Which Pieces Does Unigram Tokenization Really Need?

Anonymous ACL submission

Abstract

The Unigram tokenization algorithm offers a probabilistic alternative to the greedy heuristics of Byte-Pair Encoding. Despite its theoretical elegance, its implementation in practice is complex, limiting its adoption to the SentencePiece package and adapters thereof. We bridge this gap between theory and practice by providing a clear guide to implementation and parameter choices. We also identify a simpler algorithm that accepts slightly higher training loss in exchange for improved compression.

github.com/code/after/review

1 Introduction

The Unigram language model for tokenization (Kudo, 2018) was introduced as a principled, probabilistic alternative to the greedy heuristics of Byte-Pair Encoding (BPE; Sennrich et al., 2016). Despite its theoretical elegance, its implementation in practice is daunting: the original paper presents the training algorithm at a very high level of abstraction, online tutorials are often oversimplified to the point of being incorrect, and the only authoritative reference is a complex C++ implementation in SentencePiece (Kudo and Richardson, 2018). This has practical consequences: researchers often default to BPE or treat SentencePiece as a black box, limiting experimentation and exploration of tokenizer algorithms beyond small variations in BPE (Provilkov et al., 2020; Chizhov et al., 2024, inter alia).

This paper explains the reference Unigram training algorithm, suggests alternatives to key steps, and explores the effect of configurable and hard-coded parameters on tokenizer performance. We find that costly defaults such as multiple expectation-maximization steps have little effect on the output vocabulary. We also present a better heuristic for constructing seed vocabularies and a simplified pruning algorithm that accepts slightly higher loss while compressing almost as well as BPE.

2 The Unigram Model in Theory

The Unigram model[1] assumes that the probability $P(\mathbf{x})$ of a tokenized sequence $\mathbf{x} = (x_1, \ldots, x_k)$ is the product of the token probabilities $p(x_i)$:

$$P(\mathbf{x}) = \prod_{i=1}^{k} p(x_i). \tag{1}$$

When tokenizing text with a trained model, the Viterbi algorithm is typically used to determine the sequence of tokens which maximizes $P(\mathbf{x})$. During LLM training, this can be extended to sample lower-probability token sequences for regularization. In tokenizer training, both the vocabulary $\mathcal{V}$ and the token probabilities $p(x)$ for all $x \in \mathcal{V}$ are jointly optimized to maximize the marginal log-likelihood over a training corpus C. For a given text X, let $\mathcal{S}(X)$ be the set of all possible segmentations using the vocabulary. The objective function $\mathcal{L}$ to minimize is:

$$\mathcal{L} = \frac{-1}{\sum_{X \in C} |X|} \sum_{X \in C} \log \left(\sum_{\mathbf{x} \in \mathcal{S}(X)} P(\mathbf{x}) \right), \tag{2}$$

where $|X|$ is the length of X measured in atomic tokens (e.g. bytes). We normalize by the number of atomic tokens rather than the number of texts to ensure the metric remains independent of how the corpus is subdivided. For a more detailed mathematical treatment, see Meister (2025).

3 The Unigram Model in Practice

Since the vocabulary $\mathcal{V}$ is not known in advance, the training process must simultaneously discover a good vocabulary and estimate token probabilities.

[1] Sometimes referred to as *UnigramLM*. We use the shorter term as it is unambiguous in this context.

The algorithm described by Kudo (2018) is an iterative procedure that starts with a large number of seed tokens and gradually reduces them to the target vocabulary size. Both the original description and our summary of the reference implementation are listed in Algorithm 1.[2]

3.1 Seed Vocabulary Initialization

The SentencePiece implementation uses a combination of a suffix array and a longest prefix algorithm to extract frequent substrings. In short, the suffix array-based approach encodes a compact representation of the corpus' sorted suffixes, which then allows for efficient frequency counting with a stack-based emission algorithm. Algorithm 2 describes these steps in more detail.

Unlike many other tokenizers, SentencePiece does not pretokenize (i.e., split text into *pretokens*) before training. Instead, it operates on the entire corpus, rejecting candidate tokens during initialization based on validity criteria such as not containing internal whitespace. As a result, the stack-based emission algorithm in the reference implementation (cf. Algorithm 2, lines 12–18) may exclude certain frequent tokens from the set of seed tokens. This happens when the shared prefix is not a valid token, often due to trailing spaces. For example, given the corpus "the␣old␣man␣the␣boat", the substring "the␣" will be emitted, and then rejected due to its trailing space, while "the" is never emitted as it is not a maximal prefix. We propose and test a 'Maximal Valid Prefix Recovery' variation of the algorithm where the longest valid prefixes of rejected seed tokens are included (see Algorithm 3). Note that in both variants, a single corpus occurrence of a word can introduce new high-frequency tokens in the seed vocabulary, by making a prefix 'maximal' in at least one case.

3.2 The EM Algorithm

Given a fixed vocabulary, token probabilities $p(x)$ are estimated using the expectation-maximization algorithm, which alternates between expectation and maximization steps. In the **E**xpectation step, the forward-backward algorithm (Baum, 1972) is used to efficiently determine the expected count of each token, averaged over all segmentations of the corpus under the current probabilities $p(x_i)$. The **M**aximization step updates the token probabilities $p(x_i)$ based on these expected counts. In SentencePiece, at the start of the M-step, tokens with an expected count below $\tau_{\text{mp}} = 0.5$ are removed, which we refer to as the 'early prune step'. This hyperparameter is sensitive to the absolute corpus size, i.e. duplicating a corpus would change thresholding behavior despite identical token distributions.

3.3 Iterative Vocabulary Pruning

After a fixed number N_{em} of EM steps,[3] the main pruning step is performed. For each token, the pruning algorithm determines whether the token represents its own optimal tokenization and removes it otherwise. For the remaining tokens, the algorithm determines each token's second-best tokenization and calculates the effect on total log-likelihood if all occurrences of the token were replaced with this second-best tokenization. Tokens that are least critical to maintain high overall likelihood are discarded. During this step, the probabilities $p(x_i)$ are only used to inform the optimal tokenization paths.[4] The fraction of vocabulary preserved at each step is controlled by a hyperparameter α_{prune} (typically 0.75–0.9), ensuring that pruning happens gradually over several iterations.

In the reference implementation, the second-best tokenization is determined with a generic algorithm for determining the top k tokenizations. We note that a simple alternative is to apply the Viterbi algorithm over all paths over the token text except for the singleton.

3.4 Final Pruning

Once the iterative EM and pruning process reduces the vocabulary to at most α_{inter} times the target size (typically 1.1, i.e. 10% over), the SentencePiece implementation performs a final vocabulary selection step. At this stage, tokens with the lowest estimated probabilities $p(x_i)$ are pruned regardless of token interactions.

4 Evaluating Unigram Tokenizers

In this section, we evaluate the effect of various algorithm and parameter choices on Unigram tokenizer performance.

Tokenizers are generally difficult to evaluate, as intrinsic metrics such as compression rate do not

[2] We defer detailed expansions such as pseudocode algorithms to the appendix, but the descriptions in the main text should suffice for understanding the experiments.

[3] The SentencePiece implementation uses $N_{\text{em}} = 2$.

[4] This is a local heuristic: it cannot account for cases where, for example, `thou` is removed from the vocabulary and now a segmentation like `court|house` becomes optimal.

Method	Loss	#Tokens	Overlap
Mean over Monolingual 30 MB Corpora			
Pretokens (Ours)	1.325	5.43	—
Pretok., Recovery	$1.326^{\uparrow 0.02}$	$5.43^{\downarrow 0.01}$	90.2
Full-text (SP)	$1.333^{\uparrow 0.54}$	$5.47^{\uparrow 0.75}$	79.1
Full-text, Recov.	$1.326^{\uparrow 0.03}$	$5.43^{\downarrow 0.01}$	88.6
Mean over Monolingual 300 MB Corpora			
Pretokens (Ours)	1.337	54.02	—
Pretok., Recovery	$1.338^{\uparrow 0.02}$	$54.02^{\uparrow 0.00}$	91.6
Full-text (SP)	$1.338^{\uparrow 0.05}$	$54.05^{\uparrow 0.05}$	95.7
Full-text, Recov.	$1.338^{\uparrow 0.03}$	$54.02^{\downarrow 0.01}$	91.6

Table 1: Seed vocabulary initialization algorithms. Comparing pretoken-based (Ours) and full-text (SentencePiece-style) initialization, with and without Maximal Valid Prefix Recovery (Algorithm 3). Superscripts show % change vs baseline; 'Overlap' is vocabulary overlap with baseline. Pretoken-based initialization achieves better loss and compression on both corpus sizes. The recovery variation has minimal effect, suggesting that missing prefixes are recovered naturally in sufficiently large corpora.

Method	n	Train Tok.	FW Tok.	Morph.
Baseline	16K	58.7	65.0	0.620
	32K	54.0	59.6	0.611
	64K	50.6	55.7	0.644
FSP	16K	$58.2^{\downarrow 0.8}$	$64.1^{\downarrow 1.4}$	0.497
	32K	$53.7^{\downarrow 0.6}$	$58.9^{\downarrow 1.2}$	0.514
	64K	$50.3^{\downarrow 0.5}$	$55.1^{\downarrow 1.1}$	0.559
BPE	16K	$58.5^{\downarrow 0.4}$	$64.0^{\downarrow 1.5}$	0.221
	32K	$53.8^{\downarrow 0.3}$	$58.8^{\downarrow 1.3}$	0.186
	64K	$50.4^{\downarrow 0.4}$	$54.9^{\downarrow 1.4}$	0.165

Table 2: Compression (M tokens, lower is better, superscripts show % change vs baseline at same n) and morphological alignment across vocabulary sizes n. Baseline is Unigram with our default settings, while FSP uses Final-Style Pruning. *Train Tok.*: trained and evaluated on same 300 MB corpora. *FW Tok.*: trained on FineWiki 1 GB, evaluated on 300 MB corpora. Note that the gap with Baseline widens, while BPE gains a small edge over FSP. *Morph.*: MorphScore boundary recall on English (higher is better), where Unigram dominates.

always correlate well with downstream model performance (Schmidt et al., 2024), but are nevertheless important for inference speed. We evaluate Unigram tokenizer training algorithms based on (1) the training objective (marginal log-likelihood, Equation 2) and (2) compression as measured by corpus token count. For both, lower is better.

4.1 Experimental Setup

We test the effect of training hyperparameters using 300 MB monolingual corpora in English, German, Korean, Chinese, Arabic, and Hindi. We use SCRIPT encoding and pretokenization with character boundaries (Land and Arnett, 2025) and a trained vocabulary size of 32,768 for all experiments.

4.2 Seed Vocabulary Initialization

We implement a pretoken-based initialization algorithm using a pretoken-to-count mapping, following the common approach in efficient BPE implementations for storing corpora. This approach uses suffix arrays to track both text and indices, largely sidestepping the issue of incorrectly-rejected prefixes (§3.1) while improving efficiency. We also test SentencePiece-style initialization with the Maximal Valid Prefix Recovery variant described in Algorithm 3. Results in Table 1 show that both the pretoken-count approach and the recovery variation improve both compression and objective optimization over the SentencePiece-style approach. However, the recovery variation has only a minor effect, ranging from mildly beneficial to mildly detrimental. We hypothesize that in larger corpora, the few remaining important seed tokens that are not caught by the corpus-based approach are likely to appear at least once, mitigating the original issue, while adding too many similar seed tokens may crowd out other candidates. Therefore, in the rest of our experiments we use the more efficient pretoken-count approach without recovery.

The only other parameter in initialization is the seed vocabulary size, which we implement as a relative factor $n_{\text{seed}} = \beta_{\text{seed}} \cdot n$, where n is the target vocabulary size. While our default of $\beta_{\text{seed}} = 10$ is sufficient, performance drops significantly at lower values. This is particularly relevant given that SentencePiece's default value (10^6) corresponds to $\beta_{\text{seed}} \approx 4$ for large multilingual tokenizers with 250k tokens.

4.3 Iterative Vocabulary Pruning

The main pruning loop involves several hyperparameters, many of which have minimal effect. Varying the number of EM iterations within the 1–5 range shows no consistent differences. SentencePiece applies a digamma transformation to expected counts in the M-step (cf. Algorithm 1) which suppresses low-count tokens; omitting this transformation yields essentially no difference.

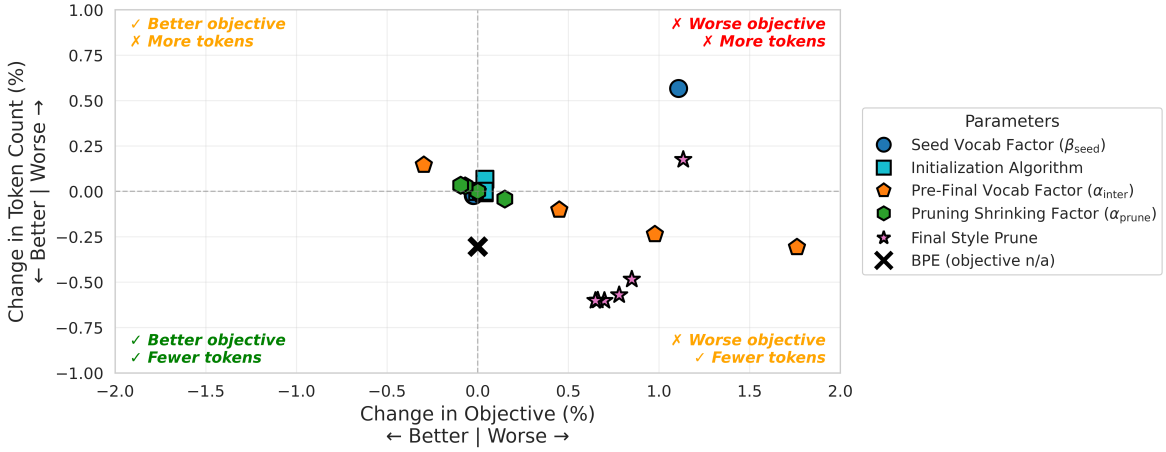

Figure 1: Effect of training hyperparameters. The origin is our baseline settings, as defined in Appendix C. Hyperparameters with minimal impact on both metrics omitted. The parameters with larger effects reveal a fundamental tradeoff: improvements in compression come at the cost of Unigram objective. Final-Style Pruning achieves the best compression. No configuration in our search significantly improves both metrics simultaneously.

Similarly, the early prune threshold τ_{mp} (cf. subsection 3.2) can be varied between 0 and 10 with no apparent effect.

Among the more impactful parameters, a slower shrinking factor α_{prune} yields better objective values but worse compression, though both effects are small. A similar tradeoff appears when halting pruning earlier via a larger α_{inter}. Motivated by these results, we investigate Final-Style Pruning (FSP), which replaces *all* pruning steps with the simpler probability-based algorithm. Figure 1 visualizes all parameter variations, revealing that (1) parameters that significantly affect performance exhibit a clear tradeoff with no configuration improving both metrics, and (2) FSP is the only variation which achieves compression exceeding BPE.

Table 2 validates these findings across vocabulary sizes (16K–64K) and training on FineWiki (Penedo, 2025, 1 GB subsets across the same 6 languages). The compression–objective tradeoff persists across all settings. Notably, when training on different datasets, BPE appears to generalize better. We also report MorphScore (Arnett et al., 2025, higher is better) for morphological alignment: Unigram methods substantially outperform BPE, consistent with Unigram's original motivation for subword regularization. Notably, FSP's compression gains come at the cost of reduced morphological alignment, positioning it between standard Unigram and BPE on this metric. Appendix D shows some examples of segmentations.

5 Conclusion

We provide a detailed guide to Unigram tokenization, clarifying implementation details that bridge the gap between the original high-level description and the SentencePiece reference implementation.

Our analysis reveals that Unigram is remarkably robust to hyperparameter choices. Several computationally expensive settings, including the number of EM iterations, digamma transformation, and early pruning thresholds, have minimal impact and can be simplified to improve training efficiency. We also demonstrate that pretoken-based initialization consistently outperforms the full-text approach in both objective and compression.

We identify a fundamental tradeoff between training objective and compression: no parameter configuration improves both metrics simultaneously. Our proposed Final-Style Pruning achieves compression closer to BPE by simplifying the pruning procedure, though at the cost of a marginally higher loss. Given the uncertain relationship between intrinsic tokenization metrics and downstream performance, this simpler alternative warrants consideration when compression is prioritized. BPE appears to achieve slightly better compression in out-of-domain settings, which warrants further investigation.

Overall, Unigram compression closely matches BPE, suggesting the choice between them may depend more on the need for probabilistic segmentation than on compression. We hope this work lowers the barrier to broader research and experimentation in Unigram tokenization.

Limitations

Our evaluation focuses on intrinsic metrics (training objective and compression) rather than downstream language model performance, which remains the ultimate measure of tokenizer effectiveness.

References

Catherine Arnett, Marisa Hudspeth, and Brendan O'Connor. 2025. Evaluating morphological alignment of tokenizers in 70 languages. *Preprint*, arXiv:2507.06378.

Leonard E Baum. 1972. An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, 3(1):1–8.

Pavel Chizhov, Catherine Arnett, Elizaveta Korotkova, and Ivan P. Yamshchikov. 2024. Bpe gets picky: Efficient vocabulary refinement during tokenizer training. *Preprint*, arXiv:2409.04599.

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.

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

Sander Land and Catherine Arnett. 2025. BPE stays on SCRIPT: Structured encoding for robust multilingual pretokenization. *Preprint*, arXiv:2505.24689.

Clara Meister. 2025. UnigramLM: An attempt at writing the missing manual. Blog post.

Guilherme Penedo. 2025. Finewiki. Source: Wikimedia Enterprise Snapshot API (https://api.enterprise.wikimedia.com/v2/snapshots). Text licensed under CC BY-SA 4.0 with attribution to Wikipedia contributors.

Ivan Provilkov, Dmitrii Emelianenko, and Elena Voita. 2020. Bpe-dropout: Simple and effective subword regularization. *Preprint*, arXiv:1910.13267.

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.

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.

A The Unigram Tokenizer Training Algorithm

Algorithm 1 High-Level Unigram Model Training Algorithms, with hardcoded and configurable parameters highlighted.

Inputs and Parameters:

C ▷ *Mandatory input: Training corpus*

n ▷ *Mandatory parameter: Target vocabulary size*

$n_{\text{seed}} \leftarrow 10^6$ ▷ *Initial seed vocabulary size*

$N_{\text{em}} \leftarrow 2$ ▷ *Number of EM sub-iterations per pruning iteration*

$\alpha_{\text{prune}} \leftarrow 0.75$ ▷ *Reduction factor for vocabulary size during pruning*

$\alpha_{\text{inter}} \leftarrow 1.1$ ▷ *Pre-final vocab size as multiple of n*

$\tau_{\text{mp}} \leftarrow 0.5$ ▷ *Minimum expected count threshold in M-step pruning*

Output: Final vocabulary $\mathcal{V}$ and probabilities $p(x_i)$.

Conceptual algorithm from Kudo (2018)

Initialize: Heuristically make a reasonably big seed vocabulary from the training corpus

while $|\mathcal{V}| > n$ **do** ▷ *Pruning Loop*

(a) Fixing the set of vocabulary, optimize $p(x)$ with the EM algorithm.

(b) Compute the loss for each subword x_i, which represents how likely the likelihood $\mathcal{L}$ is reduced when the subword x_i is removed from the current vocabulary.

(c) Sort the symbols by loss, and keep the top 80% of subwords.

SentencePiece Implementation

Initialize: $\mathcal{V} \leftarrow V_{\text{seed}}(C, n_{\text{seed}})$ using suffix arrays (Algorithm 2).

$n_{\text{inter}} \leftarrow \alpha_{\text{inter}} \cdot n$ ▷ *Set an intermediate target size slightly larger than final.*

while True **do** ▷ *Pruning loop*

for $i = 1$ **to** N_{em} **do** ▷ *EM sub-iterations*

E-step: Compute expected counts c_j for each token $x_j \in \mathcal{V}$ using forward-backward algorithm

Fast-prune step: Remove tokens x_j with expected count $c_j < \tau_{\text{mp}}$

M-step: Update token probabilities $p(x_j) \leftarrow F(c_j)/\sum_k F(c_k)$ ▷ *digamma function F*

if $|V| \leq n_{\text{inter}}$ **then break**

Pruning step:

$n_{\text{new}} \leftarrow \max(n_{\text{inter}}, \alpha_{\text{prune}} \cdot |\mathcal{V}|)$

For each token x_j, calculate the effect on $\mathcal{L}$ if it were replaced by its optimal sub-tokenization

Prune $\mathcal{V}$ to size n_{new} by removing tokens with the lowest effect on the objective $\mathcal{L}$ if removed

Finalize: Prune vocabulary V to size n by selecting those with highest $p(x_i)$

B Seed Vocabulary Creation Algorithms

The algorithm below uses a suffix array to find frequent substrings efficiently. A *suffix array* sorts all suffixes of the corpus lexicographically, placing suffixes with shared prefixes adjacent to each other. The *longest common prefix* (LCP) between adjacent suffixes tells us how many characters they share. The algorithm maintains a stack to track ranges of suffixes sharing a common prefix, known as "open intervals". When the LCP drops below a stack entry's height, that prefix's interval has closed, and we emit the pattern with its frequency.

Algorithm 2 SentencePiece Seed Vocabulary Creation

1: **Input:** Corpus C, maximum token length $\ell_{\max}$, Target seed vocabulary size n_{seed}
2: **Output:** Seed vocabulary V_{seed} with initial scores
3: **Initialization**
4: $C_{\text{concat}} \leftarrow$ Concatenation of C, separated by boundary markers.
5: $\text{SA} \leftarrow \text{SuffixArray}(C_{\text{concat}})$ ▷ *SA[r] is the index of rth suffix in lexicographical order*
6: $V_{\text{seed}} \leftarrow$ empty queue ▷ *Bounded priority queue, size n_{seed}, sorted by score*
7: stack $\leftarrow$ empty stack
8: **Stack-based prefix emission loop**
9: **for** $r_{\text{curr}} = 1$ to $|\text{SA}| + 1$ **do** ▷ *Process all adjacent suffix pairs in lexicographical order*
10: $\quad h \leftarrow \text{LCP}(\text{SA}[\text{r}_{\text{curr}} - 1], \text{SA}[\text{r}_{\text{curr}}])$ ▷ *Length of common prefix. $h = 0$ in last iteration*
11: $\quad r_{\text{lb}} \leftarrow r_{\text{curr}} - 1$
12: $\quad$ **while not** $\text{IsEmpty}(\text{stack})$ **and** $\text{Top}(\text{stack}).\text{height} > \text{h}$ **do**
13: $\quad\quad (\ell, r_{\text{start}}) \leftarrow \text{Pop}(\text{stack})$ ▷ *pattern length ℓ, interval start*
14: $\quad\quad$ freq $\leftarrow r_{\text{curr}} - r_{\text{start}}$ ▷ *Number of suffixes sharing prefix of length $\geq \ell$*
15: $\quad\quad p \leftarrow$ prefix of length ℓ of suffix at $\text{SA}[\text{r}_{\text{start}}]$
16: $\quad\quad$ **if** $1 < \ell \leq \ell_{\max}$ **and** freq ≥ 2 **and** p is a valid token **then**
17: $\quad\quad\quad$ Add $(p, \text{freq} \times \ell)$ to V_{seed} ▷ *Character coverage score: frequency $\times$ length*
18: $\quad\quad r_{\text{lb}} \leftarrow r_{\text{start}}$ ▷ *Propagate interval start for merging*
19: $\quad$ **if** IsEmpty(stack) **or** Top(stack).height $< h$ **then**
20: $\quad\quad$ Push (height $= h$, rank=r_{lb}) onto stack
21: **return** V_{seed}

Algorithm 3 Fallback variant: including longest valid prefix of rejected tokens when creating seed vocabulary. (replacing lines 13–17 in Algorithm 2)

1: $(\ell, r_{\text{start}}) \leftarrow \text{Pop}(\text{stack})$ ▷ *Pattern length ℓ, interval start rank*
2: freq $\leftarrow r_{\text{curr}} - r_{\text{start}}$
3: $\ell_s \leftarrow \text{Top}(\text{stack}).\text{height} + 1$ ▷ *if stack is empty, use $h + 1$*
4: **for** $\ell_i = \ell$ down to ℓ_s **do** ▷ ***Change:*** *Iterate over all intermediate lengths*
5: $\quad p \leftarrow$ prefix of length ℓ_i from suffix at $\text{SA}[\text{r}_{\text{start}}]$
6: $\quad$ **if** $1 < \ell_i \leq \ell_{\max}$ **and** freq ≥ 2 **and** p is a valid token **then**
7: $\quad\quad$ score $\leftarrow$ freq $\times \ell_i$
8: $\quad\quad$ Add (p, score) to V_{seed}
9: $\quad\quad$ **break** ℓ_i **loop** ▷ *Only include the longest valid token*

C Detailed results of Unigram hyperparameter variation

Table 3: Seed Vocabulary Algorithm and Final Style Prune Results: Relative Change (%) vs Baseline, lower is better. Each corpus shows loss (relative change), token count (relative change), and vocabulary overlap (absolute %, 100% = identical to baseline), with good and bad changes above 0.5% highlighted along with the largest absolute change per parameter.

(a) Seed Vocabulary Algorithm Comparison: Relative performance compared to pretoken-based without prefix recovery.

		30 MB			300 MB		
Corpus	Metric	Full text	Full text recovery	Pretokens recovery	Full text	Full text recovery	Pretokens recovery
English	*Loss*	+0.27	+0.04	-0.01	-0.00	+0.02	+0.01
	Tokens	+0.26	+0.01	-0.00	-0.01	+0.01	+0.01
	Vocabulary Overlap	82.3	89.6	93.9	96.5	91.5	91.5
German	*Loss*	+0.14	+0.17	+0.21	+0.01	+0.17	+0.19
	Tokens	+0.32	+0.02	+0.04	-0.01	-0.03	-0.02
	Vocabulary Overlap	89.4	80.3	79.9	98.6	82.3	81.3
Arabic	*Loss*	+0.39	+0.02	+0.00	+0.02	+0.00	-0.00
	Tokens	+1.24	-0.05	-0.09	+0.03	-0.01	+0.01
	Vocabulary Overlap	74.4	83.5	84.8	95.6	89.7	89.6
Hindi	*Loss*	+2.10	-0.02	-0.03	+0.09	-0.00	-0.00
	Tokens	+2.29	-0.01	-0.01	+0.23	-0.00	+0.00
	Vocabulary Overlap	57.9	88.1	89.0	89.5	90.9	91.2
Korean	*Loss*	+0.33	+0.01	+0.01	+0.02	+0.02	+0.01
	Tokens	+1.24	-0.06	-0.03	+0.09	-0.01	-0.00
	Vocabulary Overlap	82.1	93.0	95.8	96.1	96.4	97.0
Chinese	*Loss*	+0.39	+0.02	+0.01	+0.09	+0.01	+0.00
	Tokens	+0.29	+0.02	+0.01	+0.07	-0.00	+0.00
	Vocabulary Overlap	88.6	97.4	98.0	97.7	98.9	99.3
Mean	*Loss*	+0.60	+0.04	+0.03	+0.04	+0.04	+0.04
	Tokens	+0.94	-0.01	-0.02	+0.07	-0.01	+0.00
	Vocabulary Overlap	79.1	88.6	90.2	95.7	91.6	91.6

(b) Final Style Prune: Relative performance compared to default settings, using varying pruning shrinking factor α_{prune}. We use $\alpha_{\text{inter}} = 1$ for these experiments, disabling the final pruning step, as it would be identical to normal pruning steps.

Corpus	Metric	0.0	0.25	0.5	0.75	0.9	0.95
English	*Loss*	+0.72	+0.64	+0.59	+0.55	+0.53	+0.53
	Tokens	+0.34	-0.06	-0.21	-0.24	-0.25	-0.25
	Vocabulary Overlap	73.4	75.8	75.9	75.5	74.5	74.4
German	*Loss*	+1.41	+1.06	+0.96	+0.85	+0.79	+0.77
	Tokens	+0.19	-0.47	-0.60	-0.70	-0.72	-0.73
	Vocabulary Overlap	69.0	70.4	70.9	70.5	69.9	69.7
Arabic	*Loss*	+1.19	+0.82	+0.74	+0.57	+0.49	+0.47
	Tokens	+0.19	-0.97	-1.02	-0.93	-0.85	-0.84
	Vocabulary Overlap	65.1	71.3	72.9	74.7	74.9	74.8
Hindi	*Loss*	+0.66	+0.49	+0.47	+0.45	+0.44	+0.43
	Tokens	+0.53	-0.00	-0.11	-0.13	-0.13	-0.14
	Vocabulary Overlap	74.5	77.2	77.6	77.0	76.3	76.2
Korean	*Loss*	+1.77	+1.08	+0.95	+0.84	+0.79	+0.77
	Tokens	-0.10	-1.14	-1.23	-1.30	-1.31	-1.32
	Vocabulary Overlap	65.0	70.1	70.8	71.4	71.5	71.5
Chinese	*Loss*	+1.05	+1.00	+0.97	+0.94	+0.93	+0.93
	Tokens	-0.09	-0.25	-0.26	-0.31	-0.33	-0.33
	Vocabulary Overlap	74.6	74.2	74.4	74.1	74.0	73.8
Mean	*Loss*	+1.13	+0.85	+0.78	+0.70	+0.66	+0.65
	Tokens	+0.18	-0.48	-0.57	-0.60	-0.60	-0.60
	Vocabulary Overlap	70.3	73.2	73.8	73.9	73.5	73.4

Table 4: Detailed Unigram hyperparameter results. For each corpus, we report relative change (%) in Loss and Token Count (lower is better) and absolute Vocabulary Overlap (100% = identical to baseline). Significant deviations ($> 0.5\%$) are highlighted (good and bad), with the largest absolute change per metric marked. Results are shown for relative seed size (default $\beta_{\text{seed}} = 10$), pruning overshoot (default $\alpha_{\text{inter}} = 1.1$), M-step digamma transformation (default on), pruning threshold (default $\tau_{\text{mp}} = 0.5$), EM sub-iterations (default $N_{\text{em}} = 2$), and pruning shrinking factor (default $\alpha_{\text{prune}} = 0.75$).

