

LangMAP: A Language-Adaptive Approach to Tokenization

Clara Meister¹ Suchir Salhan² Andrzej Szablewski² Pietro Lesci²

Paula Buttery² Tiago Pimentel³

¹EPFL, ²University of Cambridge, ³ETH Zürich

clara.meister@epfl.ch, {sas245, as3623, pl487, pjb48}@cam.ac.uk,

tiago.pimentel@inf.ethz.ch

 [cimeister/unimixlm](https://github.com/cimeister/unimixlm)

Abstract

Language-specific tokenizers improve tokenization quality and the downstream performance of models on those languages (Rust et al., 2021; Limisiewicz et al., 2023). Using one with an existing pretrained model requires either training a new model from scratch or adapting the vocabulary of the pretrained one. We propose **Language-adaptive Maximum a Posteriori (LangMAP) Tokenization**, which extends the UnigramLM algorithm (Kudo, 2018) to the multilingual setting by replacing its single distribution over the vocabulary with one distribution per language over the same vocabulary. LangMAP can be used to train a multilingual language model from scratch, or to adapt a pretrained model’s tokenizer to individual languages without changing its vocabulary. Language labels are required to estimate the per-language distributions; at inference, the algorithm selects the highest-probability segmentation across languages, i.e., it does not need a language label. In experiments where we adapt a pretrained tokenizer with LangMAP, we observe that LangMAP’s new segmentation has consistently higher morphological boundary recall than the original tokenizer. In code data, it improves alignment with abstract syntax tree (AST) leaf boundaries in all configurations we try. Despite ostensible improvements in segmentations, in language model fine-tuning experiments, we do not observe significant differences in performance for most language–task pairs.

1 Introduction

Tokenization is a standard pre-processing step in modern language models (LMs) that converts an input string, a sequence of characters or bytes, into a sequence of tokens from a fixed, finite vocabulary. Although tokenization is in principle language-agnostic, the tokenizers used in most production systems have been optimized for English. As a result, texts from non-English lan-

guages, and in particular those with rich morphology or non-Latin scripts, are tokenized systematically worse: they are over-segmented, producing longer token sequences in which tokens often do not correspond to full words or meaningful morphemes (Petrov et al., 2023; Lee et al., 2024). Empirical studies show that these issues hurt downstream performance (Lesci et al., 2025) and inflate per-request costs for commercial APIs, introducing cross-lingual disparities in both the quality and price of available models (Ahia et al., 2023; Petrov et al., 2023, *inter alia*). Although language-specific tokenizers can address these disparities,¹ using one with an arbitrary pretrained model requires either retraining the model with a new tokenizer or adapting the model’s parameters to a new vocabulary (Downey et al., 2023; Remy et al., 2024; Feher et al., 2025). Both approaches are computationally expensive, partially offsetting the benefit of starting from a pretrained model in the first place.

Importantly, the poor segmentations induced by English-centric tokenizers are not always due to missing vocabulary coverage: in many cases, the tokenizer already contains tokens that would support a more appropriate segmentation for the target language, but their parameters induce an alternative, suboptimal decomposition.²

In this paper, we propose Language-adaptive MAP (LangMAP) Tokenization, an inherently multilingual tokenization scheme. LangMAP follows the UnigramLM tokenization algorithm (Kudo, 2018) in casting tokenization as recovering the latent segmentation of a string under a unigram

¹Language-specific tokenizers have been shown to improve segmentation quality and, after model adaptation, downstream performance for their target languages (Rust et al., 2021; Limisiewicz et al., 2023).

²As an example, the German compound Rathaus (“city hall”) has the morphological segmentation Rat + haus. In a BPE tokenizer fit mostly on English text the merge rule $t + h \rightarrow th$ has a high rank and is applied first, so even if Rat and haus are both in the vocabulary, the segmentation is not possible.

model. Given a dataset, UnigramLM learns a generative-model for it, composed of a vocabulary and its associated probability distribution. Uncovering a string’s latent segmentation then consists of selecting the maximum a posteriori (MAP) segmentation under this model. LangMAP retains this formulation but replaces the single global distribution with a collection of language-specific distributions learned over the same shared vocabulary. At inference time, we compute the MAP segmentation under each language-specific distribution and select the one attaining the highest overall likelihood across all languages. A notable consequence is that LangMAP can adapt a pretrained model’s tokenizer to individual languages without changing its vocabulary. Furthermore, although language labels are required to estimate the per-language distributions during estimation of LangMAP parameters, none are needed at tokenizer inference.

We apply LangMAP to create language-specific variants of several open-source tokenizers. Across 7 natural languages and 9 programming languages, morphological boundary recall rises consistently for the four Latin-script natural languages. Alignment with AST leaf boundaries (Meister, 2026) rises for all programming languages on every tokenizer. For the three non-Latin-script languages, the change in recall that LangMAP can induce is bottlenecked by how many entries from that script the vocabulary contains. We fine-tune Qwen2.5-0.5B and Llama-3.2-1B on five languages and measure grammatical acceptability with MultiBLiMP. For only one of the nine MultiBLiMP accuracy differences between LangMAP and the original tokenizer does the 95% confidence interval exclude zero after multiple-testing corrections. This is consistent with prior findings that tokenizers whose segmentations match a language’s morphology more closely do not necessarily give better downstream accuracy (Arnett et al., 2025). Three applications of LangMAP remain to be tested: use for fine-tuning models in the coding domain, fine-tuning for languages whose pretraining data is limited, and training a model from scratch with LangMAP rather than replacing the tokenizer of an existing model.

2 Tokenization

Let $\mathbf{s} = \langle s_1, s_2, \dots, s_N \rangle$ be a **string**, which consists of characters (or bytes) from a base alphabet Σ . Further, let $\mathcal{V}$ be a **vocabulary**, a finite set composed

of **tokens** $v \in \mathcal{V}$.³ Each of these tokens v consists of a sequence of elements from $\Sigma \cup \Gamma$, where Γ denotes a finite set of reserved symbols (e.g., whitespace markers, punctuation, end-of-string markers, etc). Thus, intuitively, each $v \in \mathcal{V}$ is itself a finite string over the extended alphabet $\Sigma \cup \Gamma$.

Tokenization is the process of converting a string $\mathbf{s}$ into a sequence of tokens $\mathbf{v} = \langle v_1, \dots, v_m \rangle$. This process, which we denote by the function $h_\phi: \Sigma^* \rightarrow \mathcal{V}^*$,⁴ can be intuitively understood as creating a different representation of the original string. A tokenization algorithm typically defines a parametrized class of such maps $\mathcal{H} \stackrel{\text{def}}{=} \{h_\phi \mid \phi \in \Phi\}$, as well as a method for learning the parameters $\phi \in \Phi$; these parameters then define (directly or indirectly) the tokenizer’s vocabulary $\mathcal{V}$. The application of h_ϕ to a string (once its parameters are learned) is referred to as **inference**.

The map h_ϕ is typically designed to be lossless. Thus, given a sequence of tokens $\mathbf{v} = h_\phi(\mathbf{s})$, we can reconstruct the input string $\mathbf{s}$ using a **detokenization function** $g: \mathcal{V}^* \rightarrow \Sigma^*$. Typically, g is as simple as concatenating the elements composing each of $\mathbf{v}$ ’s tokens, while handling its reserved symbols; formally, $g(\mathbf{v}) = v_1 \circ \dots \circ v_m$, where $\circ$ denotes string concatenation and, when applied to tokens, it indicates that the strings forming each token are concatenated together. Notably, g is generally non-injective: even when considering a fixed vocabulary $\mathcal{V}$, there are often multiple sequences $\mathbf{v}$ that yield the same $\mathbf{s}$ after the application of g . We refer to all such token sequences as **segmentations** of $\mathbf{s}$, and use $\mathcal{T}_\mathcal{V}(\mathbf{s}) \stackrel{\text{def}}{=} \{\mathbf{v} \in \mathcal{V}^* : g(\mathbf{v}) = \mathbf{s}\}$ to refer to this set. For a given h , we refer to $h(\mathbf{s})$ as the **canonical segmentation** of $\mathbf{s}$ and to any other segmentation $\mathbf{v} \in \mathcal{T}_\mathcal{V}(\mathbf{s})$ for which $\mathbf{v} \neq h(\mathbf{s})$ as a **non-canonical segmentation**. That several token sequences detokenize to the same string has motivated researchers to propose alternative tokenizer inference schemes, which replace only h —the tokenizer’s inference rules—and keep the learned vocabulary unchanged. For example, the inference strategy may find the token sequence $\mathbf{v} \in \mathcal{T}_\mathcal{V}(\mathbf{s})$ of minimal length for a given input $\mathbf{s}$ (Schmidt et al., 2024), or may greedily select the longest in-vocabulary tokens (Hofmann et al.,

³Tokens v are also sometimes called subwords; we avoid this naming because v need not align with orthographic words, in their typical definition. Further, a sequence of tokens $\mathbf{v}$ is also a string in the strict sense. We reserve the word *string* for $\mathbf{s}$ throughout.

⁴ $\{\}^*$ denotes the Kleene closure of a set. In words, Σ^* is the set of all finite strings over the alphabet Σ .

2022). Our proposed method takes this approach, adapted for the multilingual setting.

2.1 UnigramLM

The UnigramLM tokenization algorithm (Kudo, 2018) is one of the main tokenization algorithms, used, e.g., by the language models T5, PaLM 2, and XLNet (Raffel et al., 2020; Anil et al., 2023; Yang et al., 2019). It frames tokenization as a probabilistic learning problem. It assumes strings are generated via a two-stage generative model: latent token-sequences are sampled from a unigram distribution; these tokens are then concatenated to form strings. Tokenization then simply consists of uncovering a latent segmentation under this model (typically via a MAP approach). As this generative model’s parameters are unknown, UnigramLM also provides a method for estimating them. In the remainder of this section, we briefly describe the algorithm and refer to App. E for more details.

Generative Model. Let $\mathcal{V}$ be a vocabulary, and $\phi \in \Delta^{|\mathcal{V}|-1}$ denote a probability distribution over it. UnigramLM assumes strings are produced via a two stage generative model P . First, token sequences $\mathbf{v} = \langle v_1, v_2, \dots, v_m \rangle$ are generated by sampling tokens independently from ϕ :⁵

$$P(\mathbf{V}=\mathbf{v}; \phi) \stackrel{\text{def}}{\propto} \prod_{t=1}^{|\mathbf{v}|} \phi[v_t]. \quad (1)$$

where $\phi[v]$ denotes the probability of token v under ϕ . Second, these sequences deterministically induce an observed string via a fixed detokenization function g :

$$P(\mathbf{S}=\mathbf{s} \mid \mathbf{V}=\mathbf{v}) = \begin{cases} 1 & \text{if } \mathbf{s} = g(\mathbf{v}) \\ 0 & \text{else} \end{cases} \quad (2)$$

Inference Function. Given the parameters ϕ of such a generative model, UnigramLM’s inference procedure simply recovers the MAP segmentation of a string $\mathbf{s}$:

$$h_\phi(\mathbf{s}) \stackrel{\text{def}}{=} \underset{\mathbf{v} \in \mathcal{V}^*}{\operatorname{argmax}} P(\mathbf{V}=\mathbf{v} \mid \mathbf{S}=\mathbf{s}; \phi) \quad (3a)$$

$$= \underset{\mathbf{v} \in \mathcal{T}_\mathcal{V}(\mathbf{s})}{\operatorname{argmax}} \prod_{t=1}^{|\mathbf{v}|} \phi[v_t] \quad (3b)$$

⁵In order to be a valid probability distribution over token sequences, P would need to include an explicit termination mechanism (e.g., an EOS symbol). Since this is omitted here, we use $\propto$ in Eq. (1) to indicate that the expression is defined up to a normalization constant.

(A detailed derivation is shown in App. E.1.) The solution to Eq. (3) can then be found efficiently using a Viterbi-style dynamic program, which runs in $\mathcal{O}(N \cdot K_{\max})$ time, where N is the length of the input string and $K_{\max}$ is the maximal length (in characters or bytes) of any token $v \in \mathcal{V}$.

