Unsupervised Morphological Tree Tokenizer

Anonymous ACL submission

Abstract

As a cornerstone in language modeling, tokenization involves segmenting text inputs into pre-defined atomic units. Conventional statistical tokenizers often disrupt constituent boundaries within words, thereby corrupting semantic information. To address this drawback, we introduce morphological structure guidance to tokenization and propose a deep model to induce character-level structures of words. Specifically, the deep model jointly encodes internal structures and representations of words with a mechanism named *MorphOverriding* to ensure the indecomposability of morphemes. By training the model with self-supervised objectives, our method is capable of inducing character-level structures that align with morphological rules without annotated training data. Based on the induced structures, our algorithm tokenizes words through vocabulary matching in a top-down manner. Empirical results indicate that the proposed method effectively retains complete morphemes and outperforms widely adopted methods such as BPE and WordPiece on both morphological segmentation tasks and language modeling tasks. The code will be released later.

1 Introduction

Tokenization, the initial step of language modeling, segments natural language into manageable units. While this process is crucial for representing natural language, research on new tokenization methods has remained limited, particularly in contrast to the rapid advancements in language model architectures and learning approaches. Currently, the de-facto tokenizers are BPE (Sennrich et al., 2016) and WordPiece (Schuster and Nakajima, 2012), which have been widely adopted by state-of-the-art language models such as GPT (Radford et al., 2019) and BERT (Devlin et al., 2019). However, numerous studies have challenged these methods (Bostrom and Durrett, 2020; Church, 2020; Hofmann et al., 2021; Minixhofer et al., 2023), arguing that they cannot adequately capture linguistic information. They often disrupt constituent boundaries within words, leading to unnatural and fragmented token representations. Figure 1(top) demonstrates an example where BPE fails to identify the appropriate boundaries in a word.

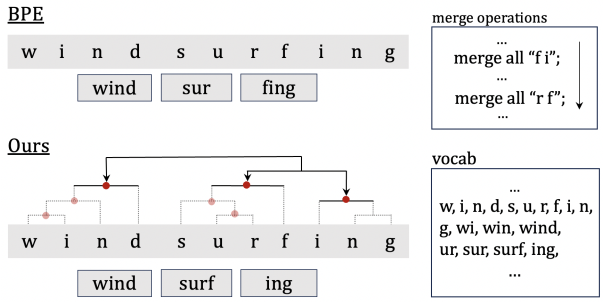

Figure 1: BPE (top) tokenizes a word through a bottom-up greedy merging approach given pre-learned merge operations, while ours (bottom) tokenizes a word via a top-down vocabulary matching while traversing a global parse tree.

Inspired by linguistic theories that words have internal structures (Selkirk, 1982; Marvin, 2002; Cotterell and Schütze, 2015), we seek to tokenize words based on their morphological structures as shown in Figure 1(bottom), aiming to enhance the alignment between tokenization and the morphological intricacies of language. Specifically, we consider an unsupervised approach to induce character-level morphological structures within words, which circumvents the need for annotated morphological data that are unavailable for many languages. Our approach draws inspiration from syntactic composition models (Maillard et al., 2017), where a sentence is encoded as a weighted sum over all composed root representations of its underlying binary parse trees via dynamic programming. Instead of composing a sentence from words, we apply composition models on characters in a word to induce its morphological parse tree. To train the composition model, we propose two self-supervised objectives akin to next token prediction and span prediction that effectively leverage both contextual information at the sentence level and semantic information at the subword level. Thus the model can learn to assign higher probabilities to morphological constituents of a word and induce the underlying morphological parse tree.

However, character sequences present a unique challenge to composition models because morphemes, the smallest meaning-bearing units in a language (Jurafsky and Martin, 2009), are indecomposable. While we can represent a constituent by composing its sub-constituents in most cases, we cannot represent a subword by composing its components if the subword is a morpheme. For example, the meaning of `windsurf` can be decomposed to `wind+surf`, but `wind` is a morpheme whose meaning is not a function of its components. To address the challenge, we propose a mechanism named *MorphOverriding*. During the bottom-up composition process in our model, upon identifying a subword that matches an entry in a heuristically constructed morpheme vocabulary, we compute the subword representation from both its components and the corresponding morpheme embedding, i.e., the model may learn to mix or override the composition with the morpheme embedding. Our experiments show that such a mechanism is critical in morphological structure induction.

Building upon the resolution of morphological structure induction, we introduce a novel tokenization algorithm named *TreeTok*, which includes both vocabulary construction and word segmentation. During vocabulary construction, TreeTok first utilizes a tree-based BPE variant to build an initial vocabulary and then applies a tree-based Unigram variant to prune the initial vocabulary to a specified size. Because TreeTok operates in a top-down manner, it does not need to retain all intermediate tokens produced by merge operations in the vocabulary as BPE does. By this means, we can build a more compact vocabulary by pruning less important subwords. During word segmentation, we employ a lightweight parser with compact parameters distilled from the composition model to parse a word into a character-level binary tree and then apply top-down vocabulary matching to enhance the tokenizer's alignment to morphological structure, as illustrated in Figure 1.

In our experiments, we train TreeTok and baselines on the Wikitext-103 corpus (McClosky et al., 2006) and assess their performance on morphological segmentation tasks and language modeling tasks. Evaluation results indicate that TreeTok consistently outperforms BPE and WordPiece across all the tasks.

In conclusion, our contributions are three-fold:

- We propose a composition model with MorphOverriding for modeling the compositionality within words.
- We propose effective self-supervised objectives to train the composition model to induce morphological structures of words highly consistent with gold morphological segmentations.
- We propose a novel tokenization algorithm based on induced morphological parse trees.

2 Related Work

Subword Tokenizers. Subword tokenization, with typical methods such as BPE (Sennrich et al., 2016) and WordPiece (Schuster and Nakajima, 2012), has become customary in most NLP fields. BPE builds its vocabulary by repeatedly merging the most frequent subword unit pairs, whereas WordPiece selects pairs using the highest mutual information. During tokenization, both apply these learned merge operations in the same order to new text initialized with characters. Unigram (Kudo, 2018), another popular tokenizer, builds its vocabulary in the opposite direction: it starts with a large set of potential subwords and prunes them based on delta entropy in a unigram language model.

Our tokenizer aims to build upon the advantages of these effective statistical tokenizers and augment them with unsupervised induced tree structures.

Unsupervised Morphological Segmentation. In the line of work on unsupervised morphological segmentation, the most well-known model is Morfessor (Creutz and Lagus, 2002), along with its multiple variants (Creutz and Lagus, 2005; Grönroos et al., 2014, 2020). In Morfessor, an online search algorithm is utilized to apply a hierarchical word splitting strategy with a Minimum Description Length (MDL) (Rissanen, 1989) cost function. However, although some studies (Ataman and Federico, 2018; Hou et al., 2023) find unsupervised morphologically motivated segmentation can improve data-driven tokenizers, most other studies (Machácek et al., 2018; Domingo et al., 2019; Sälevä and Lignos, 2021) find no reliable improvement of such methods over BPE. According to Gallé (2019), the effectiveness of BPE lies in its superior compression capability. A more detailed discussion can be found in Mielke et al. (2021).

Some other studies try to model morphological structures using Bayesian PCFGs (Johnson et al., 2007) or a non-parametric Bayesian generalization of PCFGs (Johnson et al., 2006). However, they are pure statistical models and do not utilize modern neural methodologies.

Our method differs from previous unsupervised morphological methods in our character-based structures, thereby possessing the superior compression capability of BPE. Meanwhile, our method leverages modern neural methodologies to better utilize contextual and intra-word semantic information.

Composition Model. In this work, we utilize a composition model to induce morphological structures. Composition models jointly learn representations and structures of a symbol sequence by transforming text encoding into a combinatorial optimization problem. Maillard et al. (2017) proposes a CKY-like (Cocke, 1969; Kasami, 1966; Younger, 1967) encoder, in which each constituent is represented as a weighted average of the set of composed representations computed from different splits of the constituent. Drozdov et al. (2019) proposes a deep inside-outside encoder (Baker, 1979; Lari and Young, 1990), enabling the encoder to learn underlying structures via an auto-encoding objective. Recently, a series of studies (Hu et al., 2024a,b) have been conducted to reduce the deep inside-outside encoder complexity from cubic to linear, on which our work is based.

3 Methodology

Given a word $\mathbf{x} = \{x_1, x_2, ..., x_n\}$ where x_i is the i-th character, we aim to parse it into a binary tree and then tokenize it via top-down vocabulary matching. The parser is a deep composition model capable of jointly modeling the internal structures and representations of words and is trained by optimizing self-supervised objectives. In the following sections, we sequentially introduce the composition model, training objectives, and the tree-based tokenization algorithm.

3.1 Composition Model for Word

For a given word $\mathbf{x}$, we denote $\mathbf{i}_{i,j}$ as the representation of subword $\mathbf{x}_{i:j} = \{x_i, ..., x_j\}$. The inside pass (Drozdov et al., 2019) of a composition model computes a composition vector $\bar{\mathbf{i}}_{i,j}^k$ and a compatibility score $\bar{a}_{i,j}^k$ for each pair of sub-constituents (i, k) and $(k+1, j)$. The compatibility score indicates how likely these two sub-constituents are to be merged. The constituent representation $\mathbf{i}_{i,j}$ is computed as a weighted average over composition vectors of all possible pairs of sub-constituents as follows:

$$\bar{a}_{i,j}^k, \bar{\mathbf{i}}_{i,j}^k = f_\alpha(\mathbf{i}_{i,k}, \mathbf{i}_{k+1,j}),$$
$$\hat{w}_{i,j}^k = \frac{\exp(\bar{a}_{i,j}^k)}{\sum_{k'=i}^{j-1} \exp(\bar{a}_{i,j}^{k'})}, \mathbf{i}_{i,j} = \sum_{k=i}^{j-1} \hat{w}_{i,j}^k \bar{\mathbf{i}}_{i,j}^k . \quad (1)$$

The inside pass starts with characters by initializing $\mathbf{i}_{i,i}$ with character embeddings and recursively computes constituent representations bottom up following Equation 1. Representation $\mathbf{i}_{1,n}$ of the whole word $\mathbf{x}$ is regarded as the word embedding $\text{EMB}(\mathbf{x})$. f_α is the composition function implemented with a multi-layered Transformer. An example of the bottom-up composition process is depicted in Figure 2(a). In this work, we employ a pruned version of deep inside encoder (Hu et al., 2024b) as our backbone, which is easy to scale up, thanks to the logarithmic parallel time complexity and the linear space complexity.

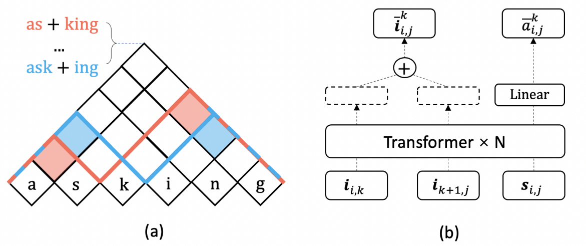

Figure 2: (a) The composition representation of asking ($\mathbf{i}_{1,6}$) is a weighted sum over all subword pairs such as ask+ing ($\bar{\mathbf{i}}_{1,6}^3$) and as + king ($\bar{\mathbf{i}}_{1,6}^2$). (b) The composition function. Take ask ($\mathbf{i}_{1,3}$) as an example. $\mathbf{s}_{1,3}$ is $\mathbf{E}_{\mathbb{V}[\text{ask}]}$ if ask $\in \mathbb{V}$. Thus the representation of ask depends not only on its components but also on $\mathbf{E}_{\mathbb{V}[\text{ask}]}$. However, if asking $\notin \mathbb{V}$, then $\mathbf{s}_{1,6}$ is $\mathbb{E}_{empty}$ and the representation of asking ($\mathbf{i}_{1,6}$) only depends on the composition representation of its components.

The limitation of this approach is that the representation of any subword is always composed of its component pairs, which is incompatible with the linguistic constraint that morphemes are the smallest meaning-bearing units and should not be decomposed further. Hence, we introduce MorphOverriding to enable a subword representation to disentangle from its component pairs when the subword is a morpheme. Specifically, we construct a morpheme vocabulary $\mathbb{V}$ heuristically using a statistical method (BPE in this work), in which each entry is associated with a learnable vector in a morpheme embedding table $\mathbf{E}$. When $\mathbf{x}_{i:j}$ hits the vocabulary $\mathbb{V}$, we insert its morpheme embedding $\mathbf{s}_{i,j}$ into the computation of $\mathbf{i}_{i,j}$, making it possible to mix or override the composition vector with the morpheme embedding. Thus, the composition vector and the compatibility scores can then be reformulated as:

$$\bar{a}_{i,j}^k, \bar{\mathbf{i}}_{i,j}^k = f_\alpha(\mathbf{i}_{i,k}, \mathbf{i}_{k+1,j}, \mathbf{s}_{i,j}),$$
$$\mathbf{s}_{i,j} = \begin{cases} \mathbf{E}_{\mathbb{V}[\mathbf{x}_{i:j}]} & \text{if } \mathbf{x}_{i:j} \in \mathbb{V} \\ \mathbf{E}_{empty} & \text{if } \mathbf{x}_{i:j} \notin \mathbb{V} \end{cases},$$

Figure 2(b) illustrates the composition function equipped with MorphOverriding. Our experiments demonstrate that this mechanism is crucial for character-level structure induction.

Tree induction. For a given span (i, j), the best split-point is k with the highest compatibility score $\bar{a}_{i,j}^k$. Thus, to derive a parse tree, we can recursively select the best split-points top-down starting from the root span $(1, n)$. As the pruned inside-outside encoder produces a lightweight parser (Hu et al., 2022) with a compact parameter set as a byproduct, we use it for efficient inference during tokenization.

3.2 Training Objectives

The overall loss for training the composition model is the summation of an auto-encoding loss $\mathcal{L}_{ae}$ and an auto-regression loss $\mathcal{L}_{ar}$. The auto-encoding loss is based on predicting each character or morpheme from the rest of a word, leveraging intra-word information. The auto-regression loss is based on next token prediction that leverages contextual information to disambiguate the underlying structures of a word. Under these objectives, the composition model learns to assign proper scores to each split point of a subword.

Auto-encoding Loss. Auto-encoding is a common practice of training a composition model. For our character-level composition model, we try to predict each character x_i based on its neighboring context representations $\mathbf{i}_{1,i-1}$ and $\mathbf{i}_{i+1,n}$ (Hu et al., 2021). However, the auto-encoding objective turns out to be empirically ineffective when training our model probably because unlike word-level auto-encoding that requires selecting from tens of thousands of words in a vocabulary, here we only need to select from tens of characters, which is much less challenging. To enhance learning efficacy, we propose predicting both individual characters and morphemes in the vocabulary $\mathbb{V}$. For instance, given the word `windsurf`, we mask out `wind` and let the model uncover the masked morpheme based on the visible part `_surf`. Analogous to the inside pass, the outside pass computes each outside representation $\mathbf{o}_{i,j}$ in a top-down manner based on context information outside span (i, j), whose details are described in Appendix A.3. we use $\mathbf{o}_{i,j}$ to predict each subword $\mathbf{x}_{i:j}$ that belongs to $\mathbb{V}$:

$$\mathcal{L}_{ae} = -\frac{1}{\mathcal{N}} \sum_{\mathbf{x}_{i:j} \in \mathbb{V}} \log \frac{\exp(\mathbf{o}_{i,j}^T \mathbf{E}_{\mathbb{V}[\mathbf{x}_{i:j}]})}{\sum_{k=1}^{|\mathbb{V}|} \exp(\mathbf{o}_{i,j}^T \mathbf{E}_k)},$$

where $\mathcal{N}$ is the total number of subwords belonging to the vocabulary [1]

Auto-regression Loss. Given a sentence $\mathbf{S} = \{\mathbf{x}^1, ..., \mathbf{x}^m\}$, whose word embedding is computed by the composition model, we feed the composed word embeddings into a causal language model and let it pick the correct next word from candidates built via in-batch sampling for each step. Let $\mathbf{h}_t$ denote the t-th hidden states of the causal language model and $\mathcal{W}$ denote a deduplicated vocabulary built on all input words in the same batch, we have the auto-regression loss defined as:

$$\mathcal{L}_{ar} = -\frac{1}{m} \sum_{t=1}^{m-1} \log \frac{\exp(\mathbf{h}_t \mathrm{EMB}(\mathbf{x}^{t+1}))}{\sum_{\mathbf{x} \in \mathcal{W}} \exp(\mathbf{h}_t \mathrm{EMB}(\mathbf{x}))}.$$

3.3 Tokenization

The proposed tree-based tokenization algorithm, TreeTok, includes segmentation and vocabulary construction procedures. As the latter depends on the former, we first discuss the segmentation procedure, followed by the vocabulary construction.

Segmentation Procedure. Given a constructed vocabulary, whose details are described later, we parse each word into a morphological tree and segment it via a *top-down matching* approach, as illustrated in Figure 1(bottom). Specifically, during the top-down traversal of a parse tree, we retain a subword and backtrack if the subword matches an entry in the vocabulary. Note that unsupervised structural learning is often imperfect, causing erroneous tokenization. For instance, an incorrect parse tree `[[[book]e]d]` may yield tokens `book` `e` `d` where `e` `d` should be merged. To address this issue, we propose a post-processing step to deal with mergeable pairs of segmented tokens. Specifically, we define the information entropy of a token

[1] Note that multiple subwords may be mutually exclusive, such as "asking" with "ask" and "king". Intuitively, it is more reasonable to predict a constituent than a distituent from its context. Therefore, we assign a constituency weight to each subword in the objective, as detailed in Appendix A.4.

t as $-\sum \log \frac{1}{\text{COUNT}(t)}$, where $\text{COUNT}(t)$ is the frequency of t in the entire corpus. Therefore, the entropy of a certain merge is the sum of the information entropies of all tokens. We find the optimal merge by searching for the one with minimal information entropy among all potential merges via dynamic programming. Detailed pseudo-code can be found in Appendix A.1.

Vocabulary Construction. One drawback of BPE and WordPiece is that they have to keep all intermediate "junk" tokens produced during the iterations of merge operations, which results in limited vocabulary space occupied by these meaningless tokens. For instance, if the corpus contains many occurrences of `low` and `lower`, the meaningless token `lo` will be added to the vocabulary before `low` and will not be removed later. However, with the top-down matching framework, we don't need bottom-up merge operations to restore tokens, allowing us to prune unnecessary tokens and create a more compact vocabulary. To build a compact vocabulary, we propose a vocabulary construction algorithm in which we employ a tree-based BPE-like algorithm to build a heuristic vocabulary and a tree-based Unigram algorithm to prune unnecessary subword units. Specifically, we initialize the token vocabulary with the character vocabulary and repeat the following steps to build a heuristic vocabulary given character-level tree structures of words:

1. Count adjacent token pairs that share the same parent in the tree structure, e.g., given `[[b[o o]]k]`, only the pair (`o`, `o`) is counted.
2. Merge adjacent symbol pairs whose counts exceed a given threshold, e.g., `[[b[o o]]k]` $\rightarrow$ `[[b oo]k]`.
3. Repeat 1-2 until there are no new symbol pairs.

In the pruning procedure, we start from the heuristic symbol vocabulary and prune it as follows:

1. Tokenize the corpus via the top-down matching according to the current vocabulary. The total entropy of the whole corpus is defined as $\mathcal{H}_{\mathbb{V}} = -\sum_{t\in\mathbb{V}} \text{COUNT}(t) \log \frac{1}{\text{COUNT}(t)}$.
2. For each token s, calculate the entropy gain after removing that word from the vocabulary denoted as $\Delta\mathcal{H}_s = \mathcal{H}_{\mathbb{V}/\{s\}} - \mathcal{H}_{\mathbb{V}}$. Intuitively, the higher $\Delta\mathcal{H}_s$ is, the more important s is.
3. Sort delta entropy of tokens and remove the lowest $k\%$ from $\mathbb{V}$. Repeat step 1-2 until $|\mathbb{V}|$ reaches the target vocabulary size.

In practice, we design a tree-based Viterbi algorithm (Viterbi, 1967) to implement the pruning procedure efficiently. The pseudo-code is presented in Appendix A.2.

4 Experiments

We evaluate the performance of TreeTok against the de-facto tokenizers such as BPE, WordPiece, and Unigram as primary baselines.

Training setups. For a fair comparison, we train all tokenizers from scratch on the lowercase version of the Wikitext-103 corpus (McClosky et al., 2006) without any word boundary marker and set the same vocabulary size of 30,000. For BPE, WordPiece, and Unigram, we use the implementation and default training paradigm provided by the HuggingFace library[2]. Regarding the composition model, we train it with a context window of up to 512 characters. We use GPT2 implemented from HuggingFace[3] as our causal language model when computing the auto-regression loss. We present detailed configurations of our model and training setup in Appendix A.5.

Evaluation datasets. We compare our tokenizer with other tokenizers for morphological alignment using two datasets with gold-standard morphological segmentation. One is from the Morpho Challenge 2010 Workshop (Kurimo et al., 2010) (Morpho), which contains 1,000 word forms with their segmentations corresponding to the surface forms of morpheme labels. The dataset contains instances of all kinds of morphological transformations, including inflection, derivation, and compounding. The other dataset is from Minixhofer et al. (2023) (Compound), which contains 759 compound words specifically designed to test the models' capabilities in decompounding. We also use these morphological segmentation datasets to evaluate the induced morphological parse trees.

In addition, we evaluate the tokenizers using statistical metrics that have been shown to strongly correlate with the performance on downstream tasks. These metrics are calculated on the validation set of Wikitext-103.

4.1 Tokenization Quality

Metrics. We measure the performance of morphological segmentation via accuracy, i.e., the ra-

[2] https://github.com/huggingface/tokenizers
[3] https://github.com/huggingface/transformers

	Morpho (Acc.) ↑	Compound (Acc.) ↑	$\|\mathbb{V}\|$
BPE	19.50	62.98	30,000
WordPiece	26.20	62.19	30,000
Unigram	27.10	53.10	30,000
TreeTok	**37.9**	**68.07**	30,000
For reference			
SECOS†	—	41.2	N.A.
Morfessor	41.30	84.85	85,000+

Table 1: Results on two morphological segmentation datasets. The result of SECOS† is copied from Minixhofer et al. (2023). Due to Morfessor's inability to specify the size of the vocabulary, its utility as a tokenizer is limited. Therefore, the results of SECOS and Morfessor are for reference because of their different vocabulary sizes.

	Rényi↑	PPL↓	avg. #tokens
BPE	44.66	107.76	26.58
WordPiece	44.54	110.97	26.60
Unigram	45.07	106.91	31.68
TreeTok	44.82	107.26	25.99

Table 2: Results for different tokenization models on WikiText103 with 30,000 vocabulary size.

tio of examples that are correctly segmented. We also consider a few statistical metrics that can directly assess the quality of tokenization, including Rényi Efficiency (Zouhar et al., 2023), average sentence-level perplexity, and average number of tokens per sentence. Rényi Efficiency is introduced by Zouhar et al. (2023) as a principled intrinsic measure of tokenization quality and is claimed to yield a Pearson correlation of 0.78 with BLEU (Papineni et al., 2002) on machine translation. Sentence-level perplexity is defined as $-\log\ p(\mathbf{s}) = -\sum_{i=1}^{n} \log\ p(s_i|s_{<i})$, where $\mathbf{s} = \{s_1, s_2, ..., s_n\}$ is a sentence with s_i being the i-th token. Since different tokenizers generate distinct segmentations leading to different numbers of tokens of the same word, sentence-level perplexity provides fairer evaluation compared with the default token-level perplexity $-\frac{1}{n}\log\ p(\mathbf{s})$.

Baselines. In addition to BPE, WordPiece, and Unigram, we also include two linguistically-motivated segmentation methods SECOS (Riedl and Biemann, 2016) and Morfessor (Creutz and Lagus, 2002) for reference.

Morphological Segmentation. According to Table 1, TreeTok significantly surpasses BPE, WordPiece, and Unigram on the two morphological segmentation datasets. The results demonstrate the efficacy of TreeTok in aligning with morphology.

The superiority of Morfessor as shown in the table mainly comes from its much larger vocabulary. On the Compound task, we observe many cases where TreeTok segments a low-frequency constituent word into smaller pieces due to its absence in vocabulary, whereas Morfessor can find it in its extensive vocabulary. Because of the huge difference in vocabulary sizes, Morfessor is not directly comparable to the other tokenizers.

Rényi efficiency & Perplexity. Table 2 reports the evaluation results in terms of Rényi efficiency and perplexity (PPL). TreeTok outperforms BPE and WordPiece on both Rényi and PPL. The improvements illustrate the benefits of TreeTok's structural constraints and more compact vocabulary. The tree structure constraints enable the segmentation of words into more morphology-aligned tokens, while the compact vocabulary allows for the inclusion of meaningful morphemes by removing intermediate tokens in the pruning process during vocabulary construction, under a top-down matching framework. Unigram performs slightly better than TreeTok, but produces 22% more tokens on average. A possible explanation for the better performance of Unigram is that Unigram tends to produce inflectional suffixes such as "`-ing`" and "`-ly`", while other methods tend to retain entire words. This difference makes it easier for Unigram to share the same stems and affixes between different word forms, thus achieving better parameter sharing. However, under the Transformer architecture, an additional 22% number of tokens means extra inference steps and nearly 1.4 times the cost of self-attention. Such additional costs only bring marginal improvements as can be seen in the table.

We also note that TreeTok achieves the shortest average token length among all the tokenizers, which is desirable as Gallé (2019) shows that *given a fixed vocabulary size budget, the fewer tokens a tokenizer needs to cover the test set, the better the translation.*

4.2 Tree Structure Quality

Since tree structures play an important role in both vocabulary construction and segmentation, we evaluate the quality of trees induced by various composition models.

Metric. We use recall of morphemes (van den Bosch and Daelemans, 1999) in a tree to assess the quality of the tree structures against gold-standard segmentations, which is defined as the percentage

	Morpho EN.	Compound EN.
Fast R2D2	67.69	48.96
Neural PCFG	39.87	58.33
TreeTok	**90.10**	**86.20**
w/o context	70.00	63.02
w/o *MorphOverriding*	75.99	46.35
w/o span loss	86.79	73.70

Table 3: Performance evaluation of our model, baseline models, and ablation studies on morphological segmentation, measured by morpheme recall rate.

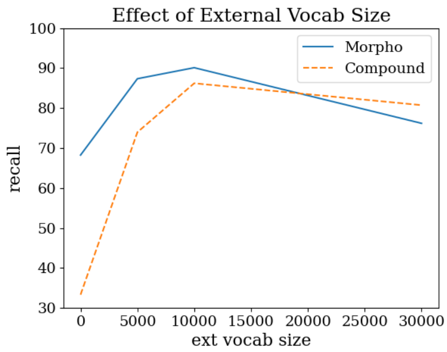

Figure 3: The effect of changing the vocabulary size learned by BPE. The initial results on both tasks show that the performance curve is a concave function where the maximum resides in the middle.

of morphemes in the gold segmentation that can be found in the spans of the evaluated tree. We discard spans that are trivial for a tree (character-level and word-level spans) and report word-level recall (averaged over word samples).

Baselines. For baseline composition models, we include Fast-R2D2 (Hu et al., 2022), which is a variant of DIORA (Drozdov et al., 2019), and an efficient variant of neural PCFG (Yang et al., 2022).

We also include four variants of our composition model for an ablation study. In w/o context, we remove the auto-regression loss from our architecture so that each representation only contains information from individual words. In w/o MorphOverriding, we degenerate $\mathbf{s}_{i,j}$ to the default empty embedding regardless of whether span $\mathbf{x}_{i:j}$ hits the external vocabulary or not. In w/o span loss, for our auto-encoding loss, we only count loss from predicting characters instead of every subword span that hits the external vocabulary.

Results and Discussions. As shown in Table 3, our model outperforms all the other composition models. Compared with Fast-R2D2, our main differences lie in the training objectives and the MorphOverriding mechanism. This result fully validates the effectiveness of these improvements. Our ablation experiments further analyze the contribution of these improvements to performance enhancement. Specifically, we have the following findings from each ablation.

Removing the auto-regression loss to prevent the model from getting feedback from contextual information significantly impacts the performance on both tasks, especially Morpho. We believe that contextual information can help the model capture the regularities of tenses and learn how to build composition representations for compound words. For example, consider how the context can help determine whether we should build the representation of `asking` as `ask+ing` or `as+king`. While either is a valid combination of morphemes, the former is more likely to be learned by our model since the context around `asking` often indicates the continuous tense or the gerund form, thus matching better with `ing`.

Removing MorphOverriding from the model results in a significant decrease of around 50% in performance on the decompounding task. The results consolidate our insight about conventional composition models violating the indecomposability of morphemes. Creating a morpheme's representation using its components' representation might make representations of disparate morphemes (e.g., `wind` and `win`) entangled together.

Removing the span loss also causes a performance drop on the two morphology tasks. This aligns well with the insight behind our design of morpheme-level loss, which augments the character-level loss by enhancing the learning of intra-word representations for most morphemes that are at an intermediate granularity.

Influence of Heuristic Vocabulary Size Additionally, we conduct experiments to investigate how the size of our heuristic morpheme vocabulary influences the performance of structure induction.

Figure 3 shows that the optimal size of an external vocabulary should be neither too large nor too small. According to our hypothesis that the compositional representation of subcomponents of a morpheme should be overridden by a high-level representation, ideally, the external vocabulary should contain all morphemes and only morphemes, because our model will trigger the soft morpheme overriding mechanism for every span that hits the

original word	bed	commonly	windsurfing	tricycles	uniquenesses
BPE	`bed`	`commonly`	`wind/sur/fing`	`tric/y/cles`	`uniqu/eness/es`
Unigram	`b/e/d`	`common/ly`	`wind/surf/ing`	`t/r/i/cycle/s`	`unique/ness/e/s`
WordPiece	`bed`	`commonly`	`winds/ur/fing`	`tric/y/cles`	`unique/ness/es`
TreeTok	`bed`	`commonly`	`wind/surf/ing`	`tri/cycles`	`unique/ness/es`

Table 4: Example tokenizations.

external vocabulary. If we use BPE to construct this external vocabulary and the vocabulary size is too small, many morphemes, particularly those in the form of long standalone words, will not have the chance to be included in the vocabulary. Conversely, if the vocabulary size is too large, BPE will continue to merge across morphemes, creating longer spans that are not the smallest meaning-bearing units.

4.3 Case Studies

To complement our quantitative evaluations of different tokenizers and further examine their difference, we list some of their tokenizations over the same words in Table 4. Since our tokenizer's performance is related to the tree structures learned by our composition model, we also include some of these tree structures in Figure 4.

Tokens produced by Unigram often include many characters. We observe that in addition to words that are out-of-domain (e.g., Japanese Romaji), single-character tokens mostly happen on short words with lengths of two or three (e.g., `as`, `it`, `to`, `oil`, etc.). This observation is consistent with notably longer tokenized sequences of Unigram observed in Table 2 compared to other tokenization methods. Note the overall likelihood of segmentation would be $\prod_{i=1}^{n} p(token_i)$ in a unigram language model. Therefore, we speculate that Unigram tends to split short words into characters since many characters have a dominantly high probability, while the length penalty resulting from the product of probabilities in short words is not significant.

BPE and WordPiece often violate morpheme boundaries and tokenize words into some intermediate "junk" tokens stored in their vocabulary during the bottom-up vocabulary construction. In Table 4, we can see many such examples such as `fing`, `cles`, and `eness`.

TreeTok aligns significantly better with morphology. Our vocabulary construction algorithm that brings the best from both BPE construction and

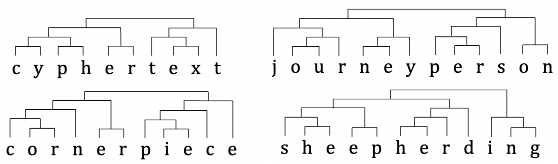

Figure 4: Example tree structures induced by our composition model.

Unigram pruning, helps eliminate "junk" tokens. In addition, with top-down matching under linguistic structure constraints, we alleviate the issue of breaking morpheme boundaries and excessively fragmenting words.

From Figure 4, we find the high-level structures of the trees learned by our model are generally accurate, whereas some of the low-level structures still do not make sense. This is because MorphOverriding allows the model to learn the most reasonable segmentation among different high-level splits based on context, while making the low-level structures less important, resulting in a certain degree of randomness within them.

