arXiv:2405.07883v1 [cs.CL] 13 May 2024

Zero-Shot Tokenizer Transfer

Benjamin Minixhofer [SEP] **Edoardo M. Ponti** [CLS] **Ivan Vulić** [SEP]
[SEP] University of Cambridge [CLS] University of Edinburgh

Abstract

Language models (LMs) are bound to their tokenizer, which maps raw text to a sequence of vocabulary items (*tokens*). This restricts their flexibility: for example, LMs trained primarily on English may still perform well in other natural and programming languages, but have vastly decreased efficiency due to their English-centric tokenizer. To mitigate this, we should be able to swap the original LM tokenizer with an arbitrary one, on the fly, without degrading performance. Hence, in this work we define a new problem: Zero-Shot Tokenizer Transfer (ZeTT). The challenge at the core of ZeTT is finding *embeddings* for the tokens in the vocabulary of the new tokenizer. Since prior heuristics for initializing embeddings often perform at chance level in a ZeTT setting, we propose a new solution: we train a hypernetwork taking a tokenizer as input and predicting the corresponding embeddings. We empirically demonstrate that the hypernetwork generalizes to new tokenizers both with encoder (e.g., XLM-R) and decoder LLMs (e.g., Mistral-7B). Our method comes close to the original models' performance in cross-lingual and coding tasks while markedly reducing the length of the tokenized sequence. We also find that the remaining gap can be quickly closed by continued training on less than 1B tokens. Finally, we show that a ZeTT hypernetwork trained for a base (L)LM can also be applied to fine-tuned variants without extra training. Overall, our results make substantial strides toward detaching LMs from their tokenizer.

1 Introduction

Language Models[1] typically operate on discrete tokens, so they need a means to map text into a sequence of tokens, namely a *tokenizer*. The vast majority of contemporary LMs use subword tokenizers (Devlin et al., 2019; Jiang et al., 2023; Touvron et al., 2023; Parmar et al., 2024, among others), whereas others use byte-level (Xue et al., 2022; Yu et al., 2023; Wang et al., 2024) or character-level tokenizers (Clark et al., 2022; Tay et al., 2022). Regardless of the chosen tokenization 'granularity', these models share a fundamental limitation: once they are trained with a particular tokenizer, inference with a different tokenizer is impossible. In other terms, a pre-trained LM is *"bound"* to the tokenizer it was trained with. This has wide-ranging implications: since the focus during pretraining is typically primarily on the English language, the tokenizer often encodes languages besides English (Rust et al., 2021) or other domains, such as code, less efficiently. This leads to large disparities in the inference cost between English and non-English text (Ahia et al., 2023; Petrov et al., 2023). Tokenizers may also be sub-optimal for domains which they were not designed to be used with, e.g. fine-tunings of the Llama models performing subpar on coding tasks (Dagan et al., 2024). Efficiency and performance are only some of the reasons to transfer models across tokenizers: methods of interaction between models, such as ensembling (Sagi & Rokach, 2018) and model merging (Wortsman et al., 2022; Ainsworth et al., 2023; Yadav et al., 2023), typically assume the same unit of representation (i.e., equivalent tokenization) across models; if two models adopt different

[1] We adopt a broad definition of LMs that also includes models that do not define a probability distribution over finite-length sequences, such as text encoders.

Preprint. Under review.

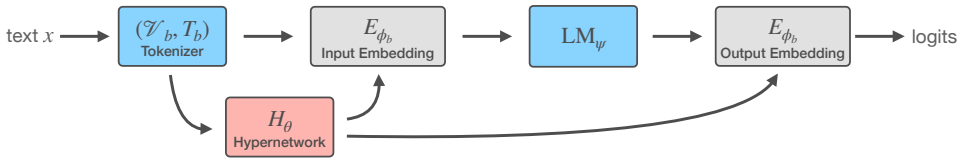

Figure 1: The hypernetwork predicts input and output embeddings based on the tokenizer.

tokenizers, they become unsuitable for ensembling or merging. Problematic artifacts of tokenization such as 'Glitch tokens' (Land & Bartolo, 2024) may also be fixed via transfer to a new tokenizer.

To address these issues, past work developed methods to equip an LM with a new tokenizer by retraining the embedding parameters, and optionally continuing to train the entire model (Artetxe et al., 2020; de Vries & Nissim, 2021). This adaptation can be made faster by initializing the embedding parameters through heuristics (Tran, 2020; Minixhofer et al., 2022; Gee et al., 2022; Dobler & de Melo, 2023; Liu et al., 2023). In this work, we formulate a new problem: given an LM, can we create an embedding matrix on-the-fly for any arbitrary tokenizer, without ever observing data for it? While past work investigated n-shot tokenizer transfer, we refer to this new problem as *zero-shot tokenizer transfer* (ZeTT). If the performance of the model can be approximately preserved, ZeTT effectively "detaches" LMs from the tokenizer they were trained with. We first evaluate the efficacy of prior (heuristic-based) approaches for ZeTT, finding that, while heuristics can preserve performance to some extent, there is generally a large gap to the original LM performance.

To close this gap, we introduce a new paradigm: We train a *hypernetwork* on a diverse distribution of tokenizers to predict the embedding parameters for any given tokenizer. By investing into the one-time-cost of training the hypernetwork, we aim to subsequently enable effective ZeTT. This proves to be possible: ZeTT via the hypernetwork preserves performance to a few percent accuracy in many cases. Furthermore, the hypernetwork can learn to rapidly adapt to a given target tokenizer by continued training on a small amount (<1B) of extra tokens, whereas previous work typically needed hundreds of billions of tokens (Dagan et al., 2024). As such, our hypernetwork provides a state-of-the-art solution to n-shot tokenizer transfer, while also establishing a competitive baseline to our newly introduced zero-shot tokenizer transfer problem. This unlocks a range of new ways to combine language models with tokenizers. For example, in this work, we zero-shot substitute the Mistral-7B tokenizer (Jiang et al., 2023) with a tokenizer that encodes code using 10% less tokens on average, while preserving functional code generation correctness to approx. 3% (Section 4.2). We also evaluate zero-shot cross-lingual transfer of the multilingual XLM-R encoder model to a range of different languages by substituting the XLM-R tokenizer with a target-language specific tokenizer and reusing adapters trained for the original XLM-R. This leads to a >16% speedup and preserves performance on XNLI (Conneau et al., 2018) to 1% on average, although the language model has never been trained with the target-language tokenizers. Finally, we show that a hypernetwork trained for a base large LM (e.g. Mistral-7B) can also be applied to fine-tunings of the same model (e.g. Mistral-7B-Instruct-v0.1), preserving capabilities to a large extent (Section 4.3). Our code and models are publicly available at `github.com/bminixhofer/zett`.

Can apply new tokenizer to finetuned variants

2 Background

Tokenizers and Embeddings. Tokenizers operate as a *tokenization function* T mapping a text to a sequence of elements in the *vocabulary* $\mathcal{V}$. By the term *tokenizer*, we henceforth refer to the tuple comprising the two crucial components, $(\mathcal{V}, T)$. Importantly, the vocabulary and the tokenization function are distinct components; given some vocabulary, there are many ways to encode text as a sequence of tokens in this vocabulary (e.g. Hofmann et al., 2022; Uzan et al., 2024). After tokenization, the model represents the sequence of tokens via a function $E_\phi : \mathcal{V} \to \mathbb{R}^{d_{\text{model}}}$ (the *embeddings*). The embeddings are typically parametrized by a matrix ϕ as a lookup table which assigns a distinct d_{model}-dimensional vector (a row of the matrix) to every element in $\mathcal{V}$. Embeddings are used twice in the language model: once at the input to map tokens to a fixed-size vector, and again at the output to compute a logit for every token, typically via a dot-product of $E_\phi(t)$ with the final hidden state of the LM. Embedding parameters may or may not be shared between the input and

the output;[2] our method works with both. We denote the entire set of embedding parameters via ϕ, denoting input embeddings as ϕ^{in} and output embeddings as ϕ^{out}, if necessary.

Contemporary language models typically use subword tokenizers via BPE (Sennrich et al., 2016) or UnigramLM (Kudo, 2018). Subword tokenization is a common choice since it can represent arbitrary sequences of text ("open-vocabulary" language modeling) while largely retaining the efficiency of word-level models (Mielke et al., 2021). However, there are a number of problems with subword tokenization, e.g. models using subword tokenization struggle parsing sequences of numbers (Golkar et al., 2023) and text with spelling mistakes (Xue et al., 2022). A recent strand of work aims to get rid of subword tokenization via byte-level (so-called "token-free") models (Xue et al., 2022; Yu et al., 2023). However, these models still operate on tokens, using the set of 256 bytes as the vocabulary, and Unicode as the tokenization function (Mielke et al., 2021). In a similar vein, some models use character-level tokenization (Tay et al., 2022; Clark et al., 2022), optionally learning to pool characters into longer tokens (Nawrot et al., 2023).[3] So far, byte- or character-level approaches have been unable to supplant subword tokenization due to reduced compute efficiency (because of longer sequences), and not necessarily being more robust (Libovický et al., 2022). Thus, although our approach is in principle applicable to any tokenizer, we focus our experiments on subword tokenizers. Specifically, we use the UnigramLM parametrization of the tokenization function, and show that other tokenizers can be converted to this parametrization later in Section 5. UnigramLM sets

$$T(x) := \underset{C \in \mathcal{C}_x}{\operatorname{argmax}} \sum_{t \in C} \log p(t)$$

where $\mathcal{C}_x$ is the set of all possible tokenizations of x (i.e., all possible decompositions of x in $\mathcal{V}$). UnigramLM provides a convenient way to represent tokens as a 2-tuple $(t, p(t)) \in (\mathcal{V}, \mathbb{R})$.

Embedding Initialization Heuristics. Prior work transfers LMs to a new tokenizer by initializing embedding parameters via a heuristic, then continuing to train the embeddings. We denote the original tokenizer as $(\mathcal{V}_a, T_a)$ and the original embedding parameters as ϕ_a. Analogously, the target tokenizer is $(\mathcal{V}_b, T_b)$ with embedding parameters ϕ_b. FVT (Gee et al., 2022) initializes embeddings for any new token $t \in \mathcal{V}_b$ as the mean of the embeddings of $T_a(t)$ i.e. the mean of the sequence of embeddings the new token is decomposed into by the previous tokenizer T_a. RAMEN (Tran, 2020), WECHSEL (Minixhofer et al., 2022) and OFA (Liu et al., 2023) require auxiliary embeddings $E_{\text{aux}} : \mathcal{V}_{\text{aux}} \to \mathbb{R}^{d_{\text{aux}}}$ with $|\mathcal{V}_{\text{aux}} \cap \mathcal{V}_a| \not\ll |\mathcal{V}_a|$ and $|\mathcal{V}_{\text{aux}} \cap \mathcal{V}_b| \not\ll |\mathcal{V}_b|$. They use E_{aux} to embed tokens in $\mathcal{V}_a$ and V_b in the same semantic space, then initialize embeddings in E_{ϕ_b} as a weighted average of embeddings in E_{ϕ_a} with weights given by their similarity in E_{aux}. FOCUS (Dobler & de Melo, 2023) initializes embeddings of tokens in $V_b \setminus V_a$ as a weighted combination of the overlapping tokens $V_a \cap V_b$, and copies the embeddings of the overlapping tokens. Weights are again computed using an auxiliary embedding matrix E_{aux}, but the only requirement is $|\mathcal{V}_{\text{aux}} \cap \mathcal{V}_b| \not\ll |\mathcal{V}_b|$. We use FOCUS as the main baseline since Dobler & de Melo (2023) show it obtains better performance without any training (i.e., zero-shot) than other heuristics, which we also confirm later in Section 4.2.

Heuristic-Free Tokenizer Transfer. While a significant amount of prior work has investigated heuristics to initialize the embedding layer, there is also research into changing the training procedure to facilitate n-shot tokenizer transfer. Marchisio et al. (2023) show that forward- and backward-propagating through a subset of the model layers is sufficient for learning embeddings for a new tokenizer. Chen et al. (2023) find that regularly resetting the embedding parameters during pretraining boosts the speed at which they are relearnt upon transfer. These approaches can be seen as orthogonal to ours. They could be freely combined with our method; we leave this to future work.

Embedding Prediction Hypernetworks. Hypernetworks are networks that predict the parameters of another network (Ha et al., 2017). Prior work uses neural networks to predict embeddings for out-of-vocabulary (Pinter et al., 2017) or rare words (Schick & Schütze, 2019) of word embedding models (Mikolov et al., 2013). Schick & Schütze (2020) extend this approach to predict embeddings for rare words in BERT models (Devlin et al., 2019). These methods can also be viewed as embedding prediction hypernetworks. In contrast, the hypernetwork we propose (i) approaches the more general problem of *transferring* to an arbitrary tokenizer, instead of *extending* the original tokenizer and (ii) can be applied to encoder, decoder, and encoder-decoder LMs, that is, it is *objective-agnostic*.

[2] Some models share the input and the output embedding parameters (e.g. Conneau et al., 2020), this has been shown to be problematic (Chung et al., 2021) and many recent LLMs (e.g. Jiang et al., 2023) separate them.

[3] See also Mielke et al. (2021) for a comprehensive overview of tokenizers.

Algorithm 1 Hypernetwork training loop for Zero-Shot Tokenizer Transfer

Input: corpus $\mathcal{D}$, tokenizer sample size n, batch size m, max. token length l, vocabulary size k, noise parameters (μ, σ), pretrained LM parameters ψ, initial hypernetwork parameters θ_{init}.

Output: Hypernetwork parameters θ.

procedure TRAINHYPERNETWORK
  $\theta \leftarrow \theta_{\text{init}}$
  $\boldsymbol{q} \leftarrow \text{queue}(x_1, .., x_n \sim \mathcal{D})$ ▷Create a pool of n texts (where $n \geq m$).

  for step in train_steps **do**
    $x_1, .., x_m \sim \mathcal{D}$
    $\boldsymbol{q} \leftarrow \text{pop}(\boldsymbol{q}, m)$ ▷Remove the least-recently-added batch.
    $\boldsymbol{q} \leftarrow \text{push}(\boldsymbol{q}, x_1, .., x_m)$ ▷Add the current batch.

    $\boldsymbol{t}, \boldsymbol{f} \leftarrow \text{substrings}(\boldsymbol{q}, l)$ ▷Compute all substrings and their frequency in $\boldsymbol{q}$.
    $\boldsymbol{f} \leftarrow \boldsymbol{f} / \sum_i f_i$ ▷Normalize frequencies to sum to one.
    $z \sim \text{Lognormal}(\mu, \sigma^2)$
    for $t, f \in (\boldsymbol{t}, \boldsymbol{f})$ **do**
      $p(t) \leftarrow f + \mathcal{N}(0, z^2)$ ▷Assign a score based on frequency + noise to the substrings.
    Sort $\boldsymbol{t}$ by $p(t)$ descending.
    $\mathcal{V}_b \leftarrow \boldsymbol{t}[:k]$ ▷Assemble the top k substrings into the tokenizer.
    $T_b \leftarrow \text{UnigramLM}(\{(t, p(t)) \mid t \in \boldsymbol{t}[:k]\})$

    $\text{loss} \leftarrow \mathcal{L}_\theta(T_b(\boldsymbol{x}), H_\theta(\mathcal{V}_b, T_b), \psi)$ ▷Compute the loss on the m texts in the current batch.

    update θ using $\nabla\theta$ w.r.t. loss.

3 Methodology

3.1 Hypernetwork Training

We aim to find parameters θ of a hypernetwork $H_\theta : (\mathcal{V}_b, T_b) \rightarrow \phi_b$ for some pretrained LM. Let ϕ_a and ψ be the embedding and inner (non-embedding) parameters of the language model, respectively. $\mathcal{L}$ is the loss of the language model as a function of the tokens, the embedding parameters, and the inner parameters, typically:

$$\mathcal{L}(t, \phi_a, \psi) = \text{CrossEntropy}(\text{LM}_\psi(E_{\phi_a}(t)), \text{label}(t)),$$

where LM_ψ is the language model and label maps the sequence of tokens to corresponding labels, e.g., shifting the sequence in case of standard (autoregressive, causal) language modeling, or masking the sequence in case of Masked Language Modeling (Devlin et al., 2019). Importantly, however, we do not make any specific assumptions on $\mathcal{L}$.

Note that the loss of the language model under the original tokenizer T_a on a text x is $\mathcal{L}(T_a(x), \phi_a, \psi)$. We train our hypernetwork to minimize the loss $\mathcal{L}_\theta(T_b(x), H_\theta(\mathcal{V}_b, T_b), \psi)$. That is, we substitute the original embedding parameters for the hypernet predictions, and substitute the original tokenizer for a tokenizer $(\mathcal{V}_b, T_b)$. Figure 1 illustrates the flow of information.

Defining Distributions over Texts and Tokenizers. We follow standard practice and sample texts uniformly from the training corpus. Tokenizer sampling is not as trivial: we would like a distribution over tokenizers $(\mathcal{V}_b, T_b)$ with high variance to encourage generalization to unseen tokenizers. To this end, we introduce a procedure to sample a diverse set of UnigramLM tokenizers. We show later in Section 5 that arbitrary tokenizers can be well-approximated via UnigramLM, motivating this choice.

We initially fill a queue $\boldsymbol{q}$ with n texts sampled randomly from the training corpus and, at every step in the training loop, push the m texts in the current batch and remove the m least recently added texts. We then compute all substrings t up to length l and their frequency in $\boldsymbol{q}$.[4][5] We add Gaussian noise to

[4] In practice, implementing $\boldsymbol{q}$ as a queue allows efficiently caching the substrings and their probability $p(t)$ at this step. They only need to be recomputed for the new m texts encountered in every batch.

[5] To ensure substrings do not cross word boundaries we pretokenize the text before computing substrings.

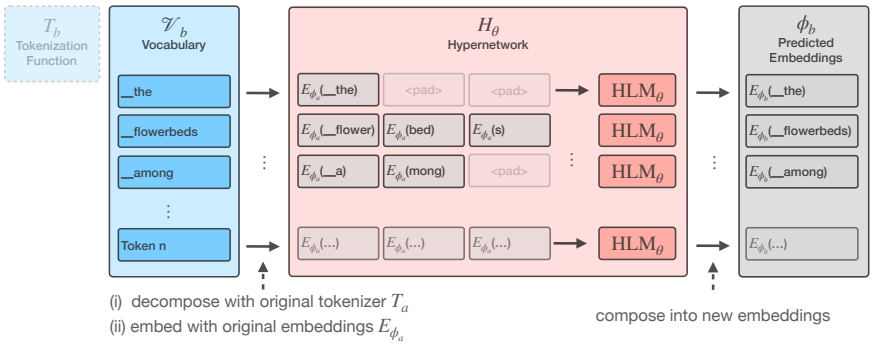

Figure 2: The hypernetwork consists of a language model HLM_θ learning to compose embeddings under the original tokenization into a new embedding and amortizes over the tokenization function.

the frequencies to arrive at a final score $p(t)$ for every token t. Finally, we assemble the tokenizer by taking the top k tokens with the highest $p(t)$ as the vocabulary and UnigramLM parametrized by $p(t)$ as the tokenization function. The training loop is summarized in Algorithm 1. The 'rolling' queue of texts $\boldsymbol{q}$ ensures high variance in the vocabulary, while the Gaussian noise added to the frequencies ensures high variance in the tokenization function.

Importantly, the texts and the tokenizer are sampled *dependently*: the batch of m texts used for training is a subset of the n texts used for sampling the tokenizer. If they were sampled independently, the probability for a token to occur would be $p(\text{token}) \propto p(\text{token} \in \mathcal{V}_b) \times p(\text{token} \in \boldsymbol{x})$. Since both these factors are small for rare tokens, $p(\text{token})$ would get vanishingly small in this case.

MIMICK-Style Warmup & Auxiliary Loss. In practice, directly minimizing $\mathcal{L}_\theta$ starting from randomly initialized θ is difficult. Thus, we include a warmup stage where we train the hypernetwork to mimic the embedding parameters of the original tokenizer, akin to MIMICK (Pinter et al., 2017).

$$\mathcal{L}_\theta^{\text{warmup}} = \|H_\theta(\mathcal{V}_a, T_a) - \phi_a)\|_2$$

The warmup stage is substantially quicker than the main stage because there is no need to propagate through the main model. We found it prevents divergence in some cases. Afterwards, we add an auxiliary loss, which, for every token in the sampled vocabulary $\mathcal{V}_b$ that also exists in the original vocabulary $\mathcal{V}_a$, penalizes the distance to the corresponding embedding in ϕ_a.

$$\mathcal{L}_\theta^{\text{aux}} = \frac{1}{|\mathcal{V}_a \cap \mathcal{V}_b|} \sum_{t \in |\mathcal{V}_a \cap \mathcal{V}_b|} \|H_\theta(\mathcal{V}_b, T_b)[\mathcal{V}_b[t]] - \phi_a[\mathcal{V}_a[t]]\|_2$$

This penalizes drift from the warmup stage. Combining it with the main loss yields the final loss.

$$\mathcal{L}_\theta^{\text{final}} = \mathcal{L}_\theta(T_b(x), H_\theta(\mathcal{V}_b, T_b), \psi) + \alpha \cdot \mathcal{L}_\theta^{\text{aux}}$$

The hyperparameter α weighs the contribution of the auxiliary loss. Since $H_\theta(\mathcal{V}_b, T_b)$ is also required for the main loss, it requires negligible extra computation. The auxiliary loss is necessary especially for models with separate input and output embedding matrices as shown in Appendix B.

3.2 Hypernetwork Architecture

It remains to define the hypernetwork architecture, that is, how to map the tokenizer $(\mathcal{V}_b, T_b)$ to the embedding parameters ϕ_b. To this end, we represent the new tokens $t_b \in \mathcal{V}_b$ by decomposing them using the original tokenization function T_a, and embedding them with the original embeddings E_{ϕ_a}.[6] This sequence of embeddings is passed through multiple Transformer layers, plus a separate prediction head for the input embeddings and output embeddings ϕ_b^{in} and ϕ_b^{out}. The hypernetwork thus consists of *another language model* which is applied separately for every token. We refer to the hypernetwork's language model as HLM_θ. HLM_θ can be thought of as learning how to compose

[6] In the multilingual case, we also append an element containing a learnable language-specific embedding.

Hypernetwork is another LM i.e HLM

Table 1: Accuracy on XNLI when *reusing* adapters trained for the original XLM-R model with new zero-shot transferred language-specific tokenizers. Also shown are the absolute change in accuracy from applying our hypernetwork (Δaccuracy) and the average decrease in token length of the language-specific tokenizers over the original tokenizer (Δlength).

	ar	**bg**	**de**	**el**	**en**	**es**	**fr**	**hi**	**ru**	**sw**	**tr**	**ur**	**vi**	**Avg.**
original	68.9	75.6	74.7	73.7	82.3	76.9	76.8	68.4	72.9	63.5	72.2	64.7	73.1	72.6
Lexical	58.7	63.1	65.3	61.7	72.8	68.4	66.7	61.8	62.3	51.8	58.5	60.0	72.0	63.3
FVT	63.9	70.3	70.9	67.4	79.0	73.9	71.9	65.7	67.8	57.1	66.3	61.7	72.9	68.4
OFA	57.3	64.2	67.3	62.8	73.6	68.6	68.4	61.8	63.1	54.8	59.7	59.3	72.3	64.1
FOCUS	64.8	71.0	71.6	67.7	79.6	74.4	72.6	64.5	68.1	55.7	67.3	61.9	72.6	68.6
ours	**67.9**	**73.9**	**74.1**	**71.4**	**81.1**	**76.2**	**74.7**	**67.7**	**70.7**	**62.3**	**68.7**	**63.2**	**73.9**	**71.2**
Δaccuracy	-1%	-2%	-1%	-2%	-1%	-1%	-2%	-1%	-2%	-1%	-3%	-2%	+1%	-1%
Δlength	-22%	-14%	-13%	-23%	-9%	-11%	-12%	-13%	-13%	-19%	-15%	-9%	-3%	-14%

Accuracy is within 3% of original

Table 2: Performance of Mistral-7B-v0.1 after zero-shot and n-shot tokenizer transfer (training on 800M tokens). We evaluate transfer to the GPT2 tokenizer on natural language benchmarks and transfer to the StarCoder tokenizer on HumanEvalPack. Note that continued training with the original tokenizer (*original@800M*) does not consistently improve performance.

#shots	**Method**	**Natural Language (→ GPT2 Tok.)**						**Code (pass@1) (→ StarCoder Tok.)**					
		PiQA	HS	ARC	BoolQ	MMLU	Avg.	HumanEvalPack js	go	py	cpp	java	Avg.
	original	80.7	81.0	79.5	83.6	59.6	76.9	28.7	20.1	29.3	29.9	32.3	28.1
	original@800M	82.1	82.7	80.6	80.6	57.8	76.8	31.7	19.5	28.7	27.4	26.2	26.7
0-shot	FOCUS	69.2	63.8	45.7	60.4	38.8	55.6	21.9	1.8	0.0	20.1	22.6	13.3
0-shot	ours	**79.7**	**77.5**	**73.0**	**81.9**	**53.0**	**73.0**	**23.8**	**17.7**	**18.9**	**28.7**	**26.8**	**23.2**
n-shot	FOCUS@800M	74.8	74.3	72.4	73.3	48.9	68.7	24.4	17.1	22.6	22.6	26.2	22.6
n-shot	ours@800M	**80.9**	**80.7**	**77.8**	**80.7**	**54.4**	**74.9**	**28.0**	**25.0**	**26.2**	**29.9**	**28.7**	**27.6**

the sequence of tokens $T_a(t)$—which any given token is decomposed into—into one embedding, as illustrated in Figure 2. Importantly, we do not take the tokenization function into account. By sampling diverse tokenizers during the training process, we aim for the hypernetwork to learn to produce a single embedding suitable to a wide variety of different tokenization functions. We analyze the impact of this choice later in Section 5. We also experiment with hypernetworks which do take the tokenization function into account in Appendix C.

