GRaMPa: Subword Regularisation by Skewing Uniform Segmentation Distributions with an Efficient Path-counting Markov Model

Thomas Bauwens[↑] and **David Kaczér**[↓] and **Miryam de Lhoneux**[↑]
[↑]LAGoM·NLP, Department of Computer Science, KU Leuven
firstname.lastname@kuleuven.be
[↓]Bonn-Aachen International Center for Information Technology, University of Bonn, Germany
flastname@bit.uni-bonn.de

Abstract

Stochastically sampling word segmentations from a subword tokeniser, also called *subword regularisation*, is a known way to increase robustness of language models to out-of-distribution inputs, such as text containing spelling errors. Recent work has observed that usual augmentations that make popular deterministic subword tokenisers stochastic still cause only a handful of all possible segmentations to be sampled. It has been proposed to uniformly sample across these instead, through rejection sampling of paths in an unweighted segmentation graph. In this paper, we argue that uniformly random segmentation in turn skews the distributions of certain segmentational properties (e.g. token lengths and amount of tokens produced) away from uniformity, which still ends up hiding meaningfully diverse tokenisations. We propose an alternative uniform sampler using the same segmentation graph, but *weighted* by counting the paths through it. Our sampling algorithm, **GRaMPa**, provides hyperparameters allowing sampled tokