terms of memory and time efficiency.

We hope that these promising results will encourage further research into sparse fine-tuning. Future work may include extending SpIEL to the embedding layers of LLMs; improving the efficiency of the backward pass for a linear layer (see Section E.4); considering more advanced dropping/growth criteria, especially those which would enable adaptive redistribution of tunable parameters across parameters tensors. This would stand in contrast to the current setup where the number of tunable parameters in each tensor is fixed.

One drawback of the current method of parameter selection is that only the mean gradient of the parameters over a fixed number of steps is used as the selection criterion. We do not, for instance, consider the variance of each parameter's gradient across batches. Doing so might enable us to improve the tradeoff between performance and efficiency by adaptively set the length of the gradient estimation phase by extending the selection phase until there are less than a certain number of parameters whose gradient magnitudes

are beneath the "cutoff" (i.e. the threshold between selection and rejection for the next growth step), but whose confidence intervals nevertheless enclose this cutoff. This would ensure that we spend the minimum necessary number of steps to reach a satisfactory level of confidence that we are in fact selecting roughly the $k(t)$ parameters with the highest expected gradient magnitudes.

This work on sparse fine-tuning has been influenced by techniques in sparse *pre-training* of neural networks (Evci et al., 2020); however, there is potential to apply our method back to the pre-training domain, specifically, in the area of post-hoc sparsification of densely pre-trained networks. For instance, Wanda (Sun et al., 2024) and SparseGPT (Frantar and Alistarh, 2023) have been proposed as highly efficient techniques for inducing sparsity in pre-trained Transformers by "pruning" a subset of parameters (i.e. setting their values to zero), in theory making them more efficient to serve. However, both methods start to incur significant performance penalties beyond around 50% sparsity. SpIEL offers a potential technique for efficiently repairing damage to a network after pruning: in the simplest case, the SpIEL algorithm can be applied without modification to briefly continue pre-training a pruned network, allowing a few pruned parameters to be regrown and the retained parameters to be modified in such a way as to best counteract the negative effects of the pruning without significantly increasing the network density. Applying LoRA, on the other hand, would cause the network to become fully dense again once the updates were merged into the base model.

The code for SpIEL is available at https://github.com/AlanAnsell/peft and for the instruction-tuning experiments at https://github.com/ducdauge/sft-llm.

Chapter 8

Conclusions

In §8.1, I summarise the motivation for and contributions of this work. I draw overarching lessons and conclusions in §8.2, and then propose some directions for future work in §8.3.

8.1 Summary

Despite larger, more thoroughly trained models becoming increasingly capable of transfer to novel tasks with fewer training examples, fine-tuning remains an indispensable step in the contemporary NLP pipeline. Furthermore, this growth in model size is making fine-tuning increasingly expensive and less accessible to those with fewer computational resources. This has stimulated interest in *parameter-efficient* fine-tuning (PEFT) techniques, which save time and GPU memory during training by fine-tuning only a small subset of the pre-trained model's weights. Fine-tunings learned through PEFT often exhibit properties of *modularity*, allowing them to be re-used for purposes such as multi-task learning (Pfeiffer et al., 2021a) or *composed* with each other to combine several skills or areas of knowledge.

PEFT techniques are well-suited for cross-lingual transfer, where we aim to exploit data from one or more (typically high-resource) *source* language(s) to improve performance of an NLP system for a set of (typically lower-resource) *target* language(s). The composability property of PEFT allows us to adapt in a zero-shot fashion to arbitrary (task, target language) pairs by combining independently trained task- or language-specific fine-tunings. By varying the degree of parameter-efficiency, we can control the capacity of a PEFT and thereby guard against overfitting, which may be a particularly serious danger in the low-resource settings often encountered in cross-lingual transfer. We have considered three avenues of improvement for cross-lingual transfer with PEFT: improvement in cross-lingual transfer performance by exploiting additional data sources; improvement in

performance through more powerful and composable PEFT methods, specifically those based on sparse fine-tuning; and improvement in efficiency of cross-lingual transfer and sparse fine-tuning.

8.1.1 Data

In Chapter 3, we consider incorporating typological data into the cross-lingual transfer pipeline by generating language-specific bottleneck adapters from vectors of typological features. We find that the resulting "MAD-G" adapter, trained simultaneously on 95 languages, performs competitively with per-language MAD-X adapters (Pfeiffer et al., 2020b) despite being around 50 times faster to train per target language. Since typological feature vectors are available for many languages which lack significant amounts of even unlabelled data, MAD-G makes it possible to generate adapters for languages for which they cannot be learned in MAD-X style. We obtain mixed results with zero-shot shot adapter generation: in the case where the unseen target language belongs to a language family introduced to the model during the MAD-G adapter training, we see significant benefits from using the generated adapter. If, on the other hand, the target language belongs to a family already seen during the base model's pre-training, or that is seen neither during pre-training nor MAD-G training, using the generated adapter does not generally result in a performance improvement.

In Chapters 3 and 4, we consider *multi-source* transfer, where we employ task training data from multiple source languages. Specifically, we train on batches of examples from one source language at a time, applying the relevant source language-specific module during each step. We find that is a highly effective strategy, and that adding more source languages improves transfer even when the total number of training examples is held constant. While there is an especially large jump in performance going from one to two source languages, there are incremental benefits even up to as many as 20 diverse source languages.

Chapter 6 investigates how best to exploit other resources often available in practical cross-lingual transfer scenarios, namely few-shot task-specific target language data and translation models or parallel data for the relevant language pairs. We propose a recipe based on multi-source transfer, where the source language task-specific data is translated into all target languages of interest, and a task-specific module is jointly trained on the source data, translated target language data and any gold-standard few-shot target data whilst language adaptation is applied as appropriate. Since the few-shot target language examples are generally of higher quality but much lesser quantity than the translated target language examples, we propose upsampling the few-shot examples to better exploit

their valuable contribution. We show that each individual data source and component of the recipe improves the downstream performance on the tasks in our evaluation. As few as 100 few-shot examples can be sufficient to significantly raise the performance in a given target language even when training on 2-3 orders of magnitude more translated target language examples.

8.1.2 Better PEFT for Cross-Lingual Transfer

Chapter 4 is primarily concerned with developing a PEFT technique which is optimised for cross-lingual transfer performance. Pfeiffer et al. (2020b) made an important advance in cross-lingual transfer by demonstrating that adapters trained for a specific task and adapters specialised for a specific language through continued pre-training could be composed to adapt a base model to an arbitrary (task, language) pair. However, we identify bottleneck adapters' lack of expressivity as a limitation, in particular their inability to modify a Transformer's attention mechanism or embedding matrix (which might be especially disadvantageous when attempting to learn the vocabulary of a new language), and propose sparse fine-tuning (SFT) as an alternative, more expressive family of PEFT methods. We introduce Lottery-Ticket sparse fine-tuning (LT-SFT), a novel method of learning SFTs which selects a sparse subset of parameters to fine-tune based on those which change the most during an initial round of full fine-tuning. We show that task- and language-specific SFTs learned through LT-SFT can be composed by summing their sparse vectors of differences from the pre-trained parameter values. This technique outperforms MAD-X over a wide range of cross-lingual transfer tasks and languages in our evaluation, and exhibits more consistent behaviour with respect to the density of the fine-tuning (or the equivalent adapter bottleneck size for MAD-X). We show that sparsity is important for performance as well as efficiency, with the best cross-lingual transfer results being obtained when the density of the language and task SFTs is no more than 20-30%. We suggest these performance benefits arise from a combination of two factors: sparsity helping to prevent overfitting, and reducing interference between the two composed fine-tunings.

8.1.3 Efficiency

In Chapter 5, we turn our attention to efficiency in cross-lingual transfer. We observe that, while MMTs typically cover in the order of 100 languages, most end users of NLP systems wish to interact with them in a single language. This suggests that compute and model capacity is often wasted supporting many unneeded languages. However,

using a monolingual target language model would preclude the use of cross-lingual transfer. We propose BiStillation, where we employ knowledge distillation to create a smaller, *bilingual* variant of a base MMT covering only the target language and intended source language. This enables us to greatly reduce the vocabulary size by discarding tokens not used in either language as well as shrinking the Transformer layers through knowledge distillation. We find that the resulting bilingual distilled models lose very little cross-lingual transfer ability despite having three to four times fewer parameters than the base model in most cases, and that they outperform comparable *multilingual* distilled models, even when those models have received much more training per language covered. We maintain modularity in both the teacher and student models by learning sparse fine-tunings for each task (and language for the teacher model).

Chapter 7 addresses some inefficiencies of sparse fine-tuning. Lottery-Ticket Sparse Fine-Tuning (LT-SFT) and other previous sparse fine-tuning methods are parameter-efficient, but not *memory*-efficient, as they require memory (in addition to that required to store the base model parameters) proportional to $d_{\boldsymbol{\theta}}$, the number of base model parameters, rather than $d_{\boldsymbol{\phi}}$, the number of trainable parameters. This excessive memory usage is a result of expensive parameter selection methods (such as an initial full fine-tuning pass for LT-SFT) as well as the typical SFT implementation strategy of calculating the dense gradient for each parameter tensor, and then masking out the gradient of non-trainable parameters during the optimiser step. We first propose an alternative implementation of SFT given a fixed set of trainable parameters, where a dense vector of parameter *deltas* is scattered into each parameter tensor at the trainable indices. This alleviates the memory problem, as gradients and optimiser state must only be stored for the $d_{\boldsymbol{\phi}}$ delta values. We then propose SpIEL (Sparse Iterative Efficient Learning), an efficient SFT algorithm which dispenses with the costly initial full-fine tuning phase of LT-SFT by starting with a randomly selected subset of trainable parameters and periodically updating this subset by dropping some parameters and "growing" new ones. Similarly to LT-SFT, we drop the parameters which have changed least from their pre-trained values, whereas we grow parameters with highest estimated gradient magnitudes. We do this by either sampling "real" gradients calculated during backpropagation, or by approximating gradients from the row- and column-wise gradient summaries accumulated by the efficient SM3 optimiser (Anil et al., 2019). We perform a thorough comparison with LoRA (Hu et al., 2022) on a range of instruction-tuning datasets and benchmarks, both with and without quantised training, and find that SpIEL outperforms LoRA in most settings with comparable training time and memory usage.

8.2 Learnings

Here I attempt to distil the most relevant learnings of my PhD in a form that will be most useful for practitioners and researchers of cross-lingual transfer and modular and parameter-efficient fine-tuning. Please note that some of the comments I make in this Section are general lessons from my experience which are not necessarily directly supported by experimental evidence presented elsewhere in the thesis.

The most important factor in cross-lingual transfer, other than model selection, is making effective use of all available data. Multi-source transfer, for instance, is a highly effective strategy, especially when one of the additional source languages is typologically or genealogically related to the target language, but even when this is not the case. We hypothesise that is because multi-source transfer exerts a regularising effect, forcing the fine-tuning to learn features which are more language-agnostic and therefore transfer better to most target languages[1]. I therefore encourage those designing datasets intended for cross-lingual transfer to prioritise gathering gold-standard training data across a range of diverse source languages, even if this somewhat reduces the total amount of data that can be obtained within budget. As we showed in Chapter 6, *target* language task-specific training data is especially valuable, with just 100 gold-standard examples offering significant incremental performance gains even when the full training set is translated into the target language using MT.

One of the main themes of this thesis has been the use of unlabelled data through language adaptation with continued pre-training. The effectiveness of language adaptation in terms of improvement in downstream task performance is inversely related to the amount of data the MMT has already seen belonging to the language in question during its pre-training. For languages unseen during pre-training, an improvement of 20 or more points is not uncommon; for high-resource languages, language adaptation may have no measurable positive effect at all, at least with our relatively modest language module training runs. For languages unseen during pre-training, the most important factor determining the effectiveness of language adaptation is the corpus size for continued pre-training (see Figure 6.2), though as little as a few hundred kilobytes can sometimes be enough to yield significant results. Relatedness of the target language to those seen

[1]This is related to another common issue in cross-lingual transfer, where the optimal checkpoint for *target* language test performance typically occurs earlier during training than the optimal checkpoint for *source* language test performance. This is because the model not only overfits to the training examples, but also to the training *language*, i.e. it starts to pay attention to language-specific task-related features at the expense of more general task-related features. This can be counteracted by having relatively short training runs and/or evaluating checkpoints on a high-resource language for which validation data is available but which is not used for training and is preferably not closely related the source language(s).

during pre-training also appears to have an effect on positive transfer from language adaptation: low-resource languages with a high-resource relative supported by the MMT often see impressive gains, perhaps because the language adaptation process helps text from the low-resource language to "plug into" subnetworks already present for handling the high-resource relative.

The "dark side" of modular language adaptation is that the model used for training is different to the one used for inference: the model is trained on source language examples (possibly whilst a source language-specific module is applied), and then used for inference on target language examples whilst a target language-specific module is applied. In the case of zero-shot transfer, the latter configuration is completely new and only used at inference time. Applying the target language module may have unintended side-effects beyond adapting to the target language, such as distorting task-specific features or causing the model to no longer pay proper attention to them. This observation strongly supports parameter-efficiency as a prior for language adaptation: the lower the capacity of the language modules, the smaller the mismatch between the training- and inference-time models is likely to be. Few-shot transfer, on the other hand, provides a means of ensuring that the inference-time model configuration is not entirely new, and that any "incompatibilities" between the task and target language models have been trained out. Recent work has considered how to address this training/inference time model mismatch problem in the absence of any target language task data: Parović et al. (2022) train modules specific to certain language *pairs*, such that the same language module can be used at training *and* inference time. This sacrifices a large degree of modularity because the task module then becomes specific to one language pair. In Parovic et al. (2023), we proposed to instead "expose" the task module to the language modules of multiple target languages during training on a single source language to reduce the chance of mismatch between the task and language modules at inference time.

Machine translation is of course one of the most important tools for cross-lingual transfer. Multilingual NMT models such as NLLB-200 (NLLB Team et al., 2022) already cover an impressive range of languages, and we show in Chapter 6 that even a modest amount of parallel data can be sufficient to adapt them to unseen languages well enough to improve downstream cross-lingual transfer performance. However, MT performance is ultimately bottlenecked by the amount of parallel data available in the language pair of interest. Fortunately, MT and unsupervised cross-lingual transfer need not be viewed as opposing, mutually exclusive approaches: as we show in Chapter 6 following Artetxe et al. (2023), an approach as simple as fine-tuning an MMT with a combination of source language and translated target language data can be sufficient to outperform either single

approach significantly. While more sophisticated methods of applying MT to cross-lingual transfer have been considered in prior work (Ponti et al., 2021c; Oh et al., 2022), and there is some work on using unlabelled data to improve adaptation of MT models to unseen languages (Ko et al., 2021), the question of how best to exploit a combination of parallel and unlabelled data for cross-lingual transfer appears under-explored and may be a fertile area for future work.

Composition of sparse fine-tunings appears to be a highly effective technique for cross-lingual transfer. Subsequent work has explored composition in a monolingual context through averaging of fully fine-tuned models and addition of "task vectors" obtained by subtracting the pre-trained parameter vector from the fine tuned one (Matena and Raffel, 2022; Ilharco et al., 2023). However, we show that sparse fine-tunings outperform dense ones in our cross-lingual transfer evaluation. This accords with the discussion above about the drawback of using different model variants for training and inference: the sparsity prior reduces the mismatch between the source and target language-adapted variants, since most parameters are constrained to have the same values in both variants. An alternative but related interpretation of the benefit of sparsity for composition is that it helps reduce interference between the two composed fine-tunings.

The findings of Chapter 5 align with the notion that the "curse of multilinguality" limits the performance of MMTs by diluting the model capacity which can be allocated to each language. Since we can easily produce a language pair-specific distilled variant one third or less[2] the size of a base MMT without a serious loss of performance, it follows that a model the same size as the base MMT could achieve better performance if its capacity were dedicated entirely to the same two languages. Bilingual distillation provides a means of benefitting from the cross-lingual transfer abilities of MMTs without having to pay the price of supporting the many languages we are not targeting.

SpIEL, as proposed in Chapter 7, offers a means of improving the efficiency of sparse fine-tuning to the point where it is suitable for use with large language models. The SpIEL algorithm was challenging both to design and to implement efficiently due to the lack of hardware and software support for efficient sparse tensor operations (Hooker, 2021), and one difficulty remains unresolved, namely realising the potential saving in the backward pass of a sparsely fine-tuned linear layer. However, the results of this first attempt at SFT for LLMs appear promising, outperforming LoRA (Hu et al., 2022), the

[2]In subsequent, unpublished experiments where we varied the hidden dimension as well as the number of layers, we achieved comparable or better results on XQuAD to the ones presented in Chapter 5 with a student model almost *eight* times smaller than the teacher; reducing the hidden size seems to offer better efficiency in general than reducing the number of layers but we unfortunately did not try this in the original work.

current *de facto* default PEFT method, in most of our experiments with comparably training and inference efficiency. While the comparison with LoRA is somewhat mixed at this point, there is plenty of opportunity to optimise SpIEL in the future with new growth/dropping criteria and other innovations.

8.3 Future Work

Parameter-efficient fine-tuning and cross-lingual transfer address the ever-present problems of data and compute scarcity, and have thus remained relevant despite recent transformative changes across Natural Language Processing as a whole. Their future appears bright. Here I propose a number of directions for future research, some following on from the work in this thesis and some more general.

8.3.1 Knowledge Modules

Noting that it is possible to capture knowledge and understanding of a given language in a dedicated module through continued pre-training on text belonging to that language, it seems natural to consider whether the specific knowledge contained in a document can be captured in a similar manner. Consider the example of a chatbot designed to answer questions about a certain product or service. Suppose the information the chatbot must refer to to perform its job is contained in a large text document. The simplest approach is to pass the document as input to the model along with each query; this is the traditional question answering setup as exemplified by the SQuAD dataset (Rajpurkar et al., 2016). However, this may be very computationally expensive for long documents, as the computational complexity of processing a sequence of length L with a Transformer is $\mathcal{O}(L^2)$. This cost can be mitigated by using a Transformer variant such as linear attention Transformers (Katharopoulos et al., 2020) with $\mathcal{O}(L)$ complexity, but it is likely to remain impractically high. However, the need to process the document in the model input for each query could be removed entirely by instead capturing its knowledge within the model parameters themselves in the form of a "knowledge module"[3] such as a LoRA or SFT.

The challenge is that continued pre-training on the document is unlikely to be as effective for learning knowledge modules as it is for learning language modules. Preliminary experiments suggest that continued pre-training results in rote-learning

[3]Retrieval-augmented generation (RAG; Lewis et al., 2020c) is an alternative approach which works by only including sections of the document relevant to the query in the input. However, it sometimes fails to retrieve sections containing essential information, amongst other drawbacks (Barnett et al., 2024).

sequences of text from the document rather than learning general-purpose representations of knowledge which can be re-used in new contexts. Therefore a novel training strategy is required which absorbs the underlying information in a document rather than its surface form.

8.3.2 Modular Distilled MMTs

In Chapter 5, we show that it is possible to create effective models for a specific cross-lingual transfer pair by distilling a smaller, bilingual model from a larger, multilingual one. However, this comes at a cost: firstly, it sacrifices modularity, as we require N_L bilingual models and $N_L N_T$ task modules to support N_T tasks in N_L languages. Secondly, it abandons the possibility of applying multi-source transfer, where we train on task examples from multiple source languages, which has proven to be a highly effective way of improving target language performance. An alternative would be to instead distil a family of N_L monolingual models from the base MMT which share some parameters with each other, but also have some private, language-specific parameters, similar to Pfeiffer et al. (2023a) who pre-train such a family of models from scratch. By training only the shared parameters on a given task using the source language variant(s) of the model, we could create a fine-tuning that should transfer well to any target language variant of the model, addressing both weaknesses of distilling independent bilingual models.

8.3.3 Combining Supervised and Unsupervised Cross-Lingual Transfer

In Chapter 6, we show that we can improve cross-lingual transfer performance by combining supervised and unsupervised techniques. Specifically, we train an MMT (i.e. unsupervised multilingual model) on a combination of source and machine-translated target language examples. However, this is a rather shallow way of combining these techniques. Ponti et al. (2022) propose a means of improving translation-based cross-lingual transfer by treating the translation as a latent random variable, allowing the translation and classification components to be trained end-to-end; Ko et al. (2021) explore the use of unlabelled data to improve translation for low-resource languages. However, my impression is that the potential for hybrid unsupervised and supervised techniques specifically for cross-lingual transfer has not yet been fully realised.

8.3.4 Representation Fine-Tuning

An interesting line of recent work in parameter-efficient fine-tuning has been "representation fine-tuning" (ReFT), or fine-tuning methods which intervene on a model's hidden representations rather than its parameters. A notable example is LoReFT (Wu et al., 2024), where a distributed interchange intervention (DII; Geiger et al., 2023) is applied to the model's hidden representations at a fixed subset of sequence positions. This subset is defined as a short prefix and suffix of the sequence, and sets LoReFT aside from previous fine-tuning methods whose behaviour is generally consistent at each sequence position, and makes training highly efficient. Wu et al. (2024)'s experiments show that it generally performs on par with or better than previous PEFT methods across a range of tasks while updating fewer parameters and training faster. This raises a number of questions for future research: what is the tradeoff between performance and the number of positions at which the intervention is applied? Would other position subsets work better? Would it be possible and beneficial to apply the intervention at a learnable rather than a fixed subset of positions? Is there something special about the DII intervention, or would other interventions would as well or better? Can ReFTs be composed effectively, and could they therefore be applied successfully to cross-lingual transfer with language adaptation?

8.3.5 Improving SFT for Large Language Models

In Chapter 7, we proposed SpIEL, a sparse fine-tuning algorithm with the necessary time- and memory-efficiency properties to make it suitable for use with LLMs. One remaining technical challenge is an efficient implementation of the backward pass of SpIEL for linear layers, as discussed in Appendix E.4. Another limitation is that so far we have not considered applying SpIEL to embedding layers, which may be quite important in language adaptation when learning the vocabulary of a new language, for instance. The difficulty here is how to apply updates to the subset of embeddings used in a particular batch of inputs efficiently, that is, in time proportional to the number of input tokens rather than the vocabulary size. It is also difficult to estimate the gradients of the embedding matrix because generally only a small subset of tokens will occur in any single batch. Therefore a specialised approach may be required for applying SpIEL to embedding layers. There is also a wide range of possible dropping/growth criteria and update schedules which could be explored to improve SpIEL's performance. So far we have considered only approaches which maintain a fixed number of trainable parameters for each tensor. However, there is no reason why it would not be possible to dynamically update the number of trainable parameters in a tensor and thereby

redistribute parameters to those tensors most critical for the fine-tuning task. This requires some way of comparing the importance of parameters across tensors. Gradient magnitude may not be ideal for this since certain tensors tend to have systematically lower or higher gradients on average despite this not necessarily reflecting their overall "importance."

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.

David Ifeoluwa Adelani, Jade Abbott, Graham Neubig, Daniel D'souza, Julia Kreutzer, Constantine Lignos, Chester Palen-Michel, Happy Buzaaba, Shruti Rijhwani, Sebastian Ruder, Stephen Mayhew, Israel Abebe Azime, Shamsuddeen H. Muhammad, Chris Chinenye Emezue, Joyce Nakatumba-Nabende, Perez Ogayo, Aremu Anuoluwapo, Catherine Gitau, Derguene Mbaye, Jesujoba Alabi, Seid Muhie Yimam, Tajuddeen Rabiu Gwadabe, Ignatius Ezeani, Rubungo Andre Niyongabo, Jonathan Mukiibi, Verrah Otiende, Iroro Orife, Davis David, Samba Ngom, Tosin Adewumi, Paul Rayson, Mofetoluwa Adeyemi, Gerald Muriuki, Emmanuel Anebi, Chiamaka Chukwuneke, Nkiruka Odu, Eric Peter Wairagala, Samuel Oyerinde, Clemencia Siro, Tobius Saul Bateesa, Temilola Oloyede, Yvonne Wambui, Victor Akinode, Deborah Nabagereka, Maurice Katusiime, Ayodele Awokoya, Mouhamadane MBOUP, Dibora Gebreyohannes, Henok Tilaye, Kelechi Nwaike, Degaga Wolde, Abdoulaye Faye, Blessing Sibanda, Orevaoghene Ahia, Bonaventure F. P. Dossou, Kelechi Ogueji, Thierno Ibrahima DIOP, Abdoulaye Diallo, Adewale Akinfaderin, Tendai Marengereke, and Salomey Osei. 2021. MasakhaNER: Named entity recognition for African languages. *Transactions of the Association for Computational Linguistics*, 9:1116–1131.

Željko Agić and Ivan Vulić. 2019. JW300: A wide-coverage parallel corpus for low-resource languages. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3204–3210, Florence, Italy. 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.

Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Merouane Debbah, Etienne Goffinet, Daniel Heslow, Julien Launay, Quentin Malartic, Badreddine Noune, Baptiste Pannier, and Guilherme Penedo. 2023. Falcon-40B: An open large language model with state-of-the-art performance. Technical report, Technology Innovation Institute.

Waleed Ammar, George Mulcaire, Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2016. Many languages, one parser. *Transactions of the Association for Computational Linguistics*, 4:431–444.

Rohan Anil, Vineet Gupta, Tomer Koren, and Yoram Singer. 2019. Memory-efficient adaptive optimization. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA. Curran Associates Inc.

Alan Ansell, Marinela Parović, Ivan Vulić, Anna Korhonen, and Edoardo Ponti. 2023a. Unifying cross-lingual transfer across scenarios of resource scarcity. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3980–3995, Singapore. Association for Computational Linguistics.

Alan Ansell, Edoardo Ponti, Anna Korhonen, and Ivan Vulić. 2022. Composable sparse fine-tuning for cross-lingual transfer. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1778–1796, Dublin, Ireland. Association for Computational Linguistics.

Alan Ansell, Edoardo Maria Ponti, Anna Korhonen, and Ivan Vulić. 2023b. Distilling efficient language-specific models for cross-lingual transfer. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 8147–8165, Toronto, Canada. Association for Computational Linguistics.

Alan Ansell, Edoardo Maria Ponti, Jonas Pfeiffer, Sebastian Ruder, Goran Glavaš, Ivan Vulić, and Anna Korhonen. 2021. MAD-G: Multilingual adapter generation for efficient cross-lingual transfer. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4762–4781, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Alan Ansell, Ivan Vulić, Hannah Sterz, Anna Korhonen, and Edoardo M. Ponti. 2024. Scaling sparse fine-tuning to large language models.

Naveen Arivazhagan, Ankur Bapna, Orhan Firat, Dmitry Lepikhin, Melvin Johnson, Maxim Krikun, Mia Xu Chen, Yuan Cao, George F. Foster, Colin Cherry, Wolfgang Macherey, Zhifeng Chen, and Yonghui Wu. 2019. Massively multilingual neural machine translation in the wild: Findings and challenges. *arXiv preprint*.

Jordi Armengol-Estapé, Ona de Gibert Bonet, and Maite Melero. 2022. On the multilingual capabilities of very large-scale English language models. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 3056–3068, Marseille, France. European Language Resources Association.

Mikel Artetxe, Vedanuj Goswami, Shruti Bhosale, Angela Fan, and Luke Zettlemoyer. 2023. Revisiting machine translation for cross-lingual classification. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 6489–6499, Singapore. Association for Computational Linguistics.

Mikel Artetxe, Gorka Labaka, and Eneko Agirre. 2016. Learning principled bilingual mappings of word embeddings while preserving monolingual invariance. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2289–2294, Austin, Texas. Association for Computational Linguistics.

Mikel Artetxe, Gorka Labaka, and Eneko Agirre. 2017. Learning bilingual word embeddings with (almost) no bilingual data. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 451–462, Vancouver, Canada. Association for Computational Linguistics.

Mikel Artetxe, Gorka Labaka, and Eneko Agirre. 2018. Unsupervised statistical machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3632–3642, Brussels, Belgium. Association for Computational Linguistics.

Mikel Artetxe, Gorka Labaka, and Eneko Agirre. 2020a. Translation artifacts in cross-lingual transfer learning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7674–7684, Online. Association for Computational Linguistics.

Mikel Artetxe, Sebastian Ruder, and Dani Yogatama. 2020b. 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.

Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Ben Mann, Nova DasSarma, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Jackson Kernion, Kamal Ndousse, Catherine Olsson, Dario Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, and Jared Kaplan. 2021. A general language assistant as a laboratory for alignment.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization.

Lucas Bandarkar, Benjamin Muller, Pritish Yuvraj, Rui Hou, Nayan Singhal, Hongjiang Lv, and Bing Liu. 2025. Layer swapping for zero-shot cross-lingual transfer in large language models. In *The Thirteenth International Conference on Learning Representations*.

Ankur Bapna and Orhan Firat. 2019. Non-parametric adaptation for neural machine translation. 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 1921–1931, Minneapolis, Minnesota. Association for Computational Linguistics.

Scott Barnett, Stefanus Kurniawan, Srikanth Thudumu, Zach Brannelly, and Mohamed Abdelrazek. 2024. Seven failure points when engineering a retrieval augmented generation system.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–9, Dublin, Ireland. Association for Computational Linguistics.

Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. 2000. A neural probabilistic language model. In *Advances in Neural Information Processing Systems*, volume 13. MIT Press.

Aishwarya Bhandare, Vamsi Sripathi, Deepthi Karkada, Vivek Menon, Sun Choi, Kushal Datta, and Vikram Saletore. 2019. Efficient 8-bit quantization of transformer neural machine language translation model.

Dan Biderman, Jacob Portes, Jose Javier Gonzalez Ortiz, Mansheej Paul, Philip Greengard, Connor Jennings, Daniel King, Sam Havens, Vitaliy Chiley, Jonathan Frankle, Cody Blakeney, and John P. Cunningham. 2024. Lora learns less and forgets less.

BigScience Workshop, :, Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, Jonathan Tow, Alexander M. Rush, Stella Biderman, Albert Webson, Pawan Sasanka Ammanamanchi, Thomas Wang, Benoît Sagot, Niklas Muennighoff, Albert Villanova del Moral, Olatunji Ruwase, Rachel Bawden, Stas Bekman, Angelina McMillan-Major, Iz Beltagy, Huu Nguyen, Lucile Saulnier, Samson Tan, Pedro Ortiz Suarez, Victor Sanh, Hugo Laurençon, Yacine Jernite, Julien Launay, Margaret Mitchell, Colin Raffel, Aaron Gokaslan, Adi Simhi, Aitor Soroa, Alham Fikri Aji, Amit Alfassy, Anna Rogers, Ariel Kreisberg Nitzav, Canwen Xu, Chenghao Mou, Chris Emezue, Christopher Klamm, Colin Leong, Daniel van Strien, David Ifeoluwa Adelani, Dragomir Radev, Eduardo González Ponferrada, Efrat Levkovizh, Ethan Kim, Eyal Bar Natan, Francesco De Toni, Gérard Dupont, Germán Kruszewski, Giada Pistilli, Hady Elsahar, Hamza Benyamina, Hieu Tran, Ian Yu, Idris Abdulmumin, Isaac Johnson, Itziar Gonzalez-Dios, Javier de la Rosa, Jenny Chim, Jesse Dodge, Jian Zhu, Jonathan Chang, Jörg Frohberg, Joseph Tobing, Joydeep Bhattacharjee, Khalid Almubarak, Kimbo Chen, Kyle Lo, Leandro Von Werra, Leon Weber, Long Phan, Loubna Ben allal, Ludovic Tanguy, Manan Dey, Manuel Romero Muñoz, Maraim Masoud, María Grandury, Mario Šaško, Max Huang, Maximin Coavoux, Mayank Singh, Mike Tian-Jian Jiang, Minh Chien Vu, Mohammad A. Jauhar, Mustafa Ghaleb, Nishant Subramani, Nora Kassner, Nurulaqilla Khamis, Olivier Nguyen, Omar Espejel, Ona de Gibert, Paulo Villegas, Peter Henderson, Pierre Colombo, Priscilla Amuok, Quentin Lhoest, Rheza Harliman, Rishi Bommasani, Roberto Luis López, Rui Ribeiro, Salomey Osei, Sampo Pyysalo, Sebastian Nagel, Shamik Bose, Shamsuddeen Hassan Muhammad, Shanya Sharma, Shayne Longpre, Somaieh Nikpoor, Stanislav Silberberg, Suhas Pai, Sydney Zink, Tiago Timponi Torrent, Timo Schick, Tristan Thrush, Valentin Danchev, Vassilina Nikoulina, Veronika Laippala, Violette Lepercq, Vrinda Prabhu, Zaid Alyafeai, Zeerak Talat, Arun Raja, Benjamin Heinzerling, Chenglei Si, Davut Emre Taşar, Elizabeth Salesky, Sabrina J. Mielke, Wilson Y. Lee, Abheesht Sharma, Andrea Santilli, Antoine Chaffin, Arnaud Stiegler, Debajyoti Datta, Eliza Szczechla, Gunjan Chhablani, Han Wang, Harshit Pandey, Hendrik Strobelt, Jason Alan Fries, Jos Rozen, Leo Gao, Lintang Sutawika, M Saiful Bari, Maged S. Al-shaibani, Matteo Manica, Nihal Nayak, Ryan Teehan, Samuel Albanie, Sheng Shen, Srulik Ben-David, Stephen H. Bach, Taewoon Kim, Tali Bers, Thibault Fevry, Trishala Neeraj, Urmish Thakker, Vikas Raunak, Xiangru Tang, Zheng-Xin Yong, Zhiqing Sun, Shaked Brody, Yallow Uri, Hadar Tojarieh, Adam Roberts, Hyung Won Chung, Jaesung Tae, Jason Phang, Ofir Press, Conglong Li, Deepak Narayanan, Hatim Bourfoune, Jared Casper, Jeff Rasley, Max Ryabinin, Mayank Mishra, Minjia Zhang, Mohammad Shoeybi, Myriam Peyrounette, Nicolas Patry, Nouamane Tazi, Omar Sanseviero, Patrick von Platen, Pierre Cornette, Pierre François Lavallée, Rémi Lacroix, Samyam Rajbhandari, Sanchit Gandhi, Shaden Smith, Stéphane Requena, Suraj Patil, Tim Dettmers, Ahmed Baruwa, Amanpreet Singh, Anastasia Cheveleva, Anne-Laure

Ligozat, Arjun Subramonian, Aurélie Névéol, Charles Lovering, Dan Garrette, Deepak Tunuguntla, Ehud Reiter, Ekaterina Taktasheva, Ekaterina Voloshina, Eli Bogdanov, Genta Indra Winata, Hailey Schoelkopf, Jan-Christoph Kalo, Jekaterina Novikova, Jessica Zosa Forde, Jordan Clive, Jungo Kasai, Ken Kawamura, Liam Hazan, Marine Carpuat, Miruna Clinciu, Najoung Kim, Newton Cheng, Oleg Serikov, Omer Antverg, Oskar van der Wal, Rui Zhang, Ruochen Zhang, Sebastian Gehrmann, Shachar Mirkin, Shani Pais, Tatiana Shavrina, Thomas Scialom, Tian Yun, Tomasz Limisiewicz, Verena Rieser, Vitaly Protasov, Vladislav Mikhailov, Yada Pruksachatkun, Yonatan Belinkov, Zachary Bamberger, Zdeněk Kasner, Alice Rueda, Amanda Pestana, Amir Feizpour, Ammar Khan, Amy Faranak, Ana Santos, Anthony Hevia, Antigona Unldreaj, Arash Aghagol, Arezoo Abdollahi, Aycha Tammour, Azadeh HajiHosseini, Bahareh Behroozi, Benjamin Ajibade, Bharat Saxena, Carlos Muñoz Ferrandis, Daniel McDuff, Danish Contractor, David Lansky, Davis David, Douwe Kiela, Duong A. Nguyen, Edward Tan, Emi Baylor, Ezinwanne Ozoani, Fatima Mirza, Frankline Ononiwu, Habib Rezanejad, Hessie Jones, Indrani Bhattacharya, Irene Solaiman, Irina Sedenko, Isar Nejadgholi, Jesse Passmore, Josh Seltzer, Julio Bonis Sanz, Livia Dutra, Mairon Samagaio, Maraim Elbadri, Margot Mieskes, Marissa Gerchick, Martha Akinlolu, Michael McKenna, Mike Qiu, Muhammed Ghauri, Mykola Burynok, Nafis Abrar, Nazneen Rajani, Nour Elkott, Nour Fahmy, Olanrewaju Samuel, Ran An, Rasmus Kromann, Ryan Hao, Samira Alizadeh, Sarmad Shubber, Silas Wang, Sourav Roy, Sylvain Viguier, Thanh Le, Tobi Oyebade, Trieu Le, Yoyo Yang, Zach Nguyen, Abhinav Ramesh Kashyap, Alfredo Palasciano, Alison Callahan, Anima Shukla, Antonio Miranda-Escalada, Ayush Singh, Benjamin Beilharz, Bo Wang, Caio Brito, Chenxi Zhou, Chirag Jain, Chuxin Xu, Clémentine Fourrier, Daniel León Periñán, Daniel Molano, Dian Yu, Enrique Manjavacas, Fabio Barth, Florian Fuhrimann, Gabriel Altay, Giyaseddin Bayrak, Gully Burns, Helena U. Vrabec, Imane Bello, Ishani Dash, Jihyun Kang, John Giorgi, Jonas Golde, Jose David Posada, Karthik Rangasai Sivaraman, Lokesh Bulchandani, Lu Liu, Luisa Shinzato, Madeleine Hahn de Bykhovetz, Maiko Takeuchi, Marc Pàmies, Maria A Castillo, Marianna Nezhurina, Mario Sänger, Matthias Samwald, Michael Cullan, Michael Weinberg, Michiel De Wolf, Mina Mihaljcic, Minna Liu, Moritz Freidank, Myungsun Kang, Natasha Seelam, Nathan Dahlberg, Nicholas Michio Broad, Nikolaus Muellner, Pascale Fung, Patrick Haller, Ramya Chandrasekhar, Renata Eisenberg, Robert Martin, Rodrigo Canalli, Rosaline Su, Ruisi Su, Samuel Cahyawijaya, Samuele Garda, Shlok S Deshmukh, Shubhanshu Mishra, Sid Kiblawi, Simon Ott, Sinee Sang-aroonsiri, Srishti Kumar, Stefan Schweter, Sushil Bharati, Tanmay Laud, Théo Gigant, Tomoya Kainuma, Wojciech Kusa, Yanis Labrak, Yash Shailesh Bajaj, Yash Venkatraman, Yifan Xu, Yingxin Xu, Yu Xu, Zhe Tan, Zhongli Xie, Zifan Ye, Mathilde Bras, Younes Belkada, and Thomas Wolf. 2023. Bloom: A 176b-parameter open-access multilingual language model.

Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. 2021. Understanding and overcoming the challenges of efficient transformer quantization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7947–7969, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

David Brambila. 1976. *Diccionario Raramuri-Castellano: Tarahumar*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. 2006. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, page 535–541, New York, NY, USA. Association for Computing Machinery.

Gina Bustamante, Arturo Oncevay, and Roberto Zariquiey. 2020. No data to crawl? monolingual corpus creation from PDF files of truly low-resource languages in Peru. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 2914–2923, Marseille, France. European Language Resources Association.

Samuel Cahyawijaya, Holy Lovenia, Alham Fikri Aji, Genta Indra Winata, Bryan Wilie, Rahmad Mahendra, Christian Wibisono, Ade Romadhony, Karissa Vincentio, Fajri Koto, Jennifer Santoso, David Moeljadi, Cahya Wirawan, Frederikus Hudi, Ivan Halim Parmonangan, Ika Alfina, Muhammad Satrio Wicaksono, Ilham Firdausi Putra, Samsul Rahmadani, Yulianti Oenang, Ali Akbar Septiandri, James Jaya, Kaustubh D. Dhole, Arie Ardiyanti Suryani, Rifki Afina Putri, Dan Su, Keith Stevens, Made Nindyatama Nityasya, Muhammad Farid Adilazuarda, Ryan Ignatius, Ryandito Diandaru, Tiezheng Yu, Vito Ghifari, Wenliang Dai, Yan Xu, Dyah Damapuspita, Cuk Tho, Ichwanul Muslim Karo Karo, Tirana Noor Fatyanosa, Ziwei Ji, Pascale Fung, Graham Neubig, Timothy Baldwin, Sebastian Ruder, Herry Sujaini, Sakriani Sakti, and Ayu Purwarianti. 2022. NusaCrowd: Open Source Initiative for Indonesian NLP Resources.

Nicholas Carlini. 2023. Playing chess with large language models.

Rich Caruana. 1997. Multitask learning. *Machine Learning*, 28:41–75.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar. Association for Computational Linguistics.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter

Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training deep nets with sublinear memory cost. *ArXiv*, abs/1604.06174.

Luis Chiruzzo, Pedro Amarilla, Adolfo Ríos, and Gustavo Giménez Lugo. 2020. Development of a Guarani - Spanish parallel corpus. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 2629–2633, Marseille, France. European Language Resources Association.

Noam Chomsky. 1965. *Aspects of the Theory of Syntax*. MIT press.

Alexandra Chronopoulou, Matthew Peters, Alexander Fraser, and Jesse Dodge. 2023. AdapterSoup: Weight averaging to improve generalization of pretrained language models. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 2054–2063, Dubrovnik, Croatia. Association for Computational Linguistics.

Jonathan H. Clark, Eunsol Choi, Michael Collins, Dan Garrette, Tom Kwiatkowski, Vitaly Nikolaev, and Jennimaria Palomaki. 2020a. TyDi QA: A benchmark for information-seeking question answering in typologically diverse languages. *Transactions of the Association for Computational Linguistics*, 8:454–470.

Kevin Clark, Thang Luong, Quoc V. Le, and Christopher Manning. 2020b. Electra: Pre-training text encoders as discriminators rather than generators. In *ICLR*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Cohere for AI. 2024. Command-r+.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167.

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, 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.

William Croft. 2002. *Typology and Universals*, 2 edition. Cambridge Textbooks in Linguistics. Cambridge University Press.

James Curran and Stephen Clark. 2003. Language independent NER using a maximum entropy tagger. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 164–167.

Rubén Cushimariano Romano and Richer C. Sebastián Q. 2008. Ñaantsipeta asháninkaki birakochaki. diccionario asháninka-castellano. versión preliminar. http://www.lengamer.org/publicaciones/diccionarios/.

Andrew M Dai and Quoc V Le. 2015. Semi-supervised sequence learning. *Advances in neural information processing systems*, 28.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. 2017. Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 933–941. PMLR.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. GPT3.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized LLMs. In *Thirty-seventh Conference on Neural Information Processing Systems*.

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.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

David M. Eberhard, Gary F. Simons, and Charles D. Fennig. 2024. Ethnologue: Languages of the world. twenty-seventh edition.

Abteen Ebrahimi, Manuel Mager, Arturo Oncevay, Vishrav Chaudhary, Luis Chiruzzo, Angela Fan, John Ortega, Ricardo Ramos, Annette Rios, Ivan Vladimir Meza Ruiz, Gustavo Giménez-Lugo, Elisabeth Mager, Graham Neubig, Alexis Palmer, Rolando Coto-Solano, Thang Vu, and Katharina Kann. 2022. AmericasNLI: Evaluating zero-shot natural language understanding of pretrained multilingual models in truly low-resource languages. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6279–6299, Dublin, Ireland. Association for Computational Linguistics.

Nicholas Evans and Stephen C. Levinson. 2009. The myth of language universals: Language diversity and its importance for cognitive science. *Behavioral and Brain Sciences*, 32(5):429–448.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. 2020. Rigging the lottery: Making all tickets winners. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 2943–2952. PMLR.

Meng Fang and Trevor Cohn. 2017. Model transfer for tagging low-resource languages using a bilingual dictionary. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 587–593, Vancouver, Canada. Association for Computational Linguistics.

Hao Fei, Meishan Zhang, and Donghong Ji. 2020. Cross-lingual semantic role labeling with high-quality translated training corpus. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7014–7026, Online. Association for Computational Linguistics.

Isaac Feldman and Rolando Coto-Solano. 2020. Neural machine translation models with back-translation for the extremely low-resource indigenous language Bribri. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 3965–3976, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Orhan Firat, Kyunghyun Cho, and Yoshua Bengio. 2016. Multi-way, multilingual neural machine translation with a shared attention mechanism. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 866–875, San Diego, California. Association for Computational Linguistics.

J. Firth. 1957. A synopsis of linguistic theory 1930-1955. In *Studies in Linguistic Analysis*. Philological Society, Oxford. Reprinted in Palmer, F. (ed. 1968) Selected Papers of J. R. Firth, Longman, Harlow.

Jonathan Frankle and Michael Carbin. 2019. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*.

Elias Frantar and Dan Alistarh. 2023. Sparsegpt: Massive language models can be accurately pruned in one-shot.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. Gptq: Accurate post-training quantization for generative pre-trained transformers.

Kunihiko Fukushima. 1975. Cognitron: A self-organizing multilayered neural network. *Biol. Cybern.*, 20(3–4):121–136.

Ana-Paula Galarreta, Andrés Melgar, and Arturo Oncevay. 2017. Corpus creation and initial SMT experiments between Spanish and Shipibo-konibo. In *Proceedings of the International Conference Recent Advances in Natural Language Processing, RANLP 2017*, pages 238–244, Varna, Bulgaria. INCOMA Ltd.

Iker García-Ferrero, Rodrigo Agerri, and German Rigau. 2022. Model and data transfer for cross-lingual sequence labelling in zero-resource settings. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 6403–6416, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Iker García-Ferrero, Rodrigo Agerri, and German Rigau. 2023. T-projection: High quality annotation projection for sequence labeling tasks. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 15203–15217, Singapore. Association for Computational Linguistics.

Atticus Geiger, Zhengxuan Wu, Christopher Potts, Thomas Icard, and Noah D. Goodman. 2023. Finding alignments between interpretable causal variables and distributed neural representations.

Goran Glavaš and Ivan Vulić. 2021. Is supervised syntactic parsing beneficial for language understanding tasks? an empirical investigation. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 3090–3104, Online. Association for Computational Linguistics.

Demi Guo, Alexander Rush, and Yoon Kim. 2021. Parameter-efficient transfer learning with diff pruning. 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 4884–4896, Online. Association for Computational Linguistics.

Ximena Gutierrez-Vasques, Gerardo Sierra, and Isaac Hernandez Pompa. 2016. Axolotl: a web accessible parallel corpus for Spanish-Nahuatl. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 4210–4214, Portorož, Slovenia. European Language Resources Association (ELRA).

David Ha, Andrew M. Dai, and Quoc V. Le. 2017. Hypernetworks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Harald Hammarström, Robert Forkel, Martin Haspelmath, and Sebastian Bank. 2024. Glottolog 5.0.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2021. Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2023. DeBERTav3: Improving deBERTa using ELECTRA-style pre-training with gradient-disentangled embedding sharing. In *The Eleventh International Conference on Learning Representations*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *International Conference on Learning Representations*.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus).

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network.

Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. 2001. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. 2021. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *The Journal of Machine Learning Research*, 22(1):10882–11005.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. 2022. Training compute-optimal large language models.

Carolin Holtermann, Paul Röttger, Timm Dill, and Anne Lauscher. 2024. Evaluating the elementary multilingual capabilities of large language models with multiq.

Sara Hooker. 2021. The hardware lottery. *Communications of the ACM*, 64(12):58–65.

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 *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 328–339, Melbourne, Australia. Association for Computational Linguistics.

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*.

Junjie Hu, Sebastian Ruder, Aditya Siddhant, Graham Neubig, Orhan Firat, and Melvin Johnson. 2020. XTREME: A massively multilingual multi-task benchmark for evaluating cross-lingual generalisation. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4411–4421. PMLR.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. 2023. Editing models with task arithmetic.