On Token Decomposition. The input to the hypernetwork consists of the sequence of tokens $T_a(t)$ that any given token is *decomposed* into. However, this decomposition is not always trivial: for example, T_a could be character-level, while the token t could be in the vocabulary of a byte-level tokenizer T_b. In this case, t could be any arbitrary sequence of bytes (not necessarily valid UTF-8). To solve this issue, we introduce a procedure to convert tokenizers to the byte level by adding a small amount of extra tokens to the vocabulary (c.f. Section 5). This guarantees that T_a can decompose arbitrary tokens. The embeddings of the extra vocabulary are initialized randomly and trainable alongside the hypernetwork parameters.

4 Experiments

4.1 Setup

Data. We use the English subset of the MADLAD-400 corpus (Kudugunta et al., 2023) and code from the StarCoder data (Li et al., 2023) for hypernetwork training. The sampling ratio of English to Code is 7:3 following Zhang et al. (2024). For the multilingual hypernetwork, we use a subset of 26 of the languages used in XGLM (Lin et al., 2022).[7] with data from MADLAD-400. We sample

[7]We exclude languages without whitespace between words since they would require language-specific pretokenizers (e.g. Sun, 2012). Although our method is also applicable to this case, we leave this to future work.

Table 3: Accuracy of Mistral-7B on XCOPA with language-specific tokenizers zero-shot transferred via FOCUS and our hypernetwork. The standard errors are between 2.1% and 2.3%.

	et	ht	id	it	qu	sw	ta	tr	vi	Avg.
original	46.6	51.6	58.0	65.8	48.4	51.4	54.4	56.4	59.0	54.6
FOCUS	52.0	53.0	51.2	49.2	**51.4**	54.6	54.0	55.2	49.8	52.3
ours	**53.4**	**57.2**	**60.0**	**65.6**	50.0	**57.2**	**55.8**	**57.4**	**57.2**	**57.1**
Δaccuracy	+7%	+6%	+2%	-0%	+1%	+6%	+1%	+1%	-2%	+3%
Δlength	-72%	-42%	-52%	-36%	-54%	-51%	-83%	-57%	-59%	-54%

Table 4: 5-shot accuracy of Mistral-7B on multilingual MMLU with the original tokenizer and language-specific tokenizers zero-shot transferred via FOCUS and our hypernetwork.

	original	FOCUS	ours	Δaccuracy	Δlength
German	51.6	26.2	**43.7**	-8%	-37%
Spanish	53.6	26.2	**45.9**	-8%	-32%
French	53.6	27.4	**44.8**	-9%	-30%
Italian	52.5	25.8	**42.7**	-10%	-36%
Russian	49.9	27.2	**35.1**	-15%	-47%

languages using a multinomial distribution as in Conneau & Lample (2019) with $\alpha = 0.1$. For the n-shot experiments, we also train on the StarCoder data, but substitute the English section of the MADLAD-400 corpus for Flan v2 (Longpre et al., 2023) sampled as in Soldaini et al. (2024).[8]

Evaluation. We use the standard benchmarks PiQA (Bisk et al., 2020), HellaSwag (HS Zellers et al., 2019), BoolQ (Clark et al., 2019), MMLU (Hendrycks et al., 2021) and the "easy" subset of ARC (Clark et al., 2018) for evaluation in English and the synthesis task of HumanEvalPack (Muennighoff et al., 2023) for coding evaluation. For multilingual evaluation, we use XNLI (Conneau et al., 2018), XCOPA (Ponti et al., 2020) and MMLU as machine-translated by Lai et al. (2023).

Models. To evaluate our method, we use Mistral-7B (Jiang et al., 2023) as the main decoder-style language model and XLM-R (Conneau et al., 2020) as a representative of encoder-style models.[9] We also experiment with the smaller TinyLlama-1.1B model (Zhang et al., 2024) in Appendix G.

Tokenizers. We transfer models to the GPT2 tokenizer (Radford et al., 2019) for evaluation on natural language benchmarks and to the StarCoder tokenizer (Li et al., 2023) for evaluation on code benchmarks.[10] For multilingual evaluation, we train language-specific monolingual tokenizers with a vocabulary size of 50k using SentencePiece (Kudo & Richardson, 2018) and evaluate transfer to these. We also verify that the hypernetwork is robust to the choice of vocabulary size in Appendix E.

Hypernetwork training. We train the hypernetwork for 200k gradient update steps (10k of which are MIMICK-style warmup) with a batch size of 128 tokens and a sequence length of 128 (we find it sufficient to use short sequence lengths for learning embedding parameters). For the multilingual decoder-style models, we start from the English + Code checkpoint and forgo MIMICK-style warmup, keeping other hyperparameters unchanged. We use a RoBERTa-style architecture i.e. bidirectional attention and Post-LayerNorm Transformer layers (Liu et al., 2019), but use a feedforward dimension of 2x the hidden dimension (instead of RoBERTa's 4x) for the hypernetwork. See Appendix D for a full list of hyperparameters.

Continued training details. To keep runtime comparable between training the model with hypernetwork and direct training (without hypernetwork), we run hypernetwork inference only for a subset of

Nice. Only 128 seq len to learn good HLM

[8]We use Flan v2 because we observed a strong decrease in accuracy from continuing to train on the MADLAD-400 data (even with the original tokenizer). The training data for most LLMs (including Mistral-7B) is not public, but it is plausible that this decrease stems from higher-quality data mixed in especially towards the end of training as in e.g. Groeneveld et al. (2024).

[9]Although (decoder-style) LLMs are the centerpiece of a large amount of current NLP research, encoder-style LMs have wide-ranging applications in e.g. retrieval (Khattab & Zaharia, 2020) and LLM distillation (Hsieh et al., 2023) due to their lower computational cost.

[10]We chose these tokenizers due to their popularity and comparatively efficient encoding of the target domain.

Table 5: Single model rating results on MT-Bench of transferring Mistral-7B-Instruct-v0.1 to the GPT2 tokenizer using the hypernetwork trained for the base Mistral-7B model. We use `gpt-3.5-turbo-1106` as a judge. *orig.* is the original fine-tuned model, *base* the model with the same tokenizer but embeddings substituted for the base models' embeddings. λ is the scaling factor for the weight differences in Task Arithmetic (Ilharco et al., 2023).

	original		**0-shot**		**n-shot**			
Embeddings	orig.	base	FOCUS	ours	ours@800			
λ	-	-	-	-	0.0	0.3	0.5	0.7
Score (1 to 10)	7.33	7.48	5.03	**6.56**	6.59	6.75	**6.82**	6.77

$k = 16384$ tokens in the continued training case. The subset consists of all tokens occurring in the batch, plus a uniform sample of those that do not occur. The language modeling loss is then only computed over this subset of tokens. We found in preliminary experiments that this causes only minor performance degradation. Furthermore, we use the zero-shot predicted embeddings as the target for the auxiliary loss instead of using the original embeddings. This stabilizes training. We train for 50k steps with a batch size of 32 and sequence length of 512, resulting in 'seeing' 819.2M tokens.

4.2 Zero-Shot and n-shot Results

Results for XLM-R are shown in Table 1. We take task adapters trained for the original XLM-R model on the English XNLI dataset via Poth et al. (2023) and substitute the tokenizer for our language-specific one. We compare our hypernetwork against a simple lexical baseline (copying the embeddings of overlapping tokens and initializing the rest randomly), FVT, OFA, and FOCUS (c.f. Section 2). We focus only on FOCUS in the following since it performs best among the baselines.[11] Our hypernetwork consistently outperforms all baselines and preserves accuracy to 1% on average, losing 3% in the worst case and improving by 1% in the best case, while sequences are on average 14% shorter for the language-specific tokenizers; inference is thus more than 16% faster.[12] We show in Appendix E that these results are robust to the target vocabulary size.

Table 2 shows results on English and Code for Mistral-7B. We find that ZeTT is more challenging in the decoder case: FOCUS performs roughly random in the worst case (-23.2% on BoolQ) and is reduced to 0% pass@1 on HumanEval in Python. The hypernetwork goes a long way in closing this gap but still falls behind on some benchmarks. However, continuing to train the hypernetwork with the target tokenizer closes the gap almost completely. In fact, continued training on 800M tokens with the StarCoder tokenizer performs *better* than continued training for the same amount of tokens with the original tokenizer, potentially because the StarCoder tokenizer is more well suited towards code; it results in approx. 10% less tokens on average. Also, notably, continued training with the original tokenizer slightly *degrades* performance on average; this may be due to a higher-quality data mix used for pretraining Mistral-7B, whereas we use public data sources (c.f. Section 4.1).

Results of the multilingual hypernetwork for Mistral-7B are shown in Table 3 and Table 4. On XCOPA, the hypernetwork on average improves performance over the original model, while also more than halving sequence length. XCOPA performance is close to random in some languages (e.g. Southern Quechua (qu) and Estonian (et)), so we also evaluate on multilingual MMLU. Here, although the hypernetwork clearly outperforms FOCUS (which performs close to random), there is still a substantial gap to the original model; this could presumably be fixed via continued training.

4.3 Applying a Hypernetwork trained for a Base Model to Fine-Tuned Models

So far, we have shown that the hypernetwork can be successfully applied for transferring the tokenizer of the base model[13] it was trained on. However, a large amount of the models used by practitioners are fine-tuned versions of base models, e.g. via SFT or RLHF (Ouyang et al., 2022). We now attempt to answer the question: *Given a hypernetwork trained for a base model, can we apply this hypernetwork to fine-tuned versions of the same model without any extra training?* This would act

[11] We note, however, that FVT comes close to FOCUS' performance without requiring auxiliary embeddings so it may be a better choice for practical applications.

[12] 1/(1-14%)=16%, plus additional speedup due to attention scaling quadratically with sequence length.

[13] We refer to models purely pretrained on the Language Modeling task as *base models*.

Table 6: Probability of pretokens sampled from the English MADLAD-400 data to be tokenized equivalently to the original tokenization when converting the tokenizer to byte-level (*To Byte-Level*) or to UnigramLM (*Unigramify*). Also shown is the LMs bits-per-character when applying the original vs. the corresponding UnigramLM tokenizer. Bits-per-character can not be measured for conversion to byte-level since extra tokens are added in this process (which there are no embeddings for).

Kind		**BERT** WordPiece	**Mistral-7B** BPE	**TinyLlama-1.1B** BPE	**GPT2** BBPE
Original	p(preserved)	100%	100%	100%	100%
	bits per char	n/a	0.675	0.747	0.930
To Byte-Level	p(preserved)	99.6%	99.9%	99.9%	100%
	Extra Tokens	162	522	362	0
Unigramify	p(preserved)	99.4%	99.8%	99.8%	99.7%
	bits per char	n/a	0.678	0.750	0.932

Table 7: Bits-per-character of GPT2 with the original tokenizer and the tokenization function being original (left), unigramified (middle) and UnigramLM with scores set to the substring frequency of the tokens (right). We compare the original embeddings with embeddings predicted from our hypernetwork, with or without Gaussian noise in the sampling process.

Model	**Embeddings**	**Tokenizer** $(\mathcal{V}, T)$ (GPT2, GPT2)	(GPT2, unigramify(GPT2))	(GPT2, UnigramLM)
GPT2	original	0.930	0.932	1.005
	ours	**0.919**	**0.920**	**0.964**
	ours (no noise)	0.925	0.926	0.978

as a multiplying factor for the hypernetwork's applicability. First, we observe that the embedding space of a fine-tuned model is compatible with that of the base model: the embeddings of the fine-tuned Mistral-7B-Instruct-v0.1 have an average cosine similarity of 98.6% to the corresponding embedding in the base model while the average cosine similarity of the mean embedding vector is 17.4%.[14] Embedding compatibility also holds true for other models (Appendix G). The predictions of a hypernetwork trained for a base model can thus be used out-of-the-box with fine-tuned models. We verify that this is the case by evaluating Mistral-7B-Instruct-v0.1 transferred to the GPT2 tokenizer on the corrected[15] version of MT-Bench (Zheng et al., 2023). For n-shot transfer, since we train the full model we also need a way to transfer the non-embedding parameters; we achieve this via Task Arithmetic (Ilharco et al., 2023). Results are shown in Table 5.

The transferred fine-tuned model performs well, coming within approx. 0.5 score of the original model. Also, curiously, the fine-tuned model with the original tokenizer performs *better* when using the embeddings of the (not fine-tuned) base model; this may be a prudent direction for future work.

5 Discussion

Converting tokenizers to byte-level. As per Section 3.2, we need a procedure to convert tokenizers to the byte level to ensure that token decomposition is always possible. This is trivial in most cases; the bytes just need to be added to the vocabulary. BPE is an exception: here, we need to change the atomic units on which merges are defined from characters to bytes. This can be achieved by adding merges to assemble the characters used by the tokenizer from their constituent bytes to the beginning of the merge table. We measure the success of the conversion to byte level as the probability that, given some pretoken sampled from a corpus, this pretoken results in the same token sequence in the original and the converted tokenizer. Results are shown in Table 6.

Converting tokenizers to UnigramLM. We also introduce a procedure to convert arbitrary tokenizers to tokenizers using UnigramLM as the tokenization function. We refer to this process as *unigramifying* (details in Appendix A). An important assumption of the hypernetwork training is that by using the UnigramLM parametrization with scores distributed as Gaussians we can cover a sufficiently

Key Assumption

[14] Averaged across the input and the output embeddings.

[15] Using the corrections from `https://github.com/InflectionAI/Inflection-Benchmarks`.

Table 8: Parameter count and FLOPs estimates for our hypernetwork (and the corresponding main model) in different setups. The relatively lower computational cost compared to parameter count is mainly due to forgoing de-embedding which contributes significantly to FLOPs (Kaplan et al., 2020).

	Model		Hypernet	
	#params	FLOPs / token	#params	FLOPs / token
GPT2	124M	253M	21M (16%)	4.5M (1.8%)
TinyLlama-1.1B	1.1B	2.1G	170M (15%)	33.1M (1.6%)
Mistral-7B	7.2G	15.4G	678M (9%)	132.1M (0.9%)

diverse distribution of tokenizers to enable the hypernetwork to generalize to e.g. BPE tokenizers. Unigramifying allows us to check if, in principle, this is possible. Luckily, we find that it is: unigramifying results in minimal performance degradation when substituting the original tokenizer with the corresponding UnigramLM tokenizer (Table 6). Although this does not guarantee that our distribution of tokenizers is sufficiently diverse, our empirical results suggest it is (cf. Section 4.2).

We believe our conversion methods to UnigramLM and to byte-level will simplify further research into tokenizer transfer, showing that *the wildly heterogeneous landscape of tokenizers can be well approximated via byte-level UnigramLM tokenizers.*

What is the effect of amortizing over the tokenization function? As described earlier in Section 3, we 'amortize' over the tokenization function, that is, the tokenization function is not an input to our hypernetwork. We find that the predicted amortized embeddings are robust to the choice of tokenization function. For example, the set of embeddings predicted for the GPT2 vocabulary has low bits-per-character for both the original GPT2 tokenization function and a different UnigramLM tokenization function with scores based on token frequencies (Table 7). This is not the case for the original GPT2 embeddings: while they (as expected) perform well with the original GPT2 tokenizer, there is significant performance degradation when switching to the frequency-based UnigramLM tokenization function. This calls into question prior work copying the embeddings of overlapping tokens for transfer across tokenizers (Dobler & de Melo, 2023; Gee et al., 2022, among others), indicating that *even if there is an exactly overlapping token in the original tokenizer, it is not necessarily the optimal initialization of the corresponding token in the new tokenizer*.

Although we amortize over most of the aspects of the tokenization function, in practice, tokenization functions rely on a considerable amount of engineering, so it is not possible to amortize over everything; we discuss remaining assumptions in Appendix H.

Analyzing computational overhead of the hypernetwork. We estimate the FLOPs per token of multiple hypernetworks in Table 8.[16] Given a batch size n and sequence length s for the main model, and using the hypernetwork to compose k token sequences of length t, the FLOPs per batch will be $n \times s \times (\frac{\text{FLOPs}}{\text{token}})_{\text{main}} + k \times t \times (\frac{\text{FLOPs}}{\text{token}})_{\text{hypernet}}$. Taking hypernet training for Mistral-7B as an example with $n = s = 128$, $k = 32768$ and $t = 7$ the FLOPs per batch will be 252T + 30T i.e. a 12% overhead from applying the hypernet. Notably, we observed in preliminary experiments that a hypernetwork size of three layers is sufficient, regardless of model size, so the relative overhead decreases with increased amounts of layers in the main model (as also evident in Table 8).

3 layers is sufficient

6 Conclusion

We have established *Zero-Shot Tokenizer Transfer (ZeTT)*, the difficult problem of transferring language models to a new tokenizer without any training. We have found that prior heuristics for embedding initialization provide a first baseline for ZeTT, but fall short in many cases. To establish a much stronger baseline, we introduced a hypernetwork-based approach that closes the gap to a large extent, and can be further improved via continued training on a few (<1B) tokens. Due to preserving the embedding space of the original model, ZeTT can be applied to e.g. reusing adapters trained for the original model with a different tokenizer, and to transferring fine-tuned models to a new tokenizer using a hypernetwork trained for the base model. In aggregate, this work is a substantial step towards *detaching* language models from their tokenizer, increasing their flexibility and reusability.

[16]We estimate FLOPs on the basis of XLA-compiled instructions using Jax (Bradbury et al., 2018).

