Objective vs. Search: Decomposing What Makes a Good Tokeniser

Anonymous ACL submission

Abstract

The two dominant tokenisation algorithms used in modern language models, byte-pair encoding (BPE) and UnigramLM, differ along two orthogonal axes: their *optimisation objective* (compression vs. log-likelihood) and their *search procedure* (bottom-up merging vs. top-down pruning). Existing comparisons confound these axes, making it unclear whether their observed differences stem from *what* is being optimised or from *how*. We disentangle the two by introducing two new tokenisers that complete this 2×2 design space: BottomUpLL, a bottom-up likelihood-based tokeniser, and TopDownComp, a top-down compression-based tokeniser. We train language models with each tokeniser varying: model size (from 100M to 1B parameters), vocabulary sizes (8k, 32k, and 128k), and domain (English-only vs. multilingual). Evaluating these models on bits-per-byte and BLiMP, we find that the search procedure—not the objective—is the dominant factor in their performance: bottom-up tokenisers consistently achieve lower bits-per-byte, while the choice of objective matters mainly in the small-vocabulary regime. This finding holds across model sizes and languages. Overall, our results help clarify the functional role of tokeniser design choices, offering concrete guidance for their more principled construction.

1 Introduction

Before a language model (LM) processes text, its raw **character string** c is mapped to a **token string** s, which defines the model's input. This mapping is performed by a **tokeniser**, a foundational component of modern language modelling pipelines used in virtually all state-of-the-art models (Qwen Team, 2026; OpenAI et al., 2025; Kimi Team et al., 2026). Two methods currently dominate the tokenisation landscape: (i) byte-pair encoding (BPE), a bottom-up compression-based method (Gage, 1994; Sennrich et al., 2016); and (ii) UnigramLM, a top-down likelihood-based method (Kudo, 2018).

		search procedure	
		bottom-up	top-down
objective	compression	BPE	TopDownComp
	log-likelihood	BottomUpLL	UnigramLM

Figure 1: Design choice factorisation for tokenisation.

Notably, several prior work investigates the properties which make up a good tokeniser (e.g., Gallé, 2019; Zouhar et al., 2023). Among these, Schmidt et al. (2024) compares language models trained with BPE and UnigramLM, finding that BPE often performs better empirically. One possible interpretation of these results is that a well-known property associated with BPE—namely, its compression optimisation objective—may be advantageous in practice. However, optimisation objectives alone may not fully explain tokeniser effectiveness.

The optimisation problem that tokenisation algorithms must solve is NP-hard (Kozma and Voderholzer, 2024; Whittington et al., 2025; Kastreva et al., 2026), and heuristic search procedures are therefore needed to approximate it. Given a dataset, BPE takes a **bottom-up** approach, greedily merging symbols to maximise compression; UnigramLM takes a **top-down** approach, starting from a large vocabulary and iteratively pruning tokens to maximise log-likelihood. Directly comparing these methods thus confounds optimisation objective and search procedure, leaving unclear which design choice explains the observed difference.

In this work, we disentangle these two axes. We introduce BottomUpLL, a bottom-up likelihood-based tokeniser that mirrors BPE's merge procedure but optimises corpus log-likelihood instead of compression. We also introduce TopDownComp, a top-down compression-based tokeniser that follows UnigramLM's pruning procedure but optimises

compression. Together with BPE and UnigramLM, these methods form a 2×2 study crossing objective (compression vs. log-likelihood) and search procedure (bottom-up vs. top-down).

We train language models with tokenisers produced by all four algorithms, evaluating them in English-only and multilingual settings. We analyse these models with a combination of extrinsic (bits-per-byte, BPB, and accuracy in a grammaticality task, BLIMP) and intrinsic evaluations (including compression, log-likelihood, vocabulary usage, vocabulary overlap, etc.). Notably, our results show that search procedure has a stronger influence in language modelling performance than objective, with bottom-up tokenisers consistently achieving the best BPB scores across experimental conditions (BPE outperforms TopDownComp, and BottomUpLL outperforms UnigramLM). The objective still matters, though, even if only more narrowly, e.g., likelihood-based tokenisers outperform compression-based ones in small-vocabulary settings, and induce vocabularies with lower entropy. Overall, our results highlight the importance of both optimisation objective and search procedure to tokeniser design.

2 Tokenisation

Language modelling starts from raw text, which we will represent as **character-strings** $\mathbf{c} \in \Sigma^*$; these are finite sequences of characters $\mathbf{c} = c_1 c_2 \ldots c_{|\mathbf{c}|}$ over alphabet Σ.[1] A tokeniser's job is then to *segment* these character-strings into **token-strings** $\mathbf{s} \in \mathcal{S}^*$: sequences $\mathbf{s} = \langle s_1, s_2, \ldots, s_{|\mathbf{s}|} \rangle$, where each symbol s_t is called a token and represents a non-empty character span. Whenever $\mathbf{s}$ is a segmentation of $\mathbf{c}$, we say these strings are equivalent, which we denote as $\mathbf{c} \overset{\circ}{=} \mathbf{s}$, i.e.:[2]

$$\mathbf{c} \overset{\circ}{=} \mathbf{s} \iff \mathbf{c} = s_1 \circ s_2 \circ \cdots \circ s_{|\mathbf{s}|}, \qquad (1)$$

Notably, tokenisers are usually not allowed to output any possible segmentation of $\mathbf{c}$, being instead constrained to segment it using only the tokens in a finite set called its **vocabulary** $\mathcal{S} \subset \Sigma^+$. To ensure that any character-string can be represented as tokens, we enforce $\Sigma \subseteq \mathcal{S}$.

How do we convert between character- and token-strings, though? Given just a vocabulary $\mathcal{S}$, there are multiple different ways a character-string could be mapped to items in that vocabulary. For example, if $\mathcal{S} = \{a, aa, aaa\}$, then the character-string $\mathbf{c} = aaa$ may be encoded as either $\langle a, aa \rangle$, $\langle aa, a \rangle$, or $\langle a, a, a \rangle$. Consequently, a tokeniser is defined not only by its vocabulary, but also by the segmentations used. Choosing which segmentation to use is the job of an **encoding function**, $\texttt{tok} : \Sigma^* \to \mathcal{S}^*$, which segments the original character-string into tokens; by definition, we have thus that $\mathbf{c} \overset{\circ}{=} \texttt{tok}(\mathbf{c})$. Finally, a tokeniser also contains a **decoding function**, $\texttt{detok} : \mathcal{S}^* \to \Sigma^*$, which converts token-strings back into characters, being typically defined as: $\texttt{detok}(\mathbf{s}) \overset{\text{def}}{=} s_1 \circ s_2 \circ \cdots \circ s_{|\mathbf{s}|}$. Formally, we thus define a tokeniser as the tuple $\mathbb{T} \overset{\text{def}}{=} \langle \mathcal{S}, \texttt{detok}, \texttt{tok} \rangle$.

3 Learning a Tokeniser: Optimisation Objective vs. Search Procedure

How do we learn the parameters of a tokeniser? Here, we cast the learning of a tokeniser as the combination of two design components: an objective function and a search procedure.

3.1 Objective Functions

Given a tokeniser $\mathbb{T}$ and a dataset $\mathcal{D} = \{\mathbf{c}_m\}_{m=1}^{M}$, an objective function $\mathfrak{G}$ assigns a score to the tokeniser. Different objectives therefore encode different notions of what makes a tokeniser useful. In this work, we focus on two standard objectives: compression and log-likelihood. The **compression** objective minimises the total length of the token strings produced by the tokeniser:

$$\mathfrak{G}_{\text{comp}}(\texttt{tok}, \mathcal{D}) \overset{\text{def}}{=} \sum_{\mathbf{c} \in \mathcal{D}} |\texttt{tok}(\mathbf{c})|, \qquad (2)$$

which is appealing because shorter token strings improve effective context usage and reduce the number of model steps needed to process a dataset. The **log-likelihood** objective instead treats tokenisation as part of a probabilistic model of text and minimises the corpus negative log-likelihood:

$$\mathfrak{G}_{\text{ll}}(\texttt{tok}, \mathcal{D}) \overset{\text{def}}{=} -\sum_{\mathbf{c} \in \mathcal{D}} \log p_{\boldsymbol{\theta}}\big(\texttt{tok}(\mathbf{c})\big). \qquad (3)$$

This objective is likewise appealing, since log-likelihood is also the training objective of the language models for which these tokenisers are designed. We follow UnigramLM in using a unigram language model as $p_{\boldsymbol{\theta}}$. For a fixed encoding function tok, let $n_s^{\mathcal{D}}$ denote the count of token s in the

[1] As usual, Σ^* denotes the Kleene star of Σ, while Σ^+ denotes the set of all non-empty strings over Σ.

[2] We restrict our discussion to *lossless* tokenisers here.

tokenised corpus, and let $N^{\mathcal{D}} \stackrel{\text{def}}{=} \sum_{s \in \mathcal{S}} n_s^{\mathcal{D}}$ be the total number of tokens. We then define $p_{\boldsymbol{\theta}}$ as

$$p_{\boldsymbol{\theta}}(\mathbf{s}) = \prod_{t=1}^{|\mathbf{s}|} p_{\boldsymbol{\theta}}(s_t), \qquad p_{\boldsymbol{\theta}}(s) = \frac{n_s^{\mathcal{D}}}{N^{\mathcal{D}}}. \tag{4}$$

3.2 Search Procedures

Given one of these objectives and a fixed vocabulary budget K, we might expect a tokeniser to be chosen by directly optimising

$$\texttt{tok}_{\texttt{opt}} = \operatorname*{argmin}_{\texttt{tok} \in \mathcal{T}_K} \mathfrak{G}(\texttt{tok}, \mathcal{D}), \tag{5}$$

where $\mathcal{T}_K$ denotes whichever class of tokenisers is under consideration. Unfortunately, this optimisation problem is computationally intractable; the compression variant is provably NP-hard (Kozma and Voderholzer, 2024; Whittington et al., 2025; Kastreva et al., 2026), and we suspect the log-likelihood variant to be as well. In practice, heuristic **search procedures** are therefore required. Such procedures specify both a restricted class of tokenisers and a strategy for selecting an element from that class. We focus on two standard search procedures: bottom-up construction and top-down pruning.

The **bottom-up** procedure builds a tokeniser from the bottom up: the vocabulary starts as the alphabet Σ and grows one token at a time through *merge operations*. Given a merge $m = \langle s', s'' \rangle$, applying it to a token-string rewrites every (non-overlapping) occurrence of bigram s', s'' into the token $s^{\text{new}} = s' \circ s''$. A merge list $\mathbf{m} = m_1, \ldots, m_k$ then defines an encoding function $\texttt{tok}_{\uparrow}[\mathbf{m}]$: starting from $\mathbf{c}$ as a token-string over Σ (one token per character), it applies $m_1, \ldots, m_k$ in order. The final token-string is our tokenised text. We write the tokeniser class defined by this procedure as $\mathcal{T}_K^{\uparrow} = \left\{\texttt{tok}_{\uparrow}[\mathbf{m}] \mid \mathbf{m} \in (\Sigma^+ \times \Sigma^+)^K\right\}$. To select an element from this class, we then start from an empty merge list, and greedily add one merge per step to it, locally optimising an objective $\mathfrak{G}$:

$$m^{\star} = \operatorname*{argmin}_{m \in \Sigma^+ \times \Sigma^+} \mathfrak{G}(\texttt{tok}_{\uparrow}[\mathbf{m}_{<k}, m], \mathcal{D}), \tag{6}$$

where $\texttt{tok}_{\uparrow}[\mathbf{m}_{<k}, m]$ denotes the encoding function obtained by applying the current merge list $\mathbf{m}_{<k}$ followed by the candidate m. The selected merge $m^{\star}$ is then appended to the merge list.