Parameter Estimation. Finally, UnigramLM also provides a way to learn the parameters ϕ of its generative model. Ideally, we would choose the ϕ that maximize the log-likelihood of a target dataset $\mathcal{C} = \{\mathbf{s}_m\}_{m=1}^M$ under our generative model: i.e., $\mathcal{L}(\mathcal{C}; \phi) = \sum_{m=1}^M \log P(\mathbf{S}=\mathbf{s}_m; \phi)$. As this is, in general, intractable, UnigramLM approximates ϕ via the EM algorithm, which lower-bounds $\mathcal{L}$ and maximizes that bound (App. E.2). Concretely, two steps are performed iteratively, either for a fixed number of steps or until changes to the objective value fall below some threshold.⁶ First, under a fixed $\phi^{(n)}$, the **E-Step** computes the expected count of each token in our corpus, i.e., the number of times it is used on average across all corpus segmentations:

$$\hat{c}_v(\mathcal{C}; \phi^{(n)}) \stackrel{\text{def}}{=} \quad (4)$$

$$\sum_{m=1}^M \sum_{\mathbf{v} \in \mathcal{T}_\mathcal{V}(\mathbf{s}_m)} P(\mathbf{V}=\mathbf{v} \mid \mathbf{S}=\mathbf{s}_m; \phi^{(n)}) c_v(\mathbf{v}) \quad (5)$$

where $c_v(\mathbf{v}) \stackrel{\text{def}}{=} \sum_{t=1}^{|\mathbf{v}|} \mathbb{1}\{v_t = v\}$ counts the occurrences of token v in $\mathbf{v}$. Second, the **M-Step** sets each parameter to that token’s normalized expected count:

$$\phi^{(n+1)}[v] = \frac{\hat{c}_v(\mathcal{C}; \phi^{(n)})}{\sum_{v' \in \mathcal{V}} \hat{c}_{v'}(\mathcal{C}; \phi^{(n)})}. \quad (6)$$

When building a tokenizer, $\mathcal{V}$ is not known in advance, and UnigramLM selects it by initializing an over-sized set and pruning after each EM iteration (App. E.3). Our algorithm uses UnigramLM’s parameter estimation but not its vocabulary selection: the vocabulary may equally be one learned by another algorithm, e.g., BPE.

2.2 Tokenization in NLP Pipelines

UnigramLM’s estimation optimizes a tokenizer for the empirical distribution of its training corpus, so a corpus dominated by a few languages or domains yields a tokenizer implicitly optimized for those

⁶ $\phi^{(0)}$ can be initialized as e.g., the uniform distribution.

regimes. Such tokenizers will logically exhibit the cross-lingual disparities described in Section 1. These disparities are relevant, as over-segmentation leads to increased API costs (Petrov et al., 2023), and tokenizer segmentation quality has been shown to affect downstream generalization in multilingual models (Hofmann et al., 2021; Klein and Tsarfaty, 2020; Liang et al., 2023; Vemula et al., 2025; Arnett and Bergen, 2025; Shani et al., 2026, *inter alia*).

A growing body of work has sought to mitigate these effects by modifying the tokenization stage or adapting pretrained models to alternative vocabularies. Broadly, existing approaches fall into two categories. The first class of approaches proposes new tokenization schemes. For example, Foroutan et al. (2025) and Limisiewicz et al. (2023) propose variants of BPE and UnigramLM, respectively, that are particularly adapted for the multilingual setting. Some other works attempt to remove or relax the dependency on a fixed token vocabulary altogether. Token-free and byte-level models operate directly on character or byte sequences (Xue et al., 2022; Ahia et al., 2024), eliminating biases incurred by vocabularies optimized for high-resource domains, but at the cost of substantially longer input sequences and increased training and inference complexity. The second class of approaches propose post-training modifications to models. Post-hoc vocabulary adaptation methods, such as merging or retrofitting token embeddings for new vocabularies (Downey et al., 2023; Remy et al., 2024; Feher et al., 2025), avoid full model retraining but still require either fine-tuning pretrained models or learning additional model parameters such as new token embeddings or auxiliary networks.

The next section introduces LangMAP, which gives every language its own parameters over one shared vocabulary. It is a tokenization scheme and an adaptation method at once: adapting a model to a further language changes neither the vocabulary nor the model’s architecture.

3 LangMAP Tokenization

We propose Language-adaptive MAP (LangMAP) Tokenization: a new tokenization algorithm that produces segmentations optimized for each language in its training set.⁷ Notably, LangMAP requires language labels during estimation, but not

at inference. LangMAP can be used either when training a language model from scratch or to adapt an already pretrained model, accepting any base vocabulary (e.g., one learned by BPE, or that of a pretrained model) without modifying it.

3.1 Learning Language-Specific Parameters

Let Ω be the set of languages under consideration and $\ell \in \Omega$ be a single language. Let $\mathcal{C}^\ell$ denote the subset of the corpus corresponding to language ℓ . Further, let $\mathcal{V}$ be a fixed base vocabulary, e.g., the vocabulary of a pretrained language model or one learned by a standard tokenization algorithm such as UnigramLM. Keeping this vocabulary fixed, we learn language-specific unigram parameters⁸ ϕ^ℓ by applying the EM procedure described in Section 2.1 independently to each language-specific corpus $\mathcal{C}^\ell$. For each language, the expected token counts in the E-step are computed (exclusively) over strings in $\mathcal{C}^\ell$, while the M-step normalizes these counts over the full (shared) vocabulary $\mathcal{V}$. As mentioned above, as $\mathcal{V}$ is fixed, there is no vocabulary learning, and the pruning step of the standard UnigramLM algorithm is skipped; only token probabilities are estimated.

3.2 Language-Adaptive Inference

At inference time, we find the most probable segmentation for each language $\ell \in \Omega$ under the unigram model with parameters ϕ^ℓ . We then compare segmentations across all languages, choosing the one with maximum unigram likelihood according to any ϕ^ℓ . Notably, this second step implicitly identifies the string’s language using UniLID, a recent method introduced by Meister et al. (2026),⁹ and then returns its segmentation. Formally:

$$\mathbf{v}_\ell = \operatorname{argmax}_{\mathbf{v} \in \mathcal{T}_\mathcal{V}(\mathbf{s})} P(\mathbf{V} = \mathbf{v}; \phi^\ell) \quad (7)$$

Standard per-language inference via UnigramLM

$$\mathbf{v}^* = \operatorname{argmax}_{\mathbf{v}_\ell \mid \ell \in \Omega} P(\mathbf{V} = \mathbf{v}_\ell; \phi^\ell) \quad (8)$$

Implicit language-identification via UniLID

Although this formulation appears to require performing the UnigramLM inference problem for each language, inference remains efficient due to the shared vocabulary. Because $\mathcal{V}$ is fixed, the

⁷We note that the algorithm is generally applicable to any partition of the corpus; we focus on languages, as LM performance according to this partition is of particular concern to the community, but perform evaluations on code as well.

⁸We re-initialize $\phi_\ell^{(0)}$ as the current token frequency distribution in the corpus.

⁹Explicitly, UniLID performs language identification using an approach identical to Eq. (8), but returning ℓ instead of $\mathbf{v}_\ell$.

segmentation lattice, which provides all valid segmentations of the input string under $\mathcal{V}$, is identical across languages and needs to be constructed only once. The per-language instantiations of the problem differ only in the edge weights induced by the language-specific token probabilities. As a result, the overall complexity scales linearly with the number of languages, requiring an additional $\mathcal{O}(|\Omega|)$ passes over a fixed lattice, rather than reconstructing the lattice itself for each model. This overhead could be reduced, if desired, by using coarser-grained categorizations, e.g., language families instead of individual languages.

3.3 Computational Complexity, Scope, and Limitations

From a computational standpoint, LangMAP differs from the standard UnigramLM tokenization algorithm only in that it maintains a separate distribution per language rather than a single shared one; its cost follows directly from this. Training amounts to running the same EM procedure once per language, each over only that language’s portion of the corpus, so the total work is comparable to a single UnigramLM run; the $|\Omega|$ fits are moreover independent and completely parallelizable. Empirically, we observed that the estimates stabilized after about one thousand documents per language, well below the 25M tokens per language on which every distribution in this paper was fit (App. D).

At inference, the segmentation lattice depends only on the (fixed) vocabulary and the input string, so it is built once and reused across all language-conditional distributions; LangMAP therefore adds only an $\mathcal{O}(|\Omega|)$ factor over standard UnigramLM decoding: one Viterbi-style pass per language over the shared lattice, plus an $\mathcal{O}(|\Omega|)$ comparison to select the most likely segmentation. Finally, because each distribution is fit independently and the vocabulary is held fixed, incremental extension to new languages or dialects can be done easily and efficiently; only the unigram distribution of the new language needs to be learned.

4 Experiments

We evaluate whether LangMAP segmentations improve tokenization *intrinsically* (measured as, e.g., compression) and *extrinsically* (measured as downstream performance after model fine-tuning). For each base tokenizer we compare intrinsic and down-

stream metrics under the unchanged base tokenizer (ORIGINAL) and LANGMAP.

4.1 Tokenizers, languages and data

Tokenizers and models. Table 4 in App. A lists the base models whose tokenizers we study here (14 in total). This list spans models from the Gemma, LLama, Mistral, and Qwen families (among others), covering distinct tokenization algorithms and vocabulary sizes. Notably, several of the model families we study use identical tokenizers across different sizes and/or versions (e.g. Gemma 2 with 2b or 9b parameters use the same tokenizer, or Llama-3.1 with 8b and Llama-3.2 with 1b). We therefore report *deduplicated* intrinsic metrics, over distinct tokenizers rather than over models. Table 1 reports MorphScore per (model family, language), measured on the LangMAP tokenizers built over each model’s own vocabulary that the downstream experiments of Section 5.3 train with, so that the intrinsic and downstream measurements describe the same tokenizers. The wider deduplicated sweep, 10 tokenizers on each of 7 languages, is reported in Table 7.

Languages and training data. We measure morphological alignment on seven natural languages, each of which has the Universal Dependencies treebank that the metric requires: German, Finnish, Hungarian, Turkish, Hindi, Arabic and Tamil. Bengali and Thai appear only in the label-free routing measurement of Table 14, and Swahili, which has no treebank, only in the additional intrinsic metrics of App. B. App. A lists all ten with their script and morphological type. We also evaluate tokenizers on nine programming languages: C, C++, C#, Go, Java, JavaScript, PHP, Python, and TypeScript, chosen to span brace-delimited vs. whitespace-sensitive and statically vs. dynamically typed conventions. The code measurements cover eight model families, listed in App. A. For LangMAP parameter estimation, we use data from FineWeb2 (Penedo et al., 2025) and The Stack (Lozhkov et al., 2024).¹⁰

Estimation. We learn each ϕ^ℓ —the per-language unigram distribution over the original tokenizer’s vocabulary—with the EM procedure of Section 3, run for $K=10$ iterations. For each language, we use 25M tokens from the aforementioned datasets (FineWeb2 for natural language; The Stack for

¹⁰<https://huggingface.co/datasets/bigcode/the-stack>

code) to perform this parameter estimation. We initialize ϕ^ℓ from a unigram distribution estimated jointly over the entire corpus via the EM procedure detailed in Section 2.1.

4.2 Tokenizer configurations

We compare five tokenizer configurations at the same training budget. ORIGINAL keeps the model’s own tokenizer. LANGMAP re-scores that same vocabulary per language. BASE-UNIGRAM applies UnigramLM segmentation over the same vocabulary with one set of language-agnostic scores, which separates the effect of the language-specific scores from the effect of switching to Unigram segmentation at all. We construct a PLACEBO variant as an additional baseline: per-language distributions are set by sorting the vocabulary by LangMAP score, dividing the sorted list into buckets of 500 pieces, and permuting the scores within each bucket. VOCABULARY-EXTENSION is the baseline for the alternative to LangMAP: changing which pieces the vocabulary contains, rather than changing the scores over a fixed vocabulary. It broadly follows the approach first proposed by Wang et al. (2020) and Chau et al. (2020). Concretely, we fit a monolingual 32k BPE tokenizer on the target language, drop the pieces already present in the base vocabulary and the pieces made of digits (165 to 210 per language), append the 16,000 most frequent of the remainder, and initialize each new embedding row from the mean of the embeddings of that piece’s decomposition under the original tokenizer (following Dobler and de Melo, 2023). This adds 14.1M trainable parameters (see App. A). The other four tokenizer configurations leave the parameter count unchanged, so the comparison between LANGMAP and VOCABULARY-EXTENSION is the comparison between re-scoring a vocabulary at no parameter cost and extending it at a parameter cost. Table 3 reports Qwen2.5-0.5B under all five configurations. Llama-3.2-1B was run with ORIGINAL and LANGMAP only.

4.3 Adaptation protocol

Extrinsic evaluations measure downstream model performance when using one of the above configurations. For these experiments, we fine-tune our base models on new language data, tokenized using the specified tokenizer.¹¹ Due to computa-

¹¹Using LangMAP out-of-the-box on a language already in the model’s pretraining data without any continued pre-training or fine-tuning of the model systematically leads to

tional constraints, we restrict these experiments to a subset of languages and a subset of the models in Table 4. We include Hindi as a negative control. Before running the experiments, we found that LangMAP had almost no effect on Hindi’s MorphScore. We therefore expected LangMAP to have little effect on Hindi in the downstream experiments. In contrast, we expected vocabulary extension to have a measurable effect. We also repeat the experiments on Llama-3.2-1B, using Turkish, Arabic, Hindi, and Hungarian, to see whether the results generalise to another model. For each combination of base model, target language, and tokenizer, we fine-tune the model on target-language FineWeb2 text, with English text making up 20% of the training data. We fine-tune with LoRA of rank 16, $\alpha=32$, dropout 0.05, learning rate 2×10^{-4} , and a fully-trainable tied embeddings. Every run takes the same number of optimizer steps. App. A gives more detail on how controls were implemented. We run 3 seeds per (language, tokenizer configuration).¹² The Llama-3.2-1B runs use the same exact training settings and data; only the base model and its vocabulary differ.

4.4 Intrinsic measurements

We evaluate LangMAP (LANGMAP) against the base tokenizer ORIGINAL using the TokEval metric suite,¹³ comparing paired (ORIGINAL, LANGMAP) tokenizations of the same input texts. For natural languages, we evaluate on the FLORES-200 parallel dataset (NLLB Team et al., 2022). For code metrics, we report on StarCoder data (Li et al., 2023). We report three of these metrics here in the body:

- **MorphScore** compares the boundaries a tokenizer places inside a word type with that word type’s gold morpheme boundaries; we report its recall, the fraction of gold boundaries the tokenizer also places, together with its precision and micro-F1, over the Universal Dependencies subsets (Arnett et al., 2025; Nivre et al., 2017).
- **AST leaf-boundary alignment** is the fraction of parse-tree leaf spans whose start and end both

worse performance; most probability mass for a string is placed on the tokenization the model saw during training, i.e., the canonical tokenization under the base tokenizer (Cao and Rimell, 2021), and so this behavior is to be expected.

¹²We extend to 5 seeds for Hungarian under a rule specified before running the experiments: namely, seeds were added when a difference has a 95% confidence interval excluding zero before correction.

¹³<https://github.com/cimeister/tokenizer-intrinsic-evals>

coincide with a token boundary.

- **Identifier fragmentation** is the fraction of programmer-defined identifiers split into more than one token.

Rényi entropy, bigram entropy, fertility, compression rate and the Gini score of per-language token counts are reported in App. B.1. We defer the reader to the TokEval library for descriptions and implementation details.

For all metrics, we report paired differences: $\Delta = \text{LANGMAP} - \text{ORIGINAL}$; the direction that indicates an improvement when using LangMAP is noted per caption. We also report 95% confidence intervals throughout. Statistical significance is assessed via a paired percentile bootstrap, with 10,000 resamples over (tokenizer, language) pairs; see App. A for details. The per-(model family, language) MorphScore values of Table 1 use a different resampling unit (see App. B.2).

4.5 Downstream measurements

We report three downstream measurements:

- **Target-language MultiBLiMP accuracy** (Jumelet et al., 2026), evaluated 0-shot (i.e., no worked examples in the prompt)
- **English Belebele accuracy** (Bandarkar et al., 2024) as a measure of English retention, evaluated 5-shot
- **FLORES+ bits per byte** in the target language and in English

MultiBLiMP is an automatically-constructed minimal-pair benchmark, derived from Universal Dependencies treebanks; each of its items presents two near-identical sentences differing only on a targeted morphosyntactic feature, and the model is scored on whether it assigns higher likelihood to the grammatical one. Belebele evaluates reading comprehension via four-way multiple-choice questions grounded in FLORES-200 passages. Target-language Belebele and Global-PIQA (Chang et al., 2025) are not reported because their accuracies in these languages are at or near chance at this model scale. We evaluate the pretrained model (NO-ADAPTATION), using its original tokenizer and without any fine-tuning. We evaluate it on the same 0-shot MultiBLiMP and 5-shot English Belebele tasks. This baseline shows how much performance changes after adapting the model to a new tokenizer. For each MultiBLiMP difference from the original tokenizer, we report the mean over seeds, the standard deviation across seeds, and a paired percentile bootstrap over items with

Lang.	<i>n</i>	$\Delta\text{micro-F1}$	$\Delta\text{Rec.}$	$\Delta\text{Prec.}$
<i>Qwen2.5-0.5B</i>				
Turkish	32,560	+0.0688*	+0.2108*	+0.0703*
Finnish	40,776	+0.0245*	+0.1332*	+0.0391*
Hungarian	7,924	+0.0350*	+0.1530*	+0.0385*
Arabic	11,256	+0.0743*	+0.0663*	+0.0350*
Hindi [†]	4,723	−0.0116*	+0.1008*	+0.0217*
<i>Llama-3.2-1B</i>				
Turkish	32,560	+0.0884*	+0.1937*	+0.0743*
Finnish	40,776	+0.0225*	+0.1151*	+0.0319*
Hungarian	7,924	+0.0413*	+0.1659*	+0.0440*
Arabic	10,603	+0.0074*	−0.0043	+0.0015
Hindi	4,723	−0.0009*	+0.0010*	−0.0000

Table 1: MorphScore difference (LANGMAP minus ORIGINAL) on the tokenizers the models of Section 5.3 were fine-tuned with; higher is better in all three columns. * marks a 95% confidence interval excluding zero. [†] Qwen2.5-0.5B Hindi recall is 1.0000 by construction: that vocabulary holds only 67 Devanagari pieces. App. B.2 gives further details, including significance testing protocol and the macro-F1 column.

seeds as replicates (10,000 resamples). Multiple testing is controlled for by the Holm procedure at $\alpha = 0.05$: over the 18 differences from the original tokenizer in Qwen2.5-0.5B, and over the 4 in Llama-3.2-1B. The bits-per-byte and Belebele numbers are reported as means over seeds. App. A states further evaluation details.

5 Results

5.1 Morphological Alignment

Table 1 reports MorphScore measured on the LangMAP tokenizers that the fine-tuned models of Section 5.3 were trained with. Recall rises in nine of the ten recall measurements, and the 95% confidence interval excludes zero in each of those nine. However, MorphScore recall on Arabic with the Llama-3.2-1B tokenizer, is negatively impacted with an interval spanning zero. The increase in recall means that LangMAP places more token boundaries near gold morpheme boundaries, while the simultaneous increase in precision suggests that these changes bring the segmentation closer to morphologically meaningful units rather than simply adding noise. However, not all observed Δ values reflect changes caused by the tokenizer itself. The two Arabic results and the Qwen2.5-0.5B Hindi result are partly properties of the MorphScore measurement rather than of the tokenizer. The Qwen2.5-0.5B Hindi result is largely determined by the tokenizer’s limited Devanagari coverage: its vocabulary contains only 67 Devana-

Metric	n	Improved	Δ
Overall AST alignment	72	100%	+0.027*
Identifier alignment	72	99%	+0.028*
Operator alignment	72	97%	+0.025*
Delimiter alignment	72	100%	+0.030*
Literal alignment	72	93%	+0.029*
Keyword alignment	72	51%	+0.003*
Ident. frag. rate	72	94%	-0.009*
Mean tokens / ident.	72	86%	-0.024*

Table 2: Change in intrinsic tokenizer code metrics (LANGMAP minus ORIGINAL) over 8 model families and 9 programming languages. The alignment rows are AST alignment stratified by node category; higher is better. *Mean tokens / ident.* is the mean number tokens per identifier; lower is better for both this metric and *Ident. frag. rate*. *Improved* is the percentage of the n measurements in which the difference improves; * marks a 95% confidence interval excluding zero.

gari pieces. With so few pieces available, any tokenizer restricted to this vocabulary must segment Hindi largely at the character level, giving a recall of 1.0000 by construction. The Llama-3.2 vocabulary contains more Devanagari pieces (958), but the Hindi Δ values are still 0. This is likely because 958 pieces remain too few to allow substantially different segmentations. In this case, LangMAP makes few or no changes to the original segmentation, so a Δ of 0 simply indicates that the two segmentations are unchanged. App. B provides a fuller analysis. These results also illustrate a limitation of LangMAP. As LangMAP re-scores a fixed vocabulary rather than adding new tokens, it can only change how the existing pieces are combined. The extent of this change therefore depends on how well the original vocabulary covers the language in the first place.

A wider sweep is reported in Table 7: 10 distinct tokenizers on 7 languages, adding German and Tamil to the five above. For all 40 recall measurements on the four Latin-script languages, the recall difference is positive. For the three non-Latin-script languages, diagnostics are performed. Results for the other intrinsic metrics are in App. B.

5.2 Code Tokenization

Table 2 reports the paired differences between LangMAP and the original tokenizer, averaged across 8 open-source tokenizers and 9 programming languages. LangMAP consistently improves alignment with AST leaf boundaries: the change is positive for every tokenizer-language pair. LangMAP also reduces the fraction of programmer-

defined identifiers that are split across multiple tokens by 0.009, and reduces identifier fertility (Mean tokens / ident). App. C reports the evaluation set, bootstrapping procedure used to compute confidence intervals, and absolute alignment scores.

5.3 Extrinsic Results

Adapting a language model. Table 3 reports target-language MultiBLiMP accuracy after adapting Qwen2.5-0.5B and Llama-3.2-1B to a single language at a time; multiple testing corrections are performed in all significance tests. Notably, the improvement from fine-tuning alone is larger than any improvements that come from the choice of tokenizer. VOCABULARY-EXTENSION is the only configuration whose accuracy is significantly above the original tokenizer’s in more than one language (in Finnish and Turkish). Hungarian under the Qwen2.5-0.5B base model is the one setting where LangMAP offers a significant MultiBLiMP improvement over the model’s original tokenizer, i.e., its 95% confidence interval excludes zero after correction. Two control configurations were run in Hungarian. The model using the BASE-UNIGRAM tokenizer scores +0.001 accuracy above the original tokenizer in Hungarian, so the improvement in Hungarian MultiBLiMP is not produced by the switch to UnigramLM segmentation. The model using the PLACEBO tokenizer, the score permutation baseline described in Section 4.2, is +0.004 above the model that uses the original tokenizer, and the 95% confidence interval for the LANGMAP-minus-PLACEBO excludes zero after correction. The Hungarian result is not reproduced in the second model family: no Llama-3.2-1B difference has an interval excluding zero after correction. App. D contains an analysis looking at whether per-language MorphScore differences are positively associated with downstream MultiBLiMP differences. There is no significant correlation between the two in our results: over the nine adapted (language, model family) combinations, the Spearman correlation is negative under all four MorphScore columns, and none of the eight correlations is distinguishable from zero at $n = 9$.