Hamish Ivison, Yizhong Wang, Valentina Pyatkin, Nathan Lambert, Matthew Peters, Pradeep Dasigi, Joel Jang, David Wadden, Noah A. Smith, Iz Beltagy, and Hannaneh Hajishirzi. 2023. Camels in a changing climate: Enhancing LM adaptation with Tulu 2. *arXiv preprint arXiv:2311.10702*, abs/2311.10702.

Alankar Jain, Bhargavi Paranjape, and Zachary C. Lipton. 2019. Entity projection via machine translation for cross-lingual NER. 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 1083–1092, Hong Kong, China. Association for Computational Linguistics.

Albert Jiang, Alexandre Sablayrolles, Alexis Tacnet, Antoine Roux, Arthur Mensch, Audrey Herblin-Stoop, Baptiste Bout, Baudouin de Monicault, Blanche Savary, Bam4d, Caroline Feldman, Devendra Singh Chaplot, Diego de las Casas, Eleonore Arcelin, Emma Bou Hanna, Etienne Metzger, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Harizo Rajaona, Jean-Malo Delignon, Jia Li, Justus Murke, Louis Martin, Louis Ternon, Lucile Saulnier, Lélio Renard Lavaud, Margaret Jennings, Marie Pellat, Marie Torelli, Marie-Anne Lachaux, Nicolas Schuhl, Patrick von Platen, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Thibaut Lavril, Timothée Lacroix, Théophile Gervet, Thomas Wang, Valera Nemychnikova, William El Sayed, and William Marshall. 2024. Mixtral 8x22b.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825*.

Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig. 2020. How can we know what language models know? *Transactions of the Association for Computational Linguistics*, 8:423–438.

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

Anders Johannsen, Héctor Martínez Alonso, and Anders Søgaard. 2015. Any-language frame-semantic parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2062–2066, Lisbon, Portugal. Association for Computational Linguistics.

Pratik Joshi, Sebastin Santy, Amar Budhiraja, Kalika Bali, and Monojit Choudhury. 2020. The state and fate of linguistic diversity and inclusion in the NLP world. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6282–6293, Online. Association for Computational Linguistics.

Iman Jundi and Gabriella Lapesa. 2022. How to translate your samples and choose your shots? analyzing translate-train & few-shot cross-lingual transfer. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 129–150, Seattle, United States. Association for Computational Linguistics.

Katharina Kann, Kyunghyun Cho, and Samuel R. Bowman. 2019. Towards realistic practices in low-resource natural language processing: The development set. 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 3342–3349, Hong Kong, China. Association for Computational Linguistics.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 5156–5165. PMLR.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Wei-Jen Ko, Ahmed El-Kishky, Adithya Renduchintala, Vishrav Chaudhary, Naman Goyal, Francisco Guzmán, Pascale Fung, Philipp Koehn, and Mona Diab. 2021. Adapting high-resource NMT models to translate low-resource related languages without parallel data. 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 802–812, Online. Association for Computational Linguistics.

Fajri Koto and Ikhwan Koto. 2020. Towards computational linguistics in Minangkabau language: Studies on sentiment analysis and machine translation. In *Proceedings of the 34th Pacific Asia Conference on Language, Information and Computation*, pages 138–148, Hanoi, Vietnam. Association for Computational Linguistics.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

Anne Lauscher, Vinit Ravishankar, Ivan Vulić, and Goran Glavaš. 2020. From zero to hero: On the limitations of zero-shot language transfer with multilingual Transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4483–4499, Online. Association for Computational Linguistics.

Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. 2017. End-to-end neural coreference resolution. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 188–197, Copenhagen, Denmark. Association for Computational Linguistics.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020a. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Patrick Lewis, Barlas Oguz, Ruty Rinott, Sebastian Riedel, and Holger Schwenk. 2020b. 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.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020c. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. 2018. Measuring the intrinsic dimension of objective landscapes.

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.

Vladislav Lialin, Vijeta Deshpande, and Anna Rumshisky. 2023. Scaling down to scale up: A guide to parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.15647*.

Xi Victoria Lin, Todor Mihaylov, Mikel Artetxe, Tianlu Wang, Shuohui Chen, Daniel Simig, Myle Ott, Naman Goyal, Shruti Bhosale, Jingfei Du, Ramakanth Pasunuru, Sam Shleifer, Punit Singh Koura, Vishrav Chaudhary, Brian O'Horo, Jeff Wang, Luke Zettlemoyer, Zornitsa Kozareva, Mona Diab, Veselin Stoyanov, and Xian Li. 2022. Few-shot learning with multilingual generative language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 9019–9052, Abu Dhabi, United Arab Emirates. 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*, pages 8–14, Valencia, Spain. Association for Computational Linguistics.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. 2022. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In *Advances in Neural Information Processing Systems*, volume 35, pages 1950–1965. Curran Associates, Inc.

Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. 2020. Multilingual denoising pre-training for neural

machine translation. *Transactions of the Association for Computational Linguistics*, 8:726–742.

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.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V. Le, Barret Zoph, Jason Wei, and Adam Roberts. 2023. The flan collection: Designing data and methods for effective instruction tuning. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 22631–22648. PMLR.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Minh-Thang Luong, Quoc V. Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. 2016. Multi-task sequence to sequence learning. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. OpenReview.net.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1064–1074, Berlin, Germany. Association for Computational Linguistics.

Manuel Mager, Diónico Carrillo, and Ivan Meza. 2018. Probabilistic finite-state morphological segmenter for wixarika (huichol) language. *Journal of Intelligent & Fuzzy Systems*, 34(5):3081–3087.

Eran Malach, Gilad Yehudai, Shai Shalev-Schwartz, and Ohad Shamir. 2020. Proving the lottery ticket hypothesis: Pruning is all you need. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 6682–6691. PMLR.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, and Sayak Paul. 2022. Peft: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Michael S Matena and Colin A Raffel. 2022. Merging models with fisher-weighted averaging. In *Advances in Neural Information Processing Systems*, volume 35, pages 17703–17716. Curran Associates, Inc.

Michael McCloskey and Neal J Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier.

Meta AI. 2024. Introducing meta llama 3: The most capable openly available llm to date.

Elena Mihas. 2011. *Añaani katonkosatzi parenini, El idioma del alto Perené*. Milwaukee, WI: Clarks Graphics.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space.

Tomas Mikolov, Quoc V Le, and Ilya Sutskever. 2013b. Exploiting similarities among languages for machine translation. *arXiv preprint*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013c. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

Nafise Sadat Moosavi, Angela Fan, Vered Shwartz, Goran Glavaš, Shafiq Joty, Alex Wang, and Thomas Wolf, editors. 2020. *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*. Association for Computational Linguistics, Online.

Subhabrata Mukherjee, Ahmed Hassan Awadallah, and Jianfeng Gao. 2021. Xtremedistiltransformers: Task transfer for task-agnostic distillation.

Mohammed Muqeeth, Haokun Liu, Yufan Liu, and Colin Raffel. 2024. Learning to route among specialized experts for zero-shot generalization.

David Nadeau and Satoshi Sekine. 2007. A survey of named entity recognition and classification. *Lingvisticæ Investigationes*, 30(1):3–26.

Nihal V. Nayak, Peilin Yu, and Stephen Bach. 2023. Learning to compose soft prompts for compositional zero-shot learning. In *The Eleventh International Conference on Learning Representations*.

NLLB Team, Marta R. Costa-jussà, James Cross, Onur Çelebi, Maha Elbayad, Kenneth Heafield, Kevin Heffernan, Elahe Kalbassi, Janice Lam, Daniel Licht, Jean Maillard, Anna Sun, Skyler Wang, Guillaume Wenzek, Al Youngblood, Bapi Akula, Loic Barrault, Gabriel Mejia-Gonzalez, Prangthip Hansanti, John Hoffman, Semarley Jarrett, Kaushik Ram Sadagopan, Dirk Rowe, Shannon Spruit, Chau Tran, Pierre Andrews, Necip Fazil Ayan, Shruti Bhosale, Sergey Edunov, Angela Fan, Cynthia Gao, Vedanuj Goswami, Francisco Guzmán, Philipp Koehn, Alexandre Mourachko, Christophe Ropers, Safiyyah Saleem, Holger Schwenk, and Jeff Wang. 2022. No language left behind: Scaling human-centered machine translation.

Jaehoon Oh, Jongwoo Ko, and Se-Young Yun. 2022. Synergy with translation artifacts for training and inference in multilingual tasks. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6747–6754, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

OpenAI. 2023. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*.

John E Ortega, Richard Alexander Castro-Mamani, and Jaime Rafael Montoya Samame. 2020. Overcoming resistance: The normalization of an Amazonian tribal language. In *Proceedings of the 3rd Workshop on Technologies for MT of Low Resource Languages*, pages 1–13, Suzhou, China. Association for Computational Linguistics.

Oleksiy Ostapenko, Zhan Su, Edoardo Maria Ponti, Laurent Charlin, Nicolas Le Roux, Matheus Pereira, Lucas Caccia, and Alessandro Sordoni. 2024. Towards modular llms by building and reusing a library of loras.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc.

Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359.

Marinela Parovic, Alan Ansell, Ivan Vulić, and Anna Korhonen. 2023. Cross-lingual transfer with target language-ready task adapters. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 176–193, Toronto, Canada. Association for Computational Linguistics.

Marinela Parović, Goran Glavaš, Ivan Vulić, and Anna Korhonen. 2022. BAD-X: Bilingual adapters improve zero-shot cross-lingual transfer. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1791–1799, Seattle, United States. Association for Computational Linguistics.

Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. 2023. Instruction tuning with GPT-4. *arXiv preprint arXiv:2304.03277*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. 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, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021a. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503, Online. Association for Computational Linguistics.

Jonas Pfeiffer, Francesco Piccinno, Massimo Nicosia, Xinyi Wang, Machel Reid, and Sebastian Ruder. 2023a. mmT5: Modular multilingual pre-training solves source language hallucinations. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 1978–2008, Singapore. Association for Computational Linguistics.

Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulić, Sebastian Ruder, Kyunghyun Cho, and Iryna Gurevych. 2020a. AdapterHub: A framework for adapting transformers. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 46–54, Online. Association for Computational Linguistics.

Jonas Pfeiffer, Sebastian Ruder, Ivan Vulić, and Edoardo Ponti. 2023b. Modular deep learning. *Transactions on Machine Learning Research*. Survey Certification.

Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. 2020b. 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. 2021b. 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.

Emmanouil Antonios Platanios, Mrinmaya Sachan, Graham Neubig, and Tom Mitchell. 2018. Contextual parameter generation for universal neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 425–435, Brussels, Belgium. Association for Computational Linguistics.

E. Ponti, Alessandro Sordoni, and Siva Reddy. 2022. Combining modular skills in multitask learning. *ArXiv*, abs/2202.13914.

Edoardo M. Ponti, Ivan Vulić, Ryan Cotterell, Marinela Parovic, Roi Reichart, and Anna Korhonen. 2021a. Parameter space factorization for zero-shot learning across tasks and languages. *Transactions of the Association for Computational Linguistics*, 9:410–428.

Edoardo Maria Ponti. 2021. *Inductive Bias and Modular Design for Sample-Efficient Neural Language Learning*. Ph.D. thesis, University of Cambridge.

Edoardo Maria Ponti, Rahul Aralikatte, Disha Shrivastava, Siva Reddy, and Anders Søgaard. 2021b. Minimax and neyman–Pearson meta-learning for outlier languages. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 1245–1260, Online. Association for Computational Linguistics.

Edoardo Maria Ponti, Goran Glavaš, Olga Majewska, Qianchu Liu, Ivan Vulić, and Anna Korhonen. 2020. XCOPA: A multilingual dataset for causal commonsense reasoning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2362–2376, Online. Association for Computational Linguistics.

Edoardo Maria Ponti, Julia Kreutzer, Ivan Vulić, and Siva Reddy. 2021c. Modelling latent translations for cross-lingual transfer.

Edoardo Maria Ponti, Helen O'Horan, Yevgeni Berzak, Ivan Vulić, Roi Reichart, Thierry Poibeau, Ekaterina Shutova, and Anna Korhonen. 2019a. Modeling language variation and universals: A survey on typological linguistics for natural language processing. *Computational Linguistics*, 45(3):559–601.

Edoardo Maria Ponti, Ivan Vulić, Ryan Cotterell, Roi Reichart, and Anna Korhonen. 2019b. Towards zero-shot language modeling. 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 2900–2910, Hong Kong, China. Association for Computational Linguistics.

Sai Prasanna, Anna Rogers, and Anna Rumshisky. 2020. When BERT Plays the Lottery, All Tickets Are Winning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3208–3229, Online. Association for Computational Linguistics.

Ayu Purwarianti and Ida Ayu Putu Ari Crisdayanti. 2019. Improving bi-lstm performance for indonesian sentiment analysis using paragraph vector. In *2019 International Conference of Advanced Informatics: Concepts, Theory and Applications (ICAICTA)*, pages 1–5.

Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. 2017. Learning to generate reviews and discovering sentiment.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

A. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. 2014. Cnn features off-the-shelf: An astounding baseline for recognition. In *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 512–519, Los Alamitos, CA, USA. IEEE Computer Society.

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. 2017. Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 506–516.

Melissa Roemmele, Cosmin Adrian Bejan, and Andrew S Gordon. 2011. Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *2011 AAAI Spring Symposium Series*.

Andreas Rücklé, Gregor Geigle, Max Glockner, Tilman Beck, Jonas Pfeiffer, Nils Reimers, and Iryna Gurevych. 2021. AdapterDrop: On the efficiency of adapters in transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7930–7946, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Sebastian Ruder. 2017. Transfer Learning - Machine Learning's Next Frontier. http://ruder.io/transfer-learning/.

Sebastian Ruder. 2018. NLP's ImageNet moment has arrived. https://ruder.io/nlp-imagenet/.

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.

Sakriani Sakti and Satoshi Nakamura. 2013. Towards language preservation: Design and collection of graphemically balanced and parallel speech corpora of Indonesian ethnic languages. In *2013 International Conference Oriental COCOSDA held jointly with 2013 Conference on Asian Spoken Language Research and Evaluation (O-COCOSDA/CASLRE)*, pages 1–5.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. 2016. Meta-learning with memory-augmented neural networks. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1842–1850, New York, New York, USA. PMLR.

Timo Schick and Hinrich Schütze. 2021. Exploiting cloze-questions for few-shot text classification and natural language inference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 255–269, Online. Association for Computational Linguistics.

Fabian David Schmidt, Ivan Vulić, and Goran Glavaš. 2022. Don't stop fine-tuning: On training regimes for few-shot cross-lingual transfer with multilingual language models. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 10725–10742, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Richard Schoonhoven, Bram Veenboer, Ben Van Werkhoven, and K. Joost Batenburg. 2022. Going green: optimizing GPUs for energy efficiency through model-steered auto-tuning . In *2022 IEEE/ACM International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 48–59, Los Alamitos, CA, USA. IEEE Computer Society.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2016. Bidirectional attention flow for machine comprehension.

Noam Shazeer. 2020. Glu variants improve transformer.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2020. Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020. AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4222–4235, Online. Association for Computational Linguistics.

Reece Shuttleworth, Jacob Andreas, Antonio Torralba, and Pratyusha Sharma. 2024. Lora vs full fine-tuning: An illusion of equivalence.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy. Association for Computational Linguistics.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. 2023. Roformer: Enhanced transformer with rotary position embedding.

Herry Sujaini. 2020. Improving the role of language model in statistical machine translation (Indonesian-Javanese). *International Journal of Electrical and Computer Engineering*, 10:2102–2109.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2024. A simple and effective pruning approach for large language models. In *The Twelfth International Conference on Learning Representations*.

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

Yi-Lin Sung, Varun Nair, and Colin Raffel. 2021. Training neural networks with fixed sparse masks. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 24193–24205.

Rich Sutton. 2019. The bitter lesson.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and Jason Wei. 2023. Challenging BIG-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13003–13051, Toronto, Canada. Association for Computational Linguistics.

Chaofan Tao, Lu Hou, Wei Zhang, Lifeng Shang, Xin Jiang, Qun Liu, Ping Luo, and Ngai Wong. 2022. Compression of generative pre-trained language models via quantization. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4821–4836, Dublin, Ireland. Association for Computational Linguistics.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford Alpaca: An instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca.

Jörg Tiedemann. 2012. Parallel data, tools and interfaces in OPUS. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 2214–2218, Istanbul, Turkey. European Language Resources Association (ELRA).

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Bayu Distiawan Trisedya and Dyah Inastra. 2014. Creating Indonesian-Javanese Parallel Corpora Using Wikipedia Articles. In *2014 International Conference on Advanced Computer Science and Information System*, pages 239–245.

Ahmet Üstün, Arianna Bisazza, Gosse Bouma, and Gertjan van Noord. 2020. UDapter: Language adaptation for truly Universal Dependency parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2302–2315, Online. Association for Computational Linguistics.

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

Vered Volansky, Noam Ordan, and Shuly Wintner. 2013. On the features of translationese. *Digital Scholarship in the Humanities*, 30(1):98–118.

A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K.J. Lang. 1989. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(3):328–339.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Haifeng Wang, Hua Wu, Zhongjun He, Liang Huang, and Kenneth Ward Church. 2022a. Progress in machine translation. *Engineering*, 18:143–153.

Thomas Wang, Adam Roberts, Daniel Hesslow, Teven Le Scao, Hyung Won Chung, Iz Beltagy, Julien Launay, and Colin Raffel. 2022b. What language model architecture and pretraining objective works best for zero-shot generalization? In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 22964–22984. PMLR.

Wenhui Wang, Hangbo Bao, Shaohan Huang, Li Dong, and Furu Wei. 2021. MiniLMv2: Multi-head self-attention relation distillation for compressing pretrained transformers. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 2140–2151, Online. Association for Computational Linguistics.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Advances in Neural Information Processing Systems*, volume 33, pages 5776–5788. Curran Associates, Inc.

Yequan Wang, Minlie Huang, Xiaoyan Zhu, and Li Zhao. 2016. Attention-based LSTM for aspect-level sentiment classification. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 606–615, Austin, Texas. Association for Computational Linguistics.

Yizhong Wang, Hamish Ivison, Pradeep Dasigi, Jack Hessel, Tushar Khot, Khyathi Chandu, David Wadden, Kelsey MacMillan, Noah A Smith, Iz Beltagy, and Hannaneh Hajishirzi. 2023. How far can camels go? exploring the state of instruction tuning on open resources. In *Advances in Neural Information Processing Systems*, volume 36, pages 74764–74786. Curran Associates, Inc.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. 2022. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*.

Xiangpeng Wei, Haoran Wei, Huan Lin, Tianhao Li, Pei Zhang, Xingzhang Ren, Mei Li, Yu Wan, Zhiwei Cao, Binbin Xie, Tianxiang Hu, Shangjie Li, Binyuan Hui, Bowen Yu, Dayiheng Liu, Baosong Yang, Fei Huang, and Jun Xie. 2023. Polylm: An open source polyglot large language model.

Bryan Wilie, Karissa Vincentio, Genta Indra Winata, Samuel Cahyawijaya, Xiaohong Li, Zhi Yuan Lim, Sidik Soleman, Rahmad Mahendra, Pascale Fung, Syafri Bahar, and Ayu Purwarianti. 2020. IndoNLU: Benchmark and resources for evaluating Indonesian natural language understanding. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 843–857, Suzhou, China. Association for Computational Linguistics.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.

Genta Winata, Shijie Wu, Mayank Kulkarni, Thamar Solorio, and Daniel Preotiuc-Pietro. 2022. Cross-lingual few-shot learning on unseen languages. In *Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 777–791, Online only. Association for Computational Linguistics.

Genta Indra Winata, Alham Fikri Aji, Samuel Cahyawijaya, Rahmad Mahendra, Fajri Koto, Ade Romadhony, Kemal Kurniawan, David Moeljadi, Radityo Eko Prasojo, Pascale Fung, Timothy Baldwin, Jey Han Lau, Rico Sennrich, and Sebastian Ruder. 2023. NusaX: Multilingual parallel sentiment dataset for 10 Indonesian local languages. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 815–834, Dubrovnik, Croatia. Association for Computational Linguistics.

Cahya Wirawan. 2022. LibriVox-Indonesia. https://huggingface.co/datasets/indonesian-nlp/librivox-indonesia.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Shijie Wu and Mark Dredze. 2019. Beto, bentz, becas: The surprising cross-lingual effectiveness of BERT. 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 833–844, Hong Kong, China. Association for Computational Linguistics.

Zhengxuan Wu, Aryaman Arora, Zheng Wang, Atticus Geiger, Dan Jurafsky, Christopher D. Manning, and Christopher Potts. 2024. Reft: Representation finetuning for language models.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 38087–38099. PMLR.

Haoran Xu and Kenton Murray. 2022. Por qué não utiliser alla språk? mixed training with gradient optimization in few-shot cross-lingual transfer. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 2043–2059, Seattle, United States. Association for Computational Linguistics.

Runxin Xu, Fuli Luo, Zhiyuan Zhang, Chuanqi Tan, Baobao Chang, Songfang Huang, and Fei Huang. 2021. Raise a child in large language model: Towards effective and generalizable fine-tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9514–9528, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

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.

Enneng Yang, Li Shen, Guibing Guo, Xingwei Wang, Xiaochun Cao, Jie Zhang, and Dacheng Tao. 2024. Model merging in llms, mllms, and beyond: Methods, theories, applications and opportunities.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. 2022. Zeroquant: Efficient and affordable post-training quantization for large-scale transformers. In *Advances in Neural Information Processing Systems*, volume 35, pages 27168–27183. Curran Associates, Inc.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8bert: Quantized 8bit bert. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*, pages 36–39.