By contrast, the **top-down** procedure starts from a large candidate vocabulary $\mathcal{S}_0$, with $|\mathcal{S}_0| \gg K$, and removes tokens until $|\mathcal{S}| = |\Sigma| + K$. Notably, this procedure does not rely on merge lists, but instead commits to an *objective-optimal* encoding function $\texttt{tok}_{\Rightarrow}[\mathcal{S}]$, which segments each $\mathbf{c}$ optimally under the given objective; this encoding function thus outputs: the shortest valid segmentation for compression, or the most probable one under the current $p_{\boldsymbol{\theta}}$ for log-likelihood.[3] Top-down's tokeniser class can then be written as: $\mathcal{T}_K^{\downarrow} = \left\{\texttt{tok}_{\Rightarrow}[\mathcal{S}] \mid \mathcal{S} \subset \Sigma^+, \Sigma \subseteq \mathcal{S}, |\mathcal{S}| = |\Sigma| + K\right\}$. To select an item in this class, we then start from a large vocabulary $\mathcal{S}_0$, from which we iteratively prune the least harmful tokens:

$$s_{\boldsymbol{\times}} = \operatorname*{argmin}_{s \in \mathcal{S}_{k-1} \setminus \Sigma} \mathfrak{G}(\texttt{tok}_{\Rightarrow}[\mathcal{S}_{k-1} \setminus \{s\}], \mathcal{D}). \tag{7}$$

where we index pruning steps by k, mirroring the bottom-up case. This yields $\mathcal{S}_k = \mathcal{S}_{k-1} \setminus \{s_{\boldsymbol{\times}}\}$. Importantly, to ensure every string remains encodable, alphabet tokens are never removed; pruning then continues until $|\mathcal{S}| = |\Sigma| + K$.[4]

4 Analysed Tokenisers: Old and New

Combining the objectives and search procedures above, we get a 2×2 design space for tokenisation algorithms (illustrated in Fig. 1). Only two of these cells is currently occupied, though (by BPE and UnigramLM). We thus introduce the two missing cells, which we call: BottomUpLL, a bottom-up likelihood-based method, and TopDownComp, a top-down compression-based method.

A loose end remains, however, before any of these methods can be run. §3 described the search procedures for $\mathbb{T}$ as greedy: at each step, the procedure picks the merge $m^{\star}$ or deletion $s_{\boldsymbol{\times}}$ which (locally) optimises objective $\mathfrak{G}(\cdot, \mathcal{D})$. It did not specify, however, how this selection is performed. Taken literally, computing the argmin operations in Eq. (6) and Eq. (7) would require running the objective function $\mathfrak{G}$ once per candidate to find the optimal one—a computationally impractical operation. Rather than recomputing $\mathfrak{G}$ every time, most tokenisation algorithms work with *incremental scores* instead:

$$m^{\star} = \operatorname*{argmax}_{\langle s^1, s^2 \rangle \in \Sigma^+ \times \Sigma^+} \Delta_{s^1, s^2}^{\mathfrak{G}(\cdot, \mathcal{D})} \tag{8a}$$

$$s_{\boldsymbol{\times}} = \operatorname*{argmin}_{s \in \mathcal{S}_{k-1} \setminus \Sigma} \Delta_{s}^{\mathfrak{G}(\cdot, \mathcal{D})} \tag{8b}$$

where $\Delta_{s^1, s^2}^{\mathfrak{G}(\cdot, \mathcal{D})}$ and $\Delta_{s}^{\mathfrak{G}(\cdot, \mathcal{D})}$ represent, respectively, a **merge's gain** and a **deletion's cost**, captur-

[3] Both can be computed efficiently.

[4] We note that, for efficiency reasons, in practice, multiple tokens are often pruned in batches.

ing a candidate's marginal effect on the objective. Notably, these equations are equivalent to §3.2's, as the pre-update objective values (e.g., $\mathfrak{G}(\texttt{tok}_{\uparrow}[\mathbf{m}_{<k}], \mathcal{D})$) do not depend on the candidates, and are thus constant. Each search procedure then derives a way to compute (or approximate) these values efficiently, as we show next.

4.1 Bottom-up tokenisers: BPE and BottomUpLL

For a bottom-up method, each candidate is scored according to its *merge gain* $\Delta^{\mathfrak{G}(\cdot,\mathcal{D})}_{s^1,s^2}$, the improvement in the objective from merging the pair $\langle s^1, s^2\rangle$. For compression, this improvement is the reduction in token count resulting from the merge; for log-likelihood, it is the corresponding gain in corpus log-likelihood. We now show how this score can be computed efficiently for both objectives.

Lemma 1. *The change in compression due to merging a token-pair can be computed as:*

$$\Delta^{\mathfrak{G}_{\text{comp}}(\cdot,\mathcal{D})}_{s^1,s^2} = n^{\mathcal{D}}_{s^1,s^2} \tag{9}$$

where $n^{\mathcal{D}}_{s^1,s^2}$ denotes the number of non-overlapping $\langle s^1, s^2\rangle$ sequences in $\mathcal{D}$.

Proof sketch. Full proof in §A. Replacing a token-pair $m = \langle s^1, s^2\rangle$ with merged token $s^1 \circ s^2$ saves 1 token. Further, the number of non-overlapping occurrences of a token-pair is its number of possible merges. A token-pair's $n^{\mathcal{D}}_{s^1,s^2}$ will thus be its resulting compression. □

Lemma 1 gives the standard BPE merge rule: the compression-optimal merge is the most (non-overlappingly) frequent token-pair. In practice, the token-pair counts $n^{\mathcal{D}}_{s^1,s^2}$ are computed once and then updated incrementally after each merge, rather than recomputed from scratch (see §B), which allows us to run BPE efficiently.

We can derive an analogous equivalence for BottomUpLL. The difference is that a merge is scored by its effect on the empirical unigram log-likelihood rather than by its immediate compression gain. When a candidate pair $\langle s^1, s^2\rangle$ is merged, each non-overlapping occurrence removes one occurrence of s^1 and one occurrence of s^2, creating one occurrence of the new token $s^1 \circ s^2$, and reducing the total token count $N^{\mathcal{D}}$ by one. Since the unigram log-likelihood depends only on token counts and the total number of tokens, we can derive a closed-form expression for the objective change.

Lemma 2. *The change in log-likelihood due to merging a token-pair can be computed as:*[5]

$$\begin{aligned}\Delta^{\mathfrak{G}_{\text{ll}}(\cdot,\mathcal{D})}_{s^1,s^2} = \quad & (10)\\ & (n^{\mathcal{D}}_{s^1} - n^{\mathcal{D}}_{s^1,s^2}) \log(n^{\mathcal{D}}_{s^1} - n^{\mathcal{D}}_{s^1,s^2}) - n^{\mathcal{D}}_{s^1} \log n^{\mathcal{D}}_{s^1}\\ & + (n^{\mathcal{D}}_{s^2} - n^{\mathcal{D}}_{s^1,s^2}) \log(n^{\mathcal{D}}_{s^2} - n^{\mathcal{D}}_{s^1,s^2}) - n^{\mathcal{D}}_{s^2} \log n^{\mathcal{D}}_{s^2}\\ & + n^{\mathcal{D}}_{s^1,s^2} \log n^{\mathcal{D}}_{s^1,s^2}\\ & - (N^{\mathcal{D}} - n^{\mathcal{D}}_{s^1,s^2}) \log(N^{\mathcal{D}} - n^{\mathcal{D}}_{s^1,s^2}) + N^{\mathcal{D}} \log N^{\mathcal{D}}.\end{aligned}$$

Proof sketch. Full proof in §C. Note that, if a token changes from count n_1 to count n_2, its contribution to the log-likelihood objective changes by: $n_2 \log n_2 - n_1 \log n_1$. The lemma then follows trivially, noting that: (i) the old tokens s^1 and s^2 change from, e.g., frequency $n^{\mathcal{D}}_{s^1}$ to $(n^{\mathcal{D}}_{s^1} - n^{\mathcal{D}}_{s^1,s^2})$; (ii) the new token $s^1 \circ s^2$ changes from frequency 0 to $n^{\mathcal{D}}_{s^1,s^2}$. Further, the total count of tokens serves as a normalisation factor in all tokens, and thus enters the log-likelihood with the opposite sign; its contribution changes from $N^{\mathcal{D}}$ to $N^{\mathcal{D}} - n^{\mathcal{D}}_{s^1,s^2}$. □

BottomUpLL can thus also be implemented efficiently, maintaining local count statistics and updating only affected candidate scores after each merge. Interestingly, an analysis of Lemma 2 shows that, for a token-pair to be selected: (i) the pair should occur often enough to matter, but (ii) it should also be favoured when its tokens co-occur more systematically than expected from their individual frequencies. To make this idea explicit, let the pointwise mutual information between two tokens be:

$$\text{PMI}_{\text{adj}}(s^1, s^2) \stackrel{\text{def}}{=} \log \frac{p_{\boldsymbol{\theta}}(s^1, s^2)}{p_{\boldsymbol{\theta}}(s^1)\, p_{\boldsymbol{\theta}}(s^2)} \tag{11a}$$

$$= \log \frac{N^{\mathcal{D}}\, n^{\mathcal{D}}_{s^1,s^2}}{n^{\mathcal{D}}_{s^1} n^{\mathcal{D}}_{s^2}}. \tag{11b}$$

This value approximates Lemma 2.

Lemma 3. *Under a first-order Taylor approximation, the change in log-likelihood due to merging a token-pair can be approximated as:*

$$\Delta^{\mathfrak{G}_{\text{ll}}(\cdot,\mathcal{D})}_{s^1,s^2} \approx n^{\mathcal{D}}_{s^1,s^2} \left(\text{PMI}_{\text{adj}}(s^1, s^2) - 1\right). \tag{12}$$

Proof sketch. Full proof in §D. Starting from Lemma 2, apply a first-order Taylor expansion to each $x \log x$ term, using $n^{\mathcal{D}}_{s^1,s^2}$ as the perturbation. Collecting the resulting terms gives a pair-count-weighted adjacent PMI term, together with the linear penalty in Eq. (12). □

[5] For the case $s^1 = s^2$, the same calculation applies albeit with a modified count update: each merge consumes two occurrences of the same token s^1 and thus its count is reduced by $2n^{\mathcal{D}}_{s^1,s^1}$ instead; other parts of the update remain unchanged.