Bits per byte and English retention. Table 10 reports FLORES+ bits per byte and pooled English Belebele accuracy for Qwen2.5-0.5B, and Table 12 repeats the bits-per-byte measurement on Llama-3.2-1B. Only VOCABULARY-EXTENSION

Language	No-adapt.	Original	Base-Unigram	LangMAP	Placebo	Vocab.-ext.
<i>Qwen2.5-0.5B</i>						
Finnish	0.6420	0.8734 $\pm$ 0.0062	0.8694 $\pm$ 0.0020	0.8712 $\pm$ 0.0039	–	0.9128* $\pm$ 0.0010
Hungarian	0.7763	0.9531 $\pm$ 0.0027	0.9538 $\pm$ 0.0083	0.9645 [†] $\pm$ 0.0012	0.9594 [†] $\pm$ 0.0030	0.9633 [†] $\pm$ 0.0036
Turkish	0.7560	0.8387 $\pm$ 0.0035	0.8305 $\pm$ 0.0029	0.8308 $\pm$ 0.0108	–	0.8603* $\pm$ 0.0058
Arabic	0.9185	0.9248 $\pm$ 0.0063	0.9300 $\pm$ 0.0033	0.9320 $\pm$ 0.0026	0.9213 $\pm$ 0.0050	0.9262 $\pm$ 0.0048
Hindi	0.8493	0.9781 $\pm$ 0.0014	0.9777 $\pm$ 0.0051	0.9790 $\pm$ 0.0029	0.9790 $\pm$ 0.0022	0.9751 $\pm$ 0.0007
<i>Llama-3.2-1B</i>						
Turkish	0.8416	0.8693 $\pm$ 0.0063	–	0.8588 $\pm$ 0.0041	–	–
Hungarian	0.9325	0.9404 [†] $\pm$ 0.0130	–	0.9448 [†] $\pm$ 0.0007	–	–
Arabic	0.8897	0.9073 $\pm$ 0.0042	–	0.9078 $\pm$ 0.0029	–	–
Hindi	0.9806	0.9719 $\pm$ 0.0052	–	0.9717 $\pm$ 0.0038	–	–

Table 3: Target-language MultiBLiMP accuracy after adapting each model to the given language at a matched training budget; higher is better. *No-adapt.* is the pretrained model with its own tokenizer and no fine-tuning (one evaluation, no seeds). Every other value is the mean over 3 seeds, with the $\pm$ value giving the standard deviation across seeds.

improves target-language likelihood: models trained with it reach the lowest target-language bits per byte in every language, at a modest cost in English bits per byte. Where the vocabulary is left unchanged and only segmentations are re-scored, likelihood gets worse rather than better: models trained with the LANGMAP and BASE-UNIGRAM tokenizers are above the original tokenizer’s bits per byte in every target language and in English. We conclude that adaptation with LangMAP does not improve model likelihood at this scale and budget. In terms of retention of English performance, Belebele accuracies differ more between the adapted and the unadapted model than between tokenizer configurations. Qwen2.5-0.5B without adaptation scores 0.573, against 0.481 to 0.500 for the five adapted configurations; the drop of 0.07 to 0.09 accuracy is shared across all five configurations. The spread between the five is within the range of their pooled standard deviations, though, implying there are not significant differences between these approaches’ impacts on English at this scale.

Adapting to five languages jointly. Adapting one model per language is not the setting LangMAP is designed for: a shared vocabulary with one distribution per language is intended for one model serving several languages. Table 11 describes this protocol, in which one model is adapted to all five languages at once (equal mix, plus English at proportion 0.2) and the segmentation is selected at inference without a language label. We observe empirically that label-free routing mechanism, i.e., the selection of a string’s segmentation without a language label (Eq. (8)), corresponds perfectly to that string’s true language. Table 14

reports an analysis of this label-free routing mechanism over 12 languages and truncated inputs on a separate base model. Models trained with the LangMAP tokenizer are slightly above the original tokenizer’s accuracy in three languages under Qwen2.5-0.5B and indistinguishable from it in the other two. These improvements do not repeat under Llama-3.2-1B. In short, while the per-language segmentation scheme works

6 Conclusion

We introduced LangMAP, a multilingual tokenization scheme that extends UnigramLM by maintaining a separate unigram distribution per language over a single, fixed vocabulary. LangMAP can be trained as a tokenizer in its own right or used to adapt a tokenizer of an existing pre-trained model without altering its vocabulary or parameters. In either case, language labels are required only during estimation; at inference, the model selects the appropriate language-specific segmentation without knowing the input language. On natural language, LangMAP raised the alignment of token boundaries with gold morpheme boundaries consistently for Latin-script languages. For programming languages, it raised alignment with abstract syntax tree leaf boundaries. Limited fine-tuning results do not show consistent improvements from using LangMAP, leaving downstream applications of LangMAP as an open research question.

Limitations

While the use of a fixed shared vocabulary ensures compatibility with pretrained models, it also bounds the representational capacity of the tokenizer. If linguistically appropriate tokens for a

given language are absent from the vocabulary, language-specific reweighting of token probabilities alone cannot recover good segmentations for those languages. This limitation is visible in our results. A natural follow up would be to pair LangMAP with vocabulary extension so that adaptation is no longer confined to the tokens a model already supports. Another limitation is that our method requires labeled training data to estimate language-specific distributions. We observe empirically that very few labeled samples are needed to get good unigram distribution estimates, but the method does not work in the complete absence of labels. We leave unsupervised or weakly supervised extensions for future work. Finally, our downstream evidence is narrow. It covers two model families at one size each (Qwen2.5-0.5B on five languages, Llama-3.2-1B on four), one adaptation budget, and 3 seeds per (language, tokenizer configuration). The one comparison in which LangMAP improves over the original tokenizer—Hungarian in the single-language setting on Qwen2.5-0.5B—does not significantly improve over the placebo control after multiple-testing correction. Bits per byte is higher for LangMAP than for the original tokenizer in every language and both the models that we fine-tuned, so we make no claim about its ability to improve base language modeling capabilities.

Acknowledgments

We thank Kris Cao for helpful conversations at the beginning of this project.

References

- Orevaoghene Ahia, Sachin Kumar, Hila Gonen, Valentin Hofmann, Tomasz Limisiewicz, Yulia Tsvetkov, and Noah A. Smith. 2024. [MAGNET: Improving the multilingual fairness of language models with adaptive gradient-based tokenization](#). In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.
- Orevaoghene Ahia, Sachin Kumar, Hila Gonen, Jungo Kasai, David Mortensen, Noah Smith, and Yulia Tsvetkov. 2023. [Do all languages cost the same? Tokenization in the era of commercial language models](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9904–9923, Singapore. Association for Computational Linguistics.
- Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Z. Chen, Eric Chu, J. Clark, Laurent El Shafey, Yanping Huang, Kathleen S. Meier-Hellstern, Gaurav Mishra, Erica Moreira, Mark Omernick, Kevin Robinson, and 109 others. 2023. [Palm 2 technical report](#). *CoRR*, abs/2305.10403.
- Catherine Arnett and Benjamin Bergen. 2025. [Why do language models perform worse for morphologically complex languages?](#) In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 6607–6623, Abu Dhabi, UAE. Association for Computational Linguistics.
- Catherine Arnett, Marisa Hudspeth, and Brendan O’Connor. 2025. [Evaluating Morphological Alignment of Tokenizers in 70 Languages](#). In *Proceedings of the ICML 2025 Tokenization Workshop (TokShop)*.
- Lucas Bandarkar, Davis Liang, Benjamin Muller, Mikel Artetxe, Satya Narayan Shukla, Donald Husa, Naman Goyal, Abhinandan Krishnan, Luke Zettlemoyer, and Madian Khabsa. 2024. [The Belebele benchmark: a parallel reading comprehension dataset in 122 language variants](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 749–775, Bangkok, Thailand. Association for Computational Linguistics.
- Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. 1993. [The mathematics of statistical machine translation: Parameter estimation](#). *Computational Linguistics*, 19(2):263–311.
- Kris Cao and Laura Rimell. 2021. [You should evaluate your language model on marginal likelihood over tokenisations](#). In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 2104–2114, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.
- Tyler A Chang, Catherine Arnett, Abdelrahman Eldes-okei, Abdelrahman Sadallah, Abeer Kashar, Abolade Daud, Abosede Grace Olanihin, Adamu Labaran Mohammed, Adeyemi Praise, Adhikarinayum Meerajita Sharma, and 1 others. 2025. [Global piqa: Evaluating physical commonsense reasoning across 100+ languages and cultures](#). *arXiv preprint arXiv:2510.24081*.
- Ethan C. Chau, Lucy H. Lin, and Noah A. Smith. 2020. [Parsing with multilingual BERT, a small corpus, and a small treebank](#). In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1324–1334, Online. Association for Computational Linguistics.
- A. P. Dempster, N. M. Laird, and D. B. Rubin. 1977. [Maximum likelihood from incomplete data via the em algorithm](#). *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38.
- Konstantin Dobler and Gerard de Melo. 2023. [FOCUS: Effective embedding initialization for monolingual](#)

- specialization of multilingual models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13440–13454, Singapore. Association for Computational Linguistics.
- C.m. Downey, Terra Blevins, Nora Goldfine, and Shane Steinert-Threlkeld. 2023. [Embedding structure matters: Comparing methods to adapt multilingual vocabularies to new languages](#). In *Proceedings of the 3rd Workshop on Multi-lingual Representation Learning (MRL)*, pages 268–281, Singapore. Association for Computational Linguistics.
- Darius Feher, Ivan Vulić, and Benjamin Minixhofer. 2025. [Retrofitting large language models with dynamic tokenization](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 29866–29883, Vienna, Austria. Association for Computational Linguistics.
- Negar Foroutan, Clara Meister, Debjit Paul, Joel Niklaus, Sina Ahmadi, Antoine Bosselut, and Rico Sennrich. 2025. [Parity-aware byte-pair encoding: Improving cross-lingual fairness in tokenization](#). *CoRR*, abs/2508.04796.
- Valentin Hofmann, Janet Pierrehumbert, and Hinrich Schütze. 2021. [Superbizarre is not superb: Derivational morphology improves BERT’s interpretation of complex words](#). 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 3594–3608, Online. Association for Computational Linguistics.
- Valentin Hofmann, Hinrich Schuetze, and Janet Pierrehumbert. 2022. [An embarrassingly simple method to mitigate undesirable properties of pretrained language model tokenizers](#). In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 385–393, Dublin, Ireland. Association for Computational Linguistics.
- Jaap Jumelet, Leonie Weissweiler, Joakim Nivre, and Arianna Bisazza. 2026. Multiblimp 1.0: A massively multilingual benchmark of linguistic minimal pairs. *Transactions of the Association for Computational Linguistics*, 14:193–216.
- Stav Klein and Reut Tsarfaty. 2020. [Getting the ##life out of living: How adequate are word-pieces for modelling complex morphology?](#) In *Proceedings of the 17th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 204–209, Online. Association for Computational Linguistics.
- Taku Kudo. 2018. [Subword regularization: Improving neural network translation models with multiple subword candidates](#). In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, Melbourne, Australia. Association for Computational Linguistics.
- Sander Land and Yuval Pinter. 2026. [Which pieces does unigram tokenization really need?](#) *CoRR*, abs/2512.12641.
- Jungseob Lee, Hyeonseok Moon, Seungjun Lee, Chanjun Park, Sugyeong Eo, Hyunwoong Ko, Jaehyung Seo, Seungyoon Lee, and Heuiseok Lim. 2024. [Length-aware byte pair encoding for mitigating over-segmentation in Korean machine translation](#). In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 2287–2303, Bangkok, Thailand. Association for Computational Linguistics.
- Pietro Lesci, Clara Meister, Thomas Hofmann, Andreas Vlachos, and Tiago Pimentel. 2025. [Causal estimation of tokenisation bias](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 28325–28340, Vienna, Austria. Association for Computational Linguistics.
- Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jiawei Li, Jenny Chim, and 1 others. 2023. StarCoder: may the source be with you! *arXiv preprint arXiv:2305.06161*.
- Davis Liang, Hila Gonen, Yuning Mao, Rui Hou, Naman Goyal, Marjan Ghazvininejad, Luke Zettlemoyer, and Madian Khabsa. 2023. [XLM-V: Overcoming the vocabulary bottleneck in multilingual masked language models](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13142–13152, Singapore. Association for Computational Linguistics.
- Tomasz Limisiewicz, Jiří Balhar, and David Mareček. 2023. [Tokenization impacts multilingual language modeling: Assessing vocabulary allocation and overlap across languages](#). In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 5661–5681, Toronto, Canada. Association for Computational Linguistics.
- Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, and 1 others. 2024. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*.
- Clara Meister. 2025. [UnigramLM: An attempt at writing the missing manual](#). Blog post.
- Clara Meister. 2026. [TokEval: A tokenizer analysis suite](#). In *Third Conference on Language Modeling*.
- Clara Meister, Ahmetcan Yavuz, Pietro Lesci, and Tiago Pimentel. 2026. [What language is this? ask your tokenizer](#). *Preprint*, arXiv:2602.17655.