Acknowledgments

This work has been supported by a Royal Society University Research Fellowship *'Inclusive and Sustainable Language Technology for a Truly Multilingual World'* (no 221137; 2022-) awarded to Ivan Vulić. Research supported with Cloud TPUs from Google's TPU Research Cloud (TRC). We thank Markus Frohmann, Marcell Fekete and Piotr Nawrot for helpful feedback on a draft of this paper, and Arduin Findeis for many valuable discussions during the entirety of this project.

References

Orevaoghene Ahia, Sachin Kumar, Hila Gonen, Jungo Kasai, David Mortensen, Noah Smith, and Yulia Tsvetkov. Do all languages cost the same? tokenization in the era of commercial language models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 9904–9923, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.614. URL https://aclanthology.org/2023.emnlp-main.614.

Samuel Ainsworth, Jonathan Hayase, and Siddhartha Srinivasa. Git re-basin: Merging models modulo permutation symmetries. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=CQsmMYmlP5T.

Mikel Artetxe, Sebastian Ruder, and Dani Yogatama. On the cross-lingual transferability of monolingual representations. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4623–4637, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.421. URL https://aclanthology.org/2020.acl-main.421.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

Sidney Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, Usvsn Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. GPT-NeoX-20B: An open-source autoregressive language model. In Angela Fan, Suzana Ilic, Thomas Wolf, and Matthias Gallé (eds.), *Proceedings of BigScience Episode #5 – Workshop on Challenges & Perspectives in Creating Large Language Models*, pp. 95–136, virtual+Dublin, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.bigscience-1.9. URL https://aclanthology.org/2022.bigscience-1.9.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Yihong Chen, Kelly Marchisio, Roberta Raileanu, David Ifeoluwa Adelani, Pontus Stenetorp, Sebastian Riedel, and Mikel Artetxe. Improving language plasticity via pretraining with active forgetting. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=jvEbQBxd8X.

Hyung Won Chung, Thibault Fevry, Henry Tsai, Melvin Johnson, and Sebastian Ruder. Rethinking embedding coupling in pre-trained language models. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=xpFFI_NtgpW.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 2924–2936, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1300. URL https://aclanthology.org/N19-1300.

Jonathan H. Clark, Dan Garrette, Iulia Turc, and John Wieting. Canine: Pre-training an efficient tokenization-free encoder for language representation. *Transactions of the Association for Computational Linguistics*, 10: 73–91, 2022. doi: 10.1162/tacl_a_00448. URL https://aclanthology.org/2022.tacl-1.5.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*, 2018.

Alexis Conneau and Guillaume Lample. Cross-lingual language model pretraining. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/c04c19c2c2474dbf5f7ac4372c5b9af1-Paper.pdf.

Alexis Conneau, Ruty Rinott, Guillaume Lample, Adina Williams, Samuel Bowman, Holger Schwenk, and Veselin Stoyanov. XNLI: Evaluating cross-lingual sentence representations. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2475–2485, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1269. URL https://aclanthology.org/D18-1269.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 8440–8451, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.747. URL https://aclanthology.org/2020.acl-main.747.

Gautier Dagan, Gabriel Synnaeve, and Baptiste Rozière. Getting the most out of your tokenizer for pre-training and domain adaptation, 2024.

Wietse de Vries and Malvina Nissim. As good as new. how to successfully recycle English GPT-2 to make models for other languages. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pp. 836–846, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-acl.74. URL https://aclanthology.org/2021.findings-acl.74.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

Konstantin Dobler and Gerard de Melo. FOCUS: Effective embedding initialization for monolingual specialization of multilingual models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 13440–13454, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.829. URL https://aclanthology.org/2023.emnlp-main.829.

Leonidas Gee, Andrea Zugarini, Leonardo Rigutini, and Paolo Torroni. Fast vocabulary transfer for language model compression. In Yunyao Li and Angeliki Lazaridou (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 409–416, Abu Dhabi, UAE, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-industry.41. URL https://aclanthology.org/2022.emnlp-industry.41.

Siavash Golkar, Mariel Pettee, Michael Eickenberg, Alberto Bietti, Miles Cranmer, Geraud Krawezik, Francois Lanusse, Michael McCabe, Ruben Ohana, Liam Parker, Bruno Régaldo-Saint Blancard, Tiberiu Tesileanu, Kyunghyun Cho, and Shirley Ho. xval: A continuous number encoding for large language models. In *NeurIPS 2023 AI for Science Workshop*, 2023. URL https://openreview.net/forum?id=KHDMZtoF4i.

Dirk Groeneveld, Iz Beltagy, Pete Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Harsh Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, Shane Arora, David Atkinson, Russell Authur, Khyathi Chandu, Arman Cohan, Jennifer Dumas, Yanai Elazar, Yuling Gu, Jack Hessel, Tushar Khot, William Merrill, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew E. Peters, Valentina Pyatkin, Abhilasha Ravichander, Dustin Schwenk, Saurabh Shah, Will Smith, Nishant Subramani, Mitchell Wortsman, Pradeep Dasigi, Nathan Lambert, Kyle Richardson, Jesse Dodge, Kyle Lo, Luca Soldaini, Noah A. Smith, and Hannaneh Hajishirzi. Olmo: Accelerating the science of language models. *Preprint*, 2024.

David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rkpACe1lx.

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

Valentin Hofmann, Hinrich Schuetze, and Janet Pierrehumbert. An embarrassingly simple method to mitigate undesirable properties of pretrained language model tokenizers. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 385–393, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-short.43. URL https://aclanthology.org/2022.acl-short.43.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 8003–8017, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.507. URL https://aclanthology.org/2023.findings-acl.507.

IBM ILOG. V22.1: User's manual for cplex. 2022.

Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=6t0Kwf8-jrj.

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. Mistral 7b, 2023.

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

Omar Khattab and Matei Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, pp. 39–48, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450380164. doi: 10.1145/3397271.3401075. URL https://doi.org/10.1145/3397271.3401075.

Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. In Iryna Gurevych and Yusuke Miyao (eds.), *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 66–75, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1007. URL https://aclanthology.org/P18-1007.

Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In Eduardo Blanco and Wei Lu (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 66–71, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-2012. URL https://aclanthology.org/D18-2012.

Sneha Kudugunta, Isaac Caswell, Biao Zhang, Xavier Garcia, Christopher A. Choquette-Choo, Katherine Lee, Derrick Xin, Aditya Kusupati, Romi Stella, Ankur Bapna, and Orhan Firat. Madlad-400: A multilingual and document-level large audited dataset, 2023.

Viet Lai, Chien Nguyen, Nghia Ngo, Thuat Nguyen, Franck Dernoncourt, Ryan Rossi, and Thien Nguyen. Okapi: Instruction-tuned large language models in multiple languages with reinforcement learning from human feedback. In Yansong Feng and Els Lefever (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 318–327, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-demo.28. URL https://aclanthology.org/2023.emnlp-demo.28.

Sander Land and Max Bartolo. Fishing for magikarp: Automatically detecting under-trained tokens in large language models, 2024.

Raymond Li, Loubna Ben allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia LI, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Joel Lamy-Poirier, Joao Monteiro, Nicolas Gontier, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Ben Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason T Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Urvashi Bhattacharyya, Wenhao Yu, Sasha Luccioni, Paulo Villegas, Fedor Zhdanov, Tony Lee, Nadav Timor, Jennifer Ding, Claire S

Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro Von Werra, and Harm de Vries. Starcoder: may the source be with you! *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=KoFOg41haE. Reproducibility Certification.

Jindřich Libovický, Helmut Schmid, and Alexander Fraser. Why don't people use character-level machine translation? In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 2470–2485, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.194. URL https://aclanthology.org/2022.findings-acl.194.

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. Few-shot learning with multilingual generative language models. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 9019–9052, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.616. URL https://aclanthology.org/2022.emnlp-main.616.

Yihong Liu, Peiqin Lin, Mingyang Wang, and Hinrich Schütze. Ofa: A framework of initializing unseen subword embeddings for efficient large-scale multilingual continued pretraining. *arXiv preprint arXiv:2311.08849*, 2023.

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

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, et al. The flan collection: Designing data and methods for effective instruction tuning. *arXiv preprint arXiv:2301.13688*, 2023.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Kelly Marchisio, Patrick Lewis, Yihong Chen, and Mikel Artetxe. Mini-model adaptation: Efficiently extending pretrained models to new languages via aligned shallow training. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 5474–5490, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.338. URL https://aclanthology.org/2023.findings-acl.338.

Sabrina J. Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Gallé, Arun Raja, Chenglei Si, Wilson Y. Lee, Benoît Sagot, and Samson Tan. Between words and characters: A brief history of open-vocabulary modeling and tokenization in nlp, 2021.

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

Benjamin Minixhofer, Fabian Paischer, and Navid Rekabsaz. WECHSEL: Effective initialization of subword embeddings for cross-lingual transfer of monolingual language models. In Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz (eds.), *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 3992–4006, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.naacl-main.293. URL https://aclanthology.org/2022.naacl-main.293.