Daniel Zeman, Joakim Nivre, Mitchell Abrams, Elia Ackermann, Noëmi Aepli, Hamid Aghaei, Željko Agić, Amir Ahmadi, Lars Ahrenberg, Chika Kennedy Ajede, Gabrielė Aleksandravičiūtė, Ika Alfina, Lene Antonsen, Katya Aplonova, Angelina Aquino, Carolina Aragon, Maria Jesus Aranzabe, Hórunn Arnardóttir, Gashaw Arutie, Jessica Naraiswari Arwidarasti, Masayuki Asahara, Luma Ateyah, Furkan Atmaca, Mohammed Attia, Aitziber Atutxa, Liesbeth Augustinus, Elena Badmaeva, Keerthana Balasubramani, Miguel Ballesteros, Esha Banerjee, Sebastian Bank, Verginica Barbu Mititelu, Victoria Basmov, Colin Batchelor, John Bauer, Seyyit Talha Bedir, Kepa Bengoetxea, Gözde Berk, Yevgeni Berzak, Irshad Ahmad Bhat, Riyaz Ahmad Bhat, Erica Biagetti, Eckhard Bick, Agnė Bielinskienė, Kristín Bjarnadóttir, Rogier Blokland, Victoria Bobicev, Loïc Boizou, Emanuel Borges Völker, Carl Börstell, Cristina Bosco, Gosse Bouma, Sam Bowman, Adriane Boyd, Kristina Brokaitė, Aljoscha Burchardt, Marie Candito, Bernard Caron, Gauthier Caron, Tatiana Cavalcanti, Gülşen Cebiroğlu Eryiğit, Flavio Massimiliano Cecchini, Giuseppe G. A. Celano, Slavomír Čéplö, Savas Cetin, Özlem Çetinoğlu, Fabricio Chalub, Ethan Chi, Yongseok Cho, Jinho Choi, Jayeol Chun, Alessandra T. Cignarella, Silvie Cinková, Aurélie Collomb, Çağrı Çöltekin, Miriam Connor, Marine Courtin, Elizabeth Davidson, Marie-Catherine de Marneffe, Valeria de Paiva, Mehmet Oguz Derin, Elvis de Souza, Arantza Diaz de Ilarraza, Carly Dickerson, Arawinda Dinakaramani, Bamba Dione, Peter Dirix, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Puneet Dwivedi, Hanne Eckhoff, Marhaba Eli, Ali Elkahky, Binyam Ephrem, Olga Erina, Tomaž Erjavec, Aline Etienne, Wograine Evelyn, Sidney Facundes, Richárd Farkas, Marília Fernanda, Hector Fernandez Alcalde, Jennifer Foster, Cláudia Freitas, Kazunori Fujita, Katarína Gajdošová, Daniel Galbraith, Marcos Garcia, Moa Gärdenfors, Sebastian Garza, Fabrício Ferraz Gerardi, Kim Gerdes, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gökırmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta González Saavedra, Bernadeta Griciūtė, Matias Grioni, Loïc Grobol, Normunds Grūzītis, Bruno Guillaume, Céline Guillot-Barbance, Tunga Güngör, Nizar Habash, Hinrik Hafsteinsson, Jan Hajič, Jan Hajič jr., Mika Hämäläinen, Linh Hà Mỹ, Na-Rae Han, Muhammad Yudistira Hanifmuti, Sam Hardwick, Kim Harris, Dag Haug, Johannes Heinecke, Oliver Hellwig, Felix Hennig, Barbora Hladká, Jaroslava Hlaváčová, Florinel Hociung, Petter Hohle, Eva Huber, Jena Hwang, Takumi Ikeda, Anton Karl Ingason, Radu Ion, Elena Irimia, Ọlájídé

Ishola, Tomáš Jelínek, Anders Johannsen, Hildur Jónsdóttir, Fredrik Jørgensen, Markus Juutinen, Sarveswaran K, Hüner Kaşıkara, Andre Kaasen, Nadezhda Kabaeva, Sylvain Kahane, Hiroshi Kanayama, Jenna Kanerva, Boris Katz, Tolga Kayadelen, Jessica Kenney, Václava Kettnerová, Jesse Kirchner, Elena Klementieva, Arne Köhn, Abdullatif Köksal, Kamil Kopacewicz, Timo Korkiakangas, Natalia Kotsyba, Jolanta Kovalevskaitė, Simon Krek, Parameswari Krishnamurthy, Sookyoung Kwak, Veronika Laippala, Lucia Lam, Lorenzo Lambertino, Tatiana Lando, Septina Dian Larasati, Alexei Lavrentiev, John Lee, Phương Lê Hồng, Alessandro Lenci, Saran Lertpradit, Herman Leung, Maria Levina, Cheuk Ying Li, Josie Li, Keying Li, Yuan Li, KyungTae Lim, Krister Lindén, Nikola Ljubešić, Olga Loginova, Andry Luthfi, Mikko Luukko, Olga Lyashevskaya, Teresa Lynn, Vivien Macketanz, Aibek Makazhanov, Michael Mandl, Christopher Manning, Ruli Manurung, Cătălina Mărănduc, David Mareček, Katrin Marheinecke, Héctor Martínez Alonso, André Martins, Jan Mašek, Hiroshi Matsuda, Yuji Matsumoto, Ryan McDonald, Sarah McGuinness, Gustavo Mendonça, Niko Miekka, Karina Mischenkova, Margarita Misirpashayeva, Anna Missilä, Cătălin Mititelu, Maria Mitrofan, Yusuke Miyao, AmirHossein Mojiri Foroushani, Amirsaeid Moloodi, Simonetta Montemagni, Amir More, Laura Moreno Romero, Keiko Sophie Mori, Shinsuke Mori, Tomohiko Morioka, Shigeki Moro, Bjartur Mortensen, Bohdan Moskalevskyi, Kadri Muischnek, Robert Munro, Yugo Murawaki, Kaili Müürisep, Pinkey Nainwani, Mariam Nakhlé, Juan Ignacio Navarro Horñiacek, Anna Nedoluzhko, Gunta Nešpore-Bērzkalne, Lương Nguyễn Thị, Huyền Nguyễn Thị Minh, Yoshihiro Nikaido, Vitaly Nikolaev, Rattima Nitisaroj, Alireza Nourian, Hanna Nurmi, Stina Ojala, Atul Kr. Ojha, Adédayọ̀ Olúòkun, Mai Omura, Emeka Onwuegbuzia, Petya Osenova, Robert Östling, Lilja Øvrelid, Şaziye Betül Özateş, Arzucan Özgür, Balkız Öztürk Başaran, Niko Partanen, Elena Pascual, Marco Passarotti, Agnieszka Patejuk, Guilherme Paulino-Passos, Angelika Peljak-Łapińska, Siyao Peng, Cenel-Augusto Perez, Natalia Perkova, Guy Perrier, Slav Petrov, Daria Petrova, Jason Phelan, Jussi Piitulainen, Tommi A Pirinen, Emily Pitler, Barbara Plank, Thierry Poibeau, Larisa Ponomareva, Martin Popel, Lauma Pretkalniņa, Sophie Prévost, Prokopis Prokopidis, Adam Przepiórkowski, Tiina Puolakainen, Sampo Pyysalo, Peng Qi, Andriela Rääbis, Alexandre Rademaker, Taraka Rama, Loganathan Ramasamy, Carlos Ramisch, Fam Rashel, Mohammad Sadegh Rasooli, Vinit Ravishankar, Livy Real, Petru Rebeja, Siva Reddy, Georg Rehm, Ivan Riabov, Michael Rießler, Erika Rimkutė, Larissa Rinaldi, Laura Rituma, Luisa Rocha, Eiríkur Rögnvaldsson, Mykhailo Romanenko, Rudolf Rosa, Valentin Roșca, Davide Rovati, Olga Rudina, Jack Rueter, Kristján Rúnarsson, Shoval Sadde, Pegah Safari, Benoît Sagot, Aleksi Sahala, Shadi Saleh, Alessio Salomoni, Tanja Samardžić, Stephanie Samson, Manuela Sanguinetti, Dage Särg, Baiba Saulīte, Yanin Sawanakunanon, Kevin Scannell, Salvatore Scarlata, Nathan Schneider, Sebastian Schuster, Djamé Seddah, Wolfgang Seeker, Mojgan Seraji, Mo Shen, Atsuko Shimada, Hiroyuki Shirasu, Muh Shohibussirri, Dmitry Sichinava, Einar Freyr Sigurðsson, Aline Silveira, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Šimková, Kiril Simov, Maria Skachedubova, Aaron Smith, Isabela Soares-Bastos, Carolyn Spadine, Steinþór Steingrímsson, Antonio Stella, Milan Straka, Emmett Strickland, Jana Strnadová, Alane Suhr, Yogi Lesmana Sulestio, Umut Sulubacak, Shingo Suzuki, Zsolt Szántó, Dima Taji, Yuta Takahashi, Fabio Tamburini, Mary Ann C. Tan, Takaaki Tanaka, Samson Tella, Isabelle Tellier, Guillaume Thomas, Liisi Torga, Marsida Toska, Trond Trosterud, Anna Trukhina, Reut Tsarfaty, Utku Türk, Francis Tyers, Sumire Uematsu, Roman Untilov, Zdeňka Urešová, Larraitz Uria, Hans Uszkoreit, Andrius

Utka, Sowmya Vajjala, Daniel van Niekerk, Gertjan van Noord, Viktor Varga, Eric Villemonte de la Clergerie, Veronika Vincze, Aya Wakasa, Joel C. Wallenberg, Lars Wallin, Abigail Walsh, Jing Xian Wang, Jonathan North Washington, Maximilan Wendt, Paul Widmer, Seyi Williams, Mats Wirén, Christian Wittern, Tsegay Woldemariam, Tak-sum Wong, Alina Wróblewska, Mary Yako, Kayo Yamashita, Naoki Yamazaki, Chunxiao Yan, Koichi Yasuoka, Marat M. Yavrumyan, Zhuoran Yu, Zdeněk Žabokrtský, Shorouq Zahra, Amir Zeldes, Hanzhi Zhu, and Anna Zhuravleva. 2020. Universal dependencies 2.7. LINDAT/CLARIAH-CZ digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University.

Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. 2020. TernaryBERT: Distillation-aware ultra-low bit BERT. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 509–521, Online. Association for Computational Linguistics.

Mengjie Zhao, Yi Zhu, Ehsan Shareghi, Ivan Vulić, Roi Reichart, Anna Korhonen, and Hinrich Schütze. 2021. A closer look at few-shot crosslingual transfer: The choice of shots matters. 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 5751–5767, Online. Association for Computational Linguistics.

Appendix A

Multilingual Adapter Generation for Efficient Cross-Lingual Transfer

A.1 Languages

A.1.1 Training Languages

Table A.1 Details of languages used for MAD-G training.

code	name	family	genus
ab	Abkhazian	Northwest Caucasian	-
ar	Arabic	Afro-Asiatic	Semitic
ary	Moroccan Arabic	Afro-Asiatic	Semitic
arz	Egyptian Arabic	Afro-Asiatic	Semitic
atj	Atikamekw	Algic	Algonquian
av	Avar	Nakh-Daghestanian	Avar-Andic-Tsezic
ay	Aymara	Aymaran	-
azb	South Azerbaijani	Turkic	Southwestern
bo	Tibetan	Sino-Tibetan	Bodic
bxr	Buryat	Mongolic	-
cdo	Min Dong	Sino-Tibetan	-
ce	Chechen	Nakh-Daghestanian	Nakh
ceb	Cebuano	Austronesian	Greater Central Philippine
cv	Chuvash	Turkic	Oghur
cy	Welsh	IE	Celtic
el	Greek	IE	Greek
en	English	IE	Germanic
et	Estonian	Uralic	Finnic
eu	Basque	Basque	-
fa	Persian	IE	Iranian
fi	Finnish	Uralic	Finnic
fr	French	IE	Romance
gn	Guarani	Tupian	Tupi-Guarani
ha	Hausa	Afro-Asiatic	West Chadic
hak	Hakka	Sino-Tibetan	-
he	Hebrew	Afro-Asiatic	Semitic
hu	Hungarian	Uralic	Ugric
hy	Armenian	IE	Armenian
id	Indonesian	Austronesian	Malayo-Sumbawan
ig	Igbo	Niger-Congo	Igboid
inh	Ingush	Nakh-Daghestanian	Nakh
ja	Japanese	Japanese	-
jv	Javanese	Austronesian	Javanese
ka	Georgian	Kartvelian	-
kab	Kabyle	Afro-Asiatic	Berber
kbd	Karbardian Circassian	Northwest Caucasian	-
kbp	Kabiye	Niger-Congo	Southern-Central Gur
kk	Kazakh	Turkic	Northwestern
km	Khmer	Austro-Asiatic	Khmer
kn	Kannada	Dravidian	Southern
ko	Korean	Korean	-
kv	Komi	Uralic	Permic
la	Latin	IE	Latin
lbe	Lak	Nakh-Daghestanian	Lak-Dargwa
lez	Lezgian	Nakh-Daghestanian	Lezgic
ln	Lingala	Niger-Congo	Bantoid
lo	Lao	Tai-Kadai	-
mg	Malagasy	Austronesian	Barito

code	name	family	genus
mhr	Meadow Mari	Uralic	Mari
min	Minangkabau	Austronesian	Malayo-Sumbawan
ml	Malayalam	Dravidian	Southern
mn	Mongolian	Mongolic	-
mrj	Hill Mari	Uralic	Mari
ms	Malay	Austronesian	Malayo-Sumbawan
mt	Maltese	Afro-Asiatic	Semitic
my	Burmese	Sino-Tibetan	Burmese-Lolo
myv	Erzya	Uralic	Mordvin
nah	Nahuatl	Uto-Aztecan	Aztecan
new	Newar	Sino-Tibetan	Mahakiranti
nso	Northern Sotho	Niger-Congo	Bantoid
nv	Navajo	Na-Dene	Athapaskan
om	Oromo	Afro-Asiatic	Lowland East Cushitic
qu	Quechua	Quechuan	-
ru	Russian	IE	Slavic
rw	Kinyarwanda	Niger-Congo	Bantoid
sah	Sakha	Turkic	Northeastern
sat	Santali	Austro-Asiatic	Munda
se	Northern Sami	Uralic	Sami
shn	Shan	Tai-Kadai	-
smn	Inari Sami	Uralic	Sami
sn	Shona	Niger-Congo	Bantoid
so	Somali	Afro-Asiatic	Lowland East Cushitic
sq	Albanian	IE	Albanian
su	Sundanese	Austronesian	Malayo-Sumbawan
sv	Swedish	IE	Germanic
sw	Swahili	Niger-Congo	Bantoid
ta	Tamil	Dravidian	Southern
tcy	Tulu	Dravidian	Southern
te	Telugu	Dravidian	South Central
th	Thai	Tai-Kadai	-
tl	Tagalog	Austronesian	Greater Central Philippine
tr	Turkish	Turkic	Southwestern
tt	Tatar	Turkic	Northwestern
tyv	Tuvan	Turkic	Northeastern
ug	Uyghur	Turkic	Southeastern
uz	Uzbek	Turkic	Southeastern
vi	Vietnamese	Austro-Asiatic	Viet-Muong
war	Waray-Waray	Austronesian	Greater Central Philippine
wuu	Wu	Sino-Tibetan	-
xal	Kalmyk	Mongolic	-
xmf	Mingrelian	Kartvelian	-
yo	Yoruba	Niger-Congo	Defoid
za	Zhuang	Tai-Kadai	-
zh	Chinese	Sino-Tibetan	-
zu	Zulu	Niger-Congo	Bantoid

A.1.2 Universal Dependencies Evaluation Languages

Table A.2 Details of languages used for POS tagging and dependency parsing evaluation. `unseen` languages have their language sub-group (`mBERT-genus`, `MAD-G-genus` or `unseen-genus`) specified.

code	name	group	treebank	family	genus
af	Afrikaans	mBERT-seen	UD_Afrikaans-AfriBooms	IE	Germanic
ajp	South Levantine Arabic	mBERT-genus	UD_South_Levantine_Arabic-MADAR	Afro-Asiatic	Semitic
akk	Akkadian	mBERT-genus	UD_Akkadian-RIAO	Afro-Asiatic	Semitic
apu	Apurina	unseen-genus	UD_Apurina-UFPA	Arawakan	-
aqz	Akuntsu	unseen-genus	UD_Akuntsu-TuDeT	Tupian	Tupari
ar	Arabic	mBERT-seen	UD_Arabic-PUD	Afro-Asiatic	Semitic
bam	Bambara	unseen-genus	UD_Bambara-CRB	Mande	-
be	Belarusian	mBERT-seen	UD_Belarusian-HSE	IE	Slavic
bg	Bulgarian	mBERT-seen	UD_Bulgarian-BTB	IE	Slavic
bho	Bhojpuri	mBERT-genus	UD_Bhojpuri-BHTB	IE	Indic
br	Breton	mBERT-seen	UD_Breton-KEB	IE	Celtic
bxr	Buryat	MAD-G-seen	UD_Buryat-BDT	Mongolic	-
ca	Catalan	mBERT-seen	UD_Catalan-AnCora	IE	Romance
ckt	Chukchi	unseen-genus	UD_Chukchi-HSE	Chukotko-Kamchatkan	-
cs	Czech	mBERT-seen	UD_Czech-PDT	IE	Slavic
cu	Old Church Slavonic	mBERT-genus	UD_Old_Church_Slavonic-PROIEL	IE	Slavic
cy	Welsh	mBERT-seen	UD_Welsh-CCG	IE	Celtic
da	Danish	mBERT-seen	UD_Danish-DDT	IE	Germanic
de	German	mBERT-seen	UD_German-HDT	IE	Germanic
el	Greek	mBERT-seen	UD_Greek-GDT	IE	Greek
en	English	mBERT-seen	UD_English-EWT	IE	Germanic
es	Spanish	mBERT-seen	UD_Spanish-AnCora	IE	Romance
et	Estonian	mBERT-seen	UD_Estonian-EDT	Uralic	Finnic
eu	Basque	mBERT-seen	UD_Basque-BDT	Basque	-
fa	Persian	mBERT-seen	UD_Persian-PerDT	IE	Iranian
fi	Finnish	mBERT-seen	UD_Finnish-TDT	Uralic	Finnic
fo	Faroese	mBERT-genus	UD_Faroese-FarPaHC	IE	Germanic
fr	French	mBERT-seen	UD_French-GSD	IE	Romance
fro	Old French	mBERT-genus	UD_Old_French-SRCMF	IE	Romance
ga	Irish	mBERT-seen	UD_Irish-IDT	IE	Celtic
gd	Scottish Gaelic	mBERT-genus	UD_Scottish_Gaelic-ARCOSG	IE	Celtic
gl	Galician	mBERT-seen	UD_Galician-TreeGal	IE	Romance
got	Gothic	mBERT-genus	UD_Gothic-PROIEL	IE	Germanic
gsw	Swiss German	mBERT-genus	UD_Swiss_German-UZH	IE	Germanic
gun	Mbya Guarani	MAD-G-genus	UD_Mbya_Guarani-Thomas	Tupian	Tupi-Guarani
gv	Manx	mBERT-genus	UD_Manx-Cadhan	IE	Celtic
he	Hebrew	mBERT-seen	UD_Hebrew-HTB	Afro-Asiatic	Semitic

code	name	group	treebank	family	genus
hi	Hindi	mBERT-seen	UD_Hindi-HDTB	IE	Indic
hr	Croatian	mBERT-seen	UD_Croatian-SET	IE	Slavic
hsb	Upper Sorbian	mBERT-genus	UD_Upper_Sorbian-UFAL	IE	Slavic
hu	Hungarian	mBERT-seen	UD_Hungarian-Szeged	Uralic	Ugric
hy	Armenian	mBERT-seen	UD_Armenian-ArmTDP	IE	Armenian
id	Indonesian	mBERT-seen	UD_Indonesian-PUD	Austronesian	Malayo-Sumbawan
is	Icelandic	mBERT-seen	UD_Icelandic-IcePaHC	IE	Germanic
it	Italian	mBERT-seen	UD_Italian-ISDT	IE	Romance
ja	Japanese	mBERT-seen	UD_Japanese-GSD	Japanese	-
kfm	Khunsari	mBERT-genus	UD_Khunsari-AHA	IE	Iranian
kk	Kazakh	mBERT-seen	UD_Kazakh-KTB	Turkic	Northwestern
kmr	Kurmanji	mBERT-genus	UD_Kurmanji-MG	IE	Iranian
ko	Korean	mBERT-seen	UD_Korean-GSD	Korean	-
koi	Komi Permyak	MAD-G-genus	UD_Komi_Permyak-UH	Uralic	Permic
kpv	Komi Zyrian	MAD-G-seen	UD_Komi_Zyrian-Lattice	Uralic	Permic
krl	Karelian	mBERT-genus	UD_Karelian-KKPP	Uralic	Finnic
la	Latin	mBERT-seen	UD_Latin-LLCT	IE	Latin
lt	Lithuanian	mBERT-seen	UD_Lithuanian-ALKSNIS	IE	Baltic
lv	Latvian	mBERT-seen	UD_Latvian-LVTB	IE	Baltic
lzh	Classical Chinese	mBERT-genus	UD_Classical_Chinese-Kyoto	Sino-Tibetan	-
mdf	Moksha	MAD-G-genus	UD_Moksha-JR	Uralic	Mordvin
mr	Marathi	mBERT-seen	UD_Marathi-UFAL	IE	Indic
mt	Maltese	MAD-G-seen	UD_Maltese-MUDT	Afro-Asiatic	Semitic
myu	Munduruku	unseen-genus	UD_Munduruku-TuDeT	Tupian	Munduruku
myv	Erzya	MAD-G-seen	UD_Erzya-JR	Uralic	Mordvin
nl	Dutch	mBERT-seen	UD_Dutch-Alpino	IE	Germanic
no	Norwegian	mBERT-seen	UD_Norwegian-Bokmaal	IE	Germanic
nyg	Nayini	mBERT-genus	UD_Nayini-AHA	IE	Iranian
olo	Livvi	mBERT-genus	UD_Livvi-KKPP	Uralic	Finnic
orv	Old East Slavic	mBERT-genus	UD_Old_Russian-RNC	IE	Slavic
pcm	Naija	unseen-genus	UD_Naija-NSC	Creole	-
pl	Polish	mBERT-seen	UD_Polish-PDB	IE	Slavic
pt	Portuguese	mBERT-seen	UD_Portuguese-GSD	IE	Romance
ro	Romanian	mBERT-seen	UD_Romanian-RRT	IE	Romance
ru	Russian	mBERT-seen	UD_Russian-GSD	IE	Slavic
sa	Sanskrit	mBERT-genus	UD_Sanskrit-UFAL	IE	Indic
sk	Slovak	mBERT-seen	UD_Slovak-SNK	IE	Slavic
sl	Slovenian	mBERT-seen	UD_Slovenian-SSJ	IE	Slavic
sme	North Sami	MAD-G-seen	UD_North_Sami-Giella	Uralic	Sami
sms	Skolt Sami	MAD-G-genus	UD_Skolt_Sami-Giellagas	Uralic	Sami
soj	Soi	mBERT-genus	UD_Soi-AHA	IE	Iranian
sq	Albanian	mBERT-seen	UD_Albanian-TSA	IE	Albanian
sr	Serbian	mBERT-seen	UD_Serbian-SET	IE	Slavic

code	name	group	treebank	family	genus
sv	Swedish	mBERT-seen	UD_Swedish-Talbanken	IE	Germanic
ta	Tamil	mBERT-seen	UD_Tamil-TTB	Dravidian	Southern
te	Telugu	mBERT-seen	UD_Telugu-MTG	Dravidian	South Central
th	Thai	mBERT-seen	UD_Thai-PUD	Tai-Kadai	-
tl	Tagalog	mBERT-seen	UD_Tagalog-TRG	Austronesian	Greater Central Philippine
tr	Turkish	mBERT-seen	UD_Turkish-GB	Turkic	Southwestern
ug	Uyghur	MAD-G-seen	UD_Uyghur-UDT	Turkic	Southeastern
uk	Ukrainian	mBERT-seen	UD_Ukrainian-IU	IE	Slavic
ur	Urdu	mBERT-seen	UD_Urdu-UDTB	IE	Indic
vi	Vietnamese	mBERT-seen	UD_Vietnamese-VTB	Austro-Asiatic	Viet-Muong
wbp	Warlpiri	unseen-genus	UD_Warlpiri-UFAL	Pama-Nyungan	-
wo	Wolof	unseen-genus	UD_Wolof-WTB	Niger-Congo	Northern Atlantic
yo	Yoruba	mBERT-seen	UD_Yoruba-YTB	Niger-Congo	Defoid
yue	Cantonese	mBERT-genus	UD_Cantonese-HK	Sino-Tibetan	-
zh	Chinese	mBERT-seen	UD_Chinese-GSD	Sino-Tibetan	-

A.2 Full Results

A.2.1 Single-source Transfer

Table A.3 Full per-language results for single-source zero-shot cross-lingual transfer experiments. POS tagging results are given as accuracy scores, dependency parsing results are unlabelled/labelled attachment scores. G = MAD-G, LS = MAD-G-LS, en = MAD-G-en, TA = TA-only, X = MAD-X, mB = mBERT-ft, R = XLM-R-ft.