- Joakim Nivre, Daniel Zeman, Filip Ginter, and Francis Tyers. 2017. [Universal Dependencies](#). In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Tutorial Abstracts*, Valencia, Spain. Association for Computational Linguistics.
- NLLB Team, Marta R. Costa-jussà, James Cross, Onur Çelebi, Maha Elbayad, Kenneth Heafield, Kevin Hefernan, 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, and 20 others. 2022. [No language left behind: Scaling human-centered machine translation](#). *arXiv preprint arXiv:2207.04672*.
- Guilherme Penedo, Hynek Kydlíček, Vinko Sabolčec, Bettina Messmer, Negar Foroutan, Amir Hossein Kargaran, Colin Raffel, Martin Jaggi, Leandro Von Werra, and Thomas Wolf. 2025. Fineweb2: One pipeline to scale them all—adapting pre-training data processing to every language. *arXiv preprint arXiv:2506.20920*.
- Aleksandar Petrov, Emanuele La Malfa, Philip H.S. Torr, and Adel Bibi. 2023. Language model tokenizers introduce unfairness between languages. In *Proceedings of the 37th International Conference on Neural Information Processing Systems, NIPS '23*, Red Hook, NY, USA. Curran Associates Inc.
- 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. *J. Mach. Learn. Res.*, 21(1).
- François Remy, Pieter Delobelle, Hayastan Avetisyan, Alfiya Khabibullina, Miryam de Lhoneux, and Thomas Demeester. 2024. [Trans-tokenization and cross-lingual vocabulary transfers: Language adaptation of llms for low-resource NLP](#). *CoRR*, abs/2408.04303.
- 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. Association for Computational Linguistics.
- Craig W Schmidt, Varshini Reddy, Haoran Zhang, Alec Alameddine, Omri Uzan, Yuval Pinter, and Chris Tanner. 2024. [Tokenization is more than compression](#). In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 678–702, Miami, Florida, USA. Association for Computational Linguistics.
- Chen Shani, Yuval Reif, Nathan Roll, Dan Jurafsky, and Ekaterina Shutova. 2026. The roots of performance disparity in multilingual language models: Intrinsic modeling difficulty or design choices? *arXiv preprint arXiv:2601.07220*.
- Saketh Reddy Vemula, Sandipan Dandapat, Dipti Sharma, and Parameswari Krishnamurthy. 2025. [Re-thinking tokenization for rich morphology: The dominance of unigram over BPE and morphological alignment](#). In *The 14th International Joint Conference on Natural Language Processing and The 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pages 232–252, Mumbai, India. Association for Computational Linguistics.
- Zihan Wang, Karthikeyan K, Stephen Mayhew, and Dan Roth. 2020. [Extending multilingual BERT to low-resource languages](#). In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2649–2656, Online. Association for Computational Linguistics.
- Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. 2022. [ByT5: Towards a token-free future with pre-trained byte-to-byte models](#). *Transactions of the Association for Computational Linguistics*, 10:291–306.
- Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2019. *XLNet: Generalized autoregressive pretraining for language understanding*. Curran Associates Inc., Red Hook, NY, USA.

A Experimental Setup Details

Evaluated models and their tokenizers. Table 4 contains information about our evaluated models and their tokenizers.

Table 4: Tokenizer and Vocabulary of Base Model tokenizers that are used for intrinsic and downstream evaluation. We additionally report results for `OLMo-3-7B` and `EuroLLM-1.7B`. The Llama 3 and Qwen 2.5 & 3 rows each cover two model versions that share one tokenizer. $|V|$ is the full vocabulary size, added and special tokens included.

Model	Algorithm	$ V $
Bloom-3b	BPE	250,880
Gemma-3-1b	BPE	262,144
Gemma-2-2b	BPE	256,000
Granite 3.0-8b	BPE	49,152
Llama 3	BPE	128,256
Mistral-7b-v0.3	BPE	32,768
Mistral-Nemo-12b	BPE	131,072
Phi-3-mini	BPE	32,064
Phi-4	BPE	100,352
Qwen 2.5 & 3	BPE	151,936
XGLM-2.9b	UnigramLM	256,008
Yi-1.5-6b	BPE	64,000
<i>Appendix-only Models</i>		
OLMo-3-7B	BPE	100,278
EuroLLM-1.7B	BPE	128,000

Paired-bootstrap differences. All reported Δ values are paired comparisons over (tokenizer, language) pairs. For each pair, Original and LangMAP are evaluated on the same test data, and we take the per-pair difference $\Delta_c = m(\text{LangMAP}_c) - m(\text{Original}_c)$ for metric m . We report the aggregate effect as the mean over the n pairs, $\bar{\Delta} = \frac{1}{n} \sum_{c=1}^n \Delta_c$, and estimate 95% confidence intervals with a nonparametric percentile bootstrap that resamples the n pairs with replacement: we draw 10,000 resamples, recompute $\bar{\Delta}$ on each, and report the 2.5th and 97.5th percentiles of the resulting distribution. An aggregate effect is significant when this interval excludes zero.

Languages. The seven languages measured for morphological alignment are German (fusional, Latin script), Finnish, Hungarian and Turkish (agglutinative, Latin script), Hindi (mixed, Devanagari), Arabic (templatic, Arabic script) and Tamil (agglutinative, Tamil script). Each has the Universal Dependencies treebank that the metric requires. Bengali (agglutinative, Bengali script) and Thai (analytic, Thai script) have treebanks as well and are measured only in the label-free routing of Ta-

ble 14. Swahili has no treebank and appears in Table 8 only.

Training budget and seeds. The adaptation budget is defined in padded training positions rather than in real tokens. Every sequence is padded to 1024 positions, and every run takes the same 3,561 optimizer steps at 49,152 positions per step, 175,030,272 positions in total. Fixing the number of optimizer steps is what makes tokenizer configurations with different compression comparable: a budget defined in real tokens would give the more compressive tokenizer more optimizer steps for the same nominal budget. The position budget was calibrated so that it corresponds to roughly 100M real target-language tokens for the median configuration. Because the padding fraction differs by tokenizer configuration and by language, the real target-language token count of a given run is below its position budget and is read from that run’s own metadata rather than fixed in advance. The seeds are 42, 43 and 44, with 45 and 46 added in Hungarian under the adaptive rule, in both model families. LoRA rank was ablated on Qwen2.5-0.5B: ranks 16 and 64, the latter at $\alpha = 128$, gave a minimal difference in downstream performance. The tied embedding is created with `peft`’s `ensure_weight_tying`, and a runtime check compares the storage of the input lookup and the output head and aborts if the two differ.

Parameters added by vocabulary extension.

Under the VOCABULARY-EXTENSION configuration, the embedding matrix gains 15,729 rows for 16,000 (the base embedding matrix was padded and so some of the padded positions could be used). The parameter count for the model under this configuration grows as follows: $15,729 \times 896 = 14.1\text{M}$ where 896 is the embedding dimension.

B Additional Natural Language Intrinsic Tokenization Results

B.1 Additional Intrinsic Metrics

Table 7 reports morphological alignment over 10 tokenizers and 7 languages: German, Finnish, Hungarian, Turkish, Hindi, Arabic and Tamil. Table 8 reports tokenization efficiency and information-theoretic metrics for German (deu), Finnish (fin), Hungarian (hun), Turkish (tur), Hindi (hin) and Swahili (swh). Notably, we see that Rényi-2.5 efficiency falls by 0.239 (95% CI $[-0.281, -0.199]$) and bigram entropy by 0.006. These results sug-

gest that LangMAP concentrates probability mass on the vocabulary subset that segments a single language well. On natural language, fertility changes are small and mixed in sign: it rises in three of the six languages (German +0.0013, Finnish +0.0017, Turkish +0.0023) and falls in the other three (Hungarian −0.0024, Hindi −0.0062, Swahili −0.0038), where a lower value is the improving direction.

B.2 MorphScore measurement, Arabic gold segmentation and Hindi script coverage

MorphScore computations. Each value in Table 1 is measured over the word types of that language’s Universal Dependencies subset. A value is marked in Table 1 and in Table 5 when its 95% confidence interval excludes zero under a paired percentile bootstrap, that is, a procedure that draws 2,000 resamples of the word types with replacement, keeps each word type’s two tokenizations together in every resample, recomputes the difference on each resample, and reports the 2.5th and 97.5th percentiles of the resulting 2,000 differences. Under this procedure 36 of the 40 values of Table 5 are marked; the four that are not are the Llama-3.2-1B Arabic recall and precision, the Llama-3.2-1B Hindi macro-F1, and the Llama-3.2-1B Hindi precision. Table 5 shows all four MorphScore metrics for the ten rows.

Arabic gold segmentation. The Arabic gold segmentation was rebuilt for Table 1 and Table 7. The released prefix, stem and suffix split is an artifact of matching a vocalized lemma against an unvocalized word form: the mean stem length is 1.77 characters and 26.5% of word types have a one-character stem. We remove the lemma’s diacritics, normalize alef and ya, and locate the result inside the word form, which aligns 16,014 of the 19,126 word types (83.7%). The 16.3% that do not align are non-concatenative forms, that is broken plurals, imperfective stem alternation and suppletion, so the Arabic values measure concatenative affixation on the alignable subset and say nothing about the rest. An earlier version of Table 1 read predicted boundaries by decoding one token id at a time, which misplaces every boundary after a token that covers part of a multi-byte character. The Arabic and Hindi rows change under the correction and the three Latin-script rows do not.

Lang.	<i>n</i>	Δ micro-F1	Δ macro-F1	Δ Rec.	Δ Prec.
<i>Qwen2.5-0.5B</i>					
Turkish	32,560	+0.0688*	+0.0733*	+0.2108*	+0.0703*
Finnish	40,776	+0.0245*	+0.0280*	+0.1332*	+0.0391*
Hungarian	7,924	+0.0350*	+0.0405*	+0.1530*	+0.0385*
Arabic	11,256	+0.0743*	+0.0775*	+0.0663*	+0.0350*
Hindi [†]	4,723	−0.0116*	−0.0124*	+0.1008*	+0.0217*
<i>Llama-3.2-1B</i>					
Turkish	32,560	+0.0884*	+0.0891*	+0.1937*	+0.0743*
Finnish	40,776	+0.0225*	+0.0257*	+0.1151*	+0.0319*
Hungarian	7,924	+0.0413*	+0.0473*	+0.1659*	+0.0440*
Arabic	10,603	+0.0074*	+0.0078*	−0.0043	+0.0015
Hindi	4,723	−0.0009*	−0.0005	+0.0010*	−0.0000

Table 5: All four MorphScore metrics for the ten (model family, language) rows of Table 1, which reports the first, third and fourth of them. Values are LANGMAP-ORIGINAL; higher is better in all four columns. * marks a value whose 95% confidence interval excludes zero under a paired percentile bootstrap over word types with 2,000 resamples, and [†] the Qwen2.5-0.5B Hindi row, whose recall is 1.0000 by construction. Llama-3.2-1B Finnish has no downstream run.