Niklas Muennighoff, Qian Liu, Armel Zebaze, Qinkai Zheng, Binyuan Hui, Terry Yue Zhuo, Swayam Singh, Xiangru Tang, Leandro von Werra, and Shayne Longpre. Octopack: Instruction tuning code large language models. *arXiv preprint arXiv:2308.07124*, 2023.

Piotr Nawrot, Jan Chorowski, Adrian Lancucki, and Edoardo Maria Ponti. Efficient transformers with dynamic token pooling. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6403–6417, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.353. URL https://aclanthology.org/2023.acl-long.353.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Gray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=TG8KACxEON.

Jupinder Parmar, Shrimai Prabhumoye, Joseph Jennings, Mostofa Patwary, Sandeep Subramanian, Dan Su, Chen Zhu, Deepak Narayanan, Aastha Jhunjhunwala, Ayush Dattagupta, Vibhu Jawa, Jiwei Liu, Ameya Mahabaleshwarkar, Osvald Nitski, Annika Brundyn, James Maki, Miguel Martinez, Jiaxuan You, John Kamalu, Patrick LeGresley, Denys Fridman, Jared Casper, Ashwath Aithal, Oleksii Kuchaiev, Mohammad Shoeybi, Jonathan Cohen, and Bryan Catanzaro. Nemotron-4 15b technical report, 2024.

Aleksandar Petrov, Emanuele La Malfa, Philip Torr, and Adel Bibi. Language model tokenizers introduce unfairness between languages. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 36963–36990. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/74bb24dca8334adce292883b4b651eda-Paper-Conference.pdf.

Yuval Pinter, Robert Guthrie, and Jacob Eisenstein. Mimicking word embeddings using subword rnns. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 102–112, 2017.

Edoardo Maria Ponti, Goran Glavaš, Olga Majewska, Qianchu Liu, Ivan Vulić, and Anna Korhonen. XCOPA: A multilingual dataset for causal commonsense reasoning. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2362–2376, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.185. URL https://aclanthology.org/2020.emnlp-main.185.

Clifton Poth, Hannah Sterz, Indraneil Paul, Sukannya Purkayastha, Leon Engländer, Timo Imhof, Ivan Vulić, Sebastian Ruder, Iryna Gurevych, and Jonas Pfeiffer. Adapters: A unified library for parameter-efficient and modular transfer learning. In Yansong Feng and Els Lefever (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 149–160, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-demo.13. URL https://aclanthology.org/2023.emnlp-demo.13.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Phillip Rust, Jonas Pfeiffer, Ivan Vulić, Sebastian Ruder, and Iryna Gurevych. How good is your tokenizer? on the monolingual performance of multilingual language models. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *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)*, pp. 3118–3135, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.243. URL https://aclanthology.org/2021.acl-long.243.

Omer Sagi and Lior Rokach. Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(4):e1249, 2018.

Timo Schick and Hinrich Schütze. Attentive mimicking: Better word embeddings by attending to informative contexts. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 489–494, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1048. URL https://aclanthology.org/N19-1048.

Timo Schick and Hinrich Schütze. BERTRAM: Improved word embeddings have big impact on contextualized model performance. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (eds.), *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 3996–4007, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.368. URL https://aclanthology.org/2020.acl-main.368.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In Katrin Erk and Noah A. Smith (eds.), *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL https://aclanthology.org/P16-1162.

Luca Soldaini, Rodney Kinney, Akshita Bhagia, Dustin Schwenk, David Atkinson, Russell Authur, Ben Bogin, Khyathi Chandu, Jennifer Dumas, Yanai Elazar, Valentin Hofmann, Ananya Harsh Jha, Sachin Kumar, Li Lucy, Xinxi Lyu, Nathan Lambert, Ian Magnusson, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew E. Peters, Abhilasha Ravichander, Kyle Richardson, Zejiang Shen, Emma Strubell, Nishant Subramani, Oyvind Tafjord, Pete Walsh, Luke Zettlemoyer, Noah A. Smith, Hannaneh Hajishirzi, Iz Beltagy, Dirk Groeneveld, Jesse Dodge, and Kyle Lo. Dolma: an Open Corpus of Three Trillion Tokens for Language Model Pretraining Research. *arXiv preprint*, 2024.

Junyi Sun. Jieba chinese word segmentation tool. 2012.

Yi Tay, Vinh Q. Tran, Sebastian Ruder, Jai Gupta, Hyung Won Chung, Dara Bahri, Zhen Qin, Simon Baumgartner, Cong Yu, and Donald Metzler. Charformer: Fast character transformers via gradient-based subword tokenization. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=JtBRnrlOEFN`.

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. Llama 2: Open foundation and fine-tuned chat models, 2023.

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

Omri Uzan, Craig W. Schmidt, Chris Tanner, and Yuval Pinter. Greed is all you need: An evaluation of tokenizer inference methods, 2024.

Junxiong Wang, Tushaar Gangavarapu, Jing Nathan Yan, and Alexander M Rush. Mambabyte: Token-free selective state space model, 2024.

Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, and Ludwig Schmidt. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 23965–23998. PMLR, 17–23 Jul 2022. URL `https://proceedings.mlr.press/v162/wortsman22a.html`.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306, 2022. doi: 10.1162/tacl_a_00461. URL `https://aclanthology.org/2022.tacl-1.17`.

Prateek Yadav, Derek Tam, Leshem Choshen, Colin Raffel, and Mohit Bansal. TIES-merging: Resolving interference when merging models. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=xtaX3WyCj1`.

Lili Yu, Daniel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. MEGABYTE: Predicting million-byte sequences with multiscale transformers. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=JTmO2V9Xpz`.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472. URL `https://aclanthology.org/P19-1472`.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. Tinyllama: An open-source small language model, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric. P Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023.

A Unigramifying: Approximating Arbitrary Tokenizers via UnigramLM

We introduce a procedure to convert arbitrary tokenizers to UnigramLM in an optimal (but lossy) way which we refer to as *unigramifying*. Given a text x and the sequence of tokens $T(x)$, for the UnigramLM tokenizer $\hat{T}$ to be equivalent to T, it is necessary that $\hat{T}$ fulfills $\sum_{t \in T(x)} \log p_{\hat{T}}(t) > \sum_{t \in C} \log p_{\hat{T}}(t)$ for all C in $\mathcal{C}_x \setminus \{T(x)\}$.[17] Thus, given a corpus of texts X we can formulate a loss

$$\mathcal{L}_{\mathcal{T}}(X, \hat{T}) = \sum_{x \in X} \sum_{C \in \mathcal{C}_x \setminus \{T(x)\}} \max\left(0, \sum_{t \in T(x)} \log p_{\hat{T}}(t) - \sum_{t \in C} \log p_{\hat{T}}(t)\right)$$

which is zero if and only if the condition above is satisfied for all texts in X. This objective is piecewise linear, so it can be converted to a standard Linear Programming (LP) form and solved via an LP solver. In practice, we use the `CPLEX v22.1` (IBM ILOG, 2022) solver. Since applying the procedure to a corpus directly would be costly, we first pre-tokenize the training corpus, then count the pretokens, and choose the top $n = 1000000$ pretokens as the set X.

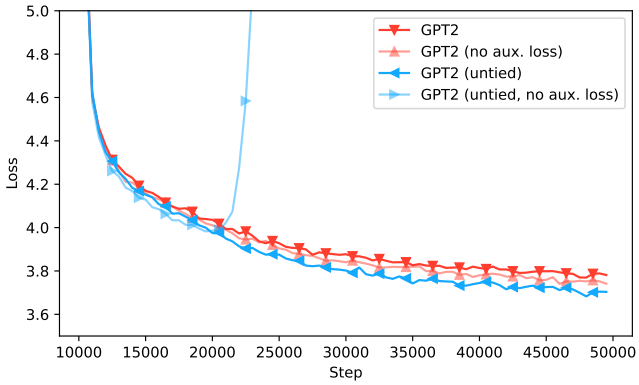

Figure 3: Language modeling loss of GPT2, and GPT2 with untied weight embeddings with and without the auxiliary loss across the first 50k training steps, excluding MIMICK-style warmup.

B Stabilization Effect of the Auxiliary Loss

We found in preliminary experiments that the auxiliary loss is necessary, especially for models that do not share embedding parameters between the input and the output (models with *untied* embeddings). To validate this hypothesis, we conducted an experiment where we manually untied the embeddings of GPT2 i.e. used a separate hypernetwork prediction head for the input and the output embeddings. Although everything else is kept the same, the untied GPT2 model diverges without the auxiliary loss, whereas the original GPT2 trains as expected, even without an auxiliary loss (Figure 3).

C Non-Amortizing Hypernetworks

We experimented with hypernetworks taking the tokenization function into account by adding *sparse inter-token attention* blocks between the self-attention and the FFN in every hypernetwork layer. Sparse inter-token attention consists of two attention blocks. The first attention block attends from a fixed amount of learnable inter-token embeddings (e.g. 16, each a vector of size d_{model}) to the ith token representation of every token sequence passed to the hypernetwork. The second block attends from the ith token representation to the inter-token embeddings. This way, we factorize the attention to e.g. one $16 \times k$ attention and one $k \times 16$ attention, instead of regular the $k \times k$ self-attention

[17] This is not sufficient for equivalence since order is ignored e.g. $T(x) = \{ab, a, b\}$ and $\hat{T}(x) = \{a, b, ab\}$ fulfill the criterion but are not equivalent.

Table 9: Performance of the hypernetwork in bits-per-byte with and without inter-token attention. *Sampled Tokenizers* are tokenizers as sampled during the training loop (c.f. Algorithm 1), *en* is an English UnigramLM tokenizer. The respective vocabulary sizes are shown in brackets.

	Sampled Tokenizers (32k)	**GPT-NeoX (50k)**	**en (30k)**
ours	1.157	**0.902**	**1.054**
ours (+ inter-token attention)	**1.118**	0.904	1.103

which would be infeasibly slow for typical vocabulary sizes. We only add inter-token attention for the first token in every sequence. This improves performance on the sampled tokenizers, but does not improve performance on 'real-world' tokenizers (Table 9); investigating this mismatch is a direction for future work.

D Additional Hyperparameters

Hyperparameters for hypernetwork training are shown in Table 10. For continued training, we use the same optimizer, but sequence length of 512, batch size of 32, training for 50k steps and a constant learning rate chosen among the set $\{1e{-}6, 3e{-}6, 6e{-}6, 1e{-}5, 3e{-}5\}$ to maximize performance. The chosen learning rate is $1e{-}6$ for the runs keeping the original tokenizer (*original@800M*), $6e{-}6$ for continued training starting from FOCUS (*FOCUS@800M*) and $3e{-}6$ for continued training with the hypernetwork (*ours@800M*).

Table 10: Hypernetwork hyperparameters.

Optimizer	AdamW (Loshchilov & Hutter, 2019)
(β_1, β_2)	(0.9, 0.95)
weight decay	0.01
Max. global gradient norm	0.1
Sequence length	128
Batch size	128
Steps	200000
of which MIMICK-style warmup steps	10000
MIMICK-style warmup learning rate schedule	linear warmup to 3-e4
Main learning rate schedule	linear warmup to 6e-5 until 10k, then cosine decay to 6e-6
Tokenizer sampling	
Vocabulary size	32768
Distribution of noise level z	$\mu = \ln(10^{-5}), \sigma = 4$
Batch size m	2048
Auxiliary loss weight	0.5
Hypernetwork	
num. layers	3
max. sequence length	7 (English + Code) or 15 (multilingual)
hidden dimension	d_{model}
FFN dimension	$2d_{\text{model}}$
num. attention heads	$\min(d_{\text{model}}/64, 32)$

E Sensitivity to Tokenizer Size

Since the tokenizers we experiment with have similar vocabulary sizes (50k for the language-specific tokenizers and for GPT2, 49k for the StarCoder tokenizer) we conduct an additional experiment to quantify the sensitivity of the performance of our hypernetwork to the size of the target tokenizer. We find that although there is slight performance degradation when increasing the size of the new tokenizers' vocabulary, the hypernetwork is fairly robust to vocabulary size (Figure 4).

F Reliance on Vocabulary Overlap

Intuitively, transfer is easier the more the target has in common with the source. One way to measure commonality between the original (source) and the target tokenizer is the fraction of tokens of the

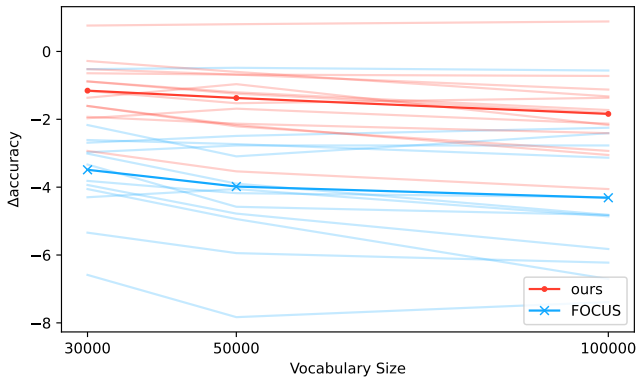

Figure 4: Difference in accuracy to the original XLM-R model on XNLI of our method and FOCUS across vocabularies with size 30k, 50k, and 100k of the new tokenizer.

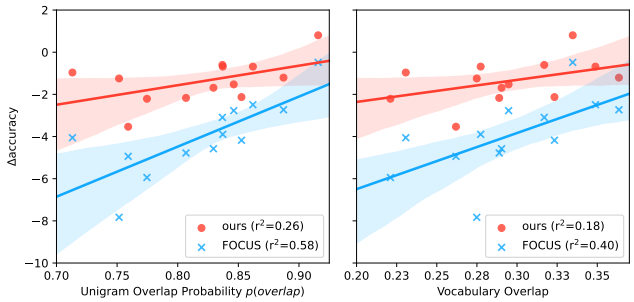

Figure 5: Correlation of the difference in accuracy to the original XLM-R model with Unigram overlap probability $p(\text{overlap})$ (left) and vocabulary overlap (right).

target vocabulary which also exist in the source vocabulary (*vocabulary overlap*). Performance correlates with vocabulary overlap, but it correlates more strongly with the probability for tokens to overlap: that is, when randomly sampling some token from a corpus tokenized with T_b, the probability that this token also exists in the vocabulary of T_a. We refer to this metric as $p(\textit{overlap})$. $p(\textit{overlap})$ has higher correlation with the performance of FOCUS, indicating that our hypernetwork depends less on overlap (Figure 5).

Table 11: Performance of TinyLlama-1.1B after zero-shot and n-shot tokenizer transfer (training on 800M tokens), compare Table 2.

#shots	Method	Natural Language (→ GPT2 Tok.)						Code (pass@1) (→ StarCoder Tok.)					
		PiQA	HS	ARC	BoolQ	MMLU	Avg.	HumanEvalPack js	go	py	cpp	java	Avg.
	original	73.1	59.1	55.2	57.2	25.5	54.0	7.3	6.7	7.3	8.5	7.9	7.5
	original@800M	73.2	59.5	63.3	65.1	26.3	57.5	9.8	7.3	9.1	8.5	10.4	9.0
0-shot	FOCUS	60.8	42.1	39.6	56.9	22.9	44.7	4.9	0.6	0.0	3.0	**7.9**	3.3
0-shot	ours	**70.5**	**55.6**	**51.4**	**62.9**	**23.7**	**52.8**	4.3	**5.5**	**4.3**	**7.3**	3.7	**5.0**
n-shot	FOCUS@800M	67.7	52.8	52.7	**66.1**	25.3	52.9	6.1	**6.1**	10.4	8.5	**8.5**	7.9
n-shot	ours@800M	**71.4**	**57.8**	**59.7**	**66.1**	**26.6**	56.3	**9.1**	**6.1**	**11.6**	**11.0**	7.3	**9.0**

Table 12: Single model rating results on MT-Bench of transferring TinyLlama-1.1B-Chat-v1.0 to the GPT2 tokenizer, compare Table 12.

	original		**0-shot**		**n-shot**			
Embeddings	orig.	base	FOCUS	ours	ours@800			
λ	-	-	-	-	0.0	0.3	0.5	0.7
Score (1 to 10)	5.5	5.7	2.7	**4.0**	4.29	4.63	**4.8**	4.43

G Additional LLM Results

Zero-shot and n-shot results for TinyLlama-1.1B are shown in Table 11 and MT-Bench results of transferring TinyLlama-1.1B-Chat-v1.0 in Table 12. We observe the same patterns as on Mistral-7B.

H Assumptions on the Tokenization Function

In practice, besides the tokenization algorithm itself (e.g. BPE, UnigramLM) tokenization functions also contain other steps, in particular *pretokenizing* text into smaller chunks (usually words) on which to apply the tokenization function (Mielke et al., 2021). In our experiments, we assume fixed pretokenization given by a regular expression based on the regular expression used by GPT2 (Radford et al., 2019), adjusted to not over-segment text in languages using characters in the Unicode Mark category within words (e.g. Hindi and Tamil). We also add a *prefix space* (i.e., a whitespace at the start of the text to tokenize) if and only if the original tokenizer also uses a prefix space. Finally, we always add whitespace characters covering sequences of consecutive whitespaces up to 16 characters long similar to Black et al. (2022) to ensure code is tokenized efficiently. These light assumptions mostly preserve the generality of our method but could be further relaxed in future work.