Interestingly, Eq. (12) is closely related to the objective used by another popular bottom-up tokeniser: WordPiece (Schuster and Nakajima, 2012), which greedily merges tokens based on their $\mathrm{PMI}_{\mathrm{adj}}(s^1, s^2)$. Both methods thus favour merging tokens whose joint occurrence is large relative to what would be expected from their frequencies; `BottomUpLL`, however, scales this value by $n^{\mathcal{D}}_{s^1,s^2}$.

4.2 Top-down tokenisers: `UnigramLM` and `TopDownComp`

For top-down methods, the candidate score is the *deletion cost* $\Delta_s^{\mathfrak{G}(\cdot,\mathcal{D})}$: the increase in the objective from removing s from the current vocabulary. Unfortunately, the cost $\Delta_s^{\mathfrak{G}(\cdot,\mathcal{D})}$ of a top-down deletion is harder to compute than in the bottom-up case. This is due to the nature of the encoding function $\texttt{tok}_{\Rightarrow}[\mathcal{S}]$ used by top-down algorithms. Unlike in the bottom-up case, deleting a token here can change the preferred segmentation of an entire character-string, not only the local decomposition of the deleted token.[6] Further, for the log-probability objective, the preferred segmentation under $\texttt{tok}_{\Rightarrow}[\mathcal{S}]$ depends on $\log p_{\boldsymbol{\theta}}$; an update to the token counts following a token-deletion, though, may alter $p_{\boldsymbol{\theta}}$ enough to have downstream effects in how this encoding function segments this and other strings (which may again impact $p_{\boldsymbol{\theta}}$).

To approximate $\Delta_s^{\mathfrak{G}(\cdot,\mathcal{D})}$, we thus rely on a **local replacement approximation**: when scoring the deletion of s, we keep the rest of the current model fixed and estimate the loss by only directly replacing occurrences of s with an alternative segmentation under $\mathcal{S} \setminus \{s\}$. For log-likelihood, this then yields the following result.

Lemma 4. *Under a local replacement approximation, the estimated increase in the negative log-likelihood objective due to deleting a token $s \in \mathcal{S} \setminus \Sigma$ can be approximated as:*[7]

$$\Delta_s^{\mathfrak{G}_{\mathrm{ll}}(\cdot,\mathcal{D})} \approx n_s^{\mathcal{D}} \left(\log p_{\boldsymbol{\theta}}(s) - \log p_{\boldsymbol{\theta}}(\mathbf{s}_s^{\mathrm{rep}})\right), \quad (13)$$

where $p_{\boldsymbol{\theta}}$ is the unigram model at step $k-1$, and $\mathbf{s}_s^{\mathrm{rep}}$ is s's log-probability–optimal replacement segmentation, i.e.,: $\mathbf{s}_s^{\mathrm{rep}} \stackrel{\text{def}}{=} \texttt{tok}_{\Rightarrow}[\mathcal{S}_{k-1} \setminus \{s\}](s)$.

[6] For example, under a compression objective, suppose $\mathcal{S} = \Sigma \cup \{abc, def, cdef\}$. Then, string $abcdef$ is segmented as $\langle abc, def\rangle$. If abc is deleted, however, the shortest segmentation under the reduced vocabulary becomes $\langle a, b, cdef\rangle$, which also changes the previously used token def.

[7] In practice, when computing this update, `UnigramLM` keeps segmentations latent and leverages $p_{\boldsymbol{\theta}}$ to compute $n_s^{\mathcal{D}}$ as the *expected* count of s, instead of its exact count under $\texttt{tok}_{\Rightarrow}$.

Proof sketch. Full proof in §E. Before deletion, each expected occurrence of s contributes $-\log p_{\boldsymbol{\theta}}(s)$ to the negative log-likelihood. Under the local replacement approximation, deleting s replaces it with $\mathbf{s}_s^{\mathrm{rep}}$, which contributes $-\log p_{\boldsymbol{\theta}}(\mathbf{s}_s^{\mathrm{rep}})$. As the number of such occurrences is $n_s^{\mathcal{D}}$, we get Eq. (13). □

After each deletion, $p_{\boldsymbol{\theta}}$ and $\texttt{tok}_{\Rightarrow}$ are then fully updated. First, we update $p_{\boldsymbol{\theta}}$, maximising the log-likelihood of $\mathcal{D}$ under it via an expectation–maximisation procedure.[8] Second, given an updated $p_{\boldsymbol{\theta}}$, the segmentations $\texttt{tok}_{\Rightarrow}$ can be trivially recomputed. We now present an analogous update for compression.

Lemma 5. *Under a local replacement approximation, the estimated change in compression due to deleting a token $s \in \mathcal{S} \setminus \Sigma$ can be approximated as:*

$$\Delta_s^{\mathfrak{G}_{\mathrm{comp}}(\cdot,\mathcal{D})} = n_s^{\mathcal{D}} \left(|\mathbf{s}_s^{\mathrm{rep}}| - 1\right), \quad (14)$$

where $\mathbf{s}_s^{\mathrm{rep}}$ is the compression-optimal replacement of s after deletion: $\mathbf{s}_s^{\mathrm{rep}} \stackrel{\text{def}}{=} \texttt{tok}_{\Rightarrow}[\mathcal{S}_{k-1} \setminus \{s\}](s)$.

Proof sketch. Full proof in §F. After deleting s, the local replacement uses $|\mathbf{s}_s^{\mathrm{rep}}|$ tokens instead, thus increasing the corpus length by $|\mathbf{s}_s^{\mathrm{rep}}| - 1$. There are $n_s^{\mathcal{D}}$ such occurrences. □

Updating this compression-based tokeniser after a deletion is easier than for log-likelihood. The only thing that must be recomputed is $\texttt{tok}_{\Rightarrow}[\mathcal{S}_k]$, which is trivial given a fixed vocabulary.

5 Experimental Setup

We will now compare our four tokenisers: `BPE`, `BottomUpLL`, `TopDownComp`, and `UnigramLM`.

English setup. For English experiments, we train our tokenisers on an subset with approximately 2B token from FineWeb-Edu (Lozhkov et al., 2024), using vocabulary sizes of 8k, 32k, and 128k. We then again use FineWeb-Edu to train language models using these tokenisers, evaluating 100M, 300M, 500M, and 1B parameter models. Results for 100M-parameter models are averaged over three random seeds, while larger models use one run per configuration.[9] Finally, unless otherwise stated, models are trained with a Chinchilla-style token budget of approximately 20× as many training tokens as model parameters.

[8] The details of this procedure fall out of our work's scope; for an accessible derivation, see Meister (2026).

[9] Full hyperparameter and architectural details are in §G.

Vocab	Tokeniser	Objective	Search	$\mathfrak{G}_{\text{comp}} \downarrow$	$\mathfrak{G}_{\text{ll}} \downarrow$	Entropy	Zipf α	BPT $\uparrow$	Tok Len	Vocab Util $\uparrow$	Cov. 50%
8k	BPE	$\mathfrak{G}_{\text{comp}}$	bottom-up	59,559,835	6.259×10^8	10.508	1.033	3.844	5.53	**99.9%**	242
	TopDownComp	$\mathfrak{G}_{\text{comp}}$	top-down	**59,304,113**	6.259×10^8	10.554	0.919	**3.861**	5.67	99.3%	268
	BottomUpLL	$\mathfrak{G}_{\text{ll}}$	bottom-up	61,170,714	$\mathbf{6.167 \times 10^8}$	10.081	1.315	3.743	6.05	99.4%	152
	UnigramLM	$\mathfrak{G}_{\text{ll}}$	top-down	68,444,350	6.221×10^8	9.089	1.093	3.345	6.72	99.3%	39
32k	BPE	$\mathfrak{G}_{\text{comp}}$	bottom-up	**50,327,096**	5.544×10^8	11.016	1.195	**4.549**	6.52	**99.8%**	269
	TopDownComp	$\mathfrak{G}_{\text{comp}}$	top-down	52,204,230	5.667×10^8	10.856	1.074	4.386	6.29	**99.8%**	192
	BottomUpLL	$\mathfrak{G}_{\text{ll}}$	bottom-up	50,920,853	$\mathbf{5.505 \times 10^8}$	10.811	1.378	4.496	6.98	99.0%	211
	UnigramLM	$\mathfrak{G}_{\text{ll}}$	top-down	58,161,017	5.697×10^8	9.796	1.272	3.937	7.03	**99.8%**	51
128k	BPE	$\mathfrak{G}_{\text{comp}}$	bottom-up	**46,859,565**	5.216×10^8	11.132	1.484	**4.886**	7.06	98.9%	204
	TopDownComp	$\mathfrak{G}_{\text{comp}}$	top-down	50,312,483	5.484×10^8	10.900	1.443	4.551	6.01	**99.8%**	149
	BottomUpLL	$\mathfrak{G}_{\text{ll}}$	bottom-up	47,111,733	$\mathbf{5.203 \times 10^8}$	11.044	1.574	4.860	7.48	96.8%	191
	UnigramLM	$\mathfrak{G}_{\text{ll}}$	top-down	55,546,423	5.552×10^8	9.996	1.717	4.122	6.46	97.1%	53

Table 1: Intrinsic tokenisation results on a held-out test set (47,384 documents). Columns are defined in the Evaluation paragraph of §5. §J reports the multilingual counterpart of this table.

Multilingual setup. For multilingual experiments, we use FineWeb2 (Penedo et al., 2025) and FineWeb-Edu (Lozhkov et al., 2024) to train tokenisers on 10% of our full training corpus with 128k vocabulary size, and we train 1B-parameter models on top of these tokenisers. Our multilingual corpus contains 10B English tokens and 2.5B for each of German, Spanish, Turkish, and Chinese.

Evaluation. We report both extrinsic and intrinsic metrics. For *extrinsic* metrics, we report our language models' bits per byte (BPB), computed on a held-out validation set, and minimal-pair grammatical accuracy, computed on BLiMP (Warstadt et al., 2020), MultiBLiMP (Jumelet et al., 2026), and ZhoBLiMP (Liu et al., 2025). For *intrinsic* metrics, we report $\mathfrak{G}_{\text{comp}}$ and $\mathfrak{G}_{\text{ll}}$. We also report the unigram entropy of our tokenised corpus, its bytes per token (BPT), average vocab token len (Tok Len), vocabulary utilisation (Vocab Util), Zipf α, and Coverage 50%. Detailed definitions are at §H.

6 Results

6.1 Intrinsic Evaluation

Compression vs. Log-likelihood. Tab. 1 presents both the $\mathfrak{G}_{\text{comp}}$ and $\mathfrak{G}_{\text{ll}}$ achieved by all our tokenisers. From this table, we can see that, for a small vocabulary (of $8k$ tokens), the compression and log-likelihood scores achieved by the different tokeniser seem to align with their objective function: BPE and TopDownComp achieve better compression, while UnigramLM and BottomUpLL achieve better log-likelihood. For larger vocabularies, however, this surprisingly does not hold, with the bottom-up tokenisers consistently achieving both better compression and log-likelihood, irrespective of their objective function. Interestingly, a tokeniser's entropy has a strong relationship to our optimisation objectives, as we can rewrite it as:

$$\text{Ent}(\texttt{tok}, \mathcal{D}) \stackrel{\text{def}}{=} -\sum_{s \in \mathcal{S}} p_{\boldsymbol{\theta}}(s) \log p_{\boldsymbol{\theta}}(s) \tag{15a}$$

$$= \frac{\mathfrak{G}_{\text{ll}}(\texttt{tok}, \mathcal{D})}{\mathfrak{G}_{\text{comp}}(\texttt{tok}, \mathcal{D})} \tag{15b}$$

Looking at the entropy values in Tab. 1 we see a similar behaviour as before: for small vocabularies, entropy results align with the choice of tokenisation objective; for a larger vocabulary, however, top-down methods achieve consistently lower entropy. Future work should further investigate this relationship between objective and search procedure.

Distributional Properties of Tokens. Tab. 1 also reports our tokenisers Zipf α and Coverage 50%, where Zipf α is the negated slope of a frequency vs. rank log-log curve, and Coverage 50% is the smallest ℓ such that the ℓ most frequent tokens together account for at least half of all token occurrences in our corpus. A larger Zipf α thus means that token frequency decreases more rapidly with rank, while lower Coverage 50% indicate that token occurrences are concentrated among fewer token types. Within each search family, the likelihood-based method induces a more concentrated token-frequency distribution than the compression-based method: BottomUpLL has higher Zipf α, lower entropy, and lower Coverage 50% than BPE, and UnigramLM shows the same pattern relative to TopDownComp. Thus, while the search procedure may dictate the values of $\mathfrak{G}_{\text{comp}}$ and $\mathfrak{G}_{\text{ll}}$, the used objective is still visible in a tokenised corpus' empirical token-frequency distribution.

Other Intrinsic Metrics. Finally, Tab. 1 also presents our tokeniser's vocabulary utilisation

	BPE	TopDownComp	BottomUpLL =	BottomUpLL ≈	UnigramLM
BPE	—	50.2%	82.3%	81.7%	41.8%
TopDownComp	50.2%	—	45.1%	45.2%	60.7%
BottomUpLL =	82.3%	45.1%	—	94.5%	41.2%
BottomUpLL ≈	81.7%	45.2%	94.5%	—	41.5%
UnigramLM	41.8%	60.7%	41.2%	41.5%	—

Table 2: Pairwise vocabulary overlap at 128k vocabulary size. See §I for an analysis of the merge dynamics in bottom-up tokenisers, and §J for a multilingual counterpart of this table.

				100M		300M		500M		1B	
Vocab	Tokeniser	Objective	Search	BLiMP ↑	BPB ↓	BLiMP ↑	BPB ↓	BLiMP ↑	BPB ↓	BLiMP ↑	BPB ↓
8k	BPE	$\mathfrak{G}_{\text{comp}}$	bottom-up	.686 ± 6e-3	1.0736 ± 2e-4	–	–	–	–	–	–
	TopDownComp	$\mathfrak{G}_{\text{comp}}$	top-down	.687 ± 4e-3	1.0750 ± 8e-4	–	–	–	–	–	–
	BottomUpLL	$\mathfrak{G}_{\text{ll}}$	bottom-up	**.703 ± 8e-3**	**1.0715 ± 6e-4**	–	–	–	–	–	–
	UnigramLM	$\mathfrak{G}_{\text{ll}}$	top-down	.698 ± 5e-3	1.0819 ± 4e-4	–	–	–	–	–	–
32k	BPE	$\mathfrak{G}_{\text{comp}}$	bottom-up	.714 ± 8e-3	1.0392 ± 5e-4	–	–	–	–	–	–
	TopDownComp	$\mathfrak{G}_{\text{comp}}$	top-down	**.727 ± 2e-2**	1.0394 ± 3e-4	–	–	–	–	–	–
	BottomUpLL	$\mathfrak{G}_{\text{ll}}$	bottom-up	.717 ± 7e-3	**1.0374 ± 1e-4**	–	–	–	–	–	–
	UnigramLM	$\mathfrak{G}_{\text{ll}}$	top-down	.719 ± 8e-3	1.0441 ± 9e-4	–	–	–	–	–	–
128k	BPE	$\mathfrak{G}_{\text{comp}}$	bottom-up	**.740 ± 4e-3**	**1.0138 ± 4e-4**	.797	**.8462**	.813	.8210	.805	**.7795**
	TopDownComp	$\mathfrak{G}_{\text{comp}}$	top-down	**.740 ± 8e-3**	1.0172 ± 4e-4	.797	.8492	**.815**	.8226	.816	.7819
	BottomUpLL	$\mathfrak{G}_{\text{ll}}$	bottom-up	.725 ± 6e-3	1.0149 ± 9e-4	.792	.8467	.798	**.8207**	.799	.7833
	UnigramLM	$\mathfrak{G}_{\text{ll}}$	top-down	.738 ± 7e-3	1.0298 ± 8e-4	**.799**	.8547	.803	.8283	**.818**	.7872

Table 3: Extrinsic tokenisation results for language models trained in English. Cells marked "–" were not run; 300M, 500M, and 1B models were trained only at 128k vocabulary. 100M parameters results are averaged over three random seeds. Lower is better for BPB; higher is better for BLiMP.

and average token length. Vocabulary utilisation is close to 100% at 8k and 32k, but at 128k, likelihood-based tokenisers have more unused tokens. Token length behaves differently across vocabulary sizes: at small vocabulary sizes, top-down present longer tokens, as they can retain these tokens from the initial seed vocabulary; at larger vocabulary sizes, though, bottom-up present longer tokens instead, as they can build these tokens from scratch through successive merges. Interestingly, across both search procedures, the likelihood-based tokenisers have longer tokens than their compression-based counterparts.

Vocabulary Composition. We now move on to analysing the composition of our tokenisers vocabularies. To this end, we compute their similarity, defined as $|\mathcal{S}_A \cap \mathcal{S}_B|/K$. Tab. 2 presents these results. From this table, we see that learned vocabularies cluster primarily by search procedure: e.g., bottom-up methods share substantially more tokens with each other than with top-down methods. This indicates that a tokeniser's search procedure has a large effect on which tokens are included in its final vocabulary. Per-example qualitative segmentations for the four tokenisers are shown in §K.

6.2 Extrinsic Evaluation

BPB. Tabs. 3 and 4 present extrinsic results for both the English and multilingual settings. From this tables, we can see that bottom-up tokenisers consistently achieve lower BPB scores than top-down. This holds true for all our experimental conditions, with the only exception of the English-only 1B models, for which TopDownComp slightly outperforms BottomUpLL; BPE still performs best in this setting though. Comparing tokenisers within a single search procedure, we see that the choice of objective is still relevant, albeit less strongly. Compression-based methods tend to outperform log-likelihood; but there are several exceptions where BottomUpLL outperforms all other methods.

Minimal-pair accuracy. Tabs. 3 and 4 also present scores on a minimal pair grammaticality task, for both the English and multilingual settings. Notably, these results do not show the same clear preference as BPB for bottom-up over top-down. In fact, in the English setting, top-down methods tend to outperform the bottom-up ones, with UnigramLM outperforming BottomUpLL and TopDownComp outperforming BPE. In the multilingual settings, no patterns arise, and all tokenisers perform best in at least one language.

Tokeniser	Objective	Search	BLiMP ↑	Minimal-pair accuracy ↑					BPB ↓				
				eng	deu	spa	tur	cmn	eng	deu	spa	tur	cmn
BPE	$\mathfrak{G}_{\text{comp}}$	bottom-up	**.812**	**.974**	.955	.958	.890	.799	**.8080**	.9426	**.9025**	.8936	1.2464
TopDownComp	$\mathfrak{G}_{\text{comp}}$	top-down	.808	.969	**.966**	.956	.853	.718	.8276	.9711	.9284	.9260	1.3430
BottomUpLL	$\mathfrak{G}_{\text{ll}}$	bottom-up	.808	.970	.952	.950	.873	**.803**	.8082	**.9416**	.9030	**.8929**	**1.2436**
UnigramLM	$\mathfrak{G}_{\text{ll}}$	top-down	.803	.973	.964	**.959**	**.897**	.728	.8311	.9683	.9303	.9252	1.3575

Table 4: Extrinsic tokenisation results for 1B parameter models trained in a multilingual setting. Minimal-pair accuracy is evaluated on BLiMP (Warstadt et al., 2020), MultiBLiMP (eng, deu, spa, tur; Jumelet et al., 2026), and ZhoBLiMP (cmn; Liu et al., 2025). Lower is better for BPB; higher is better for minimal-pair accuracy.

7 Conclusion

Our paper disentangles two important tokeniser-design choices: the objective used to score a tokeniser and the search procedure used to optimise it. We then analyse two objectives (compression vs. log-likelihood) and search procedures (bottom-up vs. top-down), introducing two new tokenisers in the process: BottomUpLL and TopDownComp. Across experimental conditions, language models trained with bottom-up tokenisers consistently outperformed top-down ones in BPB. Further, our intrinsic evaluations also highlighted a surprising pattern: bottom-up tokenisers produced both more compressed and higher log-likelihood token-strings than top-down tokenisers, independent of the used objective function. The used objective still affects the resulting tokeniser. With log-likelihood objective leaving a clear impact on the empirical rank–frequency distribution of its tokens. Overall, our results show that tokeniser quality cannot be explained by the objective alone, and that the used search procedure is an important component of tokeniser design.

Limitations

Our study has several limitations. **Scale.** First, we only train language models up to 1B parameters, and results could shift for larger models or longer training horizons. **Languages.** Second, our experiments cover English, German, Spanish, Turkish, and Chinese, but may not generalise to other scripts or typologically distinct languages. **Single seed at scale.** Third, for computational reasons, we report a single training seed results per configuration for models above 100M parameters; small differences in results across tokenisers should thus be interpreted with caution. **Tokeniser-training corpus.** Fourth, in all our experiments, each tokeniser is trained on the same corpus as the language model trained on top of it. We thus do not analyse the impact of training models on data that differ in domain or style from its tokeniser. **Approximate top-down deletion scores.** Fifth, for top-down methods, deletion costs are estimated using the local replacement approximation described in §4.2, rather than by recomputing the full objective after every possible deletion. This approximation is necessary for tractability, but it means that the top-down methods include an additional approximation in their per-step candidate scores. By contrast, the merge gains used by BPE and BottomUpLL are exact for the current greedy step, although the overall bottom-up search remains greedy and therefore not globally optimal. Investigating the impact of the local replacement approximation assumption on our tokenisers would be interesting and is left as future work. **Evaluation.** Finally, we evaluate models with BPB and minimal-pair grammatical benchmarks (BLiMP, MultiBLiMP, ZhoBLiMP). We do not measure other downstream task performance, reasoning ability, or long-context behaviour; how our models perform in those different settings is thus left open.

References

Philip Gage. 1994. A new algorithm for data compression. *C Users Journal*, 12(2):23–38.

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.

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.

Violeta Kastreva, Philip Whittington, Dennis Komm, and Tiago Pimentel. 2026. Tokenisation over

bounded alphabets is hard. In *The Fourteenth International Conference on Learning Representations*.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, S. H. Cai, Yuan Cao, Y. Charles, H. S. Che, Cheng Chen, Guanduo Chen, Huarong Chen, Jia Chen, Jiahao Chen, Jianlong Chen, Jun Chen, Kefan Chen, Liang Chen, Ruijue Chen, Xinhao Chen, and 307 others. 2026. Kimi K2.5: Visual agentic intelligence. *Preprint*, arXiv:2602.02276.