B.3 Headroom Analysis

Figure 1 shows that LangMAP’s MorphScore recall gain over the base tokenizer is largest where the base tokenizer’s recall is lowest, and near zero where base recall is already high. This pattern shows the limits of what re-scoring can do. Concretely, LangMAP re-scores a fixed vocabulary and adds no tokens, so it changes only which of the existing pieces are used to segment a string. Thus, a language can only benefit when the vocabulary already contains pieces that could yield a segmentation better aligned with that language’s morpheme boundaries. What the vocabulary covers sets not only how large the change is but what kind of change it is: with enough pieces of the script, re-scoring selects among genuine alternatives, and with almost none, the character-level segmentation is the only alternative left for it to select.

We perform an additional analysis to explore this phenomenon. In Table 6, we count, for each of the eleven tokenizers, how many vocabulary entries are Devanagari pieces and how many are English or German pieces.

How each column in Table 6 was computed. A token’s byte sequence is recovered by reversing the GPT-2 byte-to-unicode map for byte-level tokenizers (Bloom, Granite, Mistral Nemo, Phi 4, Qwen3, Llama3), or by replacing the SentencePiece metaspace marker with a space for SentencePiece tokenizers (Gemma 3, Mistral 7b, Phi 3, XGLM, Yi). Added and special tokens are excluded. Before script classification, one optional leading space

marker is stripped. We compute the columns as follows; single-byte tokens are excluded from the counts throughout, so a count measures genuine merged subwords rather than the base byte alphabet.

- *Devanagari, Complete*: tokens whose bytes decode as valid UTF-8 to a non-empty string in which every character lies in the Devanagari block U+0900 to U+097F. A single Devanagari character is three UTF-8 bytes, so single characters are included.
- *Devanagari, +Fragments*: the Complete count plus, for byte-level tokenizers only, multi-byte tokens whose bytes are not valid standalone UTF-8 but form a contiguous slice of a Devanagari UTF-8 sequence (lead byte 0xE0, second byte 0xA4 or 0xA5, continuation bytes 0x80 to 0xBF). SentencePiece tokenizers contribute no fragments because their pieces are whole characters, so the two Devanagari columns are equal for them.
- *English subwords*: tokens that decode as valid UTF-8 to a non-empty string in which every character is an ASCII letter a–z or A–Z. Digits, punctuation, mixed-script tokens, and single-byte tokens (including lone letters) are excluded.
- *German*: tokens whose bytes decode as valid UTF-8 to a non-empty string in which every character is a letter of the German alphabet, that is a–z, A–Z, or one of ä, ö, ü, Ä, Ö, Ü, ß.

Table 6 shows that the two scripts are covered unequally. Compared to German, Devanagari coverage is smaller and more variable, ranging from 3 pieces (Yi 1.5) to 16,189 (Bloom 3b), and falling below 40 for four tokenizers (Yi 1.5: 3; Granite 3.0: 16; Phi-4: 23; Phi-3 mini: 39). These counts account for both ends of the trend. Because every tokenizer contains tens of thousands of Latin pieces, German text has many alternative segmentations under the vocabulary; LangMAP’s per-language distribution selects among them, and the selected segmentations align more closely with morpheme boundaries, which is evinced by German’s mean recall gain of +0.137 over the ten tokenizers of Table 7. Hindi is the other end, and the effect there is not that re-scoring leaves the segmentation alone. A vocabulary with very few Devanagari

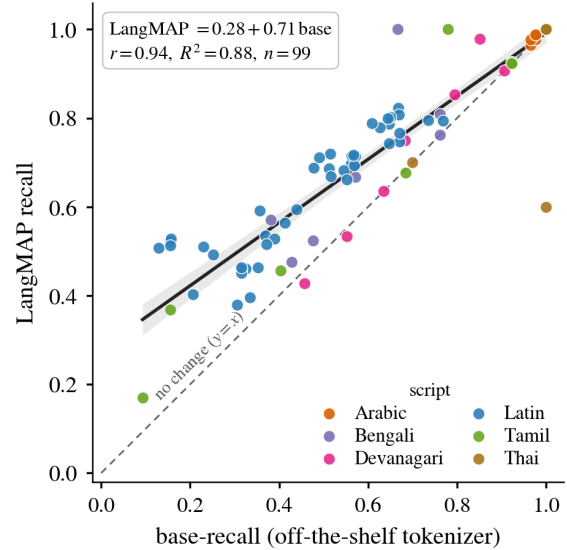


Figure 1: LangMAP MorphScore recall as a function of the original tokenizer’s MorphScore recall. We see consistent improvements (points above diagonal) and that the gains are typically largest for tokenizers with the lowest base recall, i.e., where there is the most room for improvement. Our subsequent analysis shows that this result is contingent on the vocabulary containing sufficient coverage of a script/language.

pieces gives a Devanagari string almost no segmentation other than the character-level one, which puts its MorphScore recall at 1.0000 and makes that measurement uninformative about morphological alignment. The eleven tokenizers of Table 6 span 3 to 16,189 Devanagari pieces, which is why Table 7 marks the two cases separately instead of averaging over them. Of the nine tokenizers in that table’s Hindi column, four produce a recall difference that is a character-level ceiling effect, one produces identical segmentations under both tokenizers, and the mean is taken over the four that remain.

C Additional Code Intrinsic Tokenization Results

The values used in Table 2 are shown in Table 9. A value is marked in Table 2 when its 95% confidence interval excludes zero under the percentile bootstrap procedure described in App. A.

D Additional Downstream Results

Per-language MultiBLiMP accuracy for jointly-adapted model. Table 11 shows per-language MultiBLiMP accuracy for one model adapted to all five languages jointly. With 3 seeds a per-

Table 6: Devanagari and Latin (English) subword inventories for the eleven subword tokenizers we perform experiments with. For each tokenizer, the table reports how many vocabulary entries decode to complete Devanagari characters, how many additional byte-level fragments of Devanagari characters exist, and how many entries are Latin-letter (English) subwords. The *Vocabulary* column is the full vocabulary size, matching Table 4. The four subword counts are taken over vocabulary entries after excluding added tokens, special tokens and single-byte tokens, so they do not sum to the *Vocabulary* column.

Tokenizer	Vocabulary	Devanagari subwords		Latin-script subwords	
		Complete	+Fragments	English	German
Bloom-3b	250 880	16 189	16 210	89 919	90 100
Gemma-3-1b	262 144	13 684	13 684	139 481	141 078
Granite 3.0-8b	49 152	16	19	39 727	39 825
Llama 3	128 256	958	970	71 150	71 727
Mistral 7b	32 768	44	44	24 952	25 113
Mistral Nemo	131 072	1530	1542	70 732	72 196
Phi 3 mini	32 064	39	39	24 016	24 359
Phi 4	100 352	23	30	68 643	68 897
Qwen3	151 936	67	74	68 871	69 588
XGLM 2.9b	256 008	5667	5667	108 841	112 736
Yi 1.5 6B	64 000	3	3	38 977	38 986

Table 7: **MorphScore recall difference, LANGMAP - ORIGINAL, for 10 tokenizers on 7 languages. Panel (a), Latin script.** All 40 differences are positive. **Panel (b), non-Latin scripts,** with two diagnostics beside each difference. *Pieces* counts the vocabulary entries written entirely in that language’s script, with the count of those longer than one character in parentheses. *Char-split* is the frequency-weighted fraction of word types cut at every character position, for the original tokenizer and then for LangMAP. [†] marks the 6 measurements in which the two tokenizers produce the same segmentation of every word type, producing a difference of 0. The 14 remaining non-Latin measurements show differences ranging from -0.005 to $+0.119$. The mean row of panel (b) uses only these 14, so it is not an average over all families. Yi-1.5-6b appears in panel (a) only. Yi’s vocabulary contains 3 Devanagari pieces and 0 Tamil pieces and does not have byte fallback. Those two scripts must be represented primarily by unknown tokens, and the analysis is confounded.

(a) Latin-script languages					
Tokenizer	German	Finnish	Hungarian	Turkish	Mean
Bloom-3b	+0.180	+0.089	+0.104	+0.151	+0.131
Gemma-3-1b	+0.126	+0.123	+0.130	+0.167	+0.137
Granite 3.0-8b	+0.134	+0.107	+0.171	+0.118	+0.132
Llama-3.2-1b	+0.179	+0.115	+0.166	+0.194	+0.164
Mistral-7b-v0.3	+0.090	+0.117	+0.151	+0.131	+0.122
Mistral-Nemo-12b	+0.107	+0.099	+0.142	+0.149	+0.124
Phi-4	+0.177	+0.127	+0.151	+0.175	+0.157
Qwen2.5-0.5B	+0.179	+0.133	+0.153	+0.211	+0.169
XGLM-2.9b	+0.011	+0.042	+0.053	+0.092	+0.050
Yi-1.5-6b	+0.185	+0.105	+0.148	+0.144	+0.146
Mean	+0.137	+0.106	+0.137	+0.153	+0.133

(b) Non-Latin-script languages									
Tokenizer	Hindi (Devanagari)			Arabic			Tamil		
	Δ Rec.	Pieces	Char-split	Δ Rec.	Pieces	Char-split	Δ Rec.	Pieces	Char-split
Bloom-3b	+0.030	16,189 (16,038)	0.002→0.002	+0.040	20,587 (20,394)	0.001→0.001	+0.119	6,194 (6,114)	0.000→0.000
Gemma-3-1b	+0.008	13,684 (13,519)	0.002→0.002	−0.005	8,408 (8,008)	0.002→0.002	+0.085	3,850 (3,769)	0.001→0.001
Granite 3.0-8b	+0.000 [†]	16 (0)	1.000→1.000	+0.002 [†]	79 (25)	0.157→0.181	+0.000 [†]	25 (0)	1.000→1.000
Llama-3.2-1b	+0.001 [†]	958 (845)	0.365→0.366	−0.004	3,729 (3,594)	0.002→0.002	+0.000 [†]	3 (0)	0.101→0.101
Mistral-7b-v0.3	−0.000 [†]	44 (0)	1.000→0.998	+0.000 [†]	71 (2)	0.298→0.298	+0.000 [†]	22 (0)	1.000→0.989
Mistral-Nemo-12b	+0.031	1,530 (1,412)	0.070→0.097	+0.005	9,350 (9,197)	0.001→0.001	+0.052	525 (466)	0.013→0.013
Phi-4	+0.041 [†]	23 (1)	0.613→0.807	+0.001 [†]	105 (36)	0.164→0.169	+0.000 [†]	3 (0)	0.101→0.101
Qwen2.5-0.5B	+0.101 [†]	67 (1)	0.613→0.984	+0.066	4,008 (3,449)	0.001→0.001	+0.321 [†]	28 (0)	0.101→1.000
XGLM-2.9b	+0.007	5,667 (5,536)	0.000→0.000	+0.018	13,168 (12,962)	0.001→0.001	−0.002	1,945 (1,876)	0.000→0.000
Mean	+0.019	n=4		+0.020	n=6		+0.064	n=4	

centile bootstrap over seeds returns the smallest and the largest of the three observed differences, so we report the spread as the observed range over the 3 seeds. The three per-seed dif-

Table 8: **Surface efficiency and information-theoretic metrics, per language.** $\Delta = \text{LANGMAP} - \text{ORIGINAL}$ (raw, per-checkpoint convention). Improving direction is *lower* for fertility and *higher* for the other three; small brackets are 95% CIs and * marks a CI excluding 0. Compression rate is shown $\times 10^3$.