language		Part-of-speech tagging							Dependency parsing						
code	group	G	LS	en	TA	X	mB	R	G	LS	en	TA	X	mB	R
af	mBERT-seen	81.5	84.5	86.0	86.2	82.8	85.7	88.0	61.5/47.2	67.4/53.9	68.1/55.4	68.3/55.2	61.6/47.2	65.6/52.2	63.1/49.6
ajp	mBERT-genus	55.8	58.0	56.4	58.8	56.0	54.9	63.0	48.7/27.9	48.4/28.6	48.9/30.7	50.4/33.0	50.4/31.1	46.0/28.6	26.0/13.2
akk	mBERT-genus	41.1	38.9	36.7	33.9	30.4	33.2	30.0	26.4/5.3	25.9/5.5	22.6/4.4	23.6/4.3	23.9/3.5	20.3/3.2	19.8/3.4
apu	unseen-genus	48.2	29.2	37.7	41.9	37.8	43.6	32.7	18.7/10.3	20.3/6.5	19.6/8.9	17.7/7.5	18.0/4.4	16.3/5.8	15.8/7.2
aqz	unseen-genus	32.5	25.0	27.5	27.5	21.2	33.8	16.2	32.5/6.2	26.2/2.5	28.8/11.2	26.2/11.2	28.8/13.8	21.2/7.5	30.0/11.2
ar	mBERT-seen	72.5	73.1	72.8	74.0	69.1	67.5	78.1	66.0/49.9	64.7/48.4	66.1/49.6	67.4/49.0	65.5/50.0	69.0/50.7	48.2/34.6
bam	unseen-genus	38.0	36.0	36.6	37.6	30.8	33.6	25.5	26.8/8.2	26.7/7.1	30.8/10.6	30.2/9.5	28.9/6.6	30.4/9.6	21.3/5.8
be	mBERT-seen	83.7	84.7	84.9	84.6	84.5	84.8	88.1	68.8/58.3	68.5/58.4	70.8/60.7	70.5/59.2	65.4/54.7	72.3/62.5	65.1/55.4
bg	mBERT-seen	86.3	86.4	86.6	86.4	86.4	86.2	88.8	81.6/66.5	80.9/65.7	82.1/67.0	82.4/67.0	77.2/62.0	83.1/68.4	66.8/52.7
bho	mBERT-genus	43.5	46.9	48.7	49.4	51.2	47.2	50.4	30.2/17.2	30.8/15.3	31.3/16.7	33.0/17.1	22.0/10.2	31.2/16.4	25.5/14.1
br	mBERT-seen	65.1	66.3	69.7	71.5	61.9	65.8	58.3	63.3/42.5	64.7/43.8	70.9/52.1	71.3/52.6	60.1/35.6	66.2/47.0	44.0/27.2
bxr	MAD-G-seen	68.6	66.3	58.3	59.6	70.5	55.9	59.5	41.4/22.3	39.4/19.7	39.3/19.4	41.6/19.9	38.3/23.9	41.2/19.4	35.9/17.1
ca	mBERT-seen	86.7	86.4	86.6	86.8	87.3	87.0	88.6	75.5/63.4	75.1/62.8	76.5/64.7	76.5/63.9	72.3/60.0	78.1/66.4	74.4/63.1
ckt	unseen-genus	30.7	24.8	23.5	23.6	23.2	22.6	30.3	24.9/12.0	20.3/9.1	18.5/10.9	20.4/10.3	21.0/12.4	17.6/9.1	32.4/17.6
cs	mBERT-seen	83.6	84.3	84.4	84.8	84.3	84.9	86.8	72.3/58.6	73.4/60.1	74.8/61.7	74.7/60.1	71.5/58.3	75.2/61.9	60.3/48.1
cu	mBERT-genus	34.1	33.8	35.4	37.1	34.7	30.3	45.0	31.9/12.9	30.4/11.2	32.3/13.9	32.6/14.3	27.3/9.4	28.6/12.2	31.5/15.6
cy	mBERT-seen	64.9	64.7	64.4	64.7	59.6	60.7	66.4	63.9/45.9	64.6/45.8	64.8/45.3	65.6/45.5	57.7/33.1	62.3/40.1	46.1/33.0
da	mBERT-seen	88.9	89.0	89.2	89.2	86.3	88.7	90.1	74.3/66.0	74.6/66.4	75.8/67.7	76.3/67.9	70.9/61.7	77.1/68.7	66.1/56.9
de	mBERT-seen	84.8	85.8	85.7	86.1	86.5	85.7	87.6	71.2/61.8	75.4/66.9	76.7/68.3	76.8/68.6	75.3/66.7	77.4/69.1	62.3/53.7

language		Part-of-speech tagging							Dependency parsing						
code	group	G	LS	en	TA	X	mB	R	G	LS	en	TA	X	mB	R
el	mBERT-seen	81.5	81.6	81.4	81.5	83.2	82.8	86.4	78.0/65.4	77.2/64.9	78.1/65.4	79.0/64.8	74.7/62.4	82.9/70.5	57.1/47.5
en	mBERT-seen	96.3	96.3	96.3	96.3	96.4	96.7	97.3	89.6/87.0	89.4/86.8	89.6/87.0	89.8/87.0	89.7/87.1	91.8/89.4	59.8/53.3
es	mBERT-seen	87.1	87.7	87.9	88.1	88.2	87.5	89.0	73.6/61.9	76.0/64.6	76.9/65.9	77.3/66.0	74.7/63.9	77.8/67.2	72.6/62.0
et	mBERT-seen	83.4	82.9	83.1	83.3	86.4	81.4	87.8	64.1/46.6	62.7/45.2	64.8/47.1	64.9/46.3	65.0/49.0	64.0/44.8	63.1/45.4
eu	mBERT-seen	69.8	69.0	69.0	68.9	73.4	67.4	71.1	52.6/33.4	51.3/31.7	52.7/33.4	54.0/33.8	53.8/35.3	51.2/31.6	41.8/24.6
fa	mBERT-seen	73.4	73.5	68.5	69.3	69.4	66.9	76.3	47.3/34.8	46.8/33.3	43.7/31.7	44.2/31.6	42.9/31.1	42.5/29.9	31.7/22.0
fi	mBERT-seen	83.8	83.7	83.9	84.2	71.6	82.2	88.2	66.4/50.9	65.1/49.6	66.5/51.1	66.4/50.2	51.1/32.7	68.0/51.1	61.4/45.9
fo	mBERT-genus	71.0	71.7	72.7	73.2	64.4	68.7	72.7	51.2/36.3	50.8/35.6	52.4/37.5	52.3/38.1	43.4/26.1	49.6/34.6	48.2/33.4
fr	mBERT-seen	87.0	87.0	87.0	87.7	88.1	88.4	89.1	79.1/71.0	78.8/70.6	79.2/71.1	79.1/70.6	78.5/71.2	78.5/71.8	73.1/65.6
fro	mBERT-genus	57.9	57.1	60.0	60.4	57.0	55.0	43.9	58.3/32.2	57.6/31.0	62.3/37.3	61.9/35.7	55.5/30.0	57.3/30.8	45.2/22.3
ga	mBERT-seen	51.1	57.8	71.2	71.4	74.0	65.1	69.4	31.0/15.3	44.9/22.4	63.1/41.3	64.2/42.3	61.8/44.1	60.6/37.0	48.4/32.2
gd	mBERT-genus	44.4	44.4	47.1	47.1	51.4	41.8	58.0	38.4/14.1	38.6/13.5	40.5/16.2	40.5/16.5	43.3/19.8	38.0/14.8	44.6/24.7
gl	mBERT-seen	85.9	86.5	86.6	86.8	84.5	86.3	87.6	77.5/67.1	77.9/67.7	78.9/69.1	78.9/68.4	75.1/62.8	79.6/69.8	69.5/60.4
got	mBERT-genus	23.6	24.7	22.1	22.0	21.2	18.7	11.6	27.3/8.7	28.3/5.8	28.3/8.5	28.9/9.0	27.5/6.1	26.6/7.0	22.4/6.2
gsw	mBERT-genus	52.0	56.9	60.9	63.7	60.2	52.6	43.8	45.4/29.7	52.8/33.7	56.7/39.0	60.2/42.2	54.9/36.7	46.6/29.0	31.0/14.5
gun	MAD-G-genus	36.2	35.2	30.9	30.0	30.0	30.9	26.0	20.7/6.4	20.5/6.2	14.6/5.2	17.0/6.3	12.5/3.2	11.5/3.3	12.0/4.1
gv	mBERT-genus	32.7	31.4	33.1	36.0	35.1	32.4	26.9	32.8/8.4	31.3/6.3	31.0/7.4	30.8/7.2	37.7/11.9	28.7/6.1	22.6/4.0
he	mBERT-seen	79.3	78.8	79.3	79.7	77.7	77.1	81.9	66.3/48.8	65.8/48.4	66.3/48.7	68.0/50.0	61.6/42.9	68.3/51.7	52.5/38.2
hi	mBERT-seen	40.8	67.4	68.1	68.2	70.1	67.0	69.9	16.1/6.9	39.3/25.6	42.4/29.5	44.0/30.6	29.0/17.5	46.0/31.7	35.0/22.4
hr	mBERT-seen	84.6	83.9	84.4	84.3	83.9	84.7	86.7	76.3/63.4	75.4/63.1	77.4/65.0	76.9/62.7	74.5/61.4	79.4/67.1	69.9/58.1
hsb	mBERT-genus	69.1	70.8	71.8	72.2	69.2	69.9	71.9	46.4/33.2	49.8/35.4	53.3/39.3	53.2/38.5	50.3/35.5	51.4/37.6	44.0/29.4
hu	mBERT-seen	81.4	81.5	81.5	82.1	82.3	81.8	85.1	71.0/51.6	70.3/50.4	70.9/51.4	71.1/50.8	68.3/49.1	73.0/51.9	62.7/44.7
hy	mBERT-seen	77.1	77.1	76.9	77.4	79.3	75.1	86.0	55.7/36.5	55.3/35.5	56.3/36.8	58.2/37.4	54.9/35.9	58.2/37.5	56.1/37.1
id	mBERT-seen	85.9	85.8	85.7	86.2	87.2	84.3	87.2	70.1/59.0	68.0/57.4	69.6/58.9	70.9/59.3	67.9/58.1	66.8/56.9	55.4/45.2
is	mBERT-seen	76.0	77.3	78.4	78.8	77.6	76.0	84.3	53.4/36.6	55.1/38.5	56.8/40.5	57.2/40.5	56.7/39.9	57.5/40.7	54.3/39.7
it	mBERT-seen	90.8	90.9	91.5	91.8	90.9	90.3	91.9	81.5/73.3	81.4/72.8	82.9/75.5	83.2/75.1	77.8/69.2	84.4/77.5	72.9/64.5
ja	mBERT-seen	49.2	49.1	49.1	49.9	52.5	47.6	33.6	33.7/18.5	33.8/18.3	34.1/18.8	32.9/19.0	35.2/19.4	32.5/17.0	33.4/16.4
kfm	mBERT-genus	33.8	36.5	35.1	37.8	39.2	43.2	41.9	21.6/4.1	17.6/5.4	23.0/12.2	25.7/6.8	18.9/5.4	21.6/4.1	27.0/13.5
kk	mBERT-seen	77.4	77.2	76.9	76.8	70.9	75.9	81.1	59.3/40.0	58.4/38.3	59.2/40.0	60.4/40.8	48.4/27.2	59.5/37.4	43.2/25.9
kmr	mBERT-genus	37.6	38.4	42.0	42.0	46.9	38.3	70.0	23.7/6.5	25.3/5.7	26.8/7.6	27.9/8.5	25.2/8.8	24.5/7.3	40.5/25.2
ko	mBERT-seen	64.6	64.4	64.3	64.2	64.1	63.7	67.5	41.0/27.5	40.1/25.9	41.0/27.5	43.9/29.4	42.3/28.1	38.9/24.7	30.8/20.4
koi	MAD-G-genus	44.2	43.9	41.1	41.4	40.3	41.8	48.2	33.1/17.5	26.9/14.9	28.2/12.6	32.7/15.9	27.1/11.0	26.9/9.5	28.2/13.5
kpv	MAD-G-seen	54.8	55.2	34.0	33.4	56.3	34.5	40.8	39.3/19.1	38.1/18.3	23.6/8.6	24.5/8.9	42.1/21.5	22.8/7.4	26.0/10.7
krl	mBERT-genus	65.0	66.6	66.6	67.7	53.9	62.4	68.0	48.2/25.4	46.0/23.9	47.9/27.5	45.8/25.4	37.4/15.5	44.7/23.6	40.4/21.8
la	mBERT-seen	73.0	71.8	70.7	69.9	76.6	62.6	76.0	47.5/30.6	46.6/29.8	43.9/28.3	45.8/28.8	52.1/34.1	41.0/24.1	47.6/29.4
lt	mBERT-seen	75.1	77.3	80.7	81.1	78.9	78.1	85.8	56.3/37.3	59.6/40.4	64.3/45.9	63.8/45.2	59.6/40.7	62.9/43.4	56.2/39.4
lv	mBERT-seen	77.9	79.0	80.6	80.9	83.6	78.8	85.4	61.8/42.5	65.4/46.1	67.7/48.9	68.3/48.5	65.8/47.5	66.2/45.8	55.4/38.5
lzh	mBERT-genus	50.0	50.4	50.3	49.7	48.7	49.0	27.7	46.7/27.4	47.6/27.2	48.7/29.8	48.0/28.3	45.6/27.6	49.3/30.2	25.4/9.9
mdf	MAD-G-genus	47.2	48.5	46.7	48.9	46.4	47.1	46.2	34.0/17.6	34.9/17.8	32.2/17.4	34.2/17.6	31.8/13.7	33.9/14.3	28.2/12.6
mr	mBERT-seen	71.8	73.0	74.2	72.4	60.7	70.6	80.4	48.8/28.4	48.1/26.7	48.1/28.2	46.8/27.7	25.2/14.8	44.2/26.0	40.0/23.8
mt	MAD-G-seen	71.7	72.1	27.4	26.3	75.6	24.6	24.6	61.8/43.0	61.3/43.1	29.3/6.9	32.7/7.6	65.4/49.3	28.5/5.6	20.7/3.9
myu	unseen-genus	21.4	15.5	17.3	19.9	18.8	25.1	17.3	24.0/10.3	26.9/9.2	26.6/14.4	21.8/12.2	19.9/11.4	28.4/16.6	31.7/19.6
myv	MAD-G-seen	71.0	68.7	46.7	49.0	76.9	49.5	49.0	53.2/33.3	51.5/31.9	32.5/15.5	33.6/15.4	59.3/40.5	34.3/13.7	26.4/11.4
nl	mBERT-seen	87.7	88.3	88.8	89.0	89.0	88.4	89.1	74.1/64.6	77.4/69.4	78.4/70.9	78.5/70.9	77.4/69.9	77.7/70.5	63.8/55.9
no	mBERT-seen	89.9	90.4	90.7	90.9	90.9	90.5	92.1	79.6/73.4	79.9/73.7	80.8/74.9	81.0/74.9	81.3/75.1	82.3/75.8	65.7/57.5
nyg	mBERT-genus	33.3	29.5	39.7	37.2	29.5	38.5	41.0	29.5/11.5	24.4/9.0	25.6/11.5	25.6/10.3	24.4/14.1	26.9/10.3	41.0/17.9
olo	mBERT-genus	64.9	64.4	64.7	64.7	56.5	59.6	59.8	46.0/24.0	45.6/22.9	44.0/22.4	46.0/24.3	36.7/16.7	43.1/20.0	31.8/14.0
orv	mBERT-genus	80.9	80.8	80.6	80.3	80.8	78.8	84.6	57.3/41.4	57.1/41.3	57.8/42.0	57.5/40.9	54.4/38.8	57.6/41.7	55.0/41.0
pcm	unseen-genus	45.5	45.5	45.7	46.4	43.5	44.3	45.2	49.1/26.7	49.3/26.3	49.7/27.2	52.3/27.5	46.4/23.9	50.1/27.5	31.8/14.5
pl	mBERT-seen	76.1	80.9	83.4	83.2	83.0	81.3	84.9	62.1/46.3	69.4/54.4	76.4/62.1	75.9/61.0	71.6/57.0	76.7/62.5	63.1/51.1
pt	mBERT-seen	88.4	88.6	88.8	89.1	88.1	88.5	90.1	73.0/61.8	74.4/63.2	75.4/64.7	75.8/64.8	72.5/61.1	75.5/64.9	69.7/59.0
ro	mBERT-seen	81.7	82.8	83.5	83.5	81.0	82.6	86.3	70.8/56.0	71.5/56.0	74.5/59.7	75.2/59.4	67.5/51.7	75.9/60.7	68.1/54.5
ru	mBERT-seen	83.3	83.6	83.4	83.6	84.3	83.2	87.1	74.5/63.6	73.8/62.7	74.5/63.4	75.2/63.0	71.3/60.6	77.5/65.9	62.2/51.3
sa	mBERT-genus	36.4	41.5	44.2	43.1	43.4	41.7	59.0	25.9/12.2	32.9/9.7	34.7/12.5	37.9/14.4	25.0/7.4	30.1/9.9	34.3/15.4
sk	mBERT-seen	84.0	85.0	84.6	85.0	83.9	83.9	86.4	79.0/66.4	78.9/66.1	80.4/68.0	80.3/66.8	76.1/63.8	82.1/70.2	64.4/51.7
sl	mBERT-seen	81.2	82.7	83.1	83.1	77.3	82.8	85.6	75.3/61.2	75.9/62.1	78.0/64.9	78.5/63.9	65.7/49.5	78.3/65.2	70.2/57.6
sme	MAD-G-seen	71.1	68.5	41.6	42.1	75.8	39.0	33.3	48.6/32.7	46.2/29.3	24.3/9.0	23.9/8.6	50.4/33.5	22.9/6.5	20.6/7.0

language		Part-of-speech tagging							Dependency parsing						
code	**group**	G	LS	en	TA	X	mB	R	G	LS	en	TA	X	mB	R
sms	MAD-G-genus	34.6	35.7	31.2	31.3	36.6	29.6	36.2	25.7/11.5	22.3/8.9	22.0/8.9	23.7/8.0	23.7/8.4	21.5/7.4	29.7/10.7
soj	mBERT-genus	41.8	45.5	43.6	41.8	43.6	43.6	43.6	21.8/7.3	27.3/9.1	20.0/5.5	20.0/5.5	34.5/12.7	21.8/12.7	40.0/12.7
sq	mBERT-seen	77.8	78.9	78.6	78.3	71.6	74.7	81.1	84.8/66.3	82.6/64.4	83.6/64.8	86.9/66.2	71.8/50.4	86.2/68.5	65.3/47.5
sr	mBERT-seen	84.9	84.5	84.7	84.1	84.5	85.2	86.9	77.8/66.1	76.4/64.8	78.1/67.0	78.1/64.7	75.8/63.4	80.7/68.7	71.3/60.0
sv	mBERT-seen	90.3	90.6	90.3	90.6	90.4	90.2	92.6	80.8/74.6	80.4/74.0	80.9/74.6	81.1/74.7	81.3/74.9	82.8/76.3	70.9/63.0
ta	mBERT-seen	65.4	64.5	65.5	64.7	54.1	64.9	67.9	37.9/18.4	38.3/17.8	38.2/18.4	41.1/20.2	16.9/5.1	43.2/17.5	43.3/21.4
te	mBERT-seen	75.6	75.7	76.0	75.7	67.0	76.0	85.4	70.3/51.6	64.1/46.6	70.9/53.8	73.0/53.4	43.0/29.8	59.5/42.4	53.3/34.5
th	mBERT-seen	48.7	48.5	48.6	50.0	47.9	46.4	55.1	42.4/21.1	43.4/21.4	43.7/22.3	43.5/22.9	41.7/19.2	39.9/21.7	45.8/32.7
tl	mBERT-seen	70.7	69.5	68.7	69.6	62.3	64.7	71.1	81.6/51.0	77.7/48.6	75.9/51.5	75.1/54.1	64.6/37.3	71.7/42.1	44.7/26.0
tr	mBERT-seen	74.6	74.3	74.4	74.6	78.8	70.7	80.9	64.7/43.9	63.1/40.9	64.9/43.9	67.2/45.3	62.5/42.1	60.6/37.5	43.9/28.1
ug	MAD-G-seen	58.0	60.4	35.1	34.4	57.9	28.9	73.5	33.3/17.4	29.7/13.6	16.5/6.5	21.1/7.9	36.0/16.2	17.1/7.1	50.3/30.2
uk	mBERT-seen	82.2	83.1	83.4	82.7	83.8	83.5	85.8	73.0/60.6	72.6/60.3	73.8/61.6	73.1/59.9	69.9/57.4	75.7/63.0	66.7/54.4
ur	mBERT-seen	49.9	61.2	62.2	63.5	58.7	60.4	65.6	20.6/10.1	35.7/21.4	36.7/22.7	36.7/22.6	21.3/10.7	35.2/21.9	37.9/24.1
vi	mBERT-seen	63.5	63.1	63.6	62.9	63.9	60.3	63.2	55.8/39.0	54.9/37.7	55.9/38.9	55.7/38.6	54.9/36.6	53.5/37.3	30.1/18.7
wbp	unseen-genus	25.8	27.4	32.8	32.2	33.1	37.9	22.6	24.2/8.9	26.8/10.8	32.5/13.7	30.9/14.3	15.9/4.1	47.1/17.2	44.6/19.7
wo	unseen-genus	30.3	32.2	36.8	38.0	34.1	35.2	27.1	28.1/6.3	31.5/6.5	31.8/8.7	32.7/8.9	32.9/8.8	28.4/6.3	19.9/4.5
yo	mBERT-seen	64.2	63.3	60.3	59.4	56.3	47.7	26.6	46.4/28.0	45.3/26.6	40.9/23.5	41.7/24.0	37.0/19.8	37.2/19.0	11.9/2.4
yue	mBERT-genus	62.1	62.5	61.8	63.3	62.4	63.3	53.0	45.4/27.5	44.8/27.4	45.1/27.9	46.2/27.9	45.4/28.3	45.2/28.4	32.0/18.8
zh	mBERT-seen	70.9	70.9	70.6	68.9	69.8	67.4	48.3	56.9/35.4	56.3/34.7	57.1/35.5	56.9/35.5	55.8/34.9	59.4/38.0	47.9/26.5

A.2.2 Multi-source Transfer

Table A.4 Full per-language results for multi-source zero-shot cross-lingual transfer experiments with 20 languages. POS tagging results are given as accuracy scores, dependency parsing results are unlabelled/labelled attachment scores. G = MAD-G, LS = MAD-G-LS, en = MAD-G-en, TA = TA-only, X = MAD-X, mB = mBERT-ft, R = XLM-R-ft.

language		Part-of-speech tagging							Dependency parsing						
code	**group**	G	LS	en	TA	X	mB	R	G	LS	en	TA	X	mB	R
af	mBERT-seen	85.0	86.9	87.4	88.2	83.2	88.9	89.6	66.8/54.1	68.8/55.9	69.4/57.3	69.2/57.3	66.0/53.2	71.8/59.4	67.8/55.1
ajp	mBERT-genus	63.9	66.2	66.3	64.1	65.0	64.4	73.5	58.3/41.7	53.6/34.9	55.6/39.2	54.5/36.4	55.8/38.3	56.7/39.0	34.6/21.9
akk	mBERT-genus	41.8	41.2	37.9	42.7	2.9	46.1	46.4	30.8/8.1	32.0/9.5	29.8/7.1	30.2/8.1	28.8/6.1	31.6/10.1	26.5/9.8
apu	unseen-genus	37.1	44.5	41.7	45.4	34.5	45.5	50.4	21.0/17.2	27.1/12.1	23.8/14.4	24.9/13.5	19.8/10.2	24.5/9.1	26.3/11.0
aqz	unseen-genus	30.0	27.5	20.0	30.0	22.5	22.5	32.5	35.0/15.0	27.5/10.0	23.8/10.0	25.0/5.0	33.8/12.5	30.0/8.8	27.5/16.2
ar	mBERT-seen	80.1	79.9	80.2	80.1	80.1	80.3	80.6	76.2/66.1	76.4/66.4	76.7/66.7	76.7/66.5	76.4/66.7	76.8/66.7	55.3/46.2
bam	unseen-genus	31.6	31.8	33.0	33.3	29.4	29.8	30.5	31.7/8.3	31.8/7.2	32.8/8.8	32.0/8.2	28.2/6.6	30.1/7.3	23.7/5.5
be	mBERT-seen	88.8	89.2	89.4	89.4	88.7	90.8	92.1	78.2/71.3	78.9/72.2	79.4/72.5	78.9/72.3	79.0/71.7	82.5/74.6	76.4/67.8
bg	mBERT-seen	93.5	94.1	93.9	93.6	91.3	93.2	95.3	85.2/75.3	85.4/75.4	85.2/75.3	85.9/75.7	85.6/75.4	87.6/78.5	70.9/61.1
bho	mBERT-genus	59.3	61.4	61.3	61.5	61.6	61.8	63.3	44.5/27.5	48.9/33.7	44.4/28.1	48.6/32.7	46.9/31.9	51.9/35.6	32.0/21.1
br	mBERT-seen	72.0	72.1	74.9	75.2	64.8	70.2	68.8	71.7/52.7	71.3/53.1	75.1/58.8	76.0/58.4	64.3/43.9	73.8/53.9	54.1/36.0
bxr	MAD-G-seen	73.2	72.0	63.7	63.9	74.3	62.2	67.2	51.7/32.1	52.3/31.5	47.0/25.4	47.4/26.0	54.3/34.0	49.4/25.2	41.1/22.3
ca	mBERT-seen	90.1	90.0	89.7	89.4	87.6	89.9	89.9	81.0/71.1	81.3/71.7	81.4/71.2	81.6/71.4	78.3/68.0	85.5/74.7	81.2/69.9
ckt	unseen-genus	34.5	33.7	25.4	28.3	32.0	26.2	34.4	25.8/16.5	28.8/15.7	21.4/12.8	28.0/16.3	29.3/18.0	23.5/11.6	33.3/18.1
cs	mBERT-seen	95.4	95.6	93.8	95.8	96.1	96.5	97.5	83.9/79.1	84.5/79.8	83.7/78.6	84.7/80.0	85.5/80.9	88.4/84.1	70.9/64.9
cu	mBERT-genus	36.1	36.0	37.3	37.8	36.3	37.3	51.2	33.7/16.0	34.3/15.9	33.6/16.6	38.7/19.6	33.1/16.3	34.4/16.0	44.2/24.0
cy	mBERT-seen	68.8	69.3	68.4	70.3	66.2	69.7	73.4	69.4/51.9	70.6/53.7	69.3/51.4	69.6/51.1	65.8/41.9	72.2/50.3	57.5/42.4
da	mBERT-seen	90.3	90.1	90.5	90.8	86.8	91.2	92.9	72.7/65.3	72.8/65.6	73.3/66.1	73.4/66.3	70.9/62.5	77.2/68.6	67.4/58.4
de	mBERT-seen	87.2	87.6	87.4	87.1	87.3	88.8	89.6	77.7/71.3	81.1/74.7	81.3/75.2	80.8/74.9	81.4/75.2	85.2/78.9	71.9/63.6
el	mBERT-seen	96.4	96.5	96.4	96.6	97.0	97.6	98.2	89.4/86.3	89.6/86.7	89.3/86.3	89.6/86.7	90.3/87.5	93.3/90.7	63.7/59.4
en	mBERT-seen	92.2	92.3	92.2	92.4	92.3	93.5	94.7	82.6/77.5	82.5/77.4	82.6/77.5	82.4/77.3	82.9/78.0	87.1/82.5	63.5/55.8
es	mBERT-seen	91.7	91.8	91.9	91.7	85.5	92.3	92.3	79.7/71.0	81.4/73.2	81.6/73.4	81.9/73.4	82.2/73.0	85.4/76.4	78.8/69.8
et	mBERT-seen	91.9	91.7	91.7	91.7	93.7	92.9	95.6	76.8/69.4	76.7/69.0	76.7/69.0	76.4/68.6	79.4/72.8	80.6/73.6	74.4/66.9
eu	mBERT-seen	87.9	87.8	87.7	88.0	89.9	91.2	92.8	72.8/65.4	72.7/65.2	71.9/64.4	72.9/65.6	75.1/68.5	78.0/71.4	59.0/50.5
fa	mBERT-seen	90.2	90.6	84.0	90.1	91.4	92.6	96.0	81.0/74.9	80.6/74.5	65.1/58.7	80.0/73.8	81.7/75.9	85.6/80.0	51.8/43.3
fi	mBERT-seen	87.2	87.1	87.2	86.8	74.6	86.3	91.3	74.1/65.0	74.2/65.1	74.1/64.9	74.2/64.6	60.3/47.5	77.5/68.6	64.5/56.0
fo	mBERT-genus	73.0	73.5	73.5	74.5	68.5	72.1	71.7	54.1/39.7	53.9/39.6	54.9/40.7	54.5/40.6	47.0/30.9	52.2/36.8	48.7/34.0

language		Part-of-speech tagging							Dependency parsing						
code	**group**	G	LS	en	TA	X	mB	R	G	LS	en	TA	X	mB	R
fr	mBERT-seen	96.5	96.4	96.5	96.3	96.8	97.2	97.7	87.3/83.6	87.1/83.7	87.3/83.6	87.4/83.9	87.6/83.8	91.9/88.8	84.3/79.4
fro	mBERT-genus	63.3	64.8	66.8	66.7	62.0	66.0	64.8	60.5/40.4	60.6/40.9	62.2/43.2	61.4/42.1	57.8/37.4	62.4/42.2	50.7/31.6
ga	mBERT-seen	82.5	84.6	76.1	87.7	92.3	92.3	93.7	70.4/57.5	73.9/61.5	70.8/52.3	76.5/65.5	79.7/71.2	83.3/73.6	67.4/59.0
gd	mBERT-genus	49.5	54.0	49.0	55.7	59.9	57.6	76.6	47.2/22.7	49.1/25.9	48.1/23.7	48.4/25.6	52.0/30.0	49.8/26.3	59.9/41.2
gl	mBERT-seen	91.5	91.8	91.9	91.8	87.4	91.7	92.7	80.1/72.8	80.8/73.6	81.2/74.1	80.8/73.7	78.6/68.8	83.5/76.3	73.8/66.0
got	mBERT-genus	34.4	34.6	38.0	37.7	42.1	34.6	34.8	29.0/13.4	34.2/13.3	32.2/13.2	31.2/11.3	31.9/12.6	34.0/12.7	27.5/9.6
gsw	mBERT-genus	64.5	65.7	70.0	68.2	65.4	62.6	52.7	54.5/39.1	63.0/44.6	63.2/46.3	64.2/47.5	62.5/46.8	56.0/38.2	38.9/23.4
gun	MAD-G-genus	41.4	40.7	37.8	38.4	31.5	39.8	34.5	30.4/10.5	31.3/10.7	26.6/9.0	27.2/9.0	25.8/7.4	29.2/9.2	23.8/8.6
gv	mBERT-genus	42.2	42.4	42.0	45.8	46.3	45.2	45.2	40.6/14.9	39.8/15.1	38.7/13.2	39.8/14.5	44.2/19.6	41.6/15.4	35.5/11.1
he	mBERT-seen	77.3	80.3	77.7	81.1	70.5	79.0	85.5	67.1/53.3	68.0/54.4	66.9/53.3	68.3/54.4	62.5/46.2	73.1/58.6	59.9/47.0
hi	mBERT-seen	86.9	89.3	81.9	89.9	91.4	92.0	94.6	74.9/66.3	81.0/72.8	66.8/53.0	81.6/74.2	80.4/72.9	88.2/80.6	42.3/34.2
hr	mBERT-seen	92.4	92.7	91.9	93.0	93.8	93.6	94.1	83.6/75.9	83.2/75.8	83.6/76.3	83.5/76.1	84.1/76.2	87.3/80.0	79.6/71.2
hsb	mBERT-genus	77.7	78.2	78.7	79.1	77.7	78.4	79.9	56.5/47.7	57.8/49.4	60.1/51.1	59.6/51.5	59.3/50.6	61.3/51.9	58.3/48.5
hu	mBERT-seen	93.8	93.8	93.8	93.8	94.1	95.9	97.0	82.6/76.4	82.5/76.3	82.5/76.2	81.7/75.5	83.8/77.4	88.4/82.4	69.4/61.8
hy	mBERT-seen	90.9	90.7	90.8	91.1	92.2	93.6	95.7	77.5/68.6	78.0/69.4	77.2/68.3	76.9/67.8	79.4/71.4	83.4/75.3	73.2/65.1
id	mBERT-seen	88.8	88.7	88.8	88.8	89.1	88.5	89.3	81.8/62.5	81.9/62.8	81.9/62.8	81.8/62.6	82.7/63.8	82.4/63.4	67.7/49.6
is	mBERT-seen	78.8	80.3	80.8	81.2	79.2	79.0	84.5	57.1/41.9	58.3/43.4	59.3/44.3	58.4/43.5	58.9/44.0	58.3/42.5	54.9/40.2
it	mBERT-seen	94.1	94.1	94.7	94.6	92.0	94.7	94.8	83.7/78.4	83.4/77.9	83.9/78.7	84.1/78.8	81.5/75.4	87.2/82.1	77.3/70.3
ja	mBERT-seen	92.5	92.5	92.4	92.6	93.1	95.8	96.6	81.9/77.8	82.2/78.0	81.7/77.5	81.1/77.1	82.7/78.3	91.0/87.7	83.0/78.5
kfm	mBERT-genus	43.2	43.2	40.5	41.9	41.9	51.4	41.9	40.5/20.3	37.8/21.6	40.5/18.9	37.8/18.9	29.7/14.9	28.4/14.9	17.6/9.5
kk	mBERT-seen	82.6	82.7	82.4	82.7	73.7	82.9	86.6	67.5/55.4	68.3/55.7	67.3/55.1	68.1/56.4	61.9/47.9	70.9/57.2	49.9/38.5
kmr	mBERT-genus	47.7	46.2	47.1	47.1	52.1	45.4	79.9	27.3/8.8	29.3/9.5	29.2/11.3	29.9/11.1	34.2/14.7	28.5/9.9	55.9/38.6
ko	mBERT-seen	87.4	87.6	87.1	87.3	88.6	93.8	95.1	74.5/68.5	74.7/68.4	74.2/68.1	74.4/68.2	75.6/69.4	84.7/79.3	58.7/51.5
koi	MAD-G-genus	48.2	48.4	45.3	47.1	44.5	47.7	52.3	36.1/20.6	33.8/18.5	29.7/14.9	37.7/20.7	31.7/15.9	32.1/16.4	34.0/19.2
kpv	MAD-G-seen	57.6	56.5	37.4	38.1	61.6	36.3	43.0	45.0/26.6	44.9/26.4	25.6/10.6	28.7/12.5	48.5/32.5	27.1/10.5	29.4/14.6
krl	mBERT-genus	69.9	72.5	72.4	72.9	56.6	70.3	74.9	56.2/37.0	57.5/39.7	55.5/41.5	54.4/40.2	44.3/27.5	55.6/39.3	53.4/38.7
la	mBERT-seen	95.4	94.7	93.6	94.9	96.1	97.5	98.1	74.3/70.1	74.5/70.3	72.1/67.0	74.1/69.5	76.6/72.6	84.1/80.8	79.4/74.5
lt	mBERT-seen	83.3	84.7	85.7	86.2	82.3	84.9	90.5	69.9/56.7	72.1/59.8	73.1/61.8	73.3/60.5	72.7/59.8	74.4/60.6	64.5/52.7
lv	mBERT-seen	89.0	89.4	88.1	89.8	92.3	91.8	94.6	77.0/69.6	78.1/70.8	77.5/68.6	78.5/71.1	81.7/75.3	82.0/75.4	63.4/55.6
lzh	mBERT-genus	56.1	59.8	57.1	57.5	53.3	57.8	57.4	50.8/31.4	52.5/33.5	52.5/33.0	51.7/32.5	49.0/29.5	52.9/33.2	33.1/18.3
mdf	MAD-G-genus	52.6	54.4	52.2	51.9	47.0	50.3	50.4	38.8/21.5	38.7/22.9	37.5/22.5	37.3/22.1	35.3/22.3	41.7/22.5	29.0/16.2
mr	mBERT-seen	85.9	83.4	81.6	84.0	68.7	81.0	86.5	59.5/41.0	59.0/44.7	57.5/43.2	61.9/42.5	45.6/27.4	59.5/42.2	38.6/28.2
mt	MAD-G-seen	80.2	78.8	35.4	37.1	80.4	35.7	35.9	68.6/54.4	68.1/54.0	37.1/10.8	39.0/12.1	73.1/60.4	37.4/9.8	35.9/8.0
myu	unseen-genus	26.6	28.8	29.9	27.7	21.0	22.9	35.4	24.4/8.1	29.5/11.8	30.3/15.5	31.7/12.9	25.8/10.0	29.2/14.0	37.6/19.6
myv	MAD-G-seen	73.2	71.2	51.8	51.7	78.5	51.3	50.9	63.2/46.8	62.1/43.7	35.9/19.0	36.3/19.1	67.3/52.0	40.0/19.7	29.0/15.6
nl	mBERT-seen	87.9	88.4	88.8	89.0	88.3	89.2	89.3	77.8/69.8	79.7/72.7	81.0/74.3	80.1/73.6	80.5/73.3	84.5/77.4	70.2/62.3
no	mBERT-seen	88.1	88.4	88.0	88.7	89.7	89.5	92.2	78.9/72.9	80.0/73.9	80.4/74.4	79.8/73.6	80.6/75.3	83.9/77.1	67.9/58.9
nyg	mBERT-genus	42.3	41.0	47.4	46.2	19.2	64.1	52.6	33.3/20.5	30.8/19.2	32.1/20.5	34.6/20.5	24.4/15.4	35.9/26.9	25.6/16.7
olo	mBERT-genus	70.2	72.8	70.5	71.2	58.9	68.6	71.8	57.7/39.5	57.9/41.0	54.4/37.5	54.2/37.9	43.4/27.1	57.1/39.5	47.7/31.7
orv	mBERT-genus	87.4	87.5	87.2	87.7	87.1	87.1	91.3	65.1/53.0	65.0/52.9	64.8/53.0	65.2/53.0	64.9/52.4	68.4/56.0	66.8/55.1
pcm	unseen-genus	46.3	45.8	45.9	45.7	42.4	45.2	45.7	48.8/25.9	49.3/26.5	49.8/26.5	50.4/26.7	45.0/20.6	50.3/26.3	35.9/16.5
pl	mBERT-seen	86.3	88.3	89.2	89.8	90.2	90.7	92.5	75.5/64.4	79.8/68.3	83.2/73.5	83.5/73.2	84.2/73.7	87.3/77.0	72.2/61.6
pt	mBERT-seen	90.4	90.9	90.4	90.3	88.7	90.6	91.0	79.9/70.7	80.8/71.8	80.9/71.9	81.2/72.0	79.8/69.6	83.9/74.5	76.2/66.5
ro	mBERT-seen	87.0	88.5	88.2	88.8	84.9	89.6	91.6	79.6/67.0	80.8/67.8	80.8/68.6	81.4/69.3	77.6/64.2	83.5/70.5	77.3/64.4
ru	mBERT-seen	88.6	88.9	88.7	89.2	84.8	90.3	92.6	82.6/75.2	82.5/74.9	82.8/75.3	82.4/75.0	80.9/73.2	87.6/80.3	72.4/64.3
sa	mBERT-genus	49.6	48.7	50.6	49.9	45.1	44.1	63.0	28.4/18.3	42.4/19.6	39.8/22.0	43.8/19.6	42.3/17.1	45.6/19.6	30.9/17.5
sk	mBERT-seen	92.9	93.3	92.1	94.1	94.5	94.4	95.3	87.7/82.4	88.4/83.6	87.7/82.5	88.6/84.1	88.9/84.5	90.8/86.4	72.9/66.3
sl	mBERT-seen	89.1	90.1	90.1	90.6	83.0	90.6	93.1	84.1/75.4	84.2/75.8	84.8/76.5	85.5/77.0	78.7/66.4	87.6/79.1	81.1/71.2
sme	MAD-G-seen	73.8	72.8	48.1	48.5	79.2	47.9	45.7	54.8/40.4	53.1/38.4	28.6/13.3	28.5/12.0	57.5/45.7	28.6/12.5	26.3/11.1
sms	MAD-G-genus	37.4	41.8	34.9	36.3	46.8	36.5	45.8	29.4/13.6	28.1/12.6	24.7/10.8	30.7/13.5	30.0/16.0	26.8/11.1	32.8/15.6
soj	mBERT-genus	52.7	45.5	47.3	47.3	52.7	56.4	43.6	27.3/12.7	34.5/20.0	38.2/23.6	30.9/18.2	50.9/34.5	29.1/18.2	18.2/14.5
sq	mBERT-seen	82.5	83.6	81.8	82.2	73.0	82.2	87.2	87.4/72.7	86.4/71.6	86.9/72.0	88.7/74.7	77.4/59.1	89.9/76.5	70.3/52.8
sr	mBERT-seen	92.9	93.3	93.0	93.6	95.1	94.9	94.1	84.5/77.1	84.0/76.6	84.4/77.5	84.2/76.9	85.2/77.4	87.8/79.7	81.1/72.3
sv	mBERT-seen	91.5	91.6	91.5	91.3	91.9	92.0	95.1	79.1/72.5	78.6/72.3	79.0/72.4	78.7/72.2	79.5/73.2	81.7/75.0	71.8/63.6
ta	mBERT-seen	64.4	64.9	63.7	66.2	38.9	66.3	74.2	55.8/39.6	57.0/39.1	55.9/38.9	56.4/38.4	36.8/20.0	61.6/41.2	56.3/39.3
te	mBERT-seen	80.9	81.8	81.7	82.0	59.1	81.6	86.0	82.2/66.9	82.8/67.1	82.5/67.8	83.8/66.2	63.7/48.0	82.9/67.8	59.9/45.2
th	mBERT-seen	51.4	50.4	50.6	51.4	38.0	55.5	71.9	52.6/27.8	53.0/26.3	53.1/28.3	52.7/26.7	50.4/26.0	56.3/29.1	64.6/43.1

language		Part-of-speech tagging							Dependency parsing						
code	**group**	G	LS	en	TA	X	mB	R	G	LS	en	TA	X	mB	R
tl	mBERT-seen	73.8	73.6	74.1	74.4	67.0	67.0	76.0	81.1/54.1	80.7/54.2	75.6/51.2	78.2/53.7	68.8/41.8	80.9/54.1	47.4/29.7
tr	mBERT-seen	83.6	84.1	83.6	83.6	86.2	83.6	88.1	76.1/64.7	76.6/65.0	76.1/64.4	76.3/64.1	77.7/67.4	78.5/66.2	48.3/37.2
ug	MAD-G-seen	67.8	68.8	38.5	53.1	68.4	39.2	80.5	43.1/27.7	42.6/27.4	24.5/11.9	34.4/20.3	48.2/32.9	30.7/16.0	59.3/44.7
uk	mBERT-seen	89.8	90.6	89.9	90.9	91.9	92.2	93.2	81.2/73.5	81.7/74.3	81.6/74.0	81.5/74.1	82.2/74.9	86.3/79.2	77.3/68.9
ur	mBERT-seen	74.0	80.7	76.4	83.7	77.9	83.3	89.5	41.6/29.9	62.7/51.8	54.5/40.6	65.8/54.3	61.1/50.3	74.2/62.4	52.3/43.1
vi	mBERT-seen	86.9	87.3	86.9	87.4	88.8	90.0	92.8	68.2/58.7	68.4/58.9	68.1/58.8	68.3/58.8	68.8/59.5	72.7/63.4	35.0/26.2
wbp	unseen-genus	38.2	38.2	44.3	39.2	40.1	36.9	47.1	21.3/8.6	25.2/10.2	15.6/6.4	21.3/8.3	14.3/6.7	21.7/7.6	14.3/7.6
wo	unseen-genus	40.6	39.4	42.6	41.9	41.4	39.8	38.1	37.0/11.8	39.5/12.5	37.2/13.4	37.5/12.7	36.5/11.1	38.5/12.2	31.6/9.5
yo	mBERT-seen	69.3	65.4	60.4	61.2	56.2	53.9	29.2	51.9/33.8	52.4/32.4	48.7/29.5	48.1/28.9	44.5/24.1	45.6/23.5	20.6/5.4
yue	mBERT-genus	73.2	73.0	72.2	69.7	72.0	74.7	81.7	47.7/31.4	48.1/31.8	47.5/31.0	47.0/30.3	49.0/31.9	50.5/33.7	42.5/26.3
zh	mBERT-seen	91.0	91.0	90.9	90.9	91.5	94.7	95.3	74.2/68.6	74.4/68.6	74.1/68.2	73.8/68.4	74.8/69.1	83.6/79.0	73.8/67.4

A.2.3 Fine-tuning MAD-G-Initialised Adapters

Table A.5 POS tagging accuracy scores on `unseen` languages when MAD-G-initialised (`MAD-G-ft`) or randomly initialised (`rand-ft`) language adapters are fine-tuned by MLM-ing on varying amounts of unlabelled text, specifically 1,000, 3,000, 10,000, 30,000 or 100,000 tokens.

	1,000		3,000		10,000		30,000		100,000	
language	MAD-G-ft	rand-ft	MAD-G-ft	rand-ft	MAD-G-ft	rand-ft	MAD-G-ft	rand-ft	MAD-G-ft	rand-ft
bam	31.9	31.7	31.8	27.9	31.4	30.8	31.7	30.8	32.7	31.8
bho	63.2	62.0	65.3	62.8	67.0	66.5	68.1	68.4	-	-
cu	36.3	40.0	41.3	37.2	42.3	41.2	44.5	43.5	-	-
fo	75.3	75.0	79.7	78.0	81.7	81.0	84.5	83.3	86.6	86.4
gd	54.3	56.0	57.5	54.9	60.6	58.1	64.5	64.1	67.3	67.9
got	32.3	33.7	34.9	36.1	33.8	33.0	-	-	-	-
gv	50.4	45.6	52.0	47.2	61.3	58.7	68.8	65.8	74.1	74.2
hsb	79.5	80.1	81.3	81.9	86.0	85.8	87.9	87.6	89.7	88.8
koi	53.0	51.6	56.4	52.4	59.5	54.1	60.9	56.7	-	-
mdf	55.8	53.3	60.9	57.9	66.1	61.2	-	-	-	-
olo	71.8	71.8	74.9	74.9	77.8	78.7	80.2	79.9	82.5	83.1
sa	55.9	53.1	56.1	57.7	57.9	58.9	62.6	61.3	65.3	66.8
wo	43.4	47.4	45.4	48.4	55.0	56.1	62.6	60.3	69.8	69.8
yue	73.7	71.2	72.8	72.2	71.8	72.2	71.6	70.5	73.7	72.5

Table A.6 Dependency parsing unlabelled/labelled attachment scores on `unseen` languages when MAD-G-initialised (`MAD-G-ft`) or randomly initialised (`rand-ft`) language adapters are fine-tuned by MLMing on varying amounts of unlabelled text, specifically 1,000, 3,000, 10,000, 30,000 or 100,000 tokens.