László Kozma and Johannes Voderholzer. 2024. Theoretical analysis of byte-pair encoding. *Preprint*, arXiv:2411.08671.

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.

Yikang Liu, Yeting Shen, Hongao Zhu, Lilong Xu, Zhiheng Qian, Siyuan Song, Kejia Zhang, Jialong Tang, Pei Zhang, Baosong Yang, Rui Wang, and Hai Hu. 2025. A systematic assessment of language models with linguistic minimal pairs in chinese. *Preprint*, arXiv:2411.06096.

Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. 2024. FineWeb-Edu: the finest collection of educational content.

Clara Meister. 2026. UnigramLM: An attempt at writing the missing manual. In *Proceedings of the Fourteenth International Conference on Learning Representations*. ICLR 2026 Blogpost Track.

OpenAI, Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sebastien Bubeck, Che Chang, and 107 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *Preprint*, arXiv:2508.10925.

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. *Preprint*, arXiv:2506.20920.

Qwen Team. 2026. Qwen3.5-Omni technical report. *Preprint*, arXiv:2604.15804.

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.

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

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.

Alex Warstadt, Alicia Parrish, Haokun Liu, Anhad Mohananey, Wei Peng, Sheng-Fu Wang, and Samuel R. Bowman. 2020. BLiMP: The benchmark of linguistic minimal pairs for english. *Transactions of the Association for Computational Linguistics*, 8:377–392.

Philip Whittington, Gregor Bachmann, and Tiago Pimentel. 2025. Tokenisation is NP-complete. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vienna, Austria. Association for Computational Linguistics.

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 Proof of Lemma 1

Lemma 1. *The change in compression due to merging a token-pair can be computed as:*

$$\Delta^{\mathfrak{G}_{\text{comp}}(\cdot,\mathcal{D})}_{s^1,s^2} = n^{\mathcal{D}}_{s^1,s^2} \tag{9}$$

where $n^{\mathcal{D}}_{s^1,s^2}$ *denotes the number of non-overlapping* $\langle s^1, s^2\rangle$ *sequences in* $\mathcal{D}$.

Proof. To prove this is the case, first note that replacing a token-pair $m = \langle s^1, s^2\rangle$ with merged token $s^1 \circ s^2$ saves exactly 1 token. As the number of non-overlapping occurrences of a token-pair is also the number of possible merges, it is equivalent to the compression this token-pair would achieve:

$$\mathfrak{G}(\texttt{tok}_{\uparrow}[\mathbf{m}_{<k}], \mathcal{D}) - \mathfrak{G}(\texttt{tok}_{\uparrow}[\mathbf{m}_{<k} \circ \langle s^1, s^2\rangle], \mathcal{D}) = n^{\mathcal{D}}_{s^1,s^2}. \tag{16}$$

We can now trivially conclude the proof:

$$
\begin{aligned}
m^\star &= \operatorname*{argmin}_{m \in \Sigma^+ \times \Sigma^+} \mathfrak{G}(\texttt{tok}_\uparrow[\mathbf{m}_{<k} \circ m], \mathcal{D}) && \text{(17a)}\\
&= \operatorname*{argmax}_{m \in \Sigma^+ \times \Sigma^+} -\mathfrak{G}(\texttt{tok}_\uparrow[\mathbf{m}_{<k} \circ m], \mathcal{D}) && \text{(17b)}\\
&= \operatorname*{argmax}_{m \in \Sigma^+ \times \Sigma^+} \mathfrak{G}(\texttt{tok}_\uparrow[\mathbf{m}_{<k}], \mathcal{D}) && \text{(17c)}\\
&\qquad\qquad - \mathfrak{G}(\texttt{tok}_\uparrow[\mathbf{m}_{<k} \circ m], \mathcal{D})\\
&= \operatorname*{argmax}_{\langle s^1, s^2\rangle \in \Sigma^+ \times \Sigma^+} n^{\mathcal{D}}_{s^1, s^2}. && \text{(17d)}
\end{aligned}
$$

This concludes the proof. □

B Updating Counts in bottom-up Procedures

Both BPE and BottomUpLL repeatedly select a token-pair from the current tokenised corpus. A naive implementation would recompute all token and token-pair counts after every merge, which is prohibitively expensive on large corpora. Instead, our implementation maintains these counts incrementally and updates only the parts of the corpus affected by the selected merge.

The corpus is represented as a collection of unique word types, each with an associated frequency. Each word type stores its current tokenisation using a linked token structure: every token stores pointers to its predecessor and successor, allowing merges to be applied locally without reconstructing the full token string. We maintain a mapping

$$
\begin{aligned}
&\text{pair_to_words} : \langle s^1, s^2\rangle\\
&\qquad \mapsto \{\text{word types containing } \langle s^1, s^2\rangle\}, \quad (18)
\end{aligned}
$$

which maps each token-pair to the word types in which that pair currently occurs. We also maintain a reverse index from tokens to the token-pairs containing them, which allows us to identify which candidate scores may be affected after a merge.

When a merge $m = \langle s^1, s^2\rangle$ is selected, write

$$s^{\text{new}} \stackrel{\text{def}}{=} s^1 \circ s^2.$$

Each affected word type is scanned through its linked token sequence, and non-overlapping occurrences of $\langle s^1, s^2\rangle$ are replaced by s^{new}. The non-overlap condition is enforced during this scan: once an occurrence has been merged, the scan skips over the newly created token, so overlapping occurrences of the same pair are not counted twice.

Locally, an occurrence in context

$$\langle s^\ell, s^1, s^2, s^r\rangle \qquad (19)$$

is replaced by

$$\langle s^\ell, s^{\text{new}}, s^r\rangle, \qquad (20)$$

where s^ℓ and s^r denote the immediate left and right neighbours, if they exist. Therefore, only token-pairs in this local neighbourhood can change. The old token-pairs

$$\langle s^\ell, s^1\rangle, \qquad \langle s^1, s^2\rangle, \qquad \langle s^2, s^r\rangle \qquad (21)$$

are removed, and the new token-pairs

$$\langle s^\ell, s^{\text{new}}\rangle, \qquad \langle s^{\text{new}}, s^r\rangle \qquad (22)$$

are added. Boundary cases are handled by omitting updates involving missing neighbours.

For one merged occurrence, the affected pair counts are updated as

$$
\begin{aligned}
n^{\mathcal{D}}_{s^\ell, s^1} &\leftarrow n^{\mathcal{D}}_{s^\ell, s^1} - 1, && (23)\\
n^{\mathcal{D}}_{s^1, s^2} &\leftarrow n^{\mathcal{D}}_{s^1, s^2} - 1, && (24)\\
n^{\mathcal{D}}_{s^2, s^r} &\leftarrow n^{\mathcal{D}}_{s^2, s^r} - 1, && (25)\\
n^{\mathcal{D}}_{s^\ell, s^{\text{new}}} &\leftarrow n^{\mathcal{D}}_{s^\ell, s^{\text{new}}} + 1, && (26)\\
n^{\mathcal{D}}_{s^{\text{new}}, s^r} &\leftarrow n^{\mathcal{D}}_{s^{\text{new}}, s^r} + 1. && (27)
\end{aligned}
$$

When word types have frequencies, these increments and decrements are weighted by the corresponding word-type frequency.

For BottomUpLL, we also update unigram counts, since the merge score in Lemma 2 depends on both token counts and pair counts. If $n^{\mathcal{D}}_{s^1, s^2}$ denotes the frequency-weighted number of non-overlapping occurrences of the selected pair that are actually merged, then the unigram counts are updated as

$$
\begin{aligned}
n^{\mathcal{D}}_{s^1} &\leftarrow n^{\mathcal{D}}_{s^1} - n^{\mathcal{D}}_{s^1, s^2}, && (28)\\
n^{\mathcal{D}}_{s^2} &\leftarrow n^{\mathcal{D}}_{s^2} - n^{\mathcal{D}}_{s^1, s^2}, && (29)\\
n^{\mathcal{D}}_{s^{\text{new}}} &\leftarrow n^{\mathcal{D}}_{s^1, s^2}, && (30)\\
N^{\mathcal{D}} &\leftarrow N^{\mathcal{D}} - n^{\mathcal{D}}_{s^1, s^2}. && (31)
\end{aligned}
$$

For BPE, the affected candidate scores are simply the updated pair counts, as in Lemma 1. For BottomUpLL, affected candidate scores are recomputed using the local likelihood-change expression in Lemma 2.

The priority queue of candidate token-pairs is maintained lazily. After a merge, the selected pair is removed from the pair-to-word index, affected reverse-index entries are updated, and affected token-pairs are pushed back onto the heap with fresh scores. Old heap entries are not removed immediately. Instead, stale entries are rejected when popped: the implementation checks the current count and score, and discards the heap entry if it no longer matches the current state. The heap is also periodically rebuilt to control the accumulation of stale entries.

This gives an incremental implementation in which the cost of a merge is proportional to the number of affected word types and local token updates, plus the cost of heap maintenance. In practice, this is much cheaper than recomputing all token and token-pair counts over the full corpus after every merge.

C Proof of Lemma 2

Lemma 2. *The change in log-likelihood due to merging a token-pair can be computed as:*[10]

$$\begin{aligned}
&\Delta^{\mathfrak{G}_{\mathrm{ll}}(\cdot,\mathcal{D})}_{s^1,s^2} = \qquad (10)\\
&(n^{\mathcal{D}}_{s^1} - n^{\mathcal{D}}_{s^1,s^2})\log(n^{\mathcal{D}}_{s^1} - n^{\mathcal{D}}_{s^1,s^2}) - n^{\mathcal{D}}_{s^1}\log n^{\mathcal{D}}_{s^1}\\
&+ (n^{\mathcal{D}}_{s^2} - n^{\mathcal{D}}_{s^1,s^2})\log(n^{\mathcal{D}}_{s^2} - n^{\mathcal{D}}_{s^1,s^2}) - n^{\mathcal{D}}_{s^2}\log n^{\mathcal{D}}_{s^2}\\
&+ n^{\mathcal{D}}_{s^1,s^2}\log n^{\mathcal{D}}_{s^1,s^2}\\
&- (N^{\mathcal{D}} - n^{\mathcal{D}}_{s^1,s^2})\log(N^{\mathcal{D}} - n^{\mathcal{D}}_{s^1,s^2}) + N^{\mathcal{D}}\log N^{\mathcal{D}}.
\end{aligned}$$

Proof. We prove the result by explicitly writing the corpus log-likelihood before and after applying the merge.

Let the current tokenised corpus contain $N^{\mathcal{D}}$ tokens in total. For each token $\tilde{s}$, let $n^{\mathcal{D}}_{\tilde{s}}$ denote its corpus count. Under the unigram token model, the empirical probability of $\tilde{s}$ is

$$p_{\boldsymbol{\theta}}(\tilde{s}) = \frac{n^{\mathcal{D}}_{\tilde{s}}}{N^{\mathcal{D}}}. \qquad (32)$$

Therefore, the corpus log-likelihood before applying the merge is

$$\begin{aligned}
\mathcal{L}_{\text{before}} &= \sum_{\tilde{s}} n^{\mathcal{D}}_{\tilde{s}} \log p_{\boldsymbol{\theta}}(\tilde{s}) && (33)\\
&= \sum_{\tilde{s}} n^{\mathcal{D}}_{\tilde{s}} \log \frac{n^{\mathcal{D}}_{\tilde{s}}}{N^{\mathcal{D}}} && (34)\\
&= \sum_{\tilde{s}} n^{\mathcal{D}}_{\tilde{s}} \log n^{\mathcal{D}}_{\tilde{s}} - N^{\mathcal{D}}\log N^{\mathcal{D}}. && (35)
\end{aligned}$$

Now consider merging the token-pair $\langle s^1, s^2\rangle$ into the new token $s^1 \circ s^2$. For readability, write

$$\kappa \stackrel{\text{def}}{=} n^{\mathcal{D}}_{s^1,s^2}, \qquad (36)$$

where κ is the number of non-overlapping occurrences of $\langle s^1, s^2\rangle$ in the current corpus.

Assuming $s^1 \neq s^2$, applying this merge changes only the counts of s^1, s^2, and the new token $s^1 \circ s^2$. Specifically,

$$\begin{aligned}
n^{\mathcal{D}}_{s^1} &\mapsto n^{\mathcal{D}}_{s^1} - \kappa, && (37)\\
n^{\mathcal{D}}_{s^2} &\mapsto n^{\mathcal{D}}_{s^2} - \kappa, && (38)\\
n^{\mathcal{D}}_{s^1\circ s^2} &\mapsto \kappa. && (39)
\end{aligned}$$

All other token counts remain unchanged. Since each merged occurrence replaces two tokens by one token, the total token count changes from $N^{\mathcal{D}}$ to $N^{\mathcal{D}} - \kappa$.

The log-likelihood after the merge is therefore

$$\begin{aligned}
\mathcal{L}_{\text{after}} = &\sum_{\tilde{s}\notin\{s^1,s^2,s^1\circ s^2\}} n^{\mathcal{D}}_{\tilde{s}}\log n^{\mathcal{D}}_{\tilde{s}}\\
&+ \left(n^{\mathcal{D}}_{s^1} - \kappa\right)\log\left(n^{\mathcal{D}}_{s^1} - \kappa\right)\\
&+ \left(n^{\mathcal{D}}_{s^2} - \kappa\right)\log\left(n^{\mathcal{D}}_{s^2} - \kappa\right)\\
&+ \kappa\log\kappa - (N^{\mathcal{D}} - \kappa)\log(N^{\mathcal{D}} - \kappa).
\end{aligned} \qquad (40)$$

The local log-likelihood change due to the merge is

$$\Delta^{\mathfrak{G}_{\mathrm{ll}}(\cdot,\mathcal{D})}_{s^1,s^2} = \mathcal{L}_{\text{after}} - \mathcal{L}_{\text{before}}. \qquad (41)$$

Substituting Eqs. (35) and (40), all unchanged token-count terms cancel, leaving

$$\begin{aligned}
\Delta^{\mathfrak{G}_{\mathrm{ll}}(\cdot,\mathcal{D})}_{s^1,s^2} = &\left(n^{\mathcal{D}}_{s^1} - \kappa\right)\log\left(n^{\mathcal{D}}_{s^1} - \kappa\right)\\
&- n^{\mathcal{D}}_{s^1}\log n^{\mathcal{D}}_{s^1}\\
&+ \left(n^{\mathcal{D}}_{s^2} - \kappa\right)\log\left(n^{\mathcal{D}}_{s^2} - \kappa\right)\\
&- n^{\mathcal{D}}_{s^2}\log n^{\mathcal{D}}_{s^2}\\
&+ \kappa\log\kappa\\
&- (N^{\mathcal{D}} - \kappa)\log(N^{\mathcal{D}} - \kappa) + N^{\mathcal{D}}\log N^{\mathcal{D}}.
\end{aligned} \qquad (42)$$

[10] For the case $s^1 = s^2$, the same calculation applies albeit with a modified count update: each merge consumes two occurrences of the same token s^1 and thus its count is reduced by $2n^{\mathcal{D}}_{s^1,s^1}$ instead; other parts of the update remain unchanged.

Finally substituting $\kappa = n^{\mathcal{D}}_{s^1,s^2}$ gives

$$\begin{aligned}\Delta^{\mathfrak{G}_{\mathrm{ll}}(\cdot,\mathcal{D})}_{s^1,s^2} &= \left(n^{\mathcal{D}}_{s^1} - n^{\mathcal{D}}_{s^1,s^2}\right)\log\left(n^{\mathcal{D}}_{s^1} - n^{\mathcal{D}}_{s^1,s^2}\right)\\ &\quad - n^{\mathcal{D}}_{s^1}\log n^{\mathcal{D}}_{s^1}\\ &\quad + \left(n^{\mathcal{D}}_{s^2} - n^{\mathcal{D}}_{s^1,s^2}\right)\log\left(n^{\mathcal{D}}_{s^2} - n^{\mathcal{D}}_{s^1,s^2}\right)\\ &\quad - n^{\mathcal{D}}_{s^2}\log n^{\mathcal{D}}_{s^2}\\ &\quad + n^{\mathcal{D}}_{s^1,s^2}\log n^{\mathcal{D}}_{s^1,s^2}\\ &\quad - \left(N^{\mathcal{D}} - n^{\mathcal{D}}_{s^1,s^2}\right)\log\left(N^{\mathcal{D}} - n^{\mathcal{D}}_{s^1,s^2}\right)\\ &\quad + N^{\mathcal{D}}\log N^{\mathcal{D}}. \qquad (43)\end{aligned}$$

This proves the claim. □

D Proof of Lemma 3

Lemma 3. *Under a first-order Taylor approximation, the change in log-likelihood due to merging a token-pair can be approximated as:*

$$\Delta^{\mathfrak{G}_{\mathrm{ll}}(\cdot,\mathcal{D})}_{s^1,s^2} \approx n^{\mathcal{D}}_{s^1,s^2}\left(\mathrm{PMI}_{\mathrm{adj}}(s^1, s^2) - 1\right). \qquad (12)$$

Proof. For readability, write

$$\kappa \stackrel{\text{def}}{=} n^{\mathcal{D}}_{s^1,s^2}. \qquad (44)$$

Starting from Eq. (43), we have

$$\begin{aligned}&\Delta^{\mathfrak{G}_{\mathrm{ll}}(\cdot,\mathcal{D})}_{s^1,s^2}\\ &= \left(n^{\mathcal{D}}_{s^1} - \kappa\right)\log\left(n^{\mathcal{D}}_{s^1} - \kappa\right) - n^{\mathcal{D}}_{s^1}\log n^{\mathcal{D}}_{s^1}\\ &\quad + \left(n^{\mathcal{D}}_{s^2} - \kappa\right)\log\left(n^{\mathcal{D}}_{s^2} - \kappa\right) - n^{\mathcal{D}}_{s^2}\log n^{\mathcal{D}}_{s^2}\\ &\quad + \kappa\log\kappa - (N^{\mathcal{D}} - \kappa)\log(N^{\mathcal{D}} - \kappa)\\ &\quad + N^{\mathcal{D}}\log N^{\mathcal{D}}. \qquad (45)\end{aligned}$$

Define

$$f(x) = x\log x. \qquad (46)$$

Then

$$f'(x) = \log x + 1. \qquad (47)$$

For $\kappa \ll x$, the first-order Taylor expansion gives

$$\begin{aligned}f(x-\kappa) &\approx f(x) - \kappa f'(x)\\ &= x\log x - \kappa(\log x + 1). \qquad (48)\end{aligned}$$

Applying Eq. (48) to the s^1 term gives

$$\begin{aligned}&\left(n^{\mathcal{D}}_{s^1} - \kappa\right)\log\left(n^{\mathcal{D}}_{s^1} - \kappa\right) - n^{\mathcal{D}}_{s^1}\log n^{\mathcal{D}}_{s^1}\\ &\quad \approx -\kappa\left(\log n^{\mathcal{D}}_{s^1} + 1\right). \qquad (49)\end{aligned}$$

Similarly, for the s^2 term,

$$\begin{aligned}&\left(n^{\mathcal{D}}_{s^2} - \kappa\right)\log\left(n^{\mathcal{D}}_{s^2} - \kappa\right) - n^{\mathcal{D}}_{s^2}\log n^{\mathcal{D}}_{s^2}\\ &\quad \approx -\kappa\left(\log n^{\mathcal{D}}_{s^2} + 1\right). \qquad (50)\end{aligned}$$

For the total-count term, note the negative sign:

$$\begin{aligned}&-(N^{\mathcal{D}} - \kappa)\log(N^{\mathcal{D}} - \kappa) + N^{\mathcal{D}}\log N^{\mathcal{D}}\\ &\quad = -\left[(N^{\mathcal{D}} - \kappa)\log(N^{\mathcal{D}} - \kappa) - N^{\mathcal{D}}\log N^{\mathcal{D}}\right]\\ &\quad \approx \kappa\left(\log N^{\mathcal{D}} + 1\right). \qquad (51)\end{aligned}$$

The new-token term is exact:

$$\kappa\log\kappa. \qquad (52)$$

Substituting these terms into Eq. (45), we obtain

$$\begin{aligned}\Delta^{\mathfrak{G}_{\mathrm{ll}}(\cdot,\mathcal{D})}_{s^1,s^2} &\approx -\kappa\left(\log n^{\mathcal{D}}_{s^1} + 1\right) - \kappa\left(\log n^{\mathcal{D}}_{s^2} + 1\right)\\ &\quad + \kappa\log\kappa + \kappa\left(\log N^{\mathcal{D}} + 1\right) \qquad (53)\\ &= \kappa\left(\log\frac{N^{\mathcal{D}}\,\kappa}{n^{\mathcal{D}}_{s^1}n^{\mathcal{D}}_{s^2}} - 1\right). \qquad (54)\end{aligned}$$

Substituting back $\kappa = n^{\mathcal{D}}_{s^1,s^2}$ yields

$$\Delta^{\mathfrak{G}_{\mathrm{ll}}(\cdot,\mathcal{D})}_{s^1,s^2} \approx n^{\mathcal{D}}_{s^1,s^2}\left(\log\frac{N^{\mathcal{D}}\,n^{\mathcal{D}}_{s^1,s^2}}{n^{\mathcal{D}}_{s^1}n^{\mathcal{D}}_{s^2}} - 1\right). \qquad (55)$$

By the definition of $\mathrm{PMI}_{\mathrm{adj}}(s^1, s^2)$ in Eq. (11b), this is

$$\Delta^{\mathfrak{G}_{\mathrm{ll}}(\cdot,\mathcal{D})}_{s^1,s^2} \approx n^{\mathcal{D}}_{s^1,s^2}\left(\mathrm{PMI}_{\mathrm{adj}}(s^1, s^2) - 1\right). \qquad (56)$$

This proves the approximation. □

E Proof of Lemma 4

Lemma 4. *Under a local replacement approximation, the estimated increase in the negative log-likelihood objective due to deleting a token $s \in \mathcal{S} \setminus \Sigma$ can be approximated as:*[11]

$$\Delta^{\mathfrak{G}_{\mathrm{ll}}(\cdot,\mathcal{D})}_{s} \approx n^{\mathcal{D}}_{s}\left(\log p_{\boldsymbol{\theta}}(s) - \log p_{\boldsymbol{\theta}}(\mathbf{s}^{\mathrm{rep}}_s)\right), \qquad (13)$$

where $p_{\boldsymbol{\theta}}$ is the unigram model at step $k-1$, and $\mathbf{s}^{\mathrm{rep}}_s$ is s's log-probability–optimal replacement segmentation, i.e.,: $\mathbf{s}^{\mathrm{rep}}_s \stackrel{\text{def}}{=} \mathtt{tok}_{\Rightarrow}[\mathcal{S}_{k-1} \setminus \{s\}](s)$.

Proof. We prove the result under the local replacement approximation stated in the lemma. That is, when scoring the deletion of s, we keep the current token probabilities $p_{\boldsymbol{\theta}}$ fixed, keep all other segmentation decisions fixed, and approximate the effect of deletion by replacing each expected occurrence of s with a replacement segmentation under $\mathcal{S} \setminus \{s\}$.

For UnigramLM, token usage is measured by expected counts under the current posterior over valid

[11] In practice, when computing this update, UnigramLM keeps segmentations latent and leverages $p_{\boldsymbol{\theta}}$ to compute $n^{\mathcal{D}}_s$ as the *expected* count of s, instead of its exact count under $\mathtt{tok}_{\Rightarrow}$.

segmentations. Thus, $n_s^{\mathcal{D}}$ is the expected number of times s is used in the corpus. Before deleting s, each such expected occurrence contributes

$$-\log p_{\boldsymbol{\theta}}(s) \tag{57}$$

to the negative log-likelihood objective.

After deleting s, the token can no longer be used as a single token. Under the local replacement approximation, each occurrence of s is replaced by the highest-probability valid segmentation of the same string under the reduced vocabulary:

$$\mathbf{s}_s^{\text{rep}} \stackrel{\text{def}}{=} \operatorname*{argmax}_{\substack{\tilde{\mathbf{s}} \in (\mathcal{S} \setminus \{s\})^* \\ s \stackrel{\circ}{=} \tilde{\mathbf{s}}}} p_{\boldsymbol{\theta}}(\tilde{\mathbf{s}}). \tag{58}$$

Since $p_{\boldsymbol{\theta}}$ is a unigram model, the probability of this replacement token-string is

$$p_{\boldsymbol{\theta}}(\mathbf{s}_s^{\text{rep}}) = \prod_{\tilde{s} \in \mathbf{s}_s^{\text{rep}}} p_{\boldsymbol{\theta}}(\tilde{s}). \tag{59}$$

After replacement, each expected occurrence therefore contributes

$$-\log p_{\boldsymbol{\theta}}(\mathbf{s}_s^{\text{rep}}) \tag{60}$$

to the negative log-likelihood objective.

Hence, the estimated increase in negative log-likelihood for one expected occurrence of s is

$$\begin{aligned} &-\log p_{\boldsymbol{\theta}}(\mathbf{s}_s^{\text{rep}}) - \big(-\log p_{\boldsymbol{\theta}}(s)\big) \\ &\quad = \log p_{\boldsymbol{\theta}}(s) - \log p_{\boldsymbol{\theta}}(\mathbf{s}_s^{\text{rep}}). \end{aligned} \tag{61}$$

Multiplying this per-occurrence increase by the expected number of occurrences $n_s^{\mathcal{D}}$ gives

$$\Delta_s^{\mathfrak{G}_{\text{ll}}(\cdot,\mathcal{D})} \approx n_s^{\mathcal{D}} \left(\log p_{\boldsymbol{\theta}}(s) - \log p_{\boldsymbol{\theta}}(\mathbf{s}_s^{\text{rep}})\right). \tag{62}$$

This proves the stated local approximation. □

F Proof of Lemma 5

Lemma 5. *Under a local replacement approximation, the estimated change in compression due to deleting a token $s \in \mathcal{S} \setminus \Sigma$ can be approximated as:*

$$\Delta_s^{\mathfrak{G}_{\text{comp}}(\cdot,\mathcal{D})} = n_s^{\mathcal{D}} \left(|\mathbf{s}_s^{\text{rep}}| - 1\right), \tag{14}$$

where $\mathbf{s}_s^{\text{rep}}$ is the compression-optimal replacement of s after deletion: $\mathbf{s}_s^{\text{rep}} \stackrel{\text{def}}{=} \texttt{tok}_{\Rightarrow}[\mathcal{S}_{k-1} \setminus \{s\}](s)$.

Proof. Let $\mathcal{S}$ be the current vocabulary, and suppose the corpus has already been segmented using the TopDownComp encoding rule:

$$\texttt{tok}_{\Rightarrow}[\mathcal{S}](\mathbf{c}) = \operatorname*{argmin}_{\substack{\mathbf{s} \in \mathcal{S}^* \\ \mathbf{c} \stackrel{\circ}{=} \mathbf{s}}} |\mathbf{s}|. \tag{63}$$

Now consider deleting a token $s \in \mathcal{S} \setminus \Sigma$. Before deletion, each current occurrence of s contributes exactly one token to the corpus token count. Therefore, the total contribution of all current occurrences of s before deletion is

$$C_{\text{before}}(s) = n_s^{\mathcal{D}}. \tag{64}$$

After deleting s, the token s can no longer appear as a single token. Under the local replacement calculation, each current occurrence of s is independently replaced by its shortest decomposition under the reduced vocabulary $\mathcal{S} \setminus \{s\}$, denoted $\mathbf{s}_s^{\text{rep}}$:

$$\mathbf{s}_s^{\text{rep}} = \operatorname*{argmin}_{\substack{\mathbf{s}' \in (\mathcal{S} \setminus \{s\})^* \\ s \stackrel{\circ}{=} \mathbf{s}'}} |\mathbf{s}'|. \tag{65}$$

Thus, under this local replacement calculation, each previous occurrence of s contributes $|\mathbf{s}_s^{\text{rep}}|$ tokens instead of one token. The total contribution of these occurrences after deletion is

$$C_{\text{after}}(s) = n_s^{\mathcal{D}} |\mathbf{s}_s^{\text{rep}}|. \tag{66}$$

Therefore, the estimated increase in corpus length caused by locally replacing all current occurrences of s is

$$\begin{aligned} \Delta_s^{\mathfrak{G}_{\text{comp}}(\cdot,\mathcal{D})} &= C_{\text{after}}(s) - C_{\text{before}}(s) && (67) \\ &= n_s^{\mathcal{D}} |\mathbf{s}_s^{\text{rep}}| - n_s^{\mathcal{D}} && (68) \\ &= n_s^{\mathcal{D}} \left(|\mathbf{s}_s^{\text{rep}}| - 1\right). && (69) \end{aligned}$$

This proves the local deletion-cost formula. □

G Hyperparameters and Architecture

All language models follow a Llama-style decoder-only architecture. Tab. 5 summarises the per-size architectural settings. All models use SiLU activations, grouped-query attention with separate K/V head counts, RMSNorm, and rotary positional embeddings (RoPE, $\theta = 10{,}000$).

Optimisation. We train with AdamW using learning rate 3×10^{-4}, $\beta_1 = 0.9$, $\beta_2 = 0.95$, weight decay 0.1, and a maximum gradient norm of 1.0. The learning-rate schedule is warmup–stable–decay: 2,000 warmup steps, a stable phase, and 10,000 linear-decay steps to a final learning rate of 10% of peak.

Training setup. All models are trained at sequence length 2,048 in bf16-mixed precision with FlashAttention 2. The per-device batch size is 32, and gradient accumulation is adjusted per model

Size	Hidden	Intermediate	Heads	KV heads	Layers	Tied
100M	576	1,536	9	3	30	✓
300M	960	2,560	15	5	32	✓
500M	1280	3,456	16	4	26	✓
1B	2048	5,632	32	4	22	✓

Table 5: Per-size architecture for the Llama-style language models. "Tied" denotes weight tying between input and output embeddings.

size to match a fixed effective batch size. Following Chinchilla-optimal scaling, each model is trained on roughly $20\times$ its parameter count in tokens. Training uses a single fixed random seed (42), except for the 100M setting, where we train three seeds per configuration. All experiments are implemented in PyTorch with our open-source training stack.[12]

H Detailed Metric Definitions

This appendix gives precise definitions for every metric used in the paper. Let $\mathcal{D}$ denote the held-out corpus, N_{bytes} its size in UTF-8 bytes, and N_{tokens} the number of tokens produced by a given tokeniser on $\mathcal{D}$.

Extrinsic.

- **Bits per byte (BPB)** $= -\log_2 P_\theta(\mathcal{D})/N_{\text{bytes}}$: the trained model's negative log-likelihood on $\mathcal{D}$, normalised by byte count. Lower is better. Because the denominator is a property of the raw text (and hence identical across tokenisers), BPB is directly comparable across tokenisers; token-level perplexity, by contrast, is biased toward tokenisers that produce shorter sequences.
- **Minimal-pair grammatical accuracy.** For each acceptable / unacceptable sentence pair, we mark the model correct if it assigns higher per-token log-probability to the acceptable sentence. We use BLiMP (Warstadt et al., 2020) for English, MultiBLiMP (Jumelet et al., 2026) for English, German, Spanish, and Turkish, and ZhoBLiMP (Liu et al., 2025) for Chinese. Higher is better.

Intrinsic.

- **Segmentation efficiency.** *Compression objective* $\mathfrak{G}_{\text{comp}} = N_{\text{tokens}}$ is the corpus token count (lower = more compact). *Bytes per token* (BPT) $= N_{\text{bytes}}/N_{\text{tokens}}$ is the average number of source bytes captured by one token (higher = more compact). *Fertility* (when reported) is tokens per word, with word boundaries from a pre-tokeniser.
- **Vocabulary usage.** *Vocab Util* is the fraction of vocabulary IDs that appear at least once in $\mathcal{D}$.
- **Vocabulary overlap.** For two tokenisers A and B trained at the same vocabulary size, the overlap is $|\mathcal{S}_A \cap \mathcal{S}_B|/|\mathcal{S}_A|$, where $\mathcal{S}_A$ and $\mathcal{S}_B$ are their learned vocabularies as sets of token strings. Symmetric since vocabularies have the same size.
- **Token length.** Average, median, and maximum character length of the entries in the learned vocabulary (each entry counted once, unweighted by corpus frequency).
- **Distributional structure.** *Entropy* $H = -\sum_t \hat{p}(t)\log_2 \hat{p}(t)$ of the empirical token distribution, where $\hat{p}(t) = \text{count}(t)/N_{\text{tokens}}$. The corresponding *unigram log-likelihood objective* is $\mathfrak{G}_{\text{ll}} = H \cdot N_{\text{tokens}}$ (lower = better fit under the empirical unigram model). *Zipf* α is the negated slope of a log-log OLS fit of frequency vs. rank, with tokens sorted in descending frequency; higher α means a steeper distribution. *Coverage 50%* is the smallest ℓ such that the ℓ most frequent tokens together account for at least half of all token occurrences in $\mathcal{D}$; smaller ℓ means usage is concentrated on a small core.