Lang	n	$\Delta\text{Fertility}$	$\Delta\text{Compr.} (\times 10^3)$	$\Delta\text{Bigram } H$	$\Delta\text{Rényi-2.5}$
deu	10	+0.0013* [+0.0002, +0.0023]	+0.11 [−0.00, +0.19]	−0.0068* [−0.0085, −0.0050]	−0.251* [−0.317, −0.175]
fin	10	+0.0017 [−0.0032, +0.0056]	+0.11 [−0.13, +0.33]	+0.0010 [−0.0044, +0.0056]	−0.343* [−0.464, −0.240]
hun	10	−0.0024* [−0.0049, −0.0000]	−0.09 [−0.22, +0.05]	−0.0078* [−0.0103, −0.0051]	−0.303* [−0.420, −0.193]
tur	10	+0.0023 [−0.0046, +0.0087]	+0.38 [−0.07, +0.93]	−0.0012 [−0.0065, +0.0036]	−0.109* [−0.176, −0.037]
hin	10	−0.0062 [−0.0185, +0.0003]	−0.10 [−0.34, +0.06]	−0.0086 [−0.0226, +0.0002]	−0.154 [−0.398, +0.003]
swh	10	−0.0038* [−0.0069, −0.0010]	−0.16 [−0.32, +0.00]	−0.0103* [−0.0140, −0.0064]	−0.605* [−0.778, −0.407]

Table 9: Per-(model, language) difference in *overall* AST-leaf-boundary alignment, LangMAP minus the model’s own tokenizer, for every one of the $8 \times 9 = 72$ (model, programming language) measurements. Alignment is the fraction of leaf-node spans in the parse tree whose start and end both coincide with a token boundary; higher is better. All 72 differences are positive, from +0.003 to +0.066, with a mean of +0.027. The right column and bottom row give per-model and per-language means. The evaluation set is 100 source files per language, taken in parquet row order from the StarCoder eval split.

Model	C	C++	C#	Go	Java	JS	PHP	Py	TS	Mean
BLOOM-3B	+0.052	+0.036	+0.033	+0.028	+0.029	+0.060	+0.039	+0.018	+0.041	+0.037
Gemma 2-2B	+0.066	+0.018	+0.010	+0.021	+0.011	+0.027	+0.012	+0.017	+0.016	+0.022
Granite 3.0-8B	+0.028	+0.018	+0.009	+0.022	+0.011	+0.026	+0.012	+0.019	+0.011	+0.017
Llama 3.2-1B	+0.049	+0.030	+0.021	+0.038	+0.040	+0.035	+0.016	+0.037	+0.030	+0.033
Mistral-NeMo-12B	+0.046	+0.026	+0.030	+0.040	+0.041	+0.035	+0.014	+0.043	+0.028	+0.034
Phi-4	+0.049	+0.030	+0.022	+0.038	+0.042	+0.035	+0.016	+0.037	+0.030	+0.033
Qwen 2.5-0.5B	+0.049	+0.030	+0.022	+0.037	+0.042	+0.035	+0.016	+0.036	+0.030	+0.033
XGLM-2.9B	+0.023	+0.006	+0.009	+0.015	+0.003	+0.012	+0.003	+0.008	+0.006	+0.009
Mean	+0.045	+0.024	+0.019	+0.030	+0.027	+0.033	+0.016	+0.027	+0.024	+0.027

ferences on Qwen2.5-0.5B are Turkish +0.0098, +0.0052, +0.0103; Finnish +0.0241, +0.0023, +0.0019; Hungarian +0.0095, +0.0012, +0.0189; and Arabic and Hindi change sign across seeds. On Llama-3.2-1B they are Turkish −0.0218, −0.0293, −0.0235; Finnish +0.0047, −0.0062, +0.0237; Hungarian +0.0083, +0.0059, −0.0059; Arabic +0.0041, −0.0008, −0.0058; and Hindi +0.0000, −0.0055, −0.0007.

Bits per byte on Llama-3.2-1B. Table 12 reports FLORES+ bits per byte for Llama-3.2-1B adapted to one language at a time, under the protocol of Table 3. The original tokenizer has the lower value in all four target languages and in English.

Confidence intervals and p -value computations. A paired bootstrap over evaluation items with seeds as replicates is computed alongside each difference from the original tokenizer in Table 3. Significance is only marked when the confidence interval for that difference excludes 0 after multiple testing adjustment: we use the Holm procedure at $\alpha = 0.05$, over the 18 measurements taken in the Qwen2.5-0.5B panel and the 4 in the Llama-3.2-1B panel. The Hungarian runs were extended to 5 seeds under the adaptive rule of Section 4.3.

Validation bits per byte for jointly-trained model. Bits per byte on the in-training validation split of the model trained jointly on five languages is 1.0772 for the original tokenizer and 1.0942 for LangMAP on Qwen2.5-0.5B, aggregated over the five languages and 3 seeds, a difference of +0.017 in the same direction as the single-language setting.

MorphScore differences and MultiBLiMP differences. We tested, across the nine (language, model family) combinations we adapted, whether MorphScore differences (LangMAP minus the original tokenizer) are positively associated with the MultiBLiMP differences. Each downstream difference is paired with the MorphScore difference of the tokenizer that run was trained with, taken from Table 1. The prediction is not supported. The Spearman correlation is negative under every MorphScore column, at −0.27 on recall, −0.35 on precision and −0.20 on each of the two F1 columns of Table 5. The intrinsic and downstream differences agree in sign in 4 to 6 of the nine measurements, but an exact sign test shows this result is not distinguishable from an at-chance result. Turkish has the largest MorphScore recall difference in both model families, +0.211 on Qwen2.5-0.5B

	Original	Base-Unigram	LangMAP	Placebo	Vocab.-ext.
<i>Target-language FLORES+ bits per byte (lower is better)</i>					
Finnish	1.3849	1.4278	1.4245	n/a	1.2745
Hungarian	1.3293	1.3514	1.3494	1.3607	1.2069
Turkish	1.2922	1.3108	1.3097	n/a	1.2417
Arabic	0.9335	0.9413	0.9394	0.9500	0.9324
Hindi	0.6526	0.6592	0.6591	0.6587	0.5946
<i>English FLORES+ bits per byte (lower is better)</i>					
Finnish	1.0819	1.0895	1.0907	n/a	1.1673
Hungarian	1.0746	1.0820	1.0815	1.0826	1.1060
Turkish	1.0726	1.0809	1.0816	n/a	1.1244
Arabic	1.0370	1.0458	1.0442	1.0451	1.0484
Hindi	1.0504	1.0579	1.0565	1.0569	1.0564
<i>English Belebele accuracy, pooled over languages and seeds (higher is better)</i>					
Pooled	0.4993	0.4932	0.4981	0.5000	0.4807

Table 10: FLORES+ bits per byte and pooled English Belebele accuracy after adapting Qwen2.5-0.5B to one language at a time. Each bits-per-byte value is the mean over 3 seeds and **bold** marks the lowest value in a row; the two bits-per-byte panels and the Belebele row run in opposite directions, as marked. PLACEBO was run in Arabic, Hindi and Hungarian only. The Belebele row is a mean over languages and seeds, with no bootstrap and no correction applied.

and +0.194 on Llama-3.2-1B, together with downstream differences of -0.008 and -0.011 , in the opposite direction to the prediction.

LangMAP convergence. Table 13 shows that LangMAP estimation has low sample complexity. Across German, Finnish and Turkish, re-estimating the language-specific distribution from only 1,000 documents gives a held-out fertility within 0.054% of the reference estimate, averaged over the three languages. The last row shows the scale at which every LangMAP distribution in this paper was fit, 25M tokens per language.

Label-free segmentation selection. We investigate whether the correct language’s segmentation is used at inference time in the LangMAP tokenization scheme. We evaluate on 12 languages (German, Finnish, Hungarian, Turkish, Hindi, Bengali, Tamil, Arabic, Thai, which are used for our MorphScore evaluation; and Basque, Korean, Russian). Here the input is truncated to W leading words and the base model is EuroLLM-1.7B. For 12 languages (400 sentences each), LangMAP chooses the UnigramLM segmentation from the input’s ground-truth language with 100% accuracy once 8 words of the sequence are given as input, for all 12 languages and for the Latin-script subset alike (Table 14). Below 8 words, routing performs better on Latin languages (0.803 for all languages against 0.888 for just Latin languages at one word), which we attribute to the tokenizer’s better coverage of Latin scripts. Note that this is a different

measurement from the routing result mentioned in Section 5.3.

Language	Original	LangMAP (routed)	LangMAP – Original	Routed – Oracle
<i>Qwen2.5-0.5B</i>				
Turkish	0.7907	0.7991	+0.0084	0.0000
Finnish	0.7843	0.7938	+0.0095	0.0000
Hungarian	0.8923	0.9022	+0.0099	0.0000
Arabic	0.9155	0.9158	+0.0003	0.0000
Hindi	0.9330	0.9307	−0.0023	0.0000
<i>Llama-3.2-1B</i>				
Turkish	0.8576	0.8328	−0.0249	0.0000
Finnish	0.8355	0.8429	+0.0074	0.0000
Hungarian	0.9254	0.9282	+0.0028	0.0000
Arabic	0.8936	0.8927	−0.0008	0.0000
Hindi	0.9666	0.9645	−0.0021	0.0000

Table 11: Per-language MultiBLiMP accuracy for one model adapted to all five languages jointly, in each of the two model families, means over 3 seeds; higher is better. *Routed* selects the segmentation at inference without a language label; *oracle* selects it using the true language label. The routed and oracle accuracies are identical in all 30 (model family, language, seed) combinations, and on FLORES+ each family’s tokenizer pool assigns 997 of 997 sentences to the correct language’s distribution in each of the five languages. Absolute accuracies are below the single-language values of Table 3 (Turkish 0.7907 here against 0.8387 there for Qwen2.5-0.5B); we attribute that difference to one model covering five languages rather than one. The differences in the third column are computed over 3 seeds rather than over evaluation items, for the reason and with the per-seed values given in App. D.

Language	Original	LangMAP	LangMAP – Original
<i>Target-language FLORES+ bits per byte (lower is better)</i>			
Turkish	1.1746	1.2094	+0.0348
Hungarian	1.1539	1.2053	+0.0514
Arabic	0.9118	0.9274	+0.0156
Hindi	0.5699	0.5736	+0.0036
<i>English FLORES+ bits per byte (lower is better)</i>			
Turkish	0.9822	0.9926	+0.0103
Hungarian	0.9825	0.9925	+0.0100
Arabic	0.9781	0.9812	+0.0031
Hindi	0.9738	0.9759	+0.0021

Table 12: FLORES+ bits per byte after adapting Llama-3.2-1B to one language at a time, under the protocol of Table 3; lower is better in both panels. Finnish was not run in this family, and this family was run with the original tokenizer and LangMAP only. Each value is the mean over 3 seeds, with seed standard deviations of 0.0001 to 0.0012. English Belebele is not tabulated for this family: the pretrained model scores 0.347 against a four-way chance level of 0.25, so it does not measure English retention at this scale.

Table 13: **Convergence of the language-specific distribution.** Each value is the difference in fertility between a distribution ϕ_ℓ re-estimated from the first N documents of that language’s corpus and the reference distribution, written as a percentage of the reference value. Fertility is computed on the FLORES-200 devtest set. The base vocabulary is Qwen2.5-0.5B’s and is held fixed; only the per-language distribution is re-estimated. The reference is the $N=32,000$ fit. The last row is the scale at which every LangMAP distribution in this paper was fit: 25M tokens per language.

Documents (N)	German	Finnish	Turkish
125	0.306%	0.187%	0.230%
500	0.067%	0.181%	0.040%
1,000	0.002%	0.101%	0.059%
4,000	0.026%	0.059%	0.076%
16,000	0.005%	0.047%	0.016%
32,000	<i>reference</i>		
25M tokens	0.009%	0.009%	0.011%

Table 14: **Label-free language routing accuracy.** As a function of input length (first W whitespace words), the fraction of FLORES inputs that were segmented according to the “correct” language’s unigram distribution via LangMAP’s maximum-likelihood selection rule; “full” = whole sentence). Base = EuroLLM-1.7B, 400 sentences/language. “Latin-5” restricts both the input languages and LangMAP’s language set to the five Latin-script languages.