5 Conclusion

In this work, we proposed a tree-based tokenizer (*TreeTok*) to better align with morphology. To make the method as general-purpose as possible, the most critical challenge is how to induce character-level structures without relying on human-annotated data. Our research revealed that the key to solving this problem lies in recognizing the indecomposability of morphemes. We introduce a composition model with a *MorphOverriding* mechanism to incorporate this inductive bias and propose two self-supervised objectives. Together, we can effectively induce tree structures that are highly consistent with human-labeled morphology. Ultimately, we achieved consistently positive results when comparing TreeTok against strong baselines like BPE and WordPiece on various tasks. This study provides new insights into unsupervised morphological segmentation and offers a novel approach different from traditional statistical models.

6 Limitations

Our main limitation is that we need additional training and inference overheads. Considering that the composition model only needs to be trained once and the overall time consumption is acceptable[4], we believe it is not a fatal flaw. Regarding inference cost, because a lightweight parser is produced as a byproduct, it can be afforded by even CPU environments.

References

Duygu Ataman and Marcello Federico. 2018. An evaluation of two vocabulary reduction methods for neural machine translation. In *Proceedings of the 13th Conference of the Association for Machine Translation in the Americas, AMTA 2018, Boston, MA, USA, March 17-21, 2018 - Volume 1: Research Papers*, pages 97–110. Association for Machine Translation in the Americas.

James K. Baker. 1979. Trainable grammars for speech recognition. *Journal of the Acoustical Society of America*, 65.

Kaj Bostrom and Greg Durrett. 2020. Byte pair encoding is suboptimal for language model pretraining. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4617–4624, Online. Association for Computational Linguistics.

Kenneth Ward Church. 2020. Emerging trends: Subwords, seriously? *Natural Language Engineering*, 26(3):375–382.

John Cocke. 1969. *Programming Languages and Their Compilers: Preliminary Notes*. New York University, USA.

Ryan Cotterell and Hinrich Schütze. 2015. Morphological word-embeddings. In *NAACL HLT 2015, The 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Denver, Colorado, USA, May 31 - June 5, 2015*, pages 1287–1292. The Association for Computational Linguistics.

Mathias Creutz and Krista Lagus. 2002. Unsupervised discovery of morphemes. In *Proceedings of the ACL-02 Workshop on Morphological and Phonological Learning*, pages 21–30. Association for Computational Linguistics.

Mathias Johan Philip Creutz and Krista Hannele Lagus. 2005. Inducing the morphological lexicon of a natural language from unannotated text. In *Proc. International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR'05)*, pages 106–113. International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR'05) ; Conference date: 01-01-1800.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Miguel Domingo, Mercedes García-Martínez, Alexandre Helle, Francisco Casacuberta, and Manuel Herranz. 2019. How much does tokenization affect neural machine translation? In *Computational Linguistics and Intelligent Text Processing - 20th International Conference, CICLing 2019, La Rochelle, France, April 7-13, 2019, Revised Selected Papers, Part I*, volume 13451 of *Lecture Notes in Computer Science*, pages 545–554. Springer.

Andrew Drozdov, Patrick Verga, Mohit Yadav, Mohit Iyyer, and Andrew McCallum. 2019. Unsupervised latent tree induction with deep inside-outside recursive auto-encoders. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1129–1141, Minneapolis, Minnesota. Association for Computational Linguistics.

Matthias Gallé. 2019. Investigating the effectiveness of BPE: The power of shorter sequences. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1375–1381, Hong Kong, China. Association for Computational Linguistics.

Stig-Arne Grönroos, Sami Virpioja, and Mikko Kurimo. 2020. Morfessor EM+Prune: Improved subword segmentation with expectation maximization and pruning. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 3944–3953, Marseille, France. European Language Resources Association.

Stig-Arne Grönroos, Sami Virpioja, Peter Smit, and Mikko Kurimo. 2014. Morfessor FlatCat: An HMM-based method for unsupervised and semi-supervised learning of morphology. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 1177–1185, Dublin, Ireland. Dublin City University and Association for Computational Linguistics.

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*

[4] 1 day for 8× A100 for WikiText-103

on Natural Language Processing (Volume 1: Long Papers), pages 3594–3608, Online. Association for Computational Linguistics.

Jue Hou, Anisia Katinskaia, Anh-Duc Vu, and Roman Yangarber. 2023. Effects of sub-word segmentation on performance of transformer language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 7413–7425. Association for Computational Linguistics.

Xiang Hu, Pengyu Ji, Qingyang Zhu, Wei Wu, and Kewei Tu. 2024a. Generative pretrained structured transformers: Unsupervised syntactic language models at scale.

Xiang Hu, XinYu KONG, and Kewei Tu. 2023. A multi-grained self-interpretable symbolic-neural model for single/multi-labeled text classification. In *The Eleventh International Conference on Learning Representations*.

Xiang Hu, Haitao Mi, Liang Li, and Gerard de Melo. 2022. Fast-R2D2: A pretrained recursive neural network based on pruned CKY for grammar induction and text representation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 2809–2821, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Xiang Hu, Haitao Mi, Zujie Wen, Yafang Wang, Yi Su, Jing Zheng, and Gerard de Melo. 2021. R2D2: Recursive transformer based on differentiable tree for interpretable hierarchical language modeling. 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 4897–4908, Online. Association for Computational Linguistics.

Xiang Hu, Qingyang Zhu, Kewei Tu, and Wei Wu. 2024b. Augmenting transformers with recursively composed multi-grained representations. In *The Twelfth International Conference on Learning Representations*.

Mark Johnson, Thomas Griffiths, and Sharon Goldwater. 2006. Adaptor grammars: A framework for specifying compositional nonparametric bayesian models. In *Advances in Neural Information Processing Systems*, volume 19. MIT Press.

Mark Johnson, Thomas Griffiths, and Sharon Goldwater. 2007. Bayesian inference for PCFGs via Markov chain Monte Carlo. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 139–146, Rochester, New York. Association for Computational Linguistics.

Daniel Jurafsky and James H. Martin. 2009. *Speech and Language Processing (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Tadao Kasami. 1966. An efficient recognition and syntax-analysis algorithm for context-free languages. *Coordinated Science Laboratory Report no. R-257*.

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.

Mikko Kurimo, Sami Virpioja, Ville Turunen, and Krista Lagus. 2010. Morpho challenge 2005-2010: Evaluations and results. In *Proceedings of the 11th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology*, pages 87–95, Uppsala, Sweden. Association for Computational Linguistics.

K. Lari and S.J. Young. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech & Language*, 4(1):35–56.

Dominik Machácek, Jonás Vidra, and Ondrej Bojar. 2018. Morphological and language-agnostic word segmentation for NMT. In *Text, Speech, and Dialogue - 21st International Conference, TSD 2018, Brno, Czech Republic, September 11-14, 2018, Proceedings*, volume 11107 of *Lecture Notes in Computer Science*, pages 277–284. Springer.