Seed Vocabulary Parameters

		β_{seed} (10)				α_{inter} (1.1)			
Corpus	Metric	3	25	50	100	1.0	1.25	1.5	2.0
Arabic	*Loss*	+0.60	+0.03	+0.02	+0.02	-0.30	+0.44	+1.00	+2.00
	Tokens	+0.46	-0.17	-0.20	-0.20	+0.26	-0.21	-0.37	-0.50
	Vocabulary Overlap	67.1	86.7	87.3	87.3	88.0	89.0	83.0	78.2
German	*Loss*	+2.90	-0.09	-0.10	-0.11	-0.23	+0.45	+1.10	+2.09
	Tokens	+2.51	-0.06	-0.09	-0.09	+0.21	-0.14	-0.33	-0.35
	Vocabulary Overlap	45.7	87.1	86.4	86.6	87.1	88.2	78.8	73.6
English	*Loss*	+0.92	-0.02	-0.03	-0.03	-0.21	+0.32	+0.71	+1.21
	Tokens	+0.59	-0.01	-0.02	-0.02	+0.07	-0.04	-0.10	-0.11
	Vocabulary Overlap	62.1	92.2	92.4	92.4	87.4	89.3	83.0	79.2
Hindi	*Loss*	+0.79	-0.01	-0.01	-0.01	-0.20	+0.31	+0.63	+1.07
	Tokens	+0.57	-0.01	-0.01	-0.01	+0.05	-0.00	-0.05	-0.07
	Vocabulary Overlap	65.2	91.4	91.4	91.4	87.5	89.9	84.4	80.9
Korean	*Loss*	+0.63	-0.01	+0.01	+0.03	-0.38	+0.52	+1.14	+2.14
	Tokens	-0.59	+0.09	+0.14	+0.14	+0.26	-0.31	-0.60	-0.74
	Vocabulary Overlap	77.1	91.8	91.5	91.8	87.0	87.6	79.2	74.0
Chinese	*Loss*	+0.80	-0.04	-0.04	-0.03	-0.47	+0.66	+1.29	+2.06
	Tokens	-0.13	+0.03	+0.04	+0.04	+0.02	+0.10	+0.03	-0.08
	Vocabulary Overlap	75.7	96.4	96.5	96.4	87.0	88.0	81.0	76.7
Mean	*Loss*	+1.11	-0.02	-0.03	-0.02	-0.30	+0.45	+0.98	+1.76
	Tokens	+0.57	-0.02	-0.02	-0.02	+0.15	-0.10	-0.24	-0.31
	Vocabulary Overlap	65.5	90.9	90.9	91.0	87.4	88.7	81.6	77.1

Pruning Loop Parameters

Per-language results for pruning loop parameters show similarly negligible effects across all corpora, with no single variation exceeding 0.5% change in loss or token count. The mean results below summarize these findings:

		digamma (on)	τ_{mp} (0.5)			N_{em} (2)			α_{prune} (0.75)		
Corpus	Metric	Off	0.0	2.0	10.0	1	3	5	0.5	0.9	0.95
Mean	*Loss*	-0.01	+0.00	+0.02	-0.00	+0.02	-0.00	-0.00	+0.15	-0.07	-0.09
	Tokens	-0.00	+0.00	+0.00	-0.03	+0.02	-0.00	+0.00	-0.04	+0.03	+0.03
	Vocabulary Overlap	98.0	94.0	93.8	93.7	93.3	95.7	94.1	88.8	93.2	92.4

D MorphScore Evaluation

Table 5 shows MorphScore (version 2, Arnett et al., 2025) evaluation results comparing our Baseline unigram tokenizer, the FSP (Final-Style Prune) variant, and BPE on English morphological segmentation.

Word	Freq	Gold	Baseline	FSP	BPE
All Methods Match Gold (258 words covering 9.5% of corpus)					
been	2.2%	be | en	be | en	be | en	be | en
months	0.4%	month | s	month | s	month | s	month | s
following	0.3%	follow | ing	follow | ing	follow | ing	follow | ing
Only Baseline Matches (443 words covering 6.8% of corpus)					
attached	0.4%	attach | ed	attach | ed	atta | ched	att | ached
Minutes	0.2%	Minute | s	Min | ute | s	Min | utes	Min | utes
animals	0.2%	animal | s	animal | s	anim | als	anim | als
Only FSP Matches (111 words covering 1.3% of corpus)					
Palestinians	0.1%	Palestinian | s	P | ale | stin | ians	Pal | est | inian | s	Pal | est | in | ians
instructions	0.1%	instruction | s	in | struct | ions	in | struction | s	inst | ru | ctions
quoted	0.1%	quote | d	quot | ed	quote | d	qu | oted
Only BPE Matches (68 words covering 0.8% of corpus)					
intended	0.1%	intend | ed	int | ended	int | ended	intend | ed
bigger	0.1%	big | ger	b | igger	b | igger	big | ger
seemed	0.1%	seem | ed	see | med	see | med	se | em | ed
No Method Matches (1000 words covering 12.6% of corpus)					
comments	0.2%	comment | s	com | ments	com | ments	com | ments
weapons	0.2%	weapon | s	we | ap | ons	we | ap | ons	we | ap | ons
decided	0.1%	decide | d	de | cid | ed	de | cid | ed	dec | ided

Table 5: MorphScore examples for English, grouped by which methods match gold morpheme boundaries. Each category shows the three most frequent words meeting that criterion. Freq is the word's share of total evaluation frequency.