Input (W words)	1	2	3	5	8	full
All 12 languages	.888	.961	.988	.998	1.00	1.00
Latin-5 only	.803	.922	.976	.995	1.00	1.00

E Further Details on Tokenization and the UnigramLM Algorithm

This appendix provides additional detail on the UnigramLM algorithm summarized in Section 2.1. We give the full inference and EM derivations (Apps. E.1 and E.2, respectively) and the vocabulary pruning procedure (App. E.3). These derivations are largely reproduced from Meister (2025); we refer the reader to this resource for a more detailed presentation of the algorithm.

E.1 Inference Derivation

For reference, the deterministic relationship between token sequences and strings under the UnigramLM generative model is:

$$P(\mathbf{S} = \mathbf{s} \mid \mathbf{V} = \mathbf{v}) = \begin{cases} 1 & \text{if } \mathbf{s} = g(\mathbf{v}) \\ 0 & \text{otherwise} \end{cases} \quad (9)$$

Recall from the main text that MAP inference seeks:

$$h_\phi(\mathbf{s}) \stackrel{\text{def}}{=} \operatorname{argmax}_{\mathbf{v} \in \mathcal{V}^*} P(\mathbf{V} = \mathbf{v} \mid \mathbf{S} = \mathbf{s}; \phi) \quad (3)$$

Using the relationships in Eqs. (1) and (9), we can rewrite Eq. (3) as the following equivalent optimization problem:

$$h_\phi(\mathbf{s}) \stackrel{\text{def}}{=} \operatorname{argmax}_{\mathbf{v} \in \mathcal{V}^*} P(\mathbf{V} = \mathbf{v} \mid \mathbf{S} = \mathbf{s}; \phi) \quad (3)$$

$$= \operatorname{argmax}_{\mathbf{v} \in \mathcal{T}_\mathcal{V}(\mathbf{s})} \frac{P(\mathbf{V} = \mathbf{v}; \phi)}{\sum_{\mathbf{v}' \in \mathcal{T}_\mathcal{V}(\mathbf{s})} P(\mathbf{V} = \mathbf{v}'; \phi)} \quad (10a)$$

$$= \operatorname{argmax}_{\mathbf{v} \in \mathcal{T}_\mathcal{V}(\mathbf{s})} P(\mathbf{V} = \mathbf{v}; \phi) \quad (10b)$$

$$= \operatorname{argmax}_{\mathbf{v} \in \mathcal{T}_\mathcal{V}(\mathbf{s})} \prod_{t=1}^{|\mathbf{v}|} \phi[v_t] \quad (10c)$$

where we first restrict the considered sequences of tokens to $\mathcal{T}_\mathcal{V}(\mathbf{s})$ (10a) since other sequences have probability zero given Eq. (9), and apply Bayes' rule, noting that $P(\mathbf{S} = \mathbf{s} \mid \mathbf{V} = \mathbf{v}; \phi) = 1$ for all $\mathbf{v} \in \mathcal{T}_\mathcal{V}(\mathbf{s})$. Then, we drop the denominator (10b), since it does not depend on $\mathbf{v}$ and thus does not affect the argmax . Finally, we apply the definition in Eq. (1) (10c).

E.2 Application of EM for Learning ϕ

As stated in Section 2.1, UnigramLM learns ϕ by approximately maximizing the observed data log-likelihood via EM. Here we provide the full logic behind why this is a valid approach.

Under the UnigramLM generative process, a “complete” dataset consists of $(\mathbf{s}, \mathbf{v})$ pairs, i.e., strings and the token sequences that produced them. Denoting such a dataset as $\mathcal{X} = \{(\mathbf{s}_m, \mathbf{v}_m)\}_{m=1}^M$, its log-likelihood is:

$$\mathcal{L}(\mathcal{X}; \phi) \stackrel{\text{def}}{=} \sum_{m=1}^M \log P(\mathbf{S}=\mathbf{s}_m, \mathbf{V}=\mathbf{v}_m; \phi) \quad (11)$$

which we refer to as the *complete* data log-likelihood. Crucially, this quantity cannot be optimized directly, since the token sequences $\mathbf{v}_m$ are latent: we only observe the detokenized strings $\mathbf{s} = g(\mathbf{v})$, not the segmentations that produced them. We thus instead turn to maximizing the *observed* data log-likelihood:

$$\mathcal{L}(\mathcal{C}; \phi) \stackrel{\text{def}}{=} \sum_{m=1}^M \log P(\mathbf{S}=\mathbf{s}_m; \phi) \quad (12a)$$

$$= \sum_{m=1}^M \log \sum_{\mathbf{v} \in \mathcal{T}_{\mathcal{V}}(\mathbf{s}_m)} P(\mathbf{V}=\mathbf{v}; \phi) \quad (12b)$$

where $\mathcal{C} = \{\mathbf{s}_m\}_{m=1}^M$ is the corpus of strings. However, this value is difficult to optimize directly due to the log-sum structure. This is where the expectation-maximization (EM) algorithm comes in.

By Jensen’s inequality, we can show that the *expected value* of the complete log-likelihood serves as a lower bound for the *observed* log-likelihood. We provide this derivation specifically for our setting. Let $\mathcal{Q}(\phi; \phi^{(n)})$ denote the expected complete data log-likelihood:

$$\mathcal{Q}(\phi; \phi^{(n)}) \stackrel{\text{def}}{=} \sum_{m=1}^M \mathbb{E}_{\mathbf{v} \sim \mathbf{V} | \mathbf{S}=\mathbf{s}_m; \phi^{(n)}} [\log P(\mathbf{S}=\mathbf{s}_m, \mathbf{V}=\mathbf{v}; \phi)] \quad (13)$$

For any valid probability distribution $P(\mathbf{V} = \mathbf{v})$, given our established relationships between $P(\mathbf{V} = \mathbf{v})$ and $P(\mathbf{S}=\mathbf{s}; \phi)$, we have by Jensen’s inequality that:

$$\begin{aligned} \log P(\mathbf{S}=\mathbf{s}; \phi) &= \log \sum_{\mathbf{v} \in \mathcal{T}_{\mathcal{V}}(\mathbf{s})} P(\mathbf{V} = \mathbf{v}) \frac{P(\mathbf{V}=\mathbf{v}; \phi)}{P(\mathbf{V} = \mathbf{v})} \\ &\geq \sum_{\mathbf{v} \in \mathcal{T}_{\mathcal{V}}(\mathbf{s})} P(\mathbf{V} = \mathbf{v}) \log \frac{P(\mathbf{V}=\mathbf{v}; \phi)}{P(\mathbf{V} = \mathbf{v})} \end{aligned}$$

Let’s now choose $P(\mathbf{V} = \mathbf{v})$ to be $P(\mathbf{V} = \mathbf{v} | \mathbf{S}=\mathbf{s}; \phi^{(n)})$, the posterior under our current parameter beliefs for a fixed $\mathbf{s}$. Applying Jensen’s inequality (above) to our definition of the observed data log-

likelihood from Eq. (12) gives us

$$\mathcal{L}(\mathcal{C}; \phi) = \sum_{m=1}^M \log \sum_{\mathbf{v} \in \mathcal{T}_{\mathcal{V}}(\mathbf{s}_m)} P(\mathbf{V} = \mathbf{v}; \phi) \quad (14a)$$

$$\geq \sum_{m=1}^M \sum_{\mathbf{v} \in \mathcal{T}_{\mathcal{V}}(\mathbf{s}_m)} P(\mathbf{V} = \mathbf{v} \mid \mathbf{S} = \mathbf{s}_m; \phi^{(n)}) [\log P(\mathbf{V} = \mathbf{v}; \phi) - \log P(\mathbf{V} = \mathbf{v} \mid \mathbf{S} = \mathbf{s}_m; \phi^{(n)})] \quad (14b)$$

$$= \sum_{m=1}^M \sum_{\mathbf{v} \in \mathcal{T}_{\mathcal{V}}(\mathbf{s}_m)} P(\mathbf{V} = \mathbf{v} \mid \mathbf{S} = \mathbf{s}_m; \phi^{(n)}) \log P(\mathbf{S} = \mathbf{s}_m, \mathbf{V} = \mathbf{v}; \phi) \quad (14c)$$

$$\begin{aligned} & - \sum_{m=1}^M \sum_{\mathbf{v} \in \mathcal{T}_{\mathcal{V}}(\mathbf{s}_m)} P(\mathbf{V} = \mathbf{v} \mid \mathbf{S} = \mathbf{s}_m; \phi^{(n)}) \log P(\mathbf{V} = \mathbf{v} \mid \mathbf{S} = \mathbf{s}_m; \phi^{(n)}) \\ & = \underbrace{\sum_{m=1}^M \sum_{\mathbf{v} \sim \mathbf{V} \mid \mathbf{S} = \mathbf{s}_m; \phi^{(n)}} \mathbb{E} [\log P(\mathbf{S} = \mathbf{s}_m, \mathbf{V} = \mathbf{v}; \phi)]}_{\mathcal{Q}(\phi; \phi^{(n)})} \\ & \quad + \underbrace{\sum_{m=1}^M \sum_{\mathbf{v} \sim \mathbf{V} \mid \mathbf{S} = \mathbf{s}_m; \phi^{(n)}} - \mathbb{E} [\log P(\mathbf{V} = \mathbf{v} \mid \mathbf{S} = \mathbf{s}_m; \phi^{(n)})]}_{\mathcal{H}(\mathbf{V} \mid \mathbf{S} = \mathbf{s}_m; \phi^{(n)})} \end{aligned} \quad (14d)$$

When going to Eq. (14c), we use the fact that for any $\mathbf{v} \in \mathcal{T}_{\mathcal{V}}(\mathbf{s}_m)$, it must be that $P(\mathbf{S} = \mathbf{s}_m, \mathbf{V} = \mathbf{v}; \phi) = P(\mathbf{V} = \mathbf{v}; \phi)$. Eq. (14d) simply inserts some common definitions. Namely, the definitions of expected values and (Shannon) entropy. The entropy term $\mathcal{H}(\mathbf{V} \mid \mathbf{S} = \mathbf{s}_m; \phi^{(n)})$ is constant with respect to ϕ , and so we can ignore it in our optimization.

The EM algorithm uses the above relationship and proposes a procedure for iteratively maximizing the expected complete data log-likelihood with respect to model parameters, where the expectation is taken with respect to the posterior distribution defined by current parameter beliefs. The E-step computes this expected value; the M-step maximizes it with respect to the free parameters. In our case, expected token counts are a sufficient statistic for the M-step objective, leading to the algorithm given in Section 2.1.

This procedure can be intuitively understood as iteratively performing maximum likelihood estimation for a categorical model, where observed token counts are replaced by posterior-expected counts induced by the distribution over latent segmentations given current parameter beliefs. The usage of EM for this problem is taken from prior work and its theoretical validity for problems of the same structure has been well established (Dempster et al., 1977; Brown et al., 1993).

E.3 UnigramLM Vocabulary Pruning

As our method keeps $\mathcal{V}$ fixed, we omit this step entirely; we include this description of pruning done in the original UnigramLM algorithm only for completeness. The UnigramLM algorithm was designed for the case that we do not start with a known, fixed $\mathcal{V}$. It thus incorporates a strategy for choosing $\mathcal{V}$ by initializing $\mathcal{V}$ as an over-sized set and adding a pruning step at the end of each EM iteration. This pruning step uses current parameter beliefs to estimate the log-likelihood contribution of each token to the expected observed data log-likelihood. It then prunes a percentage of tokens that contribute least, stopping once the desired vocabulary size (a hyperparameter set by the user) has been reached. Land and Pinter (2026) discuss this strategy and its effects. Our algorithm uses several of the methods from UnigramLM but is not tied to any particular method for choosing $\mathcal{V}$: the vocabulary may equally be one learned by BPE, or that of a pretrained model.