Jean Maillard, Stephen Clark, and Dani Yogatama. 2017. Jointly learning sentence embeddings and syntax with unsupervised tree-lstms. *CoRR*, abs/1705.09189.

Tatjana Marvin. 2002. *Topics in the Stress and Syntax of Words*. Ph.D. thesis, Massachusetts Institute of Technology.

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Effective self-training for parsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 152–159, New York City, USA. Association for Computational Linguistics.

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

Benjamin Minixhofer, Jonas Pfeiffer, and Ivan Vulić. 2023. CompoundPiece: Evaluating and improving decompounding performance of language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 343–359, Singapore. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

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

Martin Riedl and Chris Biemann. 2016. Unsupervised compound splitting with distributional semantics rivals supervised methods. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 617–622, San Diego, California. Association for Computational Linguistics.

Jorma Rissanen. 1989. Stochastic complexity in statistical inquiry. In *World Scientific Series in Computer Science*.

Jonne Sälevä and Constantine Lignos. 2021. The effectiveness of morphology-aware segmentation in low-resource neural machine translation. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop, EACL 2021, Online, April 19-23, 2021*, pages 164–174. Association for Computational Linguistics.

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and korean voice search. *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152.

Elisabeth Selkirk. 1982. *The Syntax of Words*. Linguistic inquiry monographs. MIT Press.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Antal van den Bosch and Walter Daelemans. 1999. Memory-based morphological analysis. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 285–292, College Park, Maryland, USA. Association for Computational Linguistics.

Andrew J. Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. Inf. Theory*, 13(2):260–269.

Songlin Yang, Wei Liu, and Kewei Tu. 2022. Dynamic programming in rank space: Scaling structured inference with low-rank HMMs and PCFGs. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4797–4809, Seattle, United States. Association for Computational Linguistics.

Daniel H Younger. 1967. Recognition and parsing of context-free languages in time n3. *Information and control*, 10(2):189–208.

Vilém Zouhar, Clara Meister, Juan Gastaldi, Li Du, Mrinmaya Sachan, and Ryan Cotterell. 2023. Tokenization and the noiseless channel. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5184–5207, Toronto, Canada. Association for Computational Linguistics.

A Appendix

A.1 Pseudo-codes of tokenization

Algorithm 1 Tokenize

Input: string $\mathbf{x}$, parse tree root r, vocabulary $\mathbb{V}$
procedure TOKENIZE($\mathbf{x}$, r, $\mathbb{V}$)
$t \leftarrow []$ ▷ tokenized subword units list
$stack \leftarrow [r]$
while $|stack| > 0$ **do**
$c \leftarrow \text{Pop}(stack)$
$i, j \leftarrow c.i, c.j$
$\bar{\mathbf{x}} \leftarrow \mathbf{x}_{i:j}$
if $\bar{\mathbf{x}} \in \mathbb{V}$ **then**
$\text{Append}(t, \bar{\mathbf{x}})$
else if $i < j$ **then** ▷ Non-terminal nodes
$\text{Push}(stack, c.right)$
$\text{Push}(stack, c.left)$
$t \leftarrow \text{PostMerge}(t, \mathbb{V})$
▷ post processing if over-split
return t

Algorithm 2 Post-Merge Algorithm

Input: tokens $\mathbf{t}$, vocab2entropy $\mathbb{V}$
procedure POSTMERGE($\mathbf{t}$, $\mathbb{V}$)
$n \leftarrow$ length of $\mathbf{t}$
if $n \leq 1$ **then**
$\mathbf{t}_{\text{MERGE}} \leftarrow \mathbf{t}$
else
$\mathcal{H}[n][n]$ init with ∞ ▷ Best entropy
$s[n][n]$ init with $[]$ ▷ Best segments
for $i \leftarrow 0$ to $n-1$ **do** ▷ Base case
$\mathcal{H}_{i,i} \leftarrow \mathbb{V}[x_i]$
$s_{i,i} \leftarrow [x_i]$
for $h \leftarrow 1$ to $n-1$ **do** ▷ Iterate tree height
for $i \leftarrow 0$ to $n-h-1$ **do**
$j \leftarrow i + h$
$k_{\text{BEST}} \leftarrow -1$
$m \leftarrow$ concatenate $t_i \dots t_j$
$\mathcal{H}_{\text{BEST}} \leftarrow \text{GET}(\mathbb{V}, m, \infty)$
for $k \leftarrow i$ to $j-1$ **do**
if $\mathcal{H}_{i,k} + \mathcal{H}_{k+1,j} \leq \mathcal{H}_{\text{BEST}}$ **then**
$k_{\text{BEST}} \leftarrow k$
$\mathcal{H}_{\text{BEST}} \leftarrow \mathcal{H}_{i,k} + \mathcal{H}_{k+1,j}$
if $k_{\text{BEST}} \neq -1$ **then**
$s_{i,j} \leftarrow s_{i,k_{\text{BEST}}} + s_{k_{\text{BEST}}+1,j}$
else
$s_{i,j} \leftarrow [m]$ ▷ Merge
$\mathcal{H}_{i,j} \leftarrow \mathcal{H}_{\text{BEST}}$
$\mathbf{t}_{\text{MERGE}} \leftarrow s_{0,n-1}$
return $\mathbf{t}_{\text{MERGE}}$

A.2 Pseudo-codes of vocab construction

Please refer to Algorithm 3 for details.

Algorithm 3 Vocabulary Construction

Input: tree-freq pair list T, vocab size k, pruning rate α
procedure VOCABULARY CONSTRUCTION(T, k, α)
procedure E-STEP($T, \mathbb{V}$)
$\mathbb{V}' \leftarrow \text{DICT}(\,)$ ▷ E-step: Update vocab freq
for $\{root, freq\} \in T$ **do**
$\_, seg \leftarrow \text{TREEVITERBI}(root, \mathbb{V}, null)$
for $token \in seg$ **do**
$\mathbb{V}'[token] \leftarrow \mathbb{V}'[token] + freq$
return $\mathbb{V}'$

procedure M-STEP($T, \mathbb{V}$)
$l \leftarrow \text{DICT}(\,)$ ▷ M-step: Update delta loss
for $\{root, freq\} \in T$ **do**
$l_{word} \leftarrow \text{DICT}(\,)$ ▷ word-level delta-loss
$\_, seg \leftarrow \text{TREEVITERBI}(root, \mathbb{V}, l_{word})$
for $token \in seg$ **do**
$loss \leftarrow l_{word}[token]$
$l[token] \leftarrow l[token] + loss * freq$
return $\mathbb{V}'$

$\mathbb{V} \leftarrow \text{INITVOCAB}(T)$ ▷ Init with a BIG vocab
while $|\mathbb{V}| > k$ **do**
$\mathbb{V} \leftarrow \text{E-STEP}(T, \mathbb{V})$ ▷ Estimate token count
$\mathbb{L} \leftarrow \text{M-STEP}(T, \mathbb{V})$ ▷ Maximize delta losses
Remove $\min(|\mathbb{V}| - k, \lfloor \alpha |\mathbb{V}| \rfloor)$ of the
tokens t with lowest L_t from $\mathbb{V}$
return $\mathbb{V}$

Algorithm 4 TreeViterbi

Input: parse tree root r, vocabulary $\mathbb{V}$, delta loss dict l
procedure TREEVITERBI($r, \mathbb{V}, l$)
$w \leftarrow r.token$
if $r.i = r.j$ **then**
$s \leftarrow \text{GET}(\mathbb{V}, w, \infty)$ ▷ Infinity entropy if $w \notin \mathbb{V}$
return $s, [w]$
else
$s_L, w_L \leftarrow \text{TREEVITERBI}(r.left, \mathbb{V}, l)$
$s_R, w_R \leftarrow \text{TREEVITERBI}(r.right, \mathbb{V}, l)$
$s \leftarrow \text{GET}(\mathbb{V}, w, \infty)$
if l **then** ▷ Enter in M-step
$l[w] \leftarrow l[w] + \text{MAX}(s_L + s_R - s, 0)$
▷ Record delta loss: Entropy increase
if $s_L + s_R > s$ **then**
return $s, [w]$
else
return $s_L + s_R, w_L + w_R$

Algorithm 5 Vocabulary Initialization

Input: tree-freq pair list T, threshold k
procedure INITVOCAB(T, k)
$\mathbb{V} \leftarrow$ All character freq
$n \leftarrow |\mathbb{V}|$
while True **do**
$\mathbb{V}' \leftarrow \text{COUNTBIGRAMS}(T, \mathbb{V})$
Prune all the entries in $\mathbb{V}'$ with freq less than k
$\mathbb{V}.\text{MERGE}(\mathbb{V}')$ ▷ Add new items in $\mathbb{V}'$ to $\mathbb{V}$
if $|\mathbb{V}| = n$ **then**
break
$n = |\mathbb{V}|$
return $\mathbb{V}$

Algorithm 6 Count Bigrams

Input: tree-freq pair list T, vocabulary $\mathbb{V}$
procedure COUNTBIGRAMS(T, $\mathbb{V}$)
  $\mathbb{V}' \leftarrow$ DICT( ) ▷ Store new merges
  procedure RECURCOUNT(r, f)
    if $r.left$ & $r.right$ **then**
      $hit_L \leftarrow$ RECURCOUNT($r.left, f$)
      $hit_R \leftarrow$ RECURCOUNT($r.right, f$)
      if hit_L and hit_R **then**
        if $r.token \in \mathbb{V}$ **then**
          return True
        else
          $\mathbb{V}'[r.token] \leftarrow f$ ▷ Merge: new entry
          return False
      else
        return False
    else
      return True
  for $\{root, freq\} \in T$ **do**
    RECURCOUNT($root, freq$)
  return $\mathbb{V}'$

A.3 The neural outside pass

The outside computation is akin to the inside pass but in a top-down manner. we denote the outside representation and score of a given span as $\bar{\mathbf{o}}_{i,j}^k$ and $\bar{b}_{i,j}^k$ respectively, whose parent span is (i, k) or (k, j) for $k > j$ or $k < i$.

$$\bar{\mathbf{o}}_{i,j}^k = \begin{cases} f_\beta(\mathbf{o}_{i,k}, \mathbf{i}_{j+1,k}) & \text{if } k > j \\ f_\beta(\mathbf{o}_{k,j}, \mathbf{i}_{k,i-1}) & \text{if } k < i \end{cases},$$
$$\bar{b}_{i,j}^k = \begin{cases} \phi_\beta(\mathbf{o}_{i,k}, \mathbf{i}_{j+1,k}) & \text{if } k > j \\ \phi_\beta(\mathbf{o}_{k,j}, \mathbf{i}_{k,i-1}) & \text{if } k < i \end{cases},$$
$$\check{w}_{i,j}^k = \frac{\exp(\bar{b}_{i,j}^k)}{\sum_{k'>j,k'<i} \exp(\bar{b}_{i,j}^{k'})}, \mathbf{o}_{i,j} = \sum_{k>j,k<i} \check{w}_{i,j}^k \bar{\mathbf{o}}_{i,j}^k .$$

A.4 Span weights

An intuitive idea is that the larger the probability of a span's existence, the greater its weight. A span exists if its parent span exists and the span is an immediate child of its parent span. Therefore, we can recursively estimate the existence probability of each span top-down (Hu et al., 2023) and formalize the auto-encoding loss as follows:

$$p_{i,j} = \sum_{k<i} p_{k,j} \hat{w}_{k,j}^i + \sum_{k>j} p_{i,k} \hat{w}_{i,k}^j , \ p_{1,n} = 1 ,$$
$$\mathcal{L}_{ae} = -\frac{1}{\sum p_{i,j}} \sum_{\mathbf{x}_{i:j} \in \mathbb{V}} p_{i,j} \log \frac{\exp(\mathbf{o}_{i,j}^T \mathbf{E}_{\mathbb{V}[\mathbf{x}_{i:j}]})}{\sum_{k=1}^{|\mathbb{V}|} \exp(\mathbf{o}_{i,j}^T \mathbf{E}_k)} .$$

A.5 Experimental Setup and Hyperparameters

Our composition function uses 4 layers of Transformer layers. For span representations, we use 128-dimensional embeddings with 4 attention heads, 512-dimensional hidden layer representations, and a vocabulary size of 7835. This vocabulary is built from concatenating 1903 most frequent characters in the training set of wikitext-103 and a 10,000-entry BPE dictionary, excluding all characters. To guide the composition function, our lightweight parser is a 4-layer Transformer model that uses 64-dimensional embeddings with 4 attention heads and 128-dimensional hidden layer representations. For the causal language model, we use a 3-layer GPT2 equipped with 128-dimensional embeddings and 4 attention heads and follow the original configuration for the rest of the hyperparameters.

Our composition models are trained on 8 PPUs with a learning rate of 1e-2 for the light-weight parser and 5e-4 for the rest. The batch size is 8×128, and for each sample, we limit the context window to 512 characters (whitespace included). The total number of training steps is ten times the number of sentences in Wikitext-103.