I Merge Dynamics

We inspect the merge trajectories of the bottom-up methods. This analysis is defined only for `BPE` and `BottomUpLL`. A token is counted as **killed** if it is created by one merge and later used inside another

[12] Code and configurations will be released upon publication.

merge. For example, if a tokeniser first creates physi and later merges it into physic, then physi is killed. Killed tokens are not necessarily unused in the final tokeniser, but they indicate how much of the merge trajectory is spent on intermediate construction steps.

Tab. 6 shows that BottomUpLL uses more intermediate merge steps than BPE. Its kill ratio is higher overall and remains higher when restricting to longer tokens. This suggests that likelihood-based merging more often builds tokens through intermediate fragments that are later absorbed into larger units.

The usage statistics qualify this pattern. Killed tokens are not necessarily wasted: most killed tokens are still used at least once in the held-out corpus. However, BottomUpLL has a larger fraction of killed tokens with zero or very low usage than BPE. Thus, BottomUpLL allocates more of its merge budget to intermediate tokens that help construct longer units but are less likely to remain frequent standalone tokens.

Overall, the merge dynamics explain why BottomUpLL can be close to BPE in compression while still learning a different vocabulary. It builds longer lexical units through more intermediate stages, which is consistent with the longer token lengths in Tab. 2. The trade-off is that this trajectory creates more low-usage intermediate vocabulary items.

J English vs. Multilingual Intrinsic Comparison

This appendix reports the multilingual counterpart of Tab. 1 and Tab. 2, using the multilingual tokenisers and the concatenation of the five per-language test splits (220,423 documents: 47,384 English, 54,969 German, 54,489 Spanish, 41,986 Turkish, 21,595 Chinese). The comparison crosses both the tokeniser-training distribution and the evaluation corpus, so it is not a controlled ablation; we include it to check which intrinsic patterns persist when the vocabulary must serve multiple languages.

Compression gaps widen in the multilingual setting. Tab. 7 restates total tokens with relative overhead against BPE. In English, UnigramLM produces 18.5% more tokens than BPE and TopDownComp 7.4% more. In multilingual, the top-down gap roughly doubles: UnigramLM +39.6%, TopDownComp +26.5%. BottomUpLL remains close to BPE in both settings (+0.5% in English, +0.9% in multilingual). The bottom-up vs. top-down separation observed in the main text therefore grows under a shared multilingual vocabulary budget.

Distributional shape changes with the evaluation corpus. Tab. 8 reports the full intrinsic statistics in both settings. All four token distributions flatten in the multilingual setting: Zipf α values decrease, entropies increase, and Coverage 50% grows by an order of magnitude (e.g., BPE: 204 $\rightarrow$ 1,483), reflecting the broader set of frequent token types in a five-language corpus. The relative pattern between objectives is preserved, however: UnigramLM retains the steepest Zipf α and lowest entropy in both settings, and likelihood-based methods stay more concentrated than their compression-based counterparts within each search family.

Vocabulary utilisation rises with corpus breadth. The unused-vocabulary tail visible in English (Vocab Util 96.8–99.8%) largely disappears in the multilingual evaluation (99.36–99.94%), because the broader, more diverse corpus exercises a larger portion of the 128k vocabulary.

Vocabulary overlap follows the same cluster structure. Tab. 9 is the multilingual counterpart of Tab. 2. BPE and BottomUpLL remain the most similar pair (80.5% overlap); TopDownComp and UnigramLM form the second-closest cluster (63.6%); cross-family overlap stays in the 30–37% range.

K Qualitative Examples

Aggregate metrics show the dominant trends, but they hide the local segmentation decisions that produce those trends. We therefore inspect a small set of representative examples covering plain English, scientific terminology, code-like text, URLs, Chinese, and multilingual accented text. These examples are illustrative rather than statistically decisive, but they make the inductive biases of the methods easier to interpret.

Tab. 10 reports the total number of tokens produced across the 10 illustrative examples. The ordering broadly matches the full compression benchmark: the bottom-up methods are shortest, TopDownComp is intermediate, and UnigramLM is longest. The purpose of this table is not to establish a new ranking, but to make the aggregate trends concrete and to motivate the example-level differences below.

Metric	BPE	BottomUpLL (Exact)	BottomUpLL (Approx)
Total killed	34,224	41,599	41,751
Kill ratio (all)	26.8%	32.6%	32.7%
Kill ratio ($\lvert s\rvert \geq 5$)	21.7%	27.8%	27.9%
Kill ratio ($\lvert s\rvert \geq 10$)	4.5%	8.5%	8.8%
Killed with zero usage	3.5%	9.2%	8.7%
Killed with low usage (≤ 10)	26.2%	36.9%	36.0%
Killed zero % of total vocab	.9%	3.0%	2.8%
Survived with zero usage	.2%	.4%	.4%

Table 6: Summary of merge-dynamics and killed-token usage statistics for the bottom-up methods. A token is killed if it is created by one merge and later used inside another merge. Low usage means at most 10 occurrences on the held-out test set.

Setting	Tokeniser	Total tokens ↓	vs. BPE
English	BPE	**46,859,565**	baseline
	TopDownComp	50,312,483	+7.37%
	BottomUpLL	47,111,733	+0.54%
	UnigramLM	55,546,423	+18.54%
Multilingual	BPE	**172,372,100**	baseline
	TopDownComp	217,983,502	+26.46%
	BottomUpLL	173,917,215	+0.90%
	UnigramLM	240,669,745	+39.62%

Table 7: Total tokens produced on each evaluation corpus, with relative overhead reported against the corresponding BPE tokeniser. The English rows use English-only tokenisers evaluated on the English held-out test set. The multilingual rows use multilingual tokenisers evaluated on the union of the five per-language held-out test sets.

Scientific and medical terminology. On examples such as `Photosynthesis converts carbon dioxide and water into glucose and oxygen.` and `Electroencephalography is used to diagnose neurological disorders.`, BottomUpLL often produces the shortest segmentation. It tends to preserve long technical units such as `Photosynthesis` or fragments of `encephalography` as large tokens. BPE is also compact, but more often splits these words into high-frequency internal pieces. This is consistent with the merge-dynamics analysis: BottomUpLL is more willing to build long lexical chains through intermediate tokens.

Code and punctuation-heavy text. On code-like or SQL-like strings, BPE is usually strongest. For example, on `SELECT * FROM users WHERE id = 1; DROP TABLE users;-`, BPE produces the fewest tokens and preserves punctuation patterns such as `;-`. UnigramLM is much more fragmented, often splitting short code tokens such as `id`, `if`, or `def` into character-level pieces. This reflects the concentration of its token distribution: unless short code tokens are strongly supported by the learned vocabulary, the model falls back to smaller pieces.

URL-like text. For URL strings such as `https://www.example.com/path/to/resource?query=value&page=1`, BPE and TopDownComp are especially compact. BPE often preserves web-specific substrings such as `https`, while BottomUpLL may split them into statistically meaningful but less URL-specific pieces such as `http+s`. This is a case where frequency-driven compression appears better matched to the surface regularities of web text than likelihood-based lexical cohesion.

Chinese text. On the short Chinese example 人工智能正在改变世界, UnigramLM produces the fewest tokens among the inspected examples. This does not contradict the multilingual BPB results in Tab. 4, where bottom-up methods are much better on Chinese overall. Rather, it shows that example-level token count and corpus-level language-model BPB can diverge: a tokeniser may preserve some non-Latin chunks well while still producing a worse distribution of tokens for language modelling across the full evaluation set.

Multilingual accented text. For `¡Hola! ¿Cómo estás? Très bien, merci. Danke schön!`, all methods produce similar total counts, but the internal boundaries differ. UnigramLM tends to preserve accented fragments such as `Cómo` and `estás` more coherently, while BPE more often splits them into smaller pieces. Thus, even when example-level

Setting	Tokeniser	Objective	Search	$\mathfrak{G}_{\text{comp}} \downarrow$	$\mathfrak{G}_{\text{ll}} \downarrow$	Entropy	Zipf α	BPT $\uparrow$	Tok Len	Vocab Util $\uparrow$	Coverage 50%
English	BPE	$\mathfrak{G}_{\text{comp}}$	bottom-up	**46,859,565**	5.216×10^8	11.132	1.484	**4.886**	7.06	98.9%	204
	TopDownComp	$\mathfrak{G}_{\text{comp}}$	top-down	50,312,483	5.484×10^8	10.900	1.443	4.551	6.01	**99.8%**	149
	BottomUpLL	$\mathfrak{G}_{\text{ll}}$	bottom-up	47,111,733	$\mathbf{5.203 \times 10^8}$	11.044	1.574	4.860	7.48	96.8%	191
	UnigramLM	$\mathfrak{G}_{\text{ll}}$	top-down	55,546,423	5.552×10^8	9.996	1.717	4.122	6.46	97.1%	53
Multilingual	BPE	$\mathfrak{G}_{\text{comp}}$	bottom-up	**172,372,100**	2.262×10^9	13.122	1.247	**4.675**	6.96	99.92%	1,483
	TopDownComp	$\mathfrak{G}_{\text{comp}}$	top-down	217,983,502	2.678×10^9	12.286	1.266	3.696	5.52	**99.94%**	618
	BottomUpLL	$\mathfrak{G}_{\text{ll}}$	bottom-up	173,917,215	$\mathbf{2.245 \times 10^9}$	12.907	1.347	4.633	7.40	99.36%	1,114
	UnigramLM	$\mathfrak{G}_{\text{ll}}$	top-down	240,669,745	2.597×10^9	10.791	1.598	3.348	6.79	99.74%	139

Table 8: Intrinsic statistics in English-only and multilingual settings, using the same column set as Tab. 1. All values are computed at 128k vocabulary size. The multilingual evaluation corpus is the union of the five per-language test splits (220,423 documents, 805.8 MB).

	BPE	TopDownComp	BottomUpLL	UnigramLM
BPE	—	37.0%	80.5%	32.3%
TopDownComp	37.0%	—	33.6%	63.6%
BottomUpLL	80.5%	33.6%	—	31.2%
UnigramLM	32.3%	63.6%	31.2%	—

Table 9: Multilingual pairwise vocabulary overlap at 128k vocabulary size; the multilingual counterpart of Tab. 2.

Method	Tokens $\downarrow$
BPE	156
TopDownComp	160
BottomUpLL (Exact)	154
BottomUpLL (Approx)	**153**
UnigramLM	173

Table 10: Total number of tokens produced across 10 illustrative test examples. Lower is better.

token counts are similar, tokenisers can impose different boundaries that may matter for downstream behaviour.

These examples illustrate the same qualitative biases seen in the aggregate analyses: BPE is robust on punctuation-heavy and web-like text, BottomUpLL tends to form longer lexical units, UnigramLM can preserve some accented or non-Latin fragments while fragmenting other domains, and TopDownComp behaves as a compression-oriented top-down method.