TOKENISATION VIA CONVEX RELAXATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Tokenisation is an integral part of the current NLP pipeline. Current tokenisation algorithms such as `BPE` and `UnigramLM` are greedy: they make locally optimal decisions without considering the resulting vocabulary as a whole. We instead formulate tokeniser construction as an integer program which we approximate using convex optimisation tools, yielding a new algorithm we call `ConvexTok`. We find that `ConvexTok` consistently improves intrinsic tokenisation metrics and the bits-per-byte (`BpB`) achieved by language models in English; downstream performance metrics and multilingual language modelling, however, show no consistent pattern. Furthermore, `ConvexTok` allows the user to upper bound how far any tokeniser is from optimal (at a certain objective), and we empirically find `ConvexTok` to be within 1% of optimal at common vocabulary sizes.

anonymised code in this link | tokenisers will be released

1 INTRODUCTION

Tokenisation is an integral part of every language modelling pipeline, taking sequences of **bytes** as input, and converting them into sequences of **tokens**, which serve as input to a language model (LM). Notably, while it is generally an open research question what makes a good tokeniser (Gowda and May, 2020; Cognetta et al., 2024b; Ali et al., 2024; Schmidt et al., 2024), a number of recent works show that a tokeniser's compression correlates (at least to some extent) with its downstream performance in LLMs (Gallé, 2019; Zouhar et al., 2023a). Given these results, we might expect researchers to test how LMs would perform given a compression-optimal tokeniser. Unfortunately, finding such optimal tokenisers is NP-hard (Kozma and Voderholzer, 2024; Whittington et al., 2025; Lim et al., 2025; Kastreva et al., 2026). Hence, in practice, approximate techniques are employed.

Currently, the de-facto standard tokenisation algorithm is byte-pair encoding (`BPE`; Gage, 1994; Sennrich et al., 2016), which originated as a compression algorithm. It is a simple greedy algorithm that iteratively merges the most frequent pair of tokens in a dataset into a new token until a desired vocabulary size is reached. As `BPE` greedily merges tokens, it may create locally optimal but globally suboptimal tokens, and compressive ability may thus be lost. Many approaches have been proposed as possible fixes to this problem (Cognetta et al., 2024a; Chizhov et al., 2024; Lian et al., 2025; Liu et al., 2025; Schmidt et al., 2025), but they all rely on relatively minor modifications or extensions to `BPE`, and do not consider optimising tokenisation beyond greedy solutions. In fact, a recent panel has deemed the design of fundamentally new tokenisation algorithms an open problem (Gu et al., 2025).

The core technical contribution of our paper is to show how to tackle tokenisation using linear programming. We first identify an integer program (IP) which is equivalent to the optimisation problem solved by tokenisation algorithms. Second, we relax this problem into a linear program (LP),[1] for which we can efficiently compute exact (or near-exact) solutions using a common solver. Such an LP solution, however, includes 'partial' or 'fractional' tokens, which need to be discretised before we can convert them into a functional tokeniser. To this end, we propose three simple rounding schemes. After rounding, constructing a tokeniser from this solution is trivial. We term our method `ConvexTok`, as it produces tokenisers via a convex relaxation of the tokenisation problem. Additionally, solving the LP provides a lower bound on the compression achieved by any tokeniser on the used dataset. Our method thus allows us to bound how close to optimal any tokeniser is.

Empirically, we evaluate our proposed tokeniser's performance in five steps. First, we analyse properties of our constructed LP and the effect of different rounding techniques on its solutions' quality. We

[1] Integer programming optimises problems over discrete sets, while linear programming operates over continuous sets. If an integer program requires a variable $x \in \{0, 1\}$, we can relax it into a linear program with $x \in [0, 1]$.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

see that even though the tokenisation problem is NP-hard, the LP's solution is not far from integral (especially at larger vocabulary sizes). Second, we use our LP's solution to certify how close to optimal various tokenisers (`ConvexTok`, `BPE`, `UnigramLM`, `Freq` and `PathPiece`) are at compression. We see that, at common vocabulary sizes, several of the tokenisers we consider are already within 1% of optimal. Third, we study how stable tokenisers are to a specific choice of training dataset, finding that `BPE` is consistently more stable than `ConvexTok`. Fourth, we evaluate our tokenisers using common intrinsic metrics (e.g., compression rate, vocabulary utilisation, and Rényi entropy). In English, `ConvexTok` outperformed all baselines in most metrics; on a multilingual dataset, however, `BPE` or `PathPiece` often performed best. Finally, we evaluate the effect of our tokenisers on downstream language modelling performance. In these experiments and again in English, a deterministic rounding scheme (`Det`) consistently performed best on bits-per-byte (`BpB`); on downstream tasks (`CORE`; Li et al., 2024) or the multilingual dataset, however, no clear pattern emerged.

Overall, LP-based tokenisation offers a flexibility not present in previous work, allowing researchers to more easily: (i) change the set of allowed tokens, (ii) change the optimised objective (at least within convex or concave alternatives), and (iii) bound the tokeniser's quality. Currently, however, it suffers from a larger memory footprint and lower stability than `BPE`. We hope that future work will improve on these aspects of `ConvexTok`, and that its flexibility will enable future work to explore a large family of new tokenisation algorithms.

2 Tokenisation

We start with some preliminary definitions. First, let Σ be some alphabet, and $\mathbf{b} \in \Sigma^*$ a **byte-string**,[2] which we can expand as $\mathbf{b} = b_1 b_2 \cdots b_{|\mathbf{b}|}$. Second, let a dataset be a multi-set of byte-strings, denoted by $\mathcal{D} = \{\mathbf{b}^{(n)}\}_{n=1}^{N}$. A tokeniser's job is to *segment* these byte-strings into substrings, which will correspond to the tokens used as input for our model. As LMs require a finite set of tokens to compose their vocabularies, we introduce this constraint directly into the tokenisation step, defining the tokeniser's **vocabulary** $\mathcal{T}$ as a finite set of byte-substrings, i.e., $\mathcal{T} \subset \Sigma^+$ with $|\mathcal{T}| < \infty$. Further, to guarantee that any byte-string has at least one valid segmentation, we also require all elements of the alphabet to be contained in the vocabulary, or formally, $\Sigma \subseteq \mathcal{T}$. In practice, we then often fix the size of the vocabulary given a **budget** K to be $|\mathcal{T}| = |\Sigma| + K$. Finally, we define a **token-string** as $\mathbf{t} \in \mathcal{T}^*$.

We can now define a tokeniser as the 3-tuple $\mathbb{T} \stackrel{\text{def}}{=} \langle \mathcal{T}, \texttt{tok}, \texttt{detok} \rangle$, where $\texttt{tok}\colon \Sigma^* \to \mathcal{T}^*$ is an **encoding function**, which segments byte-strings into token-strings, and $\texttt{detok}\colon \mathcal{T}^* \to \Sigma^*$ is a **decoding function**, which maps token-strings back to byte-strings. As we say an encoding function *segments* a byte-string, we require that, whenever $\mathbf{t} = \texttt{tok}(\mathbf{b})$, we have that $\mathbf{b} = t_1 \circ t_2 \circ \cdots \circ t_{|\mathbf{t}|}$. Further, the decoding function simply undoes the encoding mapping, being thus defined as $\texttt{detok}(\mathbf{t}) \stackrel{\text{def}}{=} t_1 \circ t_2 \circ \cdots \circ t_{|\mathbf{t}|}$. Notably, for a fixed vocabulary, many different encoding functions may successfully segment a byte-string; e.g., given the vocabulary $\mathcal{T} = \{d, o, g, do, og\}$, we could segment byte-string dog as either $\texttt{tok}(dog) = \langle do, g \rangle$, $\texttt{tok}(dog) = \langle d, og \rangle$, or $\texttt{tok}(dog) = \langle d, o, g \rangle$. A tokeniser $\mathbb{T}$ thus represents both a choice of vocabulary ($\mathcal{T}$) and of segmentation strategy ($\texttt{tok}$).

2.1 What Do We Want from Our Tokeniser?

Given the description above, we are left with a question: how should we select a tokeniser? If f is an objective function to be minimised, which, given a tokeniser, returns a score associated with it, we would ideally choose a tokeniser by computing a solution to the optimisation problem: $\operatorname{argmin}_{\mathbb{T}} f(\mathbb{T})$. Unfortunately, there are two issues with this approach: (i) it is not obvious which objective function to use, and (ii) for common objectives, we don't know how to solve this optimisation problem efficiently.

Several objective functions have been proposed in response to (i), including compression (Gallé, 2019), unigram log-likelihood (Kudo, 2018), or Rényi efficiency (Zouhar et al., 2023a). While we believe that more research should go into what optimisation function to use, we will focus on compression here, following a battery of prior work either proposing new tokenisers (Cognetta et al., 2024a; Chizhov et al., 2024; Lian et al., 2025) or theoretically analysing them (Zouhar et al., 2023b). Given a dataset ($\mathcal{D}$), we write our **compression objective function** as

$$f(\mathbb{T}) = \sum_{\mathbf{b} \in \mathcal{D}} \texttt{length}\big(\texttt{tok}(\mathbf{b})\big) \;, \tag{1}$$

where we note that the encoding function ($\texttt{tok}$) implicitly depends on the tokeniser's vocabulary ($\mathcal{T}$).

[2] Formally, Σ^* denotes the Kleene star of Σ (i.e., $\cup_{i=0}^{\infty} \Sigma^i$), and Σ^+ denotes its Kleene plus (i.e., $\cup_{i=1}^{\infty} \Sigma^i$).

Even with an objective in hand, challenge (ii) remains: finding a tokeniser that optimises it, and doing so efficiently. Unfortunately, as mentioned above, this optimisation problem is NP-hard, meaning that we cannot solve it efficiently, unless P = NP. We thus focus on approximations instead.[3] The most popular such method is **byte-pair encoding** (BPE), a greedy algorithm used in most modern LLMs (OpenAI, 2023; Touvron et al., 2023; Biderman et al., 2023; Team Olmo et al., 2026). Notably, BPE is a greedy algorithm which iteratively chooses locally optimal tokens to be merged one at a time; in general, however, these tokens may not be optimal globally, which may lead to the choice of a suboptimal vocabulary. (See App. A for a more formal description of BPE.) As a small example, consider the dataset $\mathcal{D} = \{abc, abd, abe, bc, bd, be\}$ and $K = 3$. BPE would first choose to merge $\langle a \rangle$ and $\langle b \rangle$ into a new token $\langle ab \rangle$ for a saving of 3 symbols, and any further merge can save only 1 symbol, so with two more arbitrary merges, BPE leads to 10 used tokens. It would however be optimal to choose the tokens $\langle bc \rangle, \langle bd \rangle, \langle be \rangle$ instead to save 2 symbols three times, yielding a total of 9 tokens.

3 Tokenisation via Convex Relaxations

Our paper's main contribution is an LP-based tokenisation algorithm which directly *approximates* a globally optimal tokeniser; thus avoiding locally optimal solutions. To make our LP more intuitive, however, we first express the problem as a graph problem: given a dataset ($\mathcal{D}$), we construct a directed acyclic graph whose shortest-path solutions can be easily translated into tokenisers; we then use this graph problem to introduce the LP.

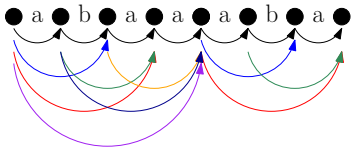


Figure 1: Tokenisation graph constructed from $\mathcal{D} = \{abaa, aba\}$. Black edges (the shortest) represent $\mathcal{E}_{\texttt{free}}$; others represent $\mathcal{E}_{\texttt{pric}}$.

We start by defining the graph. For each byte-string in our dataset, we introduce a vertex for every position in between two bytes in the string; we also introduce a vertex for the position before the first and after the last byte in the string. A string of length n thus contributes $n + 1$ vertices, with the first and last vertices designated as its start and end vertices, respectively. We then merge the last vertex of each byte-string with the first vertex of the next, essentially making them a single vertex. Next, we connect each adjacent vertex within a byte-string with what we call a **free-edge**. Finally, we connect each non-adjacent vertex within byte-strings with a **priced-edge**; this is denoted formally as:

$$\mathcal{V} \stackrel{\text{def}}{=} \{v_i^n \mid \mathbf{b}^{(n)} \in \mathcal{D}, 0 \le i \le \texttt{length}(\mathbf{b}^{(n)})\}, \qquad \forall_{\mathbf{b}^{(n)} \in \mathcal{D}} : v^n_{\texttt{length}(\mathbf{b}^{(n)})} = v_0^{n+1} \tag{2a}$$

$$\mathcal{E}_{\texttt{free}} \stackrel{\text{def}}{=} \{\langle v_i^n, v_{i+1}^n \rangle \mid \mathbf{b}^{(n)} \in \mathcal{D}, 0 \le i < \texttt{length}(\mathbf{b}^{(n)})\} \tag{2b}$$

$$\mathcal{E}_{\texttt{pric}} \stackrel{\text{def}}{=} \{\langle v_i^n, v_j^n \rangle \mid \mathbf{b}^{(n)} \in \mathcal{D}, 0 \le i < \texttt{length}(\mathbf{b}^{(n)}), i+2 \le j \le \texttt{length}(\mathbf{b}^{(n)})\} \tag{2c}$$

The set of edges in our graph is then defined as the union of free- and priced-edges: $\mathcal{E} = \mathcal{E}_{\texttt{free}} \cup \mathcal{E}_{\texttt{pric}}$. Notably, free-edges correspond to individual bytes, while priced-edges represent potential tokens that can be used to represent an entire substring $\mathbf{b}_{ij}^{(n)}$. We then colour priced-edges based on the substrings they represent. Formally, let the set of **potential tokens** in dataset $\mathcal{D}$ be denoted by $\mathcal{T}_{\texttt{all}}$. We can then define a **colour-partition** $\mathcal{C}$ as a *partition* of the priced-edges $\mathcal{E}_{\texttt{pric}}$ based on the bytes they represent:

$$\mathcal{T}_{\texttt{all}} \stackrel{\text{def}}{=} \{\mathbf{b}_{ij}^{(n)} \mid \mathbf{b}^{(n)} \in \mathcal{D}, 0 \le i < \texttt{length}(\mathbf{b}^{(n)}), i+2 \le j \le \texttt{length}(\mathbf{b}^{(n)})\} \tag{3a}$$

$$\mathcal{C} \stackrel{\text{def}}{=} \left\{ \underbrace{\{\langle v_i^n, v_j^n \rangle \mid \langle v_i^n, v_j^n \rangle \in \mathcal{E}_{\texttt{pric}}, \mathbf{b}_{ij}^{(n)} = \mathbf{b}'\}}_{\text{edges representing token } \mathbf{b}'} \;\middle|\; \mathbf{b}' \in \mathcal{T}_{\texttt{all}} \right\} \tag{3b}$$

This is exemplified in Fig. 1. Note that each set $c \in \mathcal{C}$ represents a potential token, containing all edges with a certain colour. Further, note that $\mathcal{C}$ partitions $\mathcal{E}_{\texttt{pric}}$; we thus have that $\cup_{c \in \mathcal{C}} c = \mathcal{E}_{\texttt{pric}}$ and that the sets of edges $c \in \mathcal{C}$ are disjoint. We can now recast tokenisation as a graph problem.

Definition 3.1. *We define a **tokenisation graph** to be the 3-tuple $\langle \mathcal{V}, \mathcal{E}, \mathcal{C} \rangle$. Further, we say that v_0^1 and v_L^N are, respectively, its start and end vertices, where $L = \texttt{length}(\mathbf{b}^N)$. Finally, given a budget K, we define a **graph-vocabulary** as a choice of K colours from $\mathcal{C}$, and a **graph-segmentation** as a path from v_0^1 to v_L^N using only free-edges, or priced-edges from these K colours.*

Note that, by construction, there is a one-to-one correspondence between a graph-vocabulary and a (traditional) vocabulary $\mathcal{T}$ with budget K, as the tokens in a traditional vocabulary have a direct

[3] Comparing BPE and UnigramLM, a recent paper (Yavuz et al., 2026) shows the approximation method used to find a tokeniser has a stronger impact on language modelling performance than the objective function.

mapping from the colours in a graph-vocabulary. There is also a one-to-one correspondence between a graph-segmentation and the segmentation of a dataset $\mathcal{D}$ by a tokeniser $\mathbb{T}$; we can thus build an encoding function which segments each byte-string in $\mathcal{D}$ similarly to the graph. Given a graph-vocabulary and graph-segmentation, we can thus construct an equivalent tokeniser $\mathbb{T}$.[4] Notably, the compression achieved by this tokeniser will be identical to the length of the graph-segmentation. Given this relationship between a tokeniser's compression and a graph-segmentation's path, we now define a graph problem equivalent to the problem of finding compression-optimal tokenisers.

Definition 3.2. *Consider a tokenisation graph $\langle \mathcal{V}, \mathcal{E}, \mathcal{C} \rangle$ and a budget K. The **shortest tokenisation problem** is to find a graph-segmentation in $(\mathcal{V}, \mathcal{E})$ with shortest path from v_0^1 to v_L^N using a graph-vocabulary of at most K colours.*[5]

Generalising the Tokenisation Problem. The construction above creates a tokenisation graph that is equivalent to a typical tokenisation problem. We can mildly generalise this construction to allow for more flexibility in the tokeniser's optimisation. In particular, starting from the set of vertices $\mathcal{V}$ and edges $\mathcal{E}$ as constructed above (in Eq. (2)), we could select any subset of $\mathcal{E}$ to constitute free edges, expanding it beyond only bytes. These edges would then represent tokens which must always be included in a tokeniser's vocabulary. Similarly, we could freely define the set of priced edges as any set $\mathcal{E}_{\texttt{pric}} \subseteq \mathcal{E} \setminus \mathcal{E}_{\texttt{free}}$; these edges would represent the tokens which can be added to the vocabulary. In the description of $\mathcal{E}_{\texttt{pric}}$ above (in Eq. (2c)), we have a graph with edges representing all possible byte-sequences in the dataset. However, it could be reasonable to consider only edges in $\mathcal{E}_{\texttt{pric}}$ that do not cross unicode, grapheme, or morpheme boundaries (analogously to `BoundlessBPE`; Schmidt et al., 2025), or that represent tokens with non-negligible frequencies.

3.1 Tokenisation as an Integer Program

The preceding section builds a graph problem which is, in a sense, equivalent to the problem of finding compression-optimal tokenisers. This graph problem, however, is not necessarily easier to solve than the original problem over strings. We now leverage it to build yet another equivalent problem using integer programming. While the resulting IP is still NP-hard, we will relax it into an LP which we can solve efficiently. Later we will rely on rounding schemes to recover an approximately globally optimal tokeniser. Notably, relaxing the integral constraints to continuous ones is a design choice we take here. Please see App. B for an alternative approach. Furthermore, we note that linear programming has a long history of use in approximation algorithms for combinatorial problems (Williamson and Shmoys, 2011; Vazirani, 2010; Shmoys and Tardos, 1993; Papadimitriou and Steiglitz, 1982).

Consider a tokenisation graph $\langle \mathcal{V}, \mathcal{E}, \mathcal{C} \rangle$ with $\mathcal{E}_{\texttt{free}}$ and $\mathcal{E}_{\texttt{pric}}$. To convert it into an IP, we first define a **free incidence matrix**[6] $\mathbf{F} \in \{-1, 0, 1\}^{\mathcal{V} \times \mathcal{E}_{\texttt{free}}}$ and a **priced incidence matrix** $\mathbf{P} \in \{-1, 0, 1\}^{\mathcal{V} \times \mathcal{E}_{\texttt{pric}}}$ to have elements $\mathbf{F}_{v,e}$ and $\mathbf{P}_{v,e}$ which encode whether an edge ($e \in \mathcal{E}_{\texttt{free}}$ or $e \in \mathcal{E}_{\texttt{pric}}$, respectively) starts or ends at some vertex ($v \in \mathcal{V}$). Second, we define an **edge colour matrix** $\mathbf{C} \in \{0, 1\}^{\mathcal{E}_{\texttt{pric}} \times \mathcal{C}}$ to have elements $\mathbf{C}_{e,c}$ encoding whether a priced edge $e \in \mathcal{E}_{\texttt{pric}}$ has a specific colour $c \in \mathcal{C}$. Formally:

$$\mathbf{F}_{v,e} \overset{\text{def}}{=} \begin{cases} -1 & \texttt{if } e = \langle v, v' \rangle \\ 1 & \texttt{elif } e = \langle v', v \rangle, \\ 0 & \texttt{else} \end{cases} \quad \mathbf{P}_{v,e} \overset{\text{def}}{=} \begin{cases} -1 & \texttt{if } e = \langle v, v' \rangle \\ 1 & \texttt{elif } e = \langle v', v \rangle, \\ 0 & \texttt{else} \end{cases} \quad \mathbf{C}_{e,c} \overset{\text{def}}{=} \begin{cases} 1 & \texttt{if } e \in c \\ 0 & \texttt{else} \end{cases} \tag{4}$$

Recall that each colour $c \in \mathcal{C}$ represents a potential token; $\mathbf{C}_{e,c} = 1$ then means that an edge e represents an instance of c's token. Notably, these matrices represent the structural constraints of a tokenisation problem. We now define three vectors which we will use to define a specific tokeniser: (i) a **free token-instance vector** $\mathbf{f} \in \{0, 1\}^{\mathcal{E}_{\texttt{free}}}$; (ii) a **priced token-instance vector** $\mathbf{p} \in \{0, 1\}^{\mathcal{E}_{\texttt{pric}}}$; and (iii) a **priced colour vector** $\mathbf{c} \in \{0, 1\}^{\mathcal{C}}$. These vectors represent, respectively, which free and priced edges are being used to segment a tokenisation-graph (forming, together, a graph-segmentation) and which tokens are selected as colours (forming a graph-vocabulary). As we can choose at most K colours, we introduce the constraint $\langle 1, \mathbf{c} \rangle \leq K$. Further, a priced token-instance can only be used if its priced colour is a part of the vocabulary, motivating the constraint $\mathbf{p} - \mathbf{C}\mathbf{c} \leq 0$. Finally, we introduce flow-constraints (from Ford and Fulkerson, 1962; Dantzig, 1960) to guarantee that the solution

[4] For completeness, we also define the encoding function to encode byte-strings which are not in $\mathcal{D}$ using PathPiece (Schmidt et al., 2024), which is compression-optimal for a fixed vocabulary.

[5] Related variants of Definitions 3.1 and 3.2 have been studied before, as connectivity problems of labelled graphs, or coloured graphs (Broersma et al., 2005; Hassin et al., 2006; Zhang et al., 2011; Ghaffari et al., 2017).

[6] We follow the convention that when a matrix is defined as $\{0, 1\}^{\mathcal{V} \times \mathcal{E}}$ we index it using elements of $\mathcal{V}$ and $\mathcal{E}$.

$\mathbf{p}, \mathbf{f}$ forms a valid path (or segmentation) over the graph. To this end, we define a **flow-difference** vector $\mathbf{d} \in \{-1, 0, 1\}^{\mathcal{V}}$ which is defined as $\mathbf{d}_{v_0^1} = -1$, $\mathbf{d}_{v_L^N} = 1$ for the starting and ending vertices, and as $\mathbf{d}_v = 0$ for other vertices. This vector can then be used to guarantee that the input-output flow difference in all vertices is zero, except for the starting vertex (with a -1 flow difference) and the ending vertex (with a $+1$ flow difference): $\mathbf{P}\mathbf{p} + \mathbf{F}\mathbf{f} = \mathbf{d}$. Together, these constraints enforce that any choice of $\mathbf{f}, \mathbf{p}, \mathbf{c}$ corresponds to a valid graph-segmentation and graph-vocabulary. The length of this graph-segmentation can then be measured as $\langle 1, \mathbf{p}\rangle + \langle 1, \mathbf{f}\rangle$. We can now define the IP as:

$$\begin{aligned} &\min\ \langle 1, \mathbf{p}\rangle + \langle 1, \mathbf{f}\rangle \\ &\text{s.t.}\quad \mathbf{f}, \mathbf{p}, \mathbf{c} \in \mathcal{Q}^{\texttt{IP}}, \end{aligned} \quad \texttt{where}\ \mathcal{Q}^{\texttt{IP}} \stackrel{\texttt{def}}{=} \left\{ \begin{aligned} \mathbf{f} &\in \{0,1\}^{\mathcal{E}_{\texttt{free}}} \quad \mathbf{P}\mathbf{p} + \mathbf{F}\mathbf{f} = \mathbf{d} \quad \text{\# valid segmentation} \\ \mathbf{p} &\in \{0,1\}^{\mathcal{E}_{\texttt{pric}}}: \quad \mathbf{p} - \mathbf{C}\mathbf{c} \leq 0 \quad \text{\# token in vocabulary} \\ \mathbf{c} &\in \{0,1\}^{\mathcal{C}} \qquad\quad \langle 1, \mathbf{c}\rangle \leq K \quad \text{\# vocabulary budget} \end{aligned} \right\} \tag{5}$$

Note that, as mentioned above, any choice of $\mathbf{f}, \mathbf{p}, \mathbf{c} \in \mathcal{Q}^{\texttt{IP}}$ corresponds to a valid graph-vocabulary and graph-segmentation, which, in turn, correspond to a valid tokeniser.

Observation 3.3. *Let $\mathbf{f}, \mathbf{p}, \mathbf{c}$ be any element of $\mathcal{Q}^{IP}$ (as defined in Eq. (5)). We can convert this instance into a tokeniser $\mathbb{T}$ with compression $f(\mathbb{T}) = \langle 1, \mathbf{p}\rangle + \langle 1, \mathbf{f}\rangle$.*

Observation 3.4. *Let $\mathbb{T}$ be any valid tokeniser with budget K. We can convert this tokeniser into an instance of $\mathbf{f}, \mathbf{p}, \mathbf{c} \in \mathcal{Q}^{IP}$ with path length $\langle 1, \mathbf{p}\rangle + \langle 1, \mathbf{f}\rangle = f(\mathbb{T})$.*

These observations guarantee that all elements in the space of solutions $\mathcal{Q}^{\texttt{IP}}$ correspond to valid tokenisers with equivalent compressions, and that all valid tokenisers correspond to elements in the non-relaxed polytope $\mathcal{Q}^{\texttt{IP}}$ with equivalent path-lengths. Please see App. C for an explicit mapping between the integer program and token sequences.

3.2 Approximating the Integer Program via Convex Relaxations

We now relax the IP above into an LP which can be solved efficiently. To this end, we simply relax all variables $\mathbf{f}, \mathbf{p}, \mathbf{c}$ in $\mathcal{Q}^{\texttt{IP}}$ from being constrained to the discrete space $\{0, 1\}$ to spanning the continuous $[0, 1]$ interval instead. We write $\mathcal{Q}$ for this relaxed set, which we term the **tokenisation polytope**.[7] In this relaxed version, the priced colour vector, for instance, becomes $\mathbf{c} \in [0, 1]^{\mathcal{C}}$, allowing for partial tokens to be selected. Similarly, the priced token-instance vector allows for edges to be partially included in the solution. Notably, optimising the objective $\langle 1, \mathbf{p}\rangle + \langle 1, \mathbf{f}\rangle$ over $\mathcal{Q}$ can be done efficiently.

A tokeniser's vocabulary, however, is a discrete set; thus, we cannot straightforwardly convert instances $\mathbf{f}, \mathbf{p}, \mathbf{c} \in \mathcal{Q}$ of this LP that involve fractional variables into a tokeniser. Hence, we will need to discretise the output of this LP, a process typically known as rounding. We consider three natural rounding schemes here, noting that this is far from an exhaustive list. **Deterministic rounding (`Det`)** rounds the K colours in $\mathbf{c}$ with the largest value to 1, and sets others to zero. **Biased rounding (`Bias`)** instead ranks each colour in $\mathbf{c}$ by its value divided by the length of the token it represents ($\mathbf{c}/\texttt{length}(c)$, with a slight abuse of notation to let $\texttt{length}(c)$ denote the length of the corresponding token), and then rounds these values (to 1 or 0) as before. This biases the selection towards shorter tokens when LP scores are comparable, and is motivated by the intuition that shorter tokens are more likely to generalise and occur outside the training set $\mathcal{D}$. Finally, **integral-only rounding (`Int`)** keeps only the colours $\mathbf{c}$ whose value is already essentially one (while allowing for numerical imprecisions), setting others to zero. This can be interpreted as selecting the tokens that the LP relaxation treats as forced, but it will typically return a vocabulary with fewer tokens than the allowed budget K. These rounding schemes are depicted in Fig. 7 in App. D. Given a (rounded) choice of $\mathbf{c}$, it is then trivial to compute optimal discrete values of $\mathbf{p}$ and $\mathbf{f}$ using a shortest path algorithm with the allowed colours.

3.3 Generalising the Linear Program to Other Objectives

The choice to optimise compression led to a shortest path problem and hence the linear objective used above: $\langle 1, \mathbf{p}\rangle + \langle 1, \mathbf{f}\rangle$. Our LP-based formulation, however, allows us to easily change this objective, and combine different objectives. To showcase this ability, we extend it to consider the morphological alignment between the selected tokens and a language's morphology. Let, for instance, an edge $e \in \mathcal{E}_{\texttt{pric}}$ be said to cross a morpheme boundary if the represented bytes $\mathbf{b}_{ij}^{(n)}$ encode two partial morphemes. For instance, consider string unhappiness with morpheme segmentation

[7] A polytope is defined as the finite closed intersection of halfspaces.

un|happi|ness; we say an edge crosses a boundary if it spans multiple morphemes and at least one of them only partially (e.g., unhapp), but not if it encodes full morphemes (e.g., happi or happiness) or a single partial morpheme (e.g., hap).[8] Similarly, we say an edge encodes partial UTF-8 symbols if the represented bytes $\mathbf{b}_{ij}^{(n)}$ do not cover full symbols. E.g., the symbol € is encoded in UTF-8 as E2 82 AC; an edge containing only 82 AC is thus partial. We then define a morphologically-biased compression objective as:

$$\begin{array}{ll}\min & \langle 1, \mathbf{p}\rangle + \langle 1, \mathbf{f}\rangle + \langle \mathbf{m}, \mathbf{p}\rangle \\ \text{s.t.} & \mathbf{f}, \mathbf{p}, \mathbf{c} \in \mathcal{Q}\end{array}, \quad \text{where } \mathbf{m}_e \stackrel{\text{def}}{=} \begin{cases} \lambda_1 & \texttt{if } e \text{ crosses a morpheme boundary} \\ \lambda_2 & \texttt{elif } e \text{ encodes partial UTF-8} \\ 0 & \texttt{else} \end{cases} \tag{6}$$

where $\mathbf{m} \in \mathbb{R}^{\mathcal{E}_{\text{pric}}}$ encodes a morphology-aware penalty term and $\lambda_1, \lambda_2 > 0$ are hyperparameters. We set both $\lambda_1 = \lambda_2 = 0.5$ in our experiments and note that, if either is greater than 1, simple byte-level tokenisation may achieve better objective values than some short tokens.

Notably, linear programming is an instance of a wider class called convex programming (Nesterov and Nemirovskii, 1994), meaning that one could optimise linear, minimise convex, or maximise concave objectives with this strategy. This allows us to (at least approximately) optimise several metrics previously proposed to suggest good tokenisers, such as Rényi entropy, multilingual Gini, or a code's identifier fragmentation (Meister, 2026).

4 Experimental Setup

Data. Most of our experiments focus on English, for which we used ClimbMix400B data (Karpathy, 2025). For our multilingual experiments, we use FineWeb and FineWeb2 (Penedo et al., 2024; 2025). We always use the same datasets to train tokenisers and language models. As is common in tokenisation, the text is first pretokenised using a regular expression that splits the input into linguistically and typographically meaningful chunks; in all our experiments, we use nanochat's (Karpathy, 2025) regular expression for pretokenisation.

Tokenisers. Throughout our experiments, we use BPE, UnigramLM, and PathPiece as baselines (Sennrich et al., 2016; Kudo, 2018; Schmidt et al., 2024). We also add a new baseline, termed here Freq, which blindly adds the K most frequent pretokens to its vocabulary, using them to compression-optimally tokenise new sentences with PathPiece. We compare these methods to ConvexTok with the three rounding schemes presented above: Bias, Det, and Int. In a subset of our experiments, we also compare them to ConvexTok trained with the morphology-aware objective above. We train all tokenisers with 593,920 documents, comprising 700 MB, and with vocabulary sizes: from $8k$ to $256k$ in English, in power-of-two increments; $128k$, $192k$, and $256k$ in the multilingual setting. Notably, at inference, our tokenisers use a standard shortest path algorithm (similar to UnigramLM's); ConvexTok thus has a similar inference runtime to UnigramLM and BPE.

Solving the LP. We solve the LP using the PDLP method implemented in the NVIDIA cuOpt library (NVIDIA, 2025), and use the default stopping criteria of a primal dual ratio of at most 1×10^{-5}. This results in a lower precision than other existing methods (based on, e.g., primal or dual simplex; 10^{-5} vs. 10^{-9}), but allows us to solve our relatively large LPs. We also merged identical subgraphs and adjusted the objective coefficients to preserve optimality. In English, the final LP we solved has 99,168,445 constraints in total, of which 20,093,064 are equality and 79,075,381 are inequality constraints. Furthermore, it has 105,997,943 variables corresponding to 79,075,380 priced edges and 18,096,951 free edges, with 8,825,612 different colour variables.

Language Models. Following the nanochat standard, we trained GPT-style decoder-only transformer models. Although the overall structure follows the standard GPT design, nanochat upgrades it with modern architectural techniques. We also follow nanochat's training configuration, whose hyperparameter choices are motivated by empirical scaling laws (Kaplan et al., 2020; Hoffmann et al., 2022). In our experiments, we fix these hyperparameters across tokenisers and vocabulary sizes. They do, however, vary with depth. Hyperparameters were chosen using BPE at vocabulary size 32k and reused unchanged for other tokenisers and vocabulary sizes. See App. I for training loss curves.[9]

[8] We verify this using CELEX (Baayen et al., 1995). If several analyses are available for a word, the least-penalising one is used. Words without an analysis in CELEX receive no penalty.

[9] Model training time varied from 17 minutes on 4 GH200s (for models with 135M parameters) up to 279 minutes on 16 GH200s (for models with 6.3B parameters). Training the ConvexTok tokeniser itself took about 4 hours on 1 GH200. We estimate that in total we needed about 3500 GPU hours across all of our training runs.

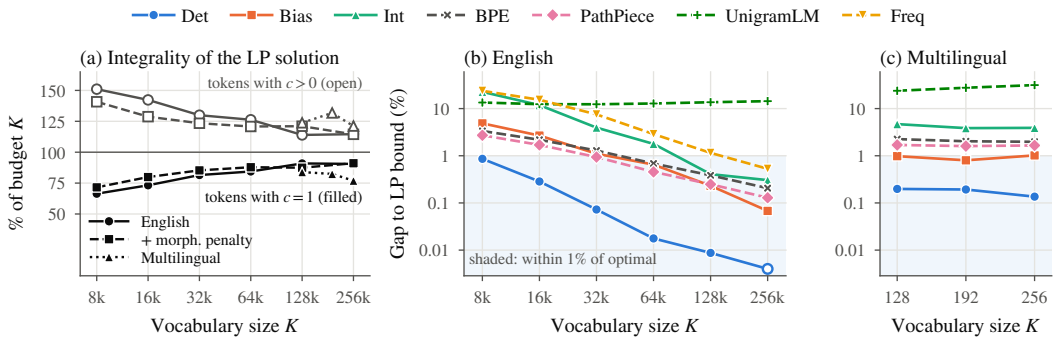


Figure 2: (a) Share of the budget K that the LP solution selects fully ($\mathbf{c} = 1$) or partially ($\mathbf{c} > 0$). (b, c) Gap between each tokeniser's training-set compression and the LP lower bound; the white circle marks a tokeniser within the solver's tolerance of the LP value. Detailed results in Tabs. 1 to 3 in App. E.

5 Results

Behaviour of Solving the Linear Programs. Figs. 2 and 3 report statistics obtained when solving the LP for different vocabulary sizes. Unsurprisingly, the LP's value (its achieved objective) decreases monotonically as the vocabulary size increases (Fig. 3). More interestingly, the number of integral tokens (with $\mathbf{c} = 1$) increases with the vocabulary size (Fig. 2 a).[10] This suggests that, at larger vocabulary sizes, the optimisation problem becomes less ambiguous, and therefore more tokens are already determined by the LP solution, leaving less work for the subsequent rounding procedure. Finally, we also note how small the number of non-zero entries is in comparison to how many variables exist.

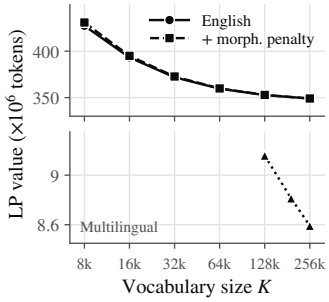


Figure 3: LP values for the English and the multilingual data.

Performance of Rounding Schemes on Compression, and Certifying Optimality. Fig. 2 (b, c) presents the gap between the compression obtained by our tokenisers and the LP values. Interestingly, it shows that the various tokenisers are close to the LP's corresponding solutions, especially `Det`. As the LP value provides a provable lower bound on the optimal compression, it follows that our tokenisers are close to optimal. Considering how close the various rounded tokenisers are to optimal, we may assume that the rounding step is not necessarily that critical to achieve optimal compression. This is especially so as already at vocabulary size of 64k the `Freq` tokeniser is within 3% of optimal, and at 256k it is less than 1% away from optimal. This trend suggests a saturation effect; after the most compression-important tokens have been included, additional tokens have diminishing marginal value. Additionally, it is worth noting how close `BPE` also is to being compression-optimal, despite being a greedy method. This is especially surprising given Kozma and Voderholzer's (2024) theoretical analysis of `BPE`, which shows it may be arbitrarily bad at compressing a dataset. Finally, `UnigramLM`'s gap-to-LP is unexpectedly stable across vocabulary sizes.

Stability of Tokenisers to Randomness on the Training Sample. We now analyse the *stability* of our analysed tokenisers, and how their vocabularies depend on a specific choice of training set. To measure this, we train tokenisers using independently sampled subsets of our training data, and compare their vocabularies using the Jaccard similarity, defined, for vocabularies $\mathcal{T}_1$ and $\mathcal{T}_2$, as $\frac{|\mathcal{T}_1 \cap \mathcal{T}_2|}{|\mathcal{T}_1 \cup \mathcal{T}_2|}$. We say a tokeniser is more **stable** if its Jaccard similarity is larger. Fig. 4 presents these results for different vocabulary and training dataset sizes, showing that stability decreases as the vocabulary size increases. This trend is likely due to larger vocabularies containing more low-frequency tokens, which are more sensitive to the sampled dataset. Another thing to note is that `BPE` is consistently more stable than `ConvexTok`. This aligns with `BPE` being a greedy and frequency-driven method; due to the power-law nature of natural language distributions, the initial high-frequency merges are unlikely to change due to resampling. The higher stability of `Int` compared to `Bias` or `Det` is also to be expected, as `Int` includes only tokens which the LP deems "critical." Compared to `UnigramLM`, however, all other tokenisers are fairly stable, except at the largest vocabulary sizes.

[10] Due to numerical imprecision, we used 0.999 as an approximation for 1 and 0.001 for 0.

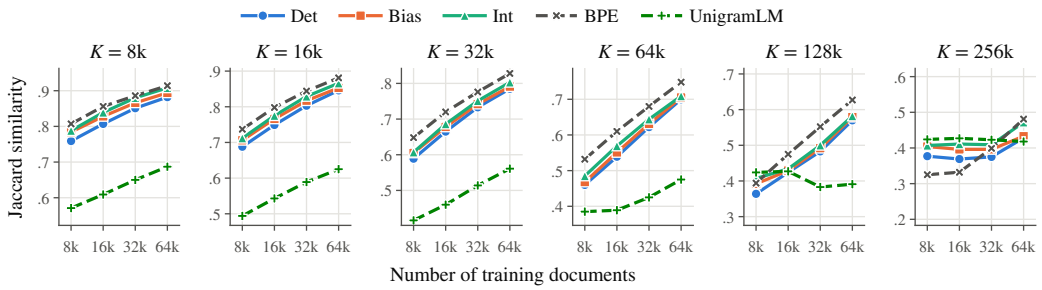


Figure 4: Average Jaccard similarity of vocabularies when retraining a tokeniser on independently sampled subsets of the English dataset.

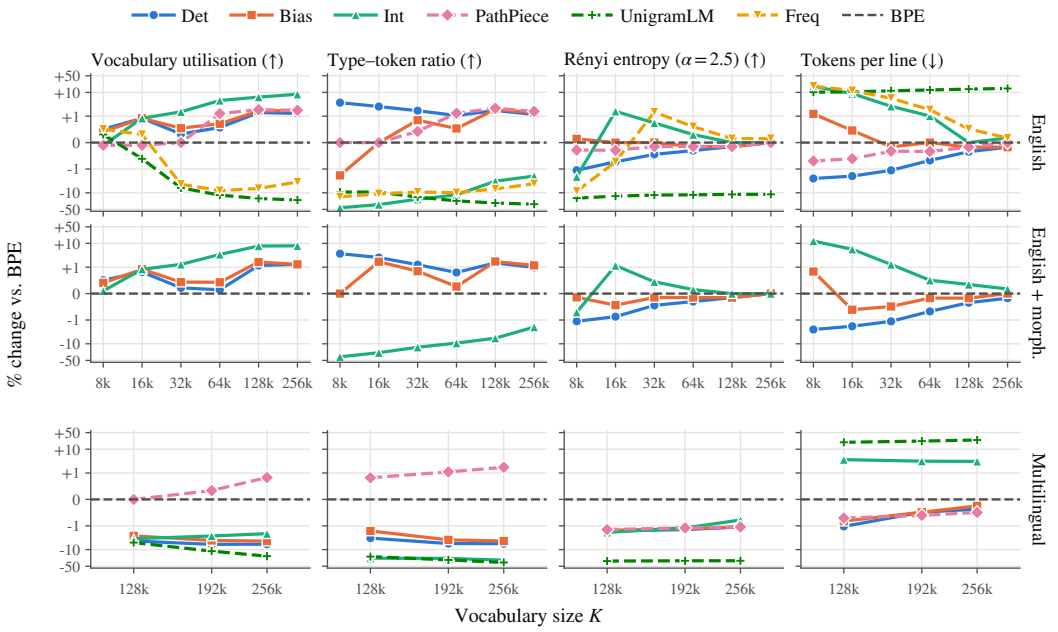


Figure 5: Tokeniser performance on intrinsic metrics computed on held-out data. Results are presented in comparison to `BPE`'s. English+morph marks `ConvexTok` with the morphology-aware objective.

Performance of Tokenisers on Various Intrinsic Metrics. Fig. 5 compares our tokenisers against `BPE` on a suite of classical intrinsic tokenisation metrics, computed using the library by Meister (2026) on a held-out subset of our datasets that was not used to train the tokenisers. (See Tab. 4 in App. F for the raw results, and Tab. 5 in App. G for similar metrics computed on the tokenisers' training data itself.) This figure shows that, in English, the `Det` and `Bias` variants of `ConvexTok` (both with a pure compression, or the morphology-aware objective) tend to outperform `BPE` in vocabulary utilisation, type-token ratio, and tokens per line. In terms of Rényi entropy, results are reversed, with `BPE` often outperforming both `Bias` and `Det`. Notably, in the multilingual dataset, we see different patterns. While `ConvexTok` still outperforms `BPE` in tokens per line, `BPE` or `PathPiece` dominate the other metrics. We suspect this may be due to the larger diversity inherent to multilingual data, which might require more data to stabilise `ConvexTok`.

Performance of Models on `BpB`. Fig. 6 shows the performance (both in terms of `BpB` and `CORE` metrics) of LMs trained with a selection of tokenisers on our English and multilingual settings. These results show that, in English and at standard vocabulary sizes of 32k tokens or more, the `ConvexTok` tokenisers consistently outperform `BPE` in terms of `BpB`. In particular for larger models (of depth 18 and 24), `Det` consistently outperforms `PathPiece` as well. In fact, a paired permutation test shows that across conditions `Det` significantly outperforms `BPE`, `UnigramLM`, and `PathPiece` (at a $\alpha < .01$ confidence). Further, due to computational constraints, we only trained models with the morphology-aware objective for depth 24 and vocabularies of size 64k and above. For those cases, however, these models also often outperformed baselines, behaving on par with `Det`. In the multilingual data the story seems more mixed. `Bias` seems to clearly lag behind `BPE`, `PathPiece`,

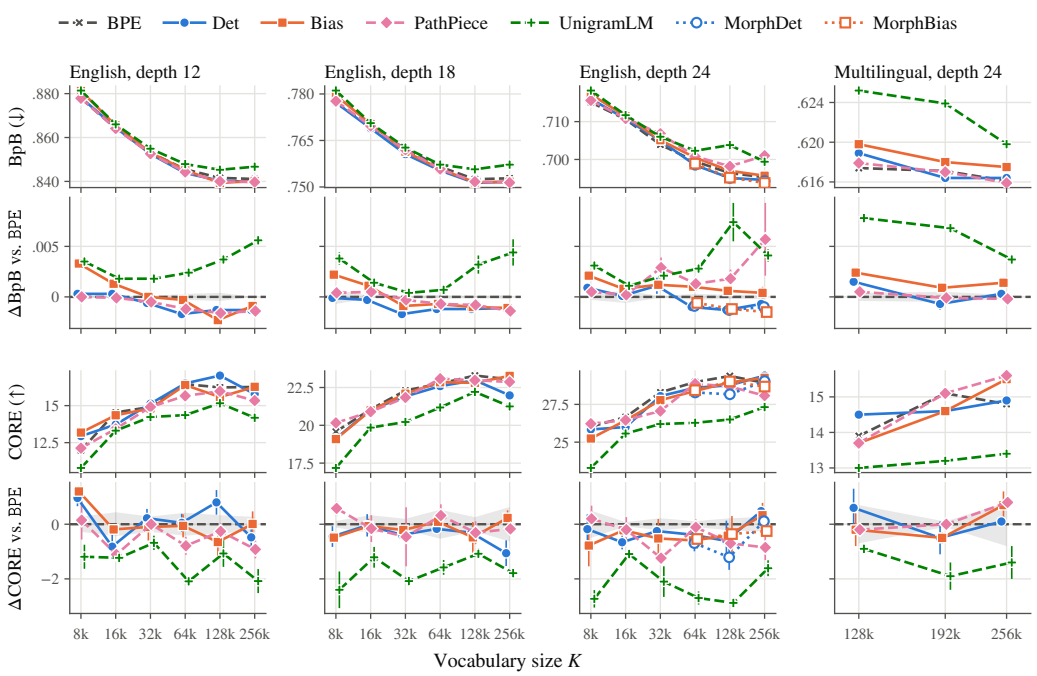


Figure 6: Language model performances. Raw BpB and CORE (×100), followed by their differences to BPE. Error bars show ±1 standard error over 3 seeds. MorphDet and MorphBias were only trained at depth 24 and for vocabulary sizes of 64k and above. Raw values are in Tab. 6 in App. H.

and Det, while these three tokenisers seem to present similar BpB. Finally, across both settings, ConvexTok seems to benefit from a minimum vocabulary size of 32k or above.

Performance of Models on CORE. Fig. 6 also presents our models' CORE performance in downstream tasks. (CORE is a benchmark of downstream tasks covering reasoning, multiple choice and common sense questions; Li et al., 2024.) For the CORE metric, there is no clear trend in results. In fact, a paired permutation test shows no significant differences between any of the analysed tokenisers, with the exception of UnigramLM, which trails behind all others.

6 Conclusion

In this paper, we reinterpreted tokenisation as a graph problem and formulated its compression objective as an IP. We then studied its corresponding relaxed formulation (i.e., its LP) experimentally, including both its solvability and three rounding procedures for converting fractional LP solutions into discrete tokenisers. In English, our experiments show that the ConvexTok tokenisers are strong both in terms of intrinsic metrics and BpB; results on CORE metrics and in a multilingual setting are more mixed, but suggest ConvexTok still at least matches BPE's performance. Overall, our results suggest that LP-based tokenisation is a promising alternative to the greedy methods (BPE and UnigramLM) currently dominating the tokenisation landscape. We hope future work will explore the flexibility offered by ConvexTok to propose and explore new avenues for tokenisation.

Limitations and Future Work. The focus of this paper is on compression; however, Tab. 3 shows that getting tokenisers with near-optimal compression on a dataset is not too hard, as at vocabulary sizes of 128k and 256k our approach achieves results within 1% of the LP lower bound. Significantly improving training-data compression is thus senseless under similar constraints. Recent approaches like SuperBPE (Liu et al., 2025), however, propose methods which partially ignore pretokenisation; it would be interesting to extend our analyses to those settings. Further, we explore here one alternative objective to compression, biasing our tokeniser towards morphologically aligned tokens. Extending our LP-based approach to other objective functions f could also lead to novel and interesting tokenisers. Finally, we find that ConvexTok is only on par with BPE in a multilingual setting (not improving over it in this setting). We suspect this may be due to the size of the datasets used to train our tokenisers, limited by the size of the LP we could feasibly solve. Improving the LP's memory efficiency, or proposing lossless reductions, would allow larger datasets to be used and potentially improve its results.

AI USAGE

In this research paper we used AI to assist with writing code, to brainstorm the morphological alignment function, to suggest formats for tables and figures, and to search for citations. All AI-written code was carefully checked by the authors, the content of the tables and figures (when reformatted using AI) was verified by the authors, and all the suggested citations were fully read before being incorporated. We did not use generative AI to do any of the following: generate synthetic datasets, develop theoretical models or conceptual frameworks, propose or refine hypotheses, design or provide feedback on research methodology or experiments, formulate mathematical claims, assist with translation, clean or reformat datasets, support qualitative and thematic data analysis, interpret results. The following are not relevant to our submission: provide critical ingredients for proving mathematical claims, or assist in the writing of proofs. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Mehdi Ali, Michael Fromm, Klaudia Thellmann, Richard Rutmann, Max Lübbering, Johannes Leveling, Katrin Klug, Jan Ebert, Niclas Doll, Jasper Buschhoff, Charvi Jain, Alexander Weber, Lena Jurkschat, Hammam Abdelwahab, Chelsea John, Pedro Ortiz Suarez, Malte Ostendorff, Samuel Weinbach, Rafet Sifa, Stefan Kesselheim, and Nicolas Flores-Herr. 2024. Tokenizer choice for LLM training: Negligible or crucial? In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 3907–3924, Mexico City, Mexico. Association for Computational Linguistics.

David L. Applegate, Robert E. Bixby, Vašek Chvatál, and William J. Cook. 2006. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press.

R. Harald Baayen, Richard Piepenbrock, and Leon Gulikers. 1995. CELEX2 Lexical Database. Web Download. Philadelphia: Linguistic Data Consortium. LDC96L14.

Stella Biderman, Hailey Schoelkopf, Quentin Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar Van Der Wal. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23.

Hajo Broersma, Xueliang Li, Gerhard Woeginger, and Shenggui Zhang. 2005. Paths and cycles in colored graphs. *Australasian journal of combinatorics*, 31(1):299–311.

Pavel Chizhov, Catherine Arnett, Elizaveta Korotkova, and Ivan P. Yamshchikov. 2024. BPE gets picky: Efficient vocabulary refinement during tokenizer training. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16587–16604, Miami, Florida, USA. Association for Computational Linguistics.

Marco Cognetta, Tatsuya Hiraoka, Rico Sennrich, Yuval Pinter, and Naoaki Okazaki. 2024a. An analysis of BPE vocabulary trimming in neural machine translation. In *Proceedings of the Fifth Workshop on Insights from Negative Results in NLP*, pages 48–50, Mexico City, Mexico. Association for Computational Linguistics.

Marco Cognetta, Vilém Zouhar, Sangwhan Moon, and Naoaki Okazaki. 2024b. Two counterexamples to tokenization and the noiseless channel. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 16897–16906, Torino, Italia. ELRA and ICCL.

George B. Dantzig. 1960. On the shortest route through a network. *Management Science*, 6(2):187–190.

L. R. Ford and D. R. Fulkerson. 1962. *Flows in Networks*. Princeton University Press.

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.

Mohsen Ghaffari, David R. Karger, and Debmalya Panigrahi. 2017. Random contractions and sampling for hypergraph and hedge connectivity. In *Proceedings of the 2017 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1101–1114. ACM-SIAM.

Thamme Gowda and Jonathan May. 2020. Finding the optimal vocabulary size for neural machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3955–3964, Online. Association for Computational Linguistics.

Albert Gu, Yuval Pinter, Sander Land, Kris Cao, and Alisa Liu. 2025. Panel: Future of tokenization. Video presentation at the 42nd International Conference on Machine Learning (ICML). Accessed: 2026-09-16.

Refael Hassin, Jérôme Monnot, and Danny Segev. 2006. Approximation algorithms and hardness results for labeled connectivity problems. *Journal of Combinatorial Optimization*, 14:437–453.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack W. Rae, and Laurent Sifre. 2022. Training compute-optimal large language models. In *Advances in Neural Information Processing Systems*.

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

Andrej Karpathy. 2025. nanochat: The best ChatGPT that $100 can buy.

Violeta Kastreva, Philip Whittington, Dennis Komm, and Tiago Pimentel. 2026. Tokenisation over bounded alphabets is hard. In *The Fourteenth International Conference on Learning Representations*.

Ed Klotz and Alexandra M. Newman. 2013. Practical guidelines for solving difficult mixed integer linear programs. *Surveys in Operations Research and Management Science*, 18(1–2):18–32.

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.

A. H. Land and A. G. Doig. 1960. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Yitzhak Gadre, Hritik Bansal, Etash Kumar Guha, Sedrick Keh, Kushal Arora, Saurabh Garg, Rui Xin, Niklas Muennighoff, Reinhard Heckel, Jean Mercat, Mayee F Chen, Suchin Gururangan, Mitchell Wortsman, Alon Albalak, Yonatan Bitton, Marianna Nezhurina, Amro Kamal Mohamed Abbas, Cheng-Yu Hsieh, Dhruba Ghosh, Joshua P Gardner, Maciej Kilian, Hanlin Zhang, Rulin Shao, Sarah M Pratt, Sunny Sanyal, Gabriel Ilharco, Giannis Daras, Kalyani Marathe, Aaron Gokaslan, Jieyu Zhang, Khyathi Chandu, Thao Nguyen, Igor Vasiljevic, Sham M. Kakade, Shuran Song, Sujay Sanghavi, Fartash Faghri, Sewoong Oh, Luke Zettlemoyer, Kyle Lo, Alaaeldin El-Nouby, Hadi Pouransari, Alexander T Toshev, Stephanie Wang, Dirk Groeneveld, Luca Soldaini, Pang Wei Koh, Jenia Jitsev, Thomas Kollar, Alex Dimakis, Yair Carmon, Achal Dave, Ludwig Schmidt, and Vaishaal Shankar. 2024. Datacomp-LM: In search of the next generation of training sets for language models. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Haoran Lian, Yizhe Xiong, Jianwei Niu, Shasha Mo, Zhenpeng Su, Zijia Lin, Hui Chen, Jungong Han, and Guiguang Ding. 2025. Scaffold-BPE: Enhancing byte pair encoding for large language models with simple and effective scaffold token removal. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(23):24539–24548.

Jia Peng Lim, Shawn Tan, Davin Choo, and Hady W. Lauw. 2025. A partition cover approach to tokenization. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Alisa Liu, Jonathan Hayase, Valentin Hofmann, Sewoong Oh, Noah A. Smith, and Yejin Choi. 2025. SuperBPE: Space travel for language models. In *Second Conference on Language Modeling*.

Clara Meister. 2026. Tokeval: A tokenizer evaluation suite. In *Conference on Language Modeling*.

Yurii Nesterov and Arkadii Nemirovskii. 1994. *Interior-Point Polynomial Algorithms in Convex Programming*, volume 13 of *Studies in Applied and Numerical Mathematics*. Society for Industrial and Applied Mathematics, Philadelphia, PA.

NVIDIA. 2025. NVIDIA cuOpt: GPU-accelerated decision optimization.

OpenAI. 2023. GPT-4 technical report. *Preprint*, arXiv:2303.08774.

Christos H. Papadimitriou and Kenneth Steiglitz. 1982. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., USA.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. 2024. The FineWeb datasets: Decanting the web for the finest text data at scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

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.

Craig W. Schmidt, Varshini Reddy, Chris Tanner, and Yuval Pinter. 2025. Boundless byte pair encoding: Breaking the pre-tokenization barrier. In *Second Conference on Language Modeling*.

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.

David B. Shmoys and Éva Tardos. 1993. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62(1):461–474.

Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, Robert Berry, Saumya Malik, Saurabh Shah, Scott Geng, Shane Arora, Shashank Gupta, Taira Anderson, Teng Xiao, Tyler Murray, Tyler Romero, Victoria Graf, Akari Asai, Akshita Bhagia, Alexander Wettig, Alisa Liu, Aman Rangapur, Chloe Anastasiades, Costa Huang, Dustin Schwenk, Harsh Trivedi, Ian Magnusson, Jaron Lochner, Jiacheng Liu, Lester James V. Miranda, Maarten Sap, Malia Morgan, Michael Schmitz, Michal Guerquin, Michael Wilson, Regan Huff, Ronan Le Bras, Rui Xin, Rulin Shao, Sam Skjonsberg, Shannon Zejiang Shen, Shuyue Stella Li, Tucker Wilde, Valentina Pyatkin, Will Merrill, Yapei Chang, Yuling Gu, Zhiyuan Zeng, Ashish Sabharwal, Luke Zettlemoyer, Pang Wei Koh, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2026. Olmo 3. *Preprint*, arXiv:2512.13961.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and efficient foundation language models. *Preprint*, arXiv:2302.13971.

Vijay V. Vazirani. 2010. *Approximation Algorithms*. Springer Publishing Company, Incorporated.

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)*, pages 28133–28153, Vienna, Austria. Association for Computational Linguistics.

David P. Williamson and David B. Shmoys. 2011. *The Design of Approximation Algorithms*, 1st edition. Cambridge University Press, USA.

Ahmetcan Yavuz, Clara Meister, and Tiago Pimentel. 2026. Objective vs. search: Decomposing what makes a good tokeniser. *Preprint*, arXiv:2609.19145.

Peng Zhang, Jin-Yi Cai, Lin-Qing Tang, and Wen-Bo Zhao. 2011. Approximation and hardness results for label cut and related problems. *Journal of Combinatorial Optimization*, 21:192–208.

Vilém Zouhar, Clara Meister, Juan Gastaldi, Li Du, Mrinmaya Sachan, and Ryan Cotterell. 2023a. 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.

Vilém Zouhar, Clara Meister, Juan Gastaldi, Li Du, Tim Vieira, Mrinmaya Sachan, and Ryan Cotterell. 2023b. A formal perspective on byte-pair encoding. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 598–614, Toronto, Canada. Association for Computational Linguistics.

A Formal Description of BPE

Let $\texttt{merge}_{t',t''}$ be a function which, given a token-string, replaces all occurrences of the bigram t', t'' in it with the merged token $t^{\texttt{new}} \stackrel{\texttt{def}}{=} t' \circ t''$. The BPE algorithm then works as follows: it initialises a character-level tokeniser, $\mathbb{T}_0 = \langle \Sigma, \texttt{id}, \texttt{detok} \rangle$, where $\texttt{id}$ is the identity function; for a pre-specified number of steps K, it then takes the current tokeniser $\mathbb{T}_{k-1} = \langle \mathcal{T}_{k-1}, \texttt{tok}_{k-1}, \texttt{detok} \rangle$ and selects the pair of existing tokens $t', t'' \in \mathcal{T}_{k-1}$ which, if merged, will lead to maximal compression, and adds it to the vocabulary, as $\mathcal{T}_k = \mathcal{T}_{k-1} \cup \{t^{\texttt{new}}\}$, and the encoding function as $\texttt{tok}_k(\mathbf{b}) = \texttt{merge}_{t',t''}(\texttt{tok}_{k-1}(\mathbf{b}))$.

B Alternatives to Linear Programming for Solving IPs

An alternative choice to linear programming is to use methods such as branch-and-bound (Land and Doig, 1960) to solve the IP directly. While branch-and-bound algorithms perform well in practice on highly structured IPs, such as travelling salesman problems (Applegate et al., 2006), they exhibit poor scaling behaviours. Hence, we decided to use linear programming with rounding, as this method scales better (Klotz and Newman, 2013). We hope future work explores other approaches to investigate and try to approximate this IP.

C Mapping Between Vectors of the IP and the Token Sequence

In this section, we describe how to map an element of $\mathcal{Q}^{\texttt{IP}}$ to a tokeniser $\mathbb{T}$, and back. We give an informal sketch here, as the process is relatively simple.

C.1 From an IP Solution to a Tokeniser

Consider some free token-instance vector, priced token-instance vector, and priced-colour vector $\mathbf{f}, \mathbf{p}, \mathbf{c} \in \mathcal{Q}^{\texttt{IP}}$. First, we can recover the vocabulary by taking the union of all elements of $\mathbf{c}$ which are equal to 1 and the alphabet. Formally, the vocabulary is $\{t_c \in \mathcal{C} \mid \mathbf{c}_c = 1\} \cup \Sigma$, where we denote as t_c the token corresponding to colour c. Second, with the free and priced vectors $\mathbf{f}, \mathbf{p}$ one can recover a valid segmentation of the dataset $\mathcal{D}$. Recall that each potential use of a token to encode a byte-string $\mathbf{b} \in \mathcal{D}$ corresponds to exactly one edge in either $\mathbf{f}$ or $\mathbf{p}$. As such, to recover the segmentation of $\mathbf{b} \in \mathcal{D}$ by selecting a token if and only if its corresponding edge from $\mathbf{f} \cup \mathbf{p}$ is equal to 1. For completeness, given the extracted vocabulary, we use PathPiece (Schmidt et al., 2024) to define the encoding function's behaviour on other byte-strings not in the dataset, computing:

$$\forall_{\mathbf{b} \notin \mathcal{D}} : \texttt{tok}(\mathbf{b}) = \underset{\mathbf{t} \in \mathcal{T}^*}{\operatorname{argmin}} |\mathbf{t}|, \quad \texttt{s.t.} \quad \mathbf{b} \stackrel{\circ}{=} \mathbf{t} \tag{7}$$

C.2 From a Tokeniser to an IP Solution

Now we sketch how to take a tokeniser $\mathbb{T}$ and convert it to vectors contained within the IP. This is simply the process described above, but in reverse. Given the tokeniser's vocabulary, we set each element $\mathbf{c}_c$ in vector $\mathbf{c}$ to 1 if and only if its corresponding token t_c is included in the vocabulary. For the dataset $\mathcal{D}$ then, we consider each of its byte-strings $\mathbf{b} \in \mathcal{D}$ and run it under the tokeniser's encoding function to get a token-string. For each token in this token-string, we then find its correspondence in either $\mathbf{f}$ or $\mathbf{p}$ and set it to 1; the rest of these vectors are all zero.

D Algorithms for the Rounding Schemes

```
def det_rounding(K, c):
    col= sort(c, by=c,
              ascending=False)
    c' = 0
    for c in col[0 : K]:
        c'_c = 1
    return c'
```

```
def biased_rounding(K, c):
    col = sort(c, by=c_c / length(c),
               ascending=False)
    c' = 0
    for c in col[0 : K]:
        c'_c = 1
    return c'
```

```
def integral_rounding(c_c):
    c' = 0
    for c ∈ C:
        if c_c ≥ 0.999:
            c'_c = 1
    return c'
```

Figure 7: The `Det` (left), `Bias` (center), and `Int` (right) rounding schemes.

E Detailed LP Solutions and Certificate Results

E.1 Characteristics of the LP Solutions

Table 1: Characteristics of the LP's solutions. % of 1s (and of ¬ 0s) measures the ratio of **c** which are 1 (or not 0) divided by the vocabulary budget; % of ¬ 0s is thus always equal to or larger than 100%.

	Vocabulary Size	# steps	Time (sec)	LP Value	c		f		p	
					% of 1s	% of ¬ 0s	# of 1s	# of ¬ 0s	# of 1s	# of ¬ 0s
English	$8k$	65,000	889.927	427,366,252	66.41%	151.01%	1,466,122	5,889,259	577,768	16,244,391
	$16k$	5,600	182.661	393,224,648	73.22%	142.31%	1,021,750	4,724,165	750,072	15,712,089
	$32k$	6,800	196.123	371,886,133	81.5%	129.95%	678,332	3,263,220	1,247,839	13,796,121
	$64k$	6,400	181.455	359,626,839	84.39%	126.26%	349,172	2,402,815	1,380,456	12,738,868
	$128k$	9,700	226.768	352,723,064	90.91%	113.97%	204,014	1,556,595	1,717,709	10,338,653
	$256k$	9,500	230.920	349,028,128	90.53%	114.59%	121,694	1,207,058	1,560,819	9,564,059
X-ling	$128k$	3,000	93.273	9,157,900.11	83.91%	123.78%	30,131	180,274	102,774	3,372,084
	$192k$	3,000	96.167	8,812,689.23	82.40%	131.77%	23,296	140,976	927,370	3,081,093
	$256k$	3,100	125.103	8,591,165.17	76.94%	121.14%	20,524	120,565	885,685	2,879,414

Table 2: Characteristics of the solutions for the Morphologically aligned LP.

Vocabulary Size	# steps	Time (sec)	LP Value	c		f		p	
				% of 1s	% of ¬ 0s	# of 1s	# of ¬ 0s	# of 1s	# of ¬ 0s
$8k$	113,000	1405.740	430,875,712	71.56%	140.80%	1,523,062	5,754,807	703,642	15,330,185
$16k$	43,200	563.552	394,939,168	79.89%	128.75%	1,196,589	4,575,999	1,214,400	14,459,374
$32k$	6,800	176.759	372,730,826	85.30%	123.43%	753,166	3,094,186	1,535,031	12,762,789
$64k$	7,100	149.716	359,886,107	87.85%	120.80%	412,468	2,258,369	1,640,147	11,685,263
$128k$	7,500	194.177	352,909,553	87.45%	121.03%	220,082	1,712,036	1,531,328	11,054,461
$256k$	9,500	186.040	349,042,343	91.08%	114.46%	150,881	1,202,375	1,624,707	9,338,450

E.2 Comparison Between LP and Tokenisers' Compression

When reading Tab. 3, one should keep in mind that the LP values are obtained numerically and are therefore subject to solver tolerances and numerical imprecision. The gap for `Det` at vocabulary size $256k$ is within that tolerance.

Table 3: Comparison between the LP relaxation value and the value obtained by each tokeniser.

	Vocab Size	Tokeniser	LP Value	Tokenised Value	Ratio to optimal	Vocabulary Size	Tokeniser	LP Value	Tokenised Value	Ratio to optimal
English	$8k$	`BPE`	427,366,252	441,677,724	103.349%	$64k$	`BPE`	359,626,839	362,089,042	100.685%
		`Det`		431,045,026	100.861%		`Det`		359,690,431	100.018%
		`Bias`		448,151,069	104.863%		`Bias`		361,962,390	100.649%
		`Int`		524,085,422	122.631%		`Int`		366,041,138	101.784%
		`UnigramLM`		484,819,725	113.444%		`UnigramLM`		405,746,323	112.824%
		`PathPiece`		438,972,410	102.716%		`PathPiece`		361,263,876	100.455%
		`Freq`		529,028,205	123.788%		`Freq`		369,990,704	102.882%
	$16k$	`BPE`	393,224,648	401,805,519	102.182%	$128k$	`BPE`	352,723,064	354,079,724	100.385%
		`Det`		394,344,618	100.285%		`Det`		352,753,850	100.009%
		`Bias`		403,757,674	102.679%		`Bias`		353,538,264	100.231%
		`Int`		439,773,017	111.838%		`Int`		354,160,993	100.408%
		`UnigramLM`		441,966,444	112.395%		`UnigramLM`		400,811,027	113.633%
		`PathPiece`		399,902,612	101.698%		`PathPiece`		353,600,543	100.249%
		`Freq`		453,684,154	115.375%		`Freq`		356,781,397	101.151%
	$32k$	`BPE`	371,886,133	376,681,580	101.289%	$256k$	`BPE`	349,028,128	349,745,795	100.206%
		`Det`		372,156,050	100.073%		`Det`		349,020,177	99.998%
		`Bias`		375,996,735	101.105%		`Bias`		349,264,651	100.068%
		`Int`		386,575,802	103.950%		`Int`		350,090,533	100.304%
		`UnigramLM`		417,810,267	112.349%		`UnigramLM`		399,172,871	114.367%
		`PathPiece`		375,375,924	100.938%		`PathPiece`		349,477,642	100.129%
		`Freq`		399,875,524	107.526%		`Freq`		350,874,040	100.529%

	Tokeniser	$128k$			$192k$			$256k$		
		LP Value	Tokenised Value	Ratio to optimal	LP Value	Tokenised Value	Ratio to optimal	LP Value	Tokenised Value	Ratio to optimal
Multilingual	`BPE`	9,157,900	9,364,235	102.253%	8,812,689	8,991,643	102.031%	8,591,165	8,760,970	101.977%
	`Det`		9,176,047	100.198%		8,829,669	100.193%		8,602,841	100.136%
	`Bias`		9,247,705	100.981%		8,883,293	100.801%		8,678,606	101.018%
	`Int`		9,586,922	104.685%		9,152,260	103.853%		8,926,269	103.901%
	`UnigramLM`		11,343,421	123.865%		11,248,099	127.635%		11,294,082	131.462%
	`PathPiece`		9,313,079	101.694%		8,953,890	101.602%		8,733,318	101.655%

F RAW INTRINSIC TOKENISATION RESULTS

Table 4: Tokenisers' performances on intrinsic metrics computed on a held-out dataset. Bold is best for the vocabulary size, underline is best across sizes.

	Vocab Size	Tokeniser	Vocabulary Utilisation (↑)	Type-Token Ratio (↑)	Rényi Entropy (α=1) (↑)	Rényi Entropy (α=2.5) (↑)	Avg Token Rank (↓)	Token Length	Tokens per Line (↓)	Compression Rate (↑)
English	$8k$	BPE	98.4%	0.0054	10.59	7.08	1111.3	4.02	71.8	0.0013
		Freq	**98.9%**	0.0046	9.17	6.47	600.0	3.40	85.1	0.0011
		PathPiece	98.3%	0.0054	10.63	7.06	1160.5	4.04	71.3	0.0013
		UnigramLM	98.7%	0.0049	9.50	5.89	778.1	3.68	79.1	0.0012
		Det	**98.9%**	**0.0056**	10.59	7.00	1135.9	4.12	**70.0**	**0.0014**
		Bias	98.8%	0.0053	**10.66**	**7.09**	1154.0	3.96	72.7	0.0013
		Int	98.3%	0.0031	9.34	6.93	**499.1**	3.41	84.6	0.0011
	$16k$	BPE	97.8%	0.0118	10.86	6.87	1747.9	4.41	65.5	**0.0015**
		Freq	98.1%	0.0105	9.93	6.82	1096.8	3.95	73.3	0.0013
		PathPiece	97.7%	0.0118	10.89	6.85	1819.2	4.43	65.1	**0.0015**
		UnigramLM	97.2%	0.0107	9.89	5.92	1269.2	4.03	72.4	0.0013
		Det	**98.7%**	**0.0121**	10.86	6.82	1798.9	4.50	**64.2**	**0.0015**
		Bias	**98.7%**	0.0118	**10.90**	6.87	1814.4	4.39	65.8	**0.0015**
		Int	**98.7%**	0.0081	10.19	**6.98**	**1039.3**	4.06	71.3	0.0014
	$32k$	BPE	91.9%	0.0236	10.98	6.72	2438.0	4.71	61.5	**0.0016**
		Freq	87.9%	0.0214	10.55	**6.82**	1805.1	4.46	65.0	0.0015
		PathPiece	91.9%	0.0237	**11.00**	6.71	2529.4	4.72	61.3	**0.0016**
		UnigramLM	86.2%	0.0200	10.07	5.88	**1772.6**	4.26	68.7	0.0014
		Det	92.2%	**0.0240**	10.97	6.69	2505.5	4.76	**60.8**	**0.0016**
		Bias	92.4%	0.0238	10.99	6.72	2520.8	4.71	61.4	**0.0016**
		Int	**93.3%**	0.0192	10.78	6.77	1939.9	4.59	63.1	0.0015
	$64k$	BPE	70.2%	0.0375	11.00	6.64	3009.4	4.89	59.3	**0.0016**
		Freq	64.5%	0.0338	10.86	**6.68**	2537.3	4.80	60.4	**0.0016**
		PathPiece	71.1%	**0.0380**	**11.01**	6.63	3100.1	4.90	59.1	**0.0016**
		UnigramLM	61.4%	0.0293	10.13	5.83	**2126.9**	4.39	66.8	0.0015
		Det	70.6%	0.0379	10.98	6.62	3067.2	4.92	**58.9**	**0.0016**
		Bias	70.7%	0.0377	10.99	6.63	3073.3	4.89	59.3	**0.0016**
		Int	**73.4%**	0.0329	10.92	6.66	2665.7	4.84	59.9	**0.0016**
	$128k$	BPE	42.6%	0.0464	**10.97**	6.59	3317.1	4.99	58.1	**0.0017**
		Freq	39.8%	0.0432	10.93	**6.60**	3043.7	4.97	58.4	**0.0017**
		PathPiece	43.4%	**0.0474**	**10.97**	6.58	3386.4	5.00	58.0	**0.0017**
		UnigramLM	35.2%	0.0339	10.15	5.82	**2270.5**	4.44	66.1	0.0015
		Det	43.2%	0.0472	10.96	6.58	3358.7	5.01	**57.9**	**0.0017**
		Bias	43.3%	0.0473	10.96	6.58	3368.9	5.00	58.0	**0.0017**
		Int	**45.3%**	0.0449	10.96	6.59	3243.3	4.99	58.1	**0.0017**
	$256k$	BPE	22.8%	0.0504	**10.94**	6.56	3415.6	5.05	57.4	**0.0017**
		Freq	22.0%	0.0483	10.93	**6.57**	3287.5	5.04	57.5	**0.0017**
		PathPiece	23.2%	**0.0512**	10.93	6.56	3446.1	5.06	57.4	**0.0017**
		UnigramLM	18.2%	0.0352	10.15	5.80	**2310.4**	4.45	65.8	0.0015
		Det	23.1%	0.0510	10.93	6.56	3430.7	5.06	**57.3**	**0.0017**
		Bias	23.2%	0.0511	10.93	6.56	3437.6	5.06	**57.3**	**0.0017**
		Int	**24.7%**	0.0494	10.93	6.56	3356.0	5.05	57.5	**0.0017**
Multilingual	$128k$	BPE	**94.7%**	0.0122	13.65	**9.07**	11828.8	3.33	85.1	**0.0008**
		PathPiece	**94.7%**	**0.0123**	**13.70**	8.94	12650.0	3.36	84.5	**0.0008**
		UnigramLM	89.9%	0.0098	12.32	6.32	**7661.4**	2.72	101.7	0.0007
		Det	90.6%	0.0118	13.63	8.92	11669.4	3.38	**84.2**	**0.0008**
		Bias	92.2%	0.0120	13.68	8.93	12075.5	3.37	84.4	**0.0008**
		Int	91.4%	0.0094	13.21	8.90	8870.7	3.21	88.2	0.0007
	$192k$	BPE	89.7%	0.0180	13.81	**9.01**	15094.9	3.46	82.7	**0.0008**
		PathPiece	**90.0%**	**0.0182**	**13.86**	8.90	16256.3	3.49	**82.2**	**0.0008**
		UnigramLM	79.3%	0.0131	12.43	6.35	**9000.4**	2.75	100.9	0.0007
		Det	84.3%	0.0170	13.78	8.88	14630.3	3.49	82.3	**0.0008**
		Bias	86.0%	0.0173	13.82	8.89	15143.5	3.49	82.3	**0.0008**
		Int	87.3%	0.0138	13.44	8.90	11483.3	3.35	85.3	**0.0008**
	$256k$	BPE	84.3%	0.0230	13.91	**8.97**	17782.8	3.55	81.3	**0.0008**
		PathPiece	**85.0%**	**0.0234**	**13.96**	8.87	19198.9	3.58	**80.9**	**0.0008**
		UnigramLM	68.4%	0.0150	12.49	6.33	**9835.1**	2.75	101.1	0.0007
		Det	79.3%	0.0217	13.85	8.87	16711.3	3.57	81.0	**0.0008**
		Bias	80.6%	0.0220	13.89	8.87	17390.6	3.56	81.1	**0.0008**
		Int	82.5%	0.0166	13.55	8.90	12986.6	3.42	83.8	**0.0008**
English + Morphology Objective	$8k$	MorphDet	**98.9%**	**0.0056**	10.57	7.00	1124.8	4.12	**70.0**	**0.0014**
		MorphBias	98.8%	0.0054	**10.63**	**7.07**	1140.2	3.98	72.4	0.0013
		MorphInt	98.5%	0.0035	9.60	7.03	**573.0**	3.57	80.8	0.0012
	$16k$	MorphDet	98.6%	**0.0121**	10.84	6.81	1773.3	4.50	**64.3**	**0.0015**
		MorphBias	**98.7%**	0.0120	**10.88**	6.84	1803.1	4.44	65.1	**0.0015**
		MorphInt	**98.7%**	0.0090	10.38	**6.95**	**1192.1**	4.18	69.2	0.0014
	$32k$	MorphDet	92.1%	**0.0239**	10.96	6.69	2476.2	4.76	**60.8**	**0.0016**
		MorphBias	92.3%	0.0238	**10.97**	6.71	2493.1	4.73	61.2	**0.0016**
		MorphInt	**93.1%**	0.0203	10.84	**6.75**	**2063.2**	4.65	62.3	0.0015
	$64k$	MorphDet	70.3%	**0.0378**	10.98	6.62	3043.7	4.92	**58.9**	**0.0016**
		MorphBias	70.5%	0.0376	**10.99**	6.63	3049.3	4.89	59.2	**0.0016**
		MorphInt	**72.6%**	0.0340	10.94	**6.65**	**2746.6**	4.86	59.6	**0.0016**
	$128k$	MorphDet	43.1%	0.0471	**10.96**	6.58	3351.9	5.01	**57.9**	**0.0017**
		MorphBias	43.3%	**0.0472**	**10.96**	6.58	3363.3	5.00	58.0	**0.0017**
		MorphInt	**45.9%**	0.0437	10.94	**6.59**	**3132.9**	4.97	58.3	**0.0017**
	$256k$	MorphDet	23.1%	0.0509	**10.93**	**6.56**	3425.3	5.06	**57.3**	**0.0017**
		MorphBias	23.1%	**0.0510**	**10.93**	**6.56**	3433.4	5.06	57.4	**0.0017**
		MorphInt	**24.6%**	0.0494	**10.93**	**6.56**	**3350.0**	5.05	57.5	**0.0017**

G Intrinsic Tokenisation Results on Tokenisers' Training Data

Table 5: Tokenisers' performances on intrinsic metrics computed on their training set. Bold is best for the vocabulary size, underline is best across sizes.