	1,000		3,000		10,000		30,000		100,000	
language	`MAD-G-ft`	`rand-ft`	`MAD-G-ft`	`rand-ft`	`MAD-G-ft`	`rand-ft`	`MAD-G-ft`	`rand-ft`	`MAD-G-ft`	`rand-ft`
bam	32.1/8.7	29.1/7.8	31.3/8.2	29.0/4.8	31.4/8.4	29.7/7.8	31.1/9.0	29.3/7.8	31.0/9.3	28.9/5.5
bho	44.8/27.6	38.6/24.1	43.7/27.4	41.1/24.9	42.7/27.6	42.1/25.3	44.4/28.0	41.0/23.3	-/-	-/-
cu	34.0/16.9	35.6/18.8	34.8/18.2	35.5/19.2	35.9/18.7	35.7/18.6	37.8/20.0	36.9/19.2	-/-	-/-
fo	55.9/41.8	54.4/39.8	58.6/45.1	54.6/40.5	60.3/47.4	58.2/45.2	61.9/49.0	57.8/45.4	62.8/50.9	56.7/44.7
gd	50.5/25.9	45.2/22.4	51.7/27.4	48.8/24.5	55.0/31.9	52.2/28.2	59.8/37.0	53.3/29.4	61.0/40.8	53.7/32.3
got	29.7/13.2	23.8/14.1	29.6/13.7	27.0/7.4	29.5/14.0	27.5/6.9	-/-	-/-	-/-	-/-
gv	42.7/19.8	36.8/13.3	44.6/22.3	38.0/16.5	51.4/31.6	45.0/25.4	53.2/36.7	47.1/30.4	57.1/41.9	50.5/35.0
hsb	61.4/51.2	60.2/49.8	66.2/55.5	63.6/53.3	71.3/61.1	64.3/54.4	73.8/64.4	69.6/60.6	75.7/67.2	71.3/62.8
koi	41.7/25.5	34.1/19.2	40.6/25.0	33.6/19.3	43.0/28.1	37.3/20.4	43.5/29.2	37.1/24.4	-/-	-/-
mdf	41.2/25.0	33.3/23.2	46.4/30.2	42.1/26.8	50.7/36.1	48.2/32.4	-/-	-/-	-/-	-/-
olo	61.7/43.9	56.9/40.9	63.4/46.1	61.6/43.8	66.8/50.9	60.1/43.4	68.1/54.7	65.5/51.4	69.8/56.5	64.3/50.5
sa	37.5/20.8	40.8/24.4	41.9/23.2	43.7/24.7	42.9/25.0	46.6/27.1	47.6/29.9	48.3/29.1	48.0/30.3	48.9/31.9
wo	37.6/12.5	34.8/13.3	40.4/14.5	39.3/16.2	44.3/19.1	42.8/19.6	49.9/24.9	51.8/25.4	55.0/31.9	53.0/29.5
yue	48.2/31.9	43.4/28.0	47.9/31.6	44.3/28.3	47.2/30.9	43.8/28.1	46.4/31.6	44.6/29.2	47.2/31.8	45.7/30.4

Appendix B

Composable Sparse Fine-Tuning for Cross-Lingual Transfer

B.1 Languages

Table B.1 Details of the languages and data used for training and evaluation of SFTs and bottleneck adapters. The corpora of Bustamante et al. (2020) are available at https://github.com/iapucp/multilingual-data-peru; all other NLI corpora mentioned are available at https://github.com/AmericasNLP/americasnlp2021. * denotes source languages for multi-source DP training; † denotes source languages for multi-source NLI training; ‡ denotes source languages for multi-source QA training. English is the source language in all single-source task training experiments.

Task	Language	ISO Code	Family	UD Treebank	Corpus source(s)
Source	Arabic†,‡	ar	Afro-Asiatic, Semitic		Wikipedia
	Basque*	eu	Basque	BDT	
	Bulgarian†	bg	Indo-European, Slavic		
	Chinese†,‡	zh	Sino-Tibetan		
	Czech*	cs	Indo-European, Slavic	PDT	
	English*,†,‡,	en	Indo-European, Germanic	EWT	
	Estonian*	et	Uralic, Finnic	EDT	
	French*,†	fr	Indo-European, Romance	GSD	
	German†,‡	de	Indo-European, Germanic		
	Greek*,†	el	Indo-European, Greek	GDT	
	Hindi*,†,‡	hi	Indo-European, Indic	HDTB	
	Korean*	ko	Korean	GSD	
	Persian*	fa	Indo-European, Iranian	PerDT	
	Russian†	ru	Indo-European, Slavic		
	Spanish†,‡	es	Indo-European, Romance		

Task	Language	ISO Code	Family	UD Treebank	Corpus source(s)
	Swahili†	swa	Niger-Congo, Bantoid		
	Thai†	th	Tai-Kadai, Kam-Thai		
	Turkish*,†	tr	Turkic, Southwestern	BOUN	
	Urdu†	ur	Indo-European, Indic		
	Vietnamese*,‡	vi	Austro-Asiatic, Viet-Muong	VTB	
	Arabic	ar	Afro-Asiatic, Semitic	PUD	
	Bambara	bm	Mande	CRB	
	Buryat	bxr	Mongolic	BDT	
	Cantonese	yue	Sino-Tibetan	HK	
	Chinese	zh	Sino-Tibetan	GSD	
	Erzya	myv	Uralic, Mordvin	JR	
	Faroese	fo	Indo-European, Germanic	FarPaHC	
	Japanese	ja	Japanese	GSD	Wikipedia
POS/DP	Livvi	olo	Uralic, Finnic	KKPP	
	Maltese	mt	Afro-Asiatic, Semitic	MUDT	
	Manx	gv	Indo-European, Celtic	Cadhan	
	North Sami	sme	Uralic, Sami	Giella	
	Komi Zyrian	kpv	Uralic, Permic	Lattice	
	Sanskrit	sa	Indo-European, Indic	UFAL	
	Upper Sorbian	hsb	Indo-European, Slavic	UFAL	
	Uyghur	ug	Turkic, Southeastern	UDT	
	Hausa	hau	Afro-Asiatic, Chadic		Wikipedia
	Igbo	ibo	Niger-Congo, Volta-Niger		Wikipedia
	Kinyarwanda	kin	Niger-Congo, Bantu		Wikipedia
	Luganda	lug	Niger-Congo, Bantu		Wikipedia
NER	Luo	luo	Nilo-Saharan	N/A	Luo News Dataset (Adelani et al., 2021)
	Nigerian-Pidgin	pcm	English Creole		JW300 (Agić and Vulić, 2019)
	Swahili	swa	Niger-Congo, Bantu		Wikipedia
	Wolof	wol	Niger-Congo, Senegambian		Wikipedia
	Yorùbá	yor	Niger-Congo, Volta-Niger		Wikipedia
	Aymara	aym	Aymaran		Tiedemann (2012); Wikipedia
	Asháninka	cni	Arawakan		Ortega et al. (2020); Cushimariano Romano and Sebastián Q. (2008); Mihas (2011); Bustamante et al. (2020)
	Bribri	bzd	Chibchan, Talamanca		Feldman and Coto-Solano (2020)
NLI	Guarani	gn	Tupian, Tupi-Guarani	N/A	Chiruzzo et al. (2020); Wikipedia
	Náhuatl	nah	Uto-Aztecan, Aztecan		Gutierrez-Vasques et al. (2016); Wikipedia
	Otomí	oto	Oto-Manguean, Otomian		Hñähñu Online Corpus
	Quechua	quy	Quechuan		Agić and Vulić (2019); Wikipedia
	Rarámuri	tar	Uto-Aztecan, Tarahumaran		Brambila (1976)
	Shipibo-Konibo	shp	Panoan		Galarreta et al. (2017); Bustamante et al. (2020)
	Wixarika	hch	Uto-Aztecan, Corachol		Mager et al. (2018)
	Greek	el	Indo-European, Greek		
	Romanian	ro	Indo-European, Romance		
QA	Russian	ru	Indo-European, Slavic	N/A	Wikipedia

Task	Language	ISO Code	Family	UD Tree-bank	Corpus source(s)
	Thai	th	Tai-Kadai, Kam-Tai		
	Turkish	tr	Turkic, Southwestern		

B.2 Full Results

	LT-SFT	rand-SFT	MAD-X	BitFit	LT-SFT TA	MAD-X TA
ar	68.7	69.3	70.1	69.8	70.6	**70.8**
bm	**57.0**	55.6	51.0	41.7	34.2	37.2
bxr	**73.2**	71.4	71.9	64.2	59.5	62.0
fo	**87.9**	86.5	85.7	77.3	72.9	74.1
gv	**72.0**	68.4	66.9	44.3	35.4	37.5
hsb	**83.1**	82.4	81.8	77.2	69.2	69.6
ja	53.9	**54.3**	51.1	53.9	54.1	51.2
kpv	**61.8**	56.0	58.5	39.6	37.1	35.8
mt	**80.6**	77.6	73.7	53.6	32.6	30.9
myv	**80.3**	71.5	75.6	54.7	45.7	48.5
olo	**82.3**	81.7	79.7	73.1	62.2	63.4
sa	**65.3**	63.2	60.9	50.3	39.8	45.0
sme	**78.0**	70.4	72.0	50.6	43.3	39.4
ug	59.1	**64.7**	63.7	43.2	34.0	36.8
yue	**66.8**	65.6	**66.8**	66.2	64.5	64.1
zh	67.5	68.0	67.6	**69.2**	65.9	67.6
avg	**71.1**	69.2	68.6	58.1	51.3	52.1

(a) POS accuracy (%)

	LT-SFT	rand-SFT	MAD-X	BitFit	LT-SFT TA	MAD-X TA	LT-SFT MS
ar	**70.8**/**53.6**	68.7/51.6	69.5/51.5	64.0/48.6	68.7/53.0	68.6/52.3	81.5/69.8
bm	**43.1**/**16.5**	39.3/14.8	39.1/13.6	33.3/8.1	30.0/7.8	29.9/6.8	46.4/20.6
bxr	**49.2**/**25.9**	48.3/24.1	48.3/24.0	44.9/19.7	40.7/17.3	41.0/18.0	60.2/35.4
fo	**68.2**/**55.5**	65.7/53.1	66.3/52.5	57.7/43.4	54.3/39.8	53.6/38.5	67.2/55.6
gv	60.0/**42.4**	59.0/39.1	**61.2**/37.0	43.3/14.7	28.1/5.0	26.4/5.4	66.1/52.0
hsb	**73.7**/60.5	72.1/58.7	72.1/**61.1**	61.7/47.7	55.4/42.1	53.5/40.9	87.0/79.5
ja	**36.9**/**19.7**	34.8/18.9	33.0/18.9	34.4/18.8	36.0/19.3	33.8/18.3	44.0/26.9
kpv	**50.5**/**27.2**	45.1/20.7	47.3/22.6	35.8/11.3	24.7/7.5	25.4/7.1	57.1/35.9
mt	**74.6**/**55.4**	68.9/48.8	69.4/50.8	51.0/25.0	29.2/5.7	28.9/5.0	81.0/67.9
myv	**65.9**/**45.3**	59.8/36.3	59.6/35.7	42.2/17.2	32.1/11.7	30.3/10.4	73.8/57.4
olo	**66.4**/**47.8**	64.5/43.1	60.9/42.0	52.4/29.3	42.2/20.0	42.5/18.3	74.9/62.4
sa	**49.5**/**25.2**	48.9/20.8	46.8/19.5	42.8/13.9	32.5/8.7	36.0/9.9	62.1/39.5
sme	**58.0**/**42.1**	49.9/29.6	50.6/29.0	31.7/10.7	23.2/7.0	22.3/6.6	63.4/50.7
ug	36.4/16.7	37.3/15.8	**42.1**/**19.2**	35.3/13.5	21.9/7.7	23.5/8.4	56.3/35.9
yue	**51.1**/**34.0**	48.7/31.2	48.8/31.8	44.5/27.0	47.4/30.0	47.0/29.4	52.1/36.3
zh	**59.8**/37.0	58.2/35.6	58.5/**37.2**	55.9/33.7	58.4/36.3	59.1/36.9	55.3/35.9
avg	**57.1**/**37.8**	54.3/33.9	54.6/34.1	45.7/23.9	39.1/19.9	38.9/19.5	64.3/47.6

(b) DP UAS/LAS

	LT-SFT	MAD-X	BitFit	LT-SFT TA	MAD-X TA	LT-SFT MS
hau	**83.5**	83.4	50.2	46.5	44.0	
ibo	**76.7**	71.7	57.2	56.8	54.5	
kin	**67.4**	65.3	56.0	52.9	50.2	
lug	**67.9**	67.0	50.9	53.8	53.3	
luo	**54.7**	52.2	35.6	37.7	33.0	
pcm	**74.6**	72.1	66.8	74.4	71.0	
swa	**79.4**	77.6	67.4	69.5	69.6	
wol	**66.3**	65.6	45.0	37.1	29.8	
yor	**74.8**	74.0	64.7	69.3	66.6	
avg	**71.7**	69.9	54.9	55.3	52.4	

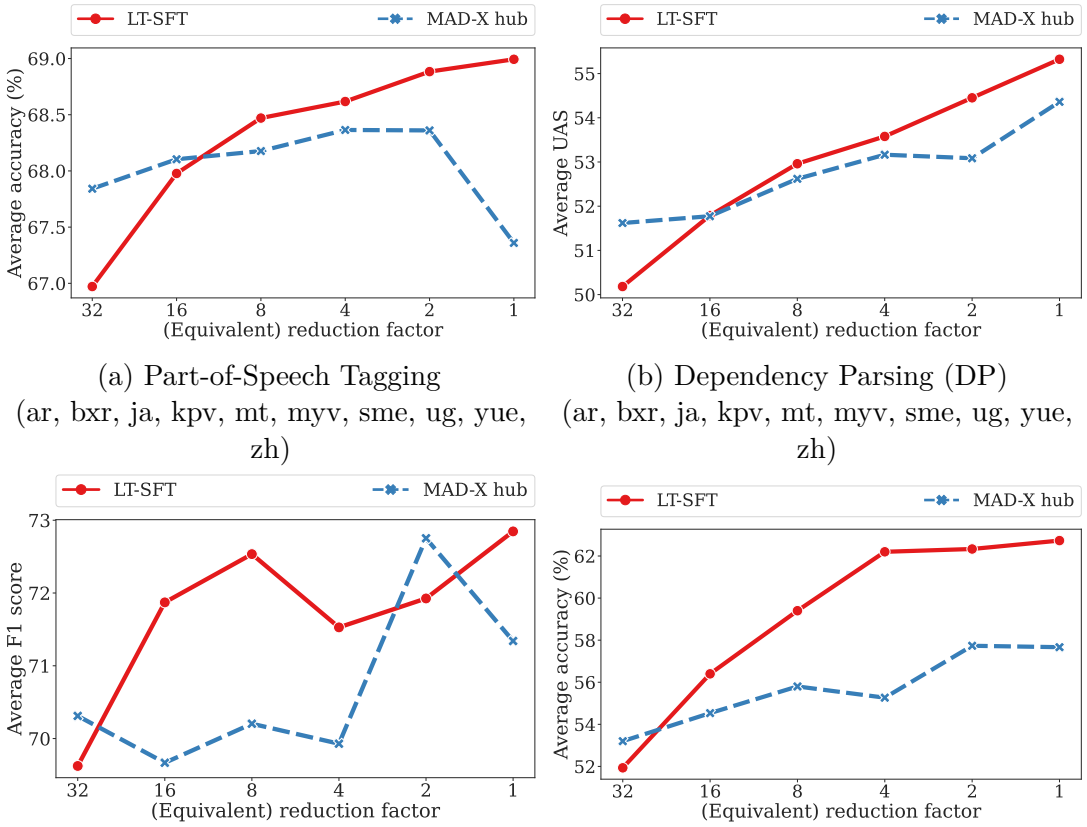
(c) NER F1-score

	LT-SFT	MAD-X	BitFit	LT-SFT TA	MAD-X TA	LT-SFT MS
aym	**57.9**	51.6	40.8	38.3	40.7	59.9
bzd	**44.4**	44.0	36.7	37.1	38.3	46.3
cni	**47.9**	47.6	34.5	40.9	44.1	50.3
gn	**63.5**	58.8	46.4	44.8	43.3	69.1
hch	**42.9**	41.5	36.3	38.4	40.7	44.4
nah	52.7	**53.7**	38.8	41.6	44.2	53.8
oto	**48.5**	46.8	39.8	39.7	40.8	43.3
quy	**62.0**	58.3	34.5	38.3	41.5	68.4
shp	**50.3**	48.9	38.8	42.1	44.4	53.2
tar	43.5	**43.9**	36.7	37.6	38.8	42.5
avg	**51.4**	49.5	38.3	39.9	41.7	53.1

(d) NLI accuracy (%)

Table B.2 Results achieved by various zero-shot cross-lingual transfer methods across all tasks for each language. For each (method, task) pair, the (equivalent) reduction factor with the best mean score is selected as shown in Table 4.2. `LT-SFT MS` denotes `LT-SFT` with multi-source training. **Bold** denotes best-performing method per language, excluding `LT-SFT MS` as its larger, more diverse dataset gives it an unfair advantage.

B.3 MAD-X Results with AdapterHub adapters



(a) Part-of-Speech Tagging (ar, bxr, ja, kpv, mt, myv, sme, ug, yue, zh)

(b) Dependency Parsing (DP) (ar, bxr, ja, kpv, mt, myv, sme, ug, yue, zh)

(c) Named Entity Recognition (NER) (swa, wol)

(d) Natural Language Inference (NLI) (gn, quy)

Fig. B.1 Zero-shot cross-lingual transfer evaluation of Lottery-Ticket Sparse Fine-Tuning (LT-SFT) and MAD-X when pre-trained language adapters from AdapterHub (Pfeiffer et al., 2020a) are used during task training and evaluation. These adapters are trained for 250,000 steps with a batch size of 64, as opposed to the 100,000 steps of batch size 8 used in our experiments. LT-SFT nevertheless maintains an edge in performance across all tasks. Since AdapterHub adapters are only available for some of the languages in our evaluation, the results shown are averaged over only the languages for which they are available, indicated in the subfigure captions.

Appendix C

Distilling Efficient Language-Specific Models for Cross-Lingual Transfer

C.1 Languages

Table C.1 Details of the languages and data used for training and evaluation of SFTs. The corpora of Bustamante et al. (2020) are available at https://github.com/iapucp/multilingual-data-peru; all other NLI corpora mentioned are available at https://github.com/AmericasNLP/americasnlp2021.

Task	Language	ISO Code	Family	UD Treebank	Corpus source(s)
Source	English	en	Indo-European, Germanic	EWT	Wikipedia
DP	Arabic	ar	Afro-Asiatic, Semitic	PUD	Wikipedia
	Bambara	bm	Mande	CRB	
	Buryat	bxr	Mongolic	BDT	
	Cantonese	yue	Sino-Tibetan	HK	
	Chinese	zh	Sino-Tibetan	GSD	
	Erzya	myv	Uralic, Mordvin	JR	
	Faroese	fo	Indo-European, Germanic	FarPaHC	
	Japanese	ja	Japanese	GSD	
	Livvi	olo	Uralic, Finnic	KKPP	
	Maltese	mt	Afro-Asiatic, Semitic	MUDT	
	Manx	gv	Indo-European, Celtic	Cadhan	
	North Sami	sme	Uralic, Sami	Giella	
	Komi Zyrian	kpv	Uralic, Permic	Lattice	
	Sanskrit	sa	Indo-European, Indic	UFAL	
	Upper Sorbian	hsb	Indo-European, Slavic	UFAL	

Task	Language	ISO Code	Family	UD Treebank	Corpus source(s)
	Uyghur	ug	Turkic, Southeastern	UDT	
NER	Hausa	hau	Afro-Asiatic, Chadic	N/A	Wikipedia
	Igbo	ibo	Niger-Congo, Volta-Niger		Wikipedia
	Kinyarwanda	kin	Niger-Congo, Bantu		Wikipedia
	Luganda	lug	Niger-Congo, Bantu		Wikipedia
	Luo	luo	Nilo-Saharan		Luo News Dataset (Adelani et al., 2021)
	Nigerian-Pidgin	pcm	English Creole		JW300 (Agić and Vulić, 2019)
	Swahili	swa	Niger-Congo, Bantu		Wikipedia
	Wolof	wol	Niger-Congo, Senegambian		Wikipedia
	Yorùbá	yor	Niger-Congo, Volta-Niger		Wikipedia
NLI	Aymara	aym	Aymaran	N/A	Tiedemann (2012); Wikipedia
	Asháninka	cni	Arawakan		Ortega et al. (2020); Cushimariano Romano and Sebastián Q. (2008); Mihas (2011); Bustamante et al. (2020)
	Bribri	bzd	Chibchan, Talamanca		Feldman and Coto-Solano (2020)
	Guarani	gn	Tupian, Tupi-Guarani		Chiruzzo et al. (2020); Wikipedia
	Náhuatl	nah	Uto-Aztecan, Aztecan		Gutierrez-Vasques et al. (2016); Wikipedia
	Otomí	oto	Oto-Manguean, Otomian		Hñähñu Online Corpus
	Quechua	quy	Quechuan		Agić and Vulić (2019); Wikipedia
	Rarámuri	tar	Uto-Aztecan, Tarahumaran		Brambila (1976)
	Shipibo-Konibo	shp	Panoan		Galarreta et al. (2017); Bustamante et al. (2020)
	Wixarika	hch	Uto-Aztecan, Corachol		Mager et al. (2018)
QA	Arabic	ar	Afro-Asiatic, Semitic	N/A	Wikipedia
	Chinese	zh	Sino-Tibetan		
	German	de	Indo-European, Germanic		
	Greek	el	Indo-European, Greek		
	Hindi	hi	Indo-European, Indic		
	Romanian	ro	Indo-European, Romance		
	Russian	ru	Indo-European, Slavic		
	Thai	th	Tai-Kadai, Kam-Tai		
	Turkish	tr	Turkic, Southwestern		
	Vietnamese	vi	Austro-Asiatic, Viet-Muong		

C.2 Additional Results

	ar	bm	bxr	fo	gv	hsb	ja	kpv	mt	myv	olo	sa	sme	ug	yue	zh	avg	avgΔ
LtSft	70.8	43.1	49.2	68.2	60.0	73.7	36.9	50.5	74.6	65.9	66.4	49.5	58.0	36.4	51.1	59.8	57.1	-
DistilmBert	65.7	34.4	42.3	63.0	52.8	67.6	32.1	42.2	65.4	58.6	59.6	44.1	51.2	29.2	47.0	56.1	50.7	-6.4
Scratch, lrf = 2	38.5	26.6	24.8	44.9	35.4	33.5	18.6	23.4	42.9	31.5	30.2	23.0	26.1	12.3	30.8	35.6	29.9	-27.2
BiStil-ST, lrf = 2	68.0	41.6	45.7	66.3	56.6	70.9	34.1	**48.2**	71.0	**64.5**	**64.3**	48.9	**57.6**	**34.5**	49.4	56.7	54.9	-2.2
BiStil-ST, lrf = 3	65.5	42.5	45.9	64.1	52.7	68.1	33.2	46.5	68.0	62.0	61.5	46.9	55.1	32.4	48.6	55.3	53.0	-4.1
BiStil-TF, lrf = 2	**70.3**	43.4	46.8	**67.1**	**57.7**	**72.4**	**34.5**	47.6	**72.7**	64.2	62.6	**50.5**	57.4	32.3	**49.8**	**58.6**	**55.5**	**-1.6**
BiStil-TF, lrf = 3	67.0	**43.9**	**47.6**	65.1	54.2	70.0	33.4	44.2	69.7	62.3	61.8	49.2	55.1	33.3	48.9	56.5	53.9	-3.3

Table C.2 DP UAS score for main experiments.

	ar	de	el	es	hi	ro	ru	th	tr	vi	zh	avg	avgΔ
LtSft	73.0	80.5	78.6	80.6	74.3	82.4	77.8	69.7	72.2	76.5	68.9	75.9	-
MiniLMv2	66.4	75.5	72.4	76.6	69.6	78.3	74.0	63.8	67.6	**73.3**	**64.6**	71.1	-4.8
BiStil-TF, lrf = 2	**69.4**	**77.4**	**73.8**	**77.6**	**69.7**	**79.1**	**75.0**	**66.7**	**68.8**	72.8	64.5	**72.3**	**-3.6**
BiStil-TF, lrf = 3	62.4	70.7	63.3	74.7	61.4	73.4	68.7	54.3	62.9	63.0	60.4	65.0	-10.9

Table C.3 XQuAD F1 score for main experiments.

Appendix D

A Holistic View of Cross-Lingual Transfer

D.1 Languages

The complete overview of languages, their codes and families, together with the monolingual data sizes and resources is provided in Table D.1. The sizes and resources for the parallel corpora used in the MT model adaptation are given in Table D.2.

Table D.1 Details of the languages and monolingual data used for training and evaluation of SFTs and adapters. The corpora of Bustamante et al. (2020) are available at https://github.com/iapucp/multilingual-data-peru; all other NLI corpora mentioned are available at https://github.com/AmericasNLP/americasnlp2021; all the SA corpora (Cahyawijaya et al., 2022) are available through https://indonlp.github.io/nusa-catalogue/.

Task	Language	Code	Family	Corpus size (MB)	Corpus source(s)
Source	English	en	Indo-European, Germanic	300,800	Wikipedia
	Indonesian	id	Austronesian, Malayo-Sumbawan	148,300	
NLI	Aymara	aym	Aymaran	2.3	Tiedemann (2012); Wikipedia
	Asháninka	cni	Arawakan	1.4	Ortega et al. (2020); Cushimariano Romano and Sebastián Q. (2008); Mihas (2011); Bustamante et al. (2020)
	Bribri	bzd	Chibchan, Talamanca	0.3	Feldman and Coto-Solano (2020)
	Guarani	gn	Tupian, Tupi-Guarani	6.9	Chiruzzo et al. (2020); Wikipedia
	Náhuatl	nah	Uto-Aztecan, Aztecan	8.1	Gutierrez-Vasques et al. (2016); Wikipedia
	Otomí	oto	Oto-Manguean, Otomian	0.4	Hñähñu Online Corpus
	Quechua	quy	Quechuan	17	Agić and Vulić (2019); Wikipedia
	Rarámuri	tar	Uto-Aztecan, Tarahumaran	0.6	Brambila (1976)
	Shipibo-Konibo	shp	Panoan	2.1	Galarreta et al. (2017); Bustamante et al. (2020)

Task	Language	Code	Family	Corpus size (MB)	Corpus source(s)
	Wixarika	hch	Uto-Aztecan, Corachol	0.5	Mager et al. (2018)
SA	Acehnese	ace	Austronesian, Malayo-Sumbawan	90	KoPI-NLLB (Cahyawijaya et al., 2022); LibriVox-Indonesia (Wirawan, 2022); NLLB-Seed (NLLB Team et al., 2022); Wikipedia
	Balinese	ban	Austronesian, Malayo-Sumbawan	42	INDspeech_NEWS_EthnicSR (Sakti and Nakamura, 2013), KoPI-NLLB (Cahyawijaya et al., 2022); LibriVox-Indonesia (Wirawan, 2022); NLLB-Seed (NLLB Team et al., 2022); Wikipedia
	Banjarese	bjn	Austronesian, Malayo-Sumbawan	28	KoPI-NLLB (Cahyawijaya et al., 2022); Korpus Nusantara (Sujaini, 2020); NLLB-Seed (NLLB Team et al., 2022); Wikipedia
	Buginese	bug	Austronesian, South Sulawesi	4.3	Korpus Nusantara (Sujaini, 2020); LibriVox-Indonesia (Wirawan, 2022); NLLB-Seed (NLLB Team et al., 2022); Wikipedia
	Javanese	jav	Austronesian, Javanese	200	Wikipedia
	Madurese	mad	Austronesian, Malayo-Sumbawan	0.8	Korpus Nusantara (Sujaini, 2020); Wikipedia
	Minangkabau	min	Austronesian, Malayo-Sumbawan	93	Indo Wiki Parallel Corpora (Trisedya and Inastra, 2014); KoPI-NLLB (Cahyawijaya et al., 2022); Korpus Nusantara (Sujaini, 2020); LibriVox-Indonesia (Wirawan, 2022); MinangNLP MT (Koto and Koto, 2020); Wikipedia
	Ngaju	nij	Austronesian, Barito	-	-
	Sundanese	sun	Austronesian, Malayo-Sumbawan	100	Wikipedia
	Toba Batak	bbc	Austronesian, Northwest Sumatra-Barrier Islands	0.4	Korpus Nusantara (Sujaini, 2020)

Target Language	Source Language	Corpus size (#sent)	Corpus source(s)
Bribri	Spanish	8,502	Feldman and Coto-Solano (2020)
Náhuatl	Spanish	16,733	Gutierrez-Vasques et al. (2016)
Otomí	Spanish	5,488	Hñähñu Online Corpus
Rarámuri	Spanish	15,714	Brambila (1976)
Shipibo-Konibo	Spanish	15,586	Galarreta et al. (2017); Bustamante et al. (2020)
Wixarika	Spanish	9,960	Mager et al. (2018)
Madurese	Indonesian	629	Winata et al. (2023)
Ngaju	Indonesian	629	
Toba Batak	Indonesian	629	

Table D.2 Details of the parallel corpora used for NLLB MT model adaptation. #sent = number of sentences in train + dev set.

D.2 Ablation Experiments

We present per language results of our ablation experiments in Table D.3. The summarised results are given in Table 6.3.

(a) AmericasNLI: accuracy

	Method	AYM	BZD	CNI	GN	HCH	NAH	OTO	QUY	SHP	TAR	avg
Adapter	ZS	53.0	42.8	44.8	59.6	40.3	50.8	41.2	55.2	50.0	40.0	47.77
	ZS – LA	40.1	36.9	42.0	42.9	39.6	42.7	41.8	40.8	43.2	35.7	40.57
	FS	55.2	47.1	47.7	58.3	40.7	55.8	45.9	59.7	52.9	47.5	51.08
	FS – LA	43.3	42.8	45.7	48.5	42.3	46.3	43.7	46.4	47.2	42.0	44.82
	FS – UPSAMPLE	50.3	45.5	46.8	58.3	40.3	48.9	**46.4**	56.3	46.0	43.9	48.27
	TTRAIN-ALL	**63.3**	**60.5**	**56.1**	**66.1**	**52.1**	**60.4**	42.4	**64.9**	**63.3**	55.3	**58.44**
	TTRAIN-ALL – LA	60.1	58.3	50.3	60.7	50.7	55.6	44.7	60.8	57.1	**56.7**	55.50
SFT	ZS	58.4	44.7	47.6	62.2	44.4	50.8	46.4	60.4	49.5	43.1	50.75
	ZS – LA	35.7	38.7	37.3	39.1	38.3	41.3	37.7	36.3	40.9	36.8	38.21
	FS	59.2	58.7	53.2	63.9	45.9	54.6	**49.1**	61.2	53.1	51.2	55.01
	FS – LA	53.3	54.9	47.6	48.9	42.9	53.8	45.5	53.3	46.5	48.4	49.51
	FS – UPSAMPLE	59.1	51.2	50.3	63.9	42.9	52.7	48.1	62.8	50.1	44.9	52.60
	TTRAIN-ALL	**65.5**	**61.9**	**56.7**	**70.5**	**55.5**	**62.2**	42.6	**68.5**	**66.0**	**58.7**	**60.81**
	TTRAIN-ALL – LA	63.6	60.7	48.0	65.6	54.7	57.3	46.9	63.6	62.7	55.7	57.88

(b) NusaX: F1

	Method	ACE	BAN	BBC	BJN	BUG	JAV	MAD	MIN	NIJ	SUN	avg
Adapter	ZS	74.9	78.0	72.3	77.6	57.6	82.9	68.5	79.9	–	80.5	74.69
	ZS – LA	68.0	70.8	37.6	78.3	31.9	80.9	63.5	77.9	66.9	78.0	65.38
	FS	79.1	80.5	77.2	**86.0**	72.0	85.0	77.5	**84.8**	–	83.7	80.64
	FS – LA	74.7	73.5	64.3	79.6	61.7	82.3	74.0	81.5	70.9	77.8	74.03
	FS – UPSAMPLE	76.5	78.4	71.9	80.6	64.5	82.5	74.6	80.1	–	81.9	76.78
	TTRAIN-ALL	**79.6**	**82.3**	**81.0**	85.0	**68.1**	**86.2**	**78.8**	83.3	–	**85.7**	**81.11**
	TTRAIN-ALL – LA	78.5	76.5	73.1	83.2	67.6	82.7	77.8	82.0	**77.4**	79.5	77.83
SFT	ZS	80.0	81.3	65.8	82.0	63.8	84.3	73.5	86.6	–	84.4	77.97
	ZS – LA	72.8	72.3	47.3	79.3	47.7	82.2	71.9	81.4	70.3	79.1	70.43
	FS	82.2	84.1	80.6	**88.3**	**77.7**	**88.0**	78.6	**89.2**	–	85.1	**83.76**
	FS – LA	78.9	76.2	68.6	81.1	71.0	86.3	78.2	84.0	76.0	82.9	78.32
	FS – UPSAMPLE	82.7	**84.5**	79.5	87.9	74.7	86.5	78.0	87.8	–	**87.0**	83.18
	TTRAIN-ALL	**83.4**	82.4	**82.7**	84.6	76.8	86.4	**81.6**	87.4	–	85.2	83.39
	TTRAIN-ALL – LA	78.9	81.0	75.1	83.0	67.1	86.2	79.7	82.8	**77.6**	82.5	79.39

Table D.3 Per-language results of ablation experiments with ZS, FS, and TTRAIN-ALL methods on NLI and SA with bottleneck adapters and SFTs: – LA denotes the absence of language adaptation (i.e. only the task module is present), while – UPSAMPLE indicates there is no upsampling of the gold-standard target shots.

D.3 Full Results with Different Number of Shots K

Here we give full results with different numbers of gold-standard target shots K, where $K \in \{0, 20, 100, 500\}$. The setting $K = 0$ corresponds to the ZS approach, while the rest of the values fall within FS. The results are shown in Table D.4, with their summary given in Figure 6.1.

(a) AmericasNLI: accuracy

	Method	AYM	BZD	CNI	GN	HCH	NAH	OTO	QUY	SHP	TAR	avg
Adapter	ZS (K = 0)	53.0	42.8	44.8	59.6	40.3	50.8	41.2	55.2	50.0	40.0	47.77
	FS (K = 20)	50.4	43.3	46.0	56.9	39.9	50.4	43.2	58.3	50.5	44.0	48.29
	FS (K = 100)	55.2	47.1	47.7	58.3	40.7	**55.8**	45.9	**59.7**	52.9	47.5	51.08
	FS (K = 500)	**57.2**	**54.0**	**53.7**	**61.6**	**45.1**	55.1	**48.0**	59.1	**54.7**	**50.5**	**53.90**
SFT	ZS (K = 0)	58.4	44.7	47.6	62.2	44.4	50.8	46.4	60.4	49.5	43.1	50.75
	FS (K = 20)	57.9	48.7	46.4	59.9	44.3	51.5	48.7	61.2	51.1	43.7	51.34
	FS (K = 100)	59.2	58.7	53.2	63.9	45.9	54.6	49.1	61.2	53.1	51.2	55.01
	FS (K = 500)	**61.1**	**59.5**	**55.7**	**65.6**	**48.7**	**58.9**	**52.5**	**64.4**	**61.5**	**54.1**	**58.20**

(b) NusaX: F1

	Method	ACE	BAN	BBC	BJN	BUG	JAV	MAD	MIN	SUN	avg
Adapter	ZS (K = 0)	74.9	78.0	72.3	77.6	57.6	82.9	68.5	79.9	80.5	74.69
	FS (K = 20)	74.6	78.9	73.6	81.0	66.1	84.3	78.3	81.3	82.4	77.83
	FS (K = 100)	79.1	80.5	77.2	86.0	72.0	85.0	77.5	84.8	83.7	80.64
	FS (K = 500)	**81.1**	**84.2**	**78.5**	**88.0**	**74.8**	**87.9**	**79.4**	**88.0**	**88.1**	**83.33**
SFT	ZS (K = 0)	80.0	81.3	65.8	82.0	63.8	84.3	73.5	86.6	84.4	77.97
	FS (K = 20)	81.9	83.1	79.0	87.9	73.4	86.3	76.0	87.0	87.2	82.42
	FS (K = 100)	82.2	84.1	80.6	**88.3**	77.7	88.0	78.6	89.2	85.1	83.76
	FS (K = 500)	**86.5**	**89.3**	**83.1**	87.9	**79.7**	**91.3**	**80.6**	**90.5**	**87.5**	**86.27**

Table D.4 Per-language results on NLI and SA with the different number of gold-standard target shots K. We consider ZS (K = 0) and FS with $K \in \{20, 100, 500\}$ with bottleneck adapters and SFTs.

Appendix E

Scaling Sparse Fine-Tuning to Large Language Models

E.1 Evaluation Setup

Following Wang et al. (2023) and Ivison et al. (2023), we evaluate the instruction-tuned LLMs on a range of benchmarks to capture the following abilities:

Factuality: Massively Multitask Language Understanding (MMLU; Hendrycks et al., 2021) requires the model to pick an answer from 4 candidates and covers 57 subjects including STEM, humanities, social sciences, and other disciplines. We evaluate models in a 5-shot setting and report their accuracy.

Reasoning: We sub-sample 40 examples per task from BIG-Bench Hard (BBH; Suzgun et al., 2023), a suite of the 23 most challenging tasks from BIG-Bench. We also randomly select a subset of 200 examples from Grade School Math (GSM; Cobbe et al., 2021), a collection of math problems in linguistic form. Both benchmarks require open-ended generation. We evaluate models in a 3-shot setting with BBH and 8-shot setting with GSM, and report their exact match (EM).

Multilinguality: We choose 100 examples per language from Typologically Diverse Question Answering (TyDiQA; Clark et al., 2020a), a dataset for extractive question answering in 11 languages. We follow the Gold Standard setup where each document is guaranteed to contain the answer span. We evaluate models in a 1-shot setting and report F1.

Coding: HumanEval (Chen et al., 2021) is a dataset for synthesizing programs from docstrings. We evaluate models with a temperature of 0.1 and report their precision at 1 (P@1)[1].

E.2 Further Experiments on Drop/Grow Schedule

After our main experiments, we performed some additional exploration of the drop/growth schedule, adjusting both the frequency γ of dropping/growth and the schedule $k(i,t)$ of number of parameters to drop/grow at each update. We considered drop/growth frequency in the range $\{10, 20, 40, 80\}$ steps and both our default linear schedule and the cosine schedule of Evci et al. (2020), where

$$k(i,t) = \frac{\xi}{2}\left(1 + \cos\left(\frac{t\pi}{T}\right)\right) d_{\phi^{(i)}}, \tag{E.1}$$

recalling that ξ is the initial update rate and T is the total number of training steps. Table E.1 presents our results for SpIEL-AG when fine-tuning LLaMA2-7b on the Flan v2 subset with the other hyperparameters as shown in Table 7.1. We find that there is not a clear preference for a linear or cosine schedule, but that a higher update frequency of 10 might be better than the 20 steps we used in our main experiments.

Schedule		Benchmark	
Type	**Steps**	**MMLU**	**TyDiQA**
Linear	10	**52.0**	56.4
	20	50.7	56.2
	40	51.5	**56.5**
	80	51.3	56.1
Cosine	10	**51.8**	56.3
	20	51.6	**56.5**
	40	50.2	**56.5**
	80	50.9	55.4

Table E.1 SpIEL-AG results when fine-tuning LLaMA2-7b on Flan v2 subset with various frequencies and update schedule types.

[1] Here we differ from Wang et al. (2023) in order to reduce the variance across evaluation runs.

E.3 Full Hyperparameter Search Results

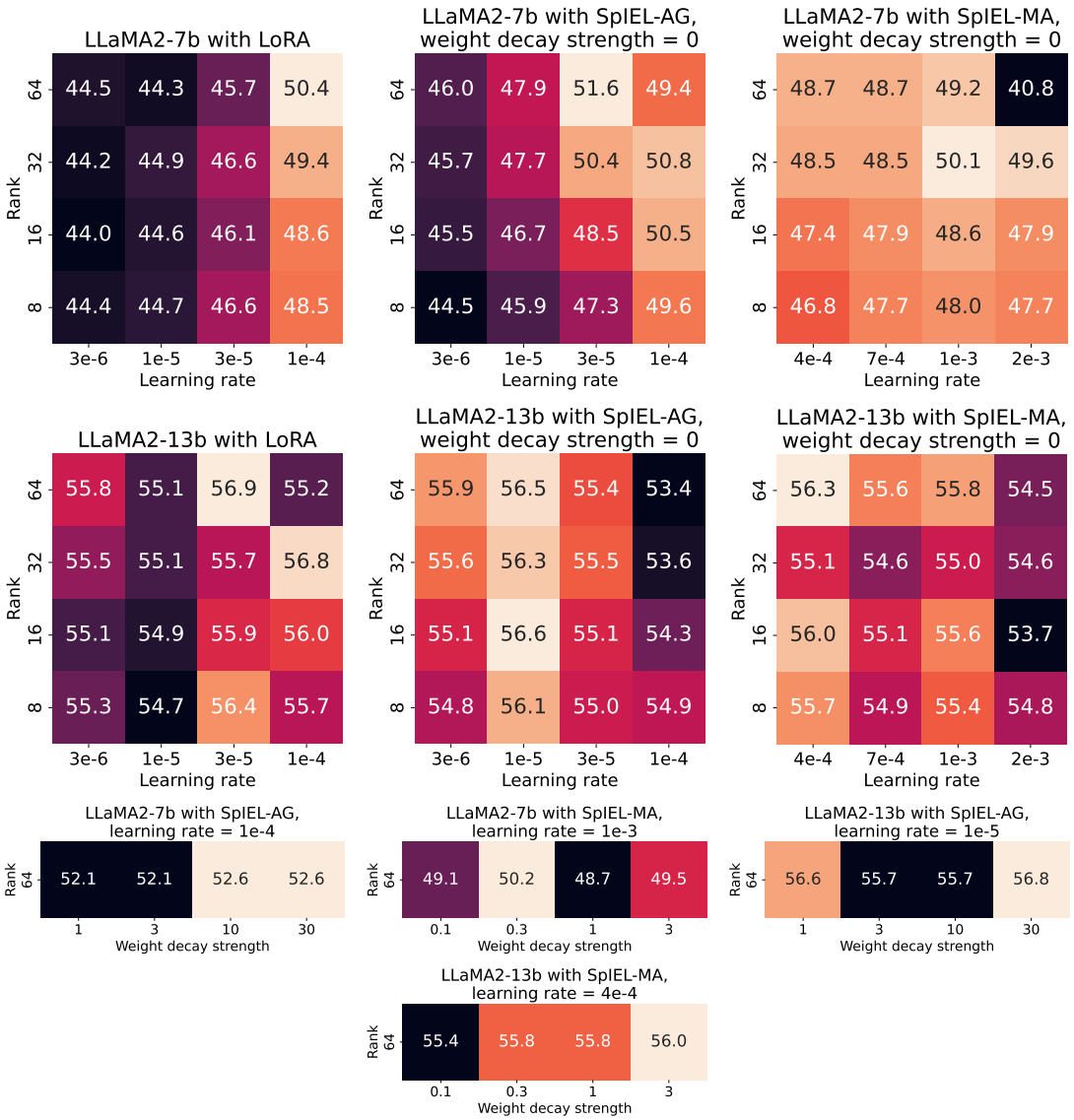


Fig. E.1 Hyperparameter search results.

E.4 Analysis of the Backward Pass for SpIEL

Consider the forward pass when sparsely fine-tuning a linear layer. We have:

$$Y = X(W + \Delta), \tag{E.2}$$

where the input $X \in \mathbb{R}^{b \times d_{\text{in}}}$, the pre-trained weight matrix W and its sparse delta $\Delta \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$, and the output $Y \in \mathbb{R}^{b \times d_{\text{out}}}$, with b, d_{in} and d_{out} being the batch size and input and output dimensions respectively. It can be shown that

$$\frac{\partial \mathcal{L}}{\partial \Delta} = X^\top \frac{\partial \mathcal{L}}{\partial Y}. \tag{E.3}$$

However, we only need the entries of $\frac{\partial \mathcal{L}}{\partial \Delta}$ corresponding to the currently active indices of Δ. Let $\boldsymbol{r} \in \{1, 2, ..., h\}^N, \boldsymbol{c} \in \{1, 2, ..., w\}^N$, where h and w are the height and width of Δ respectively, denote the N active indices of Δ, i.e. (r_i, c_i) denotes the position of the ith active index in Δ. Then, if we define g_i to be the gradient of the i-th active index of Δ, we have

$$g_i = \frac{\partial \mathcal{L}}{\partial \Delta_{r_i, c_i}} \tag{E.4}$$

$$= \left(X^\top \frac{\partial \mathcal{L}}{\partial Y} \right)_{r_i, c_i} \tag{E.5}$$

$$= X^\top_{:, r_i} \left(\frac{\partial \mathcal{L}}{\partial Y} \right)_{:, c_i}. \tag{E.6}$$

That is, the gradient of the i-th sparse update to W is given by the dot product of the r_i-th column of X and the c_i-th column of the gradient of the output Y. We can compute the gradient of the sparse updates in this manner with bN FLOPs, which is an enormous theoretical improvement over the $bd_{\text{in}}d_{\text{out}}$ FLOPs required to naively perform the full matrix multiplication in (E.3) and gather the relevant indices from the result, as the SFT density $\frac{N}{d_{\text{in}}d_{\text{out}}} \ll 1$.

While calculating the sparse delta gradient in this manner entails a great reduction in FLOPs required, it is not so easy to exploit this reduction effectively on a GPU to speed the operation up. Writing an efficient CUDA kernel for this operation is ongoing work, and the speed results presented in this thesis were obtained using the naive "gather from the full matrix product" method.

E.5 Measuring Memory and Time Requirements

To measure the memory requirements of PEFT methods, we use PyTorch's `set_per_process_memory_fraction` function to limit the total available GPU memory, and perform a binary search at 1 GiB granularity to find the lowest limit at which training can run successfully. For each limit we test, we run 30 steps of Flan v2 training with (equivalent) LoRA rank 64. We measure the time as the mean duration of each of these

30 steps for the lowest passing memory limit. All our experiments are run on a single A100 GPU.

We note that this may differ from the peak memory usage during (unconstrained) training, since deep learning frameworks such as PyTorch may allocate more memory than they actually require for efficiency reasons.

E.6 Index Overlap of SpIEL across Runs

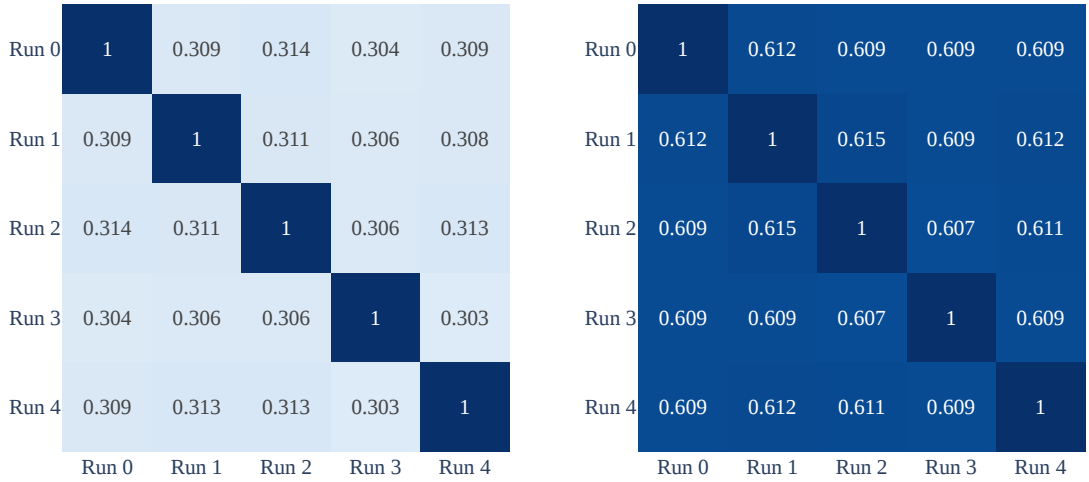


Fig. E.2 The pairwise overlap of the selected indices after training SpIEL-AG (left) and SpIEL-MA (right) with 5 different random seeds on Flan v2.

The algorithms SpIEL-AG and SpIEL-MA select a subset of indices of the parameters in the linear layers. To investigate whether the training converges to a similar set of indices for different random seeds we train with five different random seeds and report the pairwise indices overlap in Figure E.2. The overlap between the SpIEL-AG indices is notably lower than the overlap of SpIEL-MA indices. The overlap between all five runs is 11.8% for SpIEL-AG and 43.0% for SpIEL-MA. Hence, the SpIEL-MA algorithm tends to converge to a more similar set of indices more than SpIEL-AG.