	Vocabulary Size	Tokeniser	Vocabulary Utilisation (↑)	Type-Token Ratio (↑)	Rényi Entropy (α=1) (↑)	Rényi Entropy (α=2.5) (↑)	Avg Token Rank (↓)	Token Length	Tokens per Line (↓)	Compression Rate (↑)
English	$8k$	BPE	98.6%	0.0055	10.61	7.09	1121.5	3.99	63.8	**0.0014**
		Freq	**99.1%**	0.0046	9.12	6.42	590.8	3.37	76.1	0.0011
		PathPiece	98.5%	0.0055	10.65	7.07	1169.6	4.01	63.4	**0.0014**
		UnigramLM	98.9%	0.0050	9.51	5.89	781.7	3.65	70.2	0.0012
		Det	99.0%	**0.0056**	10.61	7.02	1141.3	4.09	**62.3**	**0.0014**
		Bias	99.0%	0.0054	**10.69**	**7.10**	1171.0	3.93	64.6	0.0013
		Int	98.6%	0.0031	9.31	6.90	**492.3**	3.39	75.6	0.0011
	$16k$	BPE	98.1%	0.0120	10.90	6.88	1780.4	4.38	58.1	**0.0015**
		Freq	98.1%	0.0106	9.89	6.80	1086.5	3.92	65.5	0.0013
		PathPiece	97.9%	0.0120	10.94	6.86	1855.2	4.40	57.8	**0.0015**
		UnigramLM	97.2%	0.0108	9.90	5.91	1290.8	4.00	64.1	0.0014
		Det	**98.8%**	**0.0123**	10.90	6.83	1833.8	4.46	**57.0**	**0.0015**
		Bias	**98.8%**	0.0120	**10.95**	6.88	1860.9	4.36	58.3	**0.0015**
		Int	**98.8%**	0.0082	10.18	**6.99**	**1033.6**	4.03	63.6	0.0014
	$32k$	BPE	92.1%	0.0240	11.04	6.73	2516.7	4.67	54.5	**0.0016**
		Freq	87.7%	0.0214	10.55	**6.83**	**1814.5**	4.42	57.9	0.0015
		PathPiece	92.0%	0.0241	**11.06**	6.72	2613.9	4.69	54.3	**0.0016**
		UnigramLM	86.4%	0.0203	10.11	5.87	1826.9	4.23	60.6	0.0014
		Det	92.4%	**0.0244**	11.03	6.70	2586.1	4.72	**53.9**	**0.0016**
		Bias	92.6%	0.0242	**11.06**	6.72	2609.1	4.68	54.4	**0.0016**
		Int	**93.4%**	0.0195	10.82	6.79	1975.1	4.55	56.0	0.0015
	$64k$	BPE	70.7%	0.0384	**11.07**	6.64	3138.0	4.85	52.4	**0.0017**
		Freq	64.9%	0.0344	10.90	**6.69**	2608.1	4.76	53.5	0.0016
		PathPiece	71.5%	**0.0389**	**11.07**	6.63	3234.6	4.87	52.3	**0.0017**
		UnigramLM	62.1%	0.0301	10.17	5.82	**2213.0**	4.35	58.9	0.0015
		Det	71.2%	**0.0389**	11.05	6.62	3210.4	4.88	**52.1**	**0.0017**
		Bias	71.5%	0.0388	**11.07**	6.63	3226.2	4.85	52.4	**0.0017**
		Int	**73.8%**	0.0336	10.98	6.66	2760.2	4.80	53.0	0.0016
	$128k$	BPE	43.3%	0.0481	**11.05**	6.59	3504.5	4.96	51.3	**0.0017**
		Freq	40.3%	0.0444	11.00	**6.60**	3179.5	4.93	51.6	**0.0017**
		PathPiece	44.1%	**0.0491**	11.04	6.58	3576.0	4.97	51.2	**0.0017**
		UnigramLM	35.8%	0.0351	10.19	5.80	**2382.2**	4.40	58.2	0.0015
		Det	43.8%	0.0489	11.03	6.58	3550.8	4.98	**51.1**	**0.0017**
		Bias	44.0%	0.0490	11.04	6.58	3566.7	4.97	51.2	**0.0017**
		Int	**45.8%**	0.0463	11.03	6.58	3410.6	4.96	51.3	**0.0017**
	$256k$	BPE	23.3%	0.0525	**11.02**	6.56	3635.4	5.02	50.7	**0.0017**
		Freq	22.3%	0.0501	11.00	**6.57**	3472.9	5.00	50.8	**0.0017**
		PathPiece	23.7%	**0.0534**	11.01	6.55	3672.2	5.02	**50.6**	**0.0017**
		UnigramLM	18.6%	0.0367	10.19	5.79	**2432.6**	4.42	57.9	0.0015
		Det	23.6%	0.0532	11.01	6.55	3660.9	5.03	**50.6**	**0.0017**
		Bias	23.7%	**0.0534**	11.01	6.55	3671.1	5.02	**50.6**	**0.0017**
		Int	**25.2%**	0.0514	11.00	6.56	3563.0	5.01	50.7	**0.0017**
Multilingual	$128k$	BPE	99.1%	0.0139	13.56	**9.04**	13802.7	3.45	75.8	**0.0009**
		PathPiece	98.7%	0.0139	**13.59**	8.97	14395.7	3.48	75.4	**0.0009**
		UnigramLM	99.1%	0.0114	12.20	6.46	**8193.6**	2.76	91.8	0.0007
		Det	99.8%	**0.0143**	13.53	8.94	13856.4	3.54	**74.3**	**0.0009**
		Bias	99.8%	0.0141	13.56	8.95	14048.3	3.51	74.8	**0.0009**
		Int	**99.9%**	0.0115	13.31	9.00	10987.7	3.36	77.1	0.0008
	$192k$	BPE	98.5%	0.0215	13.70	**8.96**	18570.8	3.63	73.1	**0.0009**
		PathPiece	98.0%	0.0215	**13.72**	8.91	19339.4	3.65	72.8	**0.0009**
		UnigramLM	94.7%	0.0165	12.29	6.48	**9548.3**	2.79	91.1	0.0007
		Det	99.7%	**0.0222**	13.66	8.87	18499.1	3.71	**71.8**	**0.0009**
		Bias	99.7%	0.0221	13.68	8.88	18709.2	3.68	72.2	**0.0009**
		Int	**100.0%**	0.0177	13.51	8.93	15003.2	3.56	74.0	**0.0009**
	$256k$	BPE	97.9%	0.0293	13.79	**8.90**	22999.5	3.75	71.4	**0.0009**
		PathPiece	97.3%	0.0292	**13.81**	8.86	23832.8	3.76	71.1	**0.0009**
		UnigramLM	79.0%	0.0183	12.33	6.44	**10121.3**	2.78	91.5	0.0007
		Det	**100.0%**	**0.0305**	13.73	8.82	22921.0	3.84	**70.2**	**0.0009**
		Bias	99.7%	0.0301	13.76	8.85	22848.6	3.79	70.7	**0.0009**
		Int	**100.0%**	0.0227	13.59	8.89	17692.7	3.67	72.4	**0.0009**
English + Morphology Objective	$8k$	MorphDet	**99.1%**	**0.0056**	10.59	7.01	1130.6	4.09	**62.2**	**0.0014**
		MorphBias	99.0%	0.0054	**10.65**	**7.08**	1154.2	3.95	64.3	0.0013
		MorphInt	98.7%	0.0035	9.56	7.02	**564.8**	3.54	72.3	0.0012
	$16k$	MorphDet	98.7%	**0.0123**	10.88	6.82	1810.0	4.47	**57.0**	**0.0015**
		MorphBias	**98.9%**	0.0121	**10.92**	6.85	1845.3	4.41	57.7	**0.0015**
		MorphInt	98.8%	0.0091	10.38	**6.96**	**1199.5**	4.15	61.6	0.0014
	$32k$	MorphDet	92.1%	**0.0243**	11.02	6.70	2556.2	4.72	**53.9**	**0.0016**
		MorphBias	92.4%	0.0242	**11.04**	6.71	2580.5	4.69	54.2	**0.0016**
		MorphInt	**93.1%**	0.0205	10.89	**6.76**	**2105.8**	4.61	55.3	**0.0016**
	$64k$	MorphDet	70.8%	**0.0387**	11.05	6.62	3180.7	4.88	**52.1**	**0.0017**
		MorphBias	71.1%	0.0386	**11.06**	6.63	3194.2	4.86	52.4	**0.0017**
		MorphInt	**73.0%**	0.0347	11.00	**6.65**	**2851.8**	4.83	52.7	0.0016
	$128k$	MorphDet	43.7%	0.0488	11.03	6.58	3540.9	4.98	**51.1**	**0.0017**
		MorphBias	44.0%	**0.0489**	**11.04**	6.58	3561.2	4.97	51.2	**0.0017**
		MorphInt	**46.4%**	0.0450	11.01	**6.59**	**3288.5**	4.94	51.5	**0.0017**
	$256k$	MorphDet	23.6%	0.0532	**11.01**	6.55	3657.1	5.03	**50.6**	**0.0017**
		MorphBias	23.7%	**0.0533**	**11.01**	6.55	3666.9	5.02	**50.6**	**0.0017**
		MorphInt	**25.0%**	0.0513	11.00	**6.56**	**3555.8**	5.01	50.7	**0.0017**

G.1 Visualisation of Intrinsic Results on Training Data

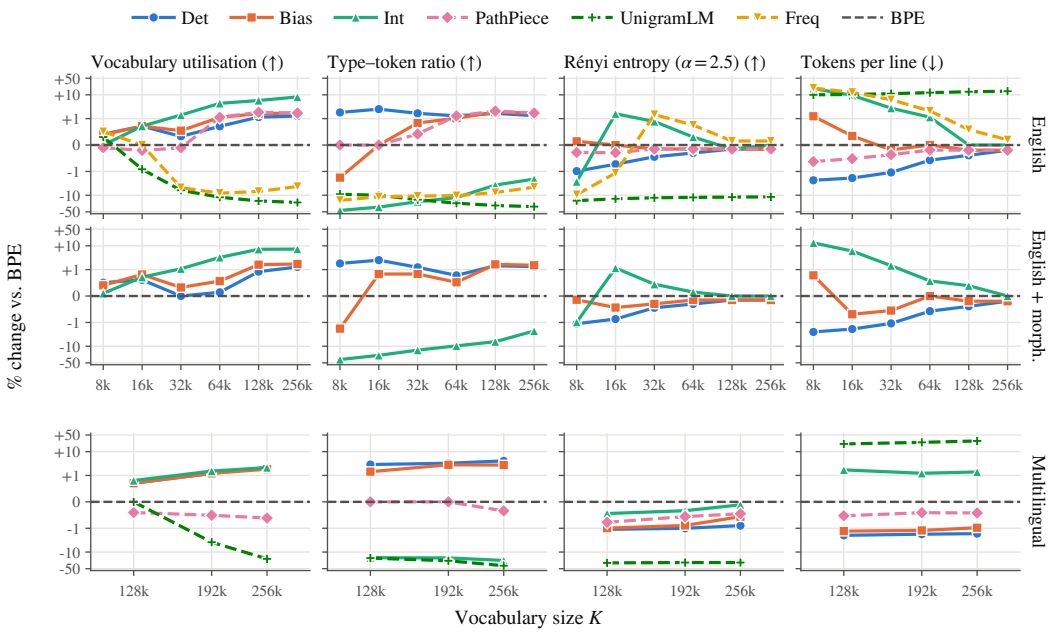


Figure 8: Tokeniser performance on intrinsic metrics as compared to BPE, computed on training data.

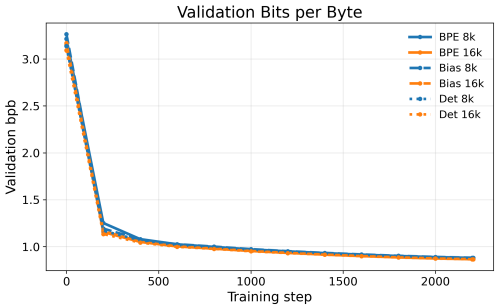
H Raw Language Modelling Performances

Table 6: Language model performances (as BpB and CORE metrics). Best scores within each vocabulary size and depth are bolded. For English results, subscripts give the standard error in units of the last digits, e.g. $.8779_{01} = 0.8779 \pm 0.0001$ and $11.98_{28} = 11.98 \pm 0.28$.

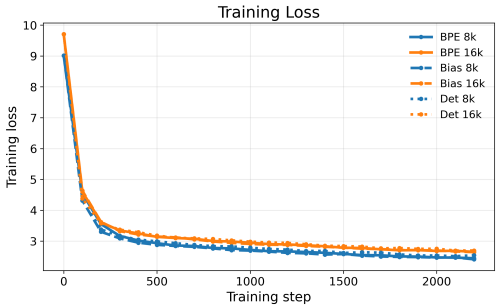
			Validation BpB (↓)						CORE ×100 (↑)					
		Tokeniser	8k	16k	32k	64k	128k	256k	8k	16k	32k	64k	128k	256k
English	Depth 12	BPE	**.8779**$_{01}$	.8642$_{01}$	.8531$_{01}$	.8455$_{03}$	.8416$_{04}$	.8411$_{00}$	11.98$_{28}$	**14.54**$_{44}$	14.91$_{28}$	16.45$_{42}$	16.23$_{33}$	16.26$_{27}$
		PathPiece	**.8779**$_{03}$	**.8641**$_{02}$	.8526$_{01}$	.8443$_{00}$	.8400$_{06}$	**.8397**$_{02}$	12.13$_{71}$	13.43$_{18}$	14.91$_{11}$	15.66$_{33}$	15.97$_{30}$	15.34$_{33}$
		UnigramLM	.8814$_{02}$	.8660$_{02}$	.8549$_{04}$	.8479$_{04}$	.8453$_{01}$	.8467$_{02}$	10.79$_{44}$	13.31$_{11}$	14.23$_{26}$	14.36$_{11}$	15.16$_{49}$	14.18$_{44}$
		Det	.8782$_{04}$	.8645$_{02}$	**.8525**$_{03}$	**.8438**$_{02}$	.8403$_{03}$	.8398$_{02}$	12.94$_{30}$	13.72$_{44}$	**15.13**$_{34}$	**16.50**$_{33}$	**17.03**$_{45}$	15.78$_{38}$
		Bias	.8812$_{01}$	.8655$_{00}$	.8531$_{02}$	.8452$_{01}$	**.8393**$_{04}$	.8402$_{04}$	**13.18**$_{14}$	14.35$_{11}$	14.81$_{50}$	16.39$_{42}$	15.58$_{50}$	**16.27**$_{46}$
	Depth 18	BPE	.7773$_{07}$	.7692$_{01}$	.7623$_{01}$	.7565$_{02}$	.7525$_{01}$	.7528$_{02}$	19.58$_{12}$	**21.06**$_{33}$	**22.30**$_{20}$	22.76$_{53}$	**23.29**$_{26}$	23.04$_{63}$
		PathPiece	.7777$_{06}$	.7697$_{04}$	.7620$_{01}$	.7558$_{02}$	.7517$_{01}$	**.7514**$_{01}$	**20.16**$_{27}$	20.90$_{49}$	21.84$_{108}$	**23.08**$_{41}$	22.98$_{42}$	22.87$_{06}$
		UnigramLM	.7811$_{06}$	.7706$_{02}$	.7627$_{02}$	.7572$_{03}$	.7557$_{09}$	.7572$_{13}$	17.18$_{67}$	19.85$_{38}$	20.22$_{18}$	21.18$_{27}$	22.21$_{19}$	21.25$_{09}$
		Det	**.7772**$_{01}$	**.7689**$_{01}$	**.7606**$_{03}$	**.7553**$_{01}$	**.7513**$_{05}$	.7517$_{03}$	19.13$_{38}$	21.02$_{41}$	21.91$_{06}$	22.59$_{14}$	22.98$_{22}$	21.98$_{47}$
		Bias	.7795$_{03}$	.7703$_{02}$	.7614$_{03}$	.7558$_{02}$	.7515$_{03}$	.7517$_{05}$	19.09$_{19}$	20.99$_{27}$	22.09$_{09}$	22.85$_{35}$	22.81$_{56}$	**23.27**$_{28}$
	Depth 24	BPE	**.7151**$_{03}$	**.7106**$_{06}$	**.7039**$_{01}$	.6995$_{03}$	.6964$_{02}$	.6953$_{02}$	26.02$_{50}$	**26.66**$_{08}$	**28.30**$_{22}$	**28.98**$_{11}$	**29.38**$_{22}$	28.93$_{27}$
		PathPiece	.7156$_{02}$	.7108$_{04}$	.7068$_{10}$	.7008$_{02}$	.6982$_{09}$	.7010$_{36}$	**26.22**$_{48}$	26.46$_{46}$	27.06$_{21}$	28.87$_{29}$	28.68$_{17}$	28.08$_{46}$
		UnigramLM	.7182$_{02}$	.7117$_{00}$	.7060$_{01}$	.7023$_{02}$	.7038$_{19}$	.6994$_{06}$	23.29$_{32}$	25.57$_{17}$	26.20$_{56}$	26.28$_{24}$	26.50$_{16}$	27.32$_{29}$
		Det	.7160$_{01}$	.7107$_{02}$	.7050$_{08}$	**.6985**$_{00}$	**.6951**$_{01}$	.6946$_{00}$	25.84$_{54}$	26.00$_{26}$	28.05$_{11}$	28.60$_{36}$	28.77$_{74}$	**29.41**$_{16}$
		Bias	.7172$_{04}$	.7114$_{04}$	.7051$_{01}$	.7005$_{02}$	.6970$_{05}$	.6957$_{06}$	25.24$_{76}$	26.44$_{18}$	27.78$_{71}$	28.39$_{20}$	28.96$_{31}$	29.26$_{45}$
		MorphBias	-	-	-	.6989$_{02}$	.6952$_{02}$	**.6938**$_{03}$	-	-	-	28.44$_{35}$	29.02$_{66}$	28.68$_{10}$
		MorphDet	-	-	-	.6986$_{03}$	**.6951**$_{01}$	.6943$_{04}$	-	-	-	28.29$_{30}$	28.17$_{46}$	29.04$_{29}$

			Validation BpB (↓)			CORE ×100 (↑)		
		Tokeniser	128k	192k	256k	128k	192k	256k
Multilingual	Depth 24	BPE	**.6174**$_{\pm.0004}$	.6171$_{\pm.0005}$	.6161$_{\pm.0002}$	13.9$_{\pm0.7}$	**15.1**$_{\pm0.1}$	14.8$_{\pm0.6}$
		PathPiece	.6179$_{\pm.0007}$	.6170$_{\pm.0003}$	**.6159**$_{\pm.0005}$	13.7$_{\pm0.0}$	**15.1**$_{\pm0.1}$	**15.6**$_{\pm0.5}$
		UnigramLM	.6252$_{\pm.0003}$	.6239$_{\pm.0004}$	.6198$_{\pm.0001}$	13.0$_{\pm0.1}$	13.2$_{\pm0.5}$	13.4$_{\pm0.6}$
		Det	.6189$_{\pm.0002}$	**.6164**$_{\pm.0006}$	.6164$_{\pm.0002}$	**14.5**$_{\pm0.7}$	14.6$_{\pm0.6}$	14.9$_{\pm0.7}$
		Bias	.6198$_{\pm.0001}$	.6180$_{\pm.0004}$	.6175$_{\pm.0002}$	13.7$_{\pm0.5}$	14.6$_{\pm0.2}$	15.5$_{\pm0.2}$

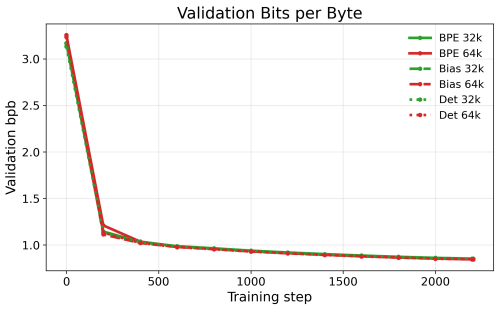
I Loss Across Training for a Subset of Our Models



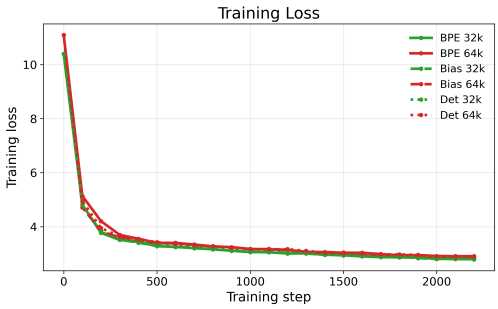
(a) Val BpB for Depth 12 $8k$ and $16k$



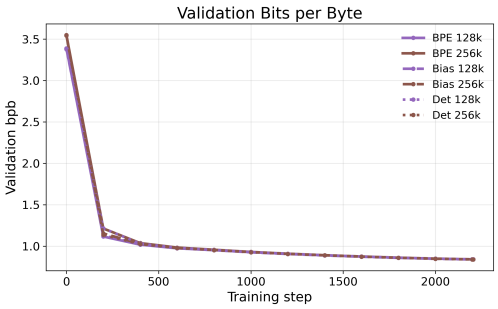
(b) Train loss for Depth 12 $8k$, and $16k$



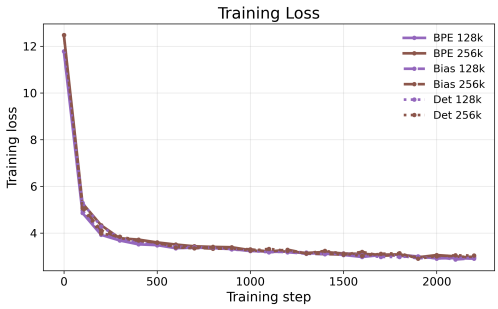
(a) Val BpB for Depth 12 $32k$, and $64k$



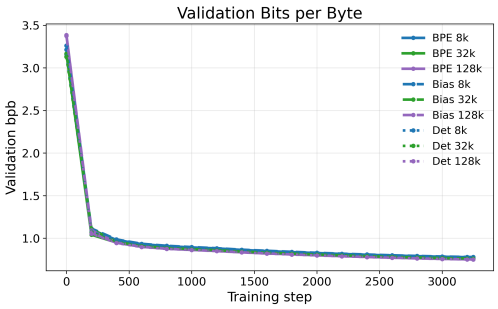
(b) Train loss for Depth 12 $32k$ and $64k$



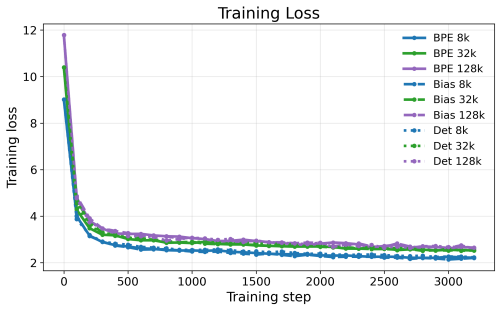
(a) Val BpB for Depth 12 $128k$, and $256k$



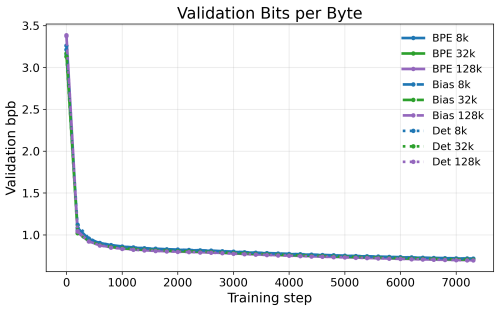
(b) Train loss for Depth 12 $128k$, and $256k$



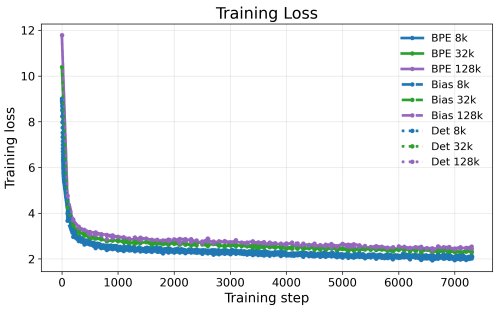
(a) Val BpB for Depth 18



(b) Train loss for Depth 18



(a) Val BpB for depth 24



(b) Train loss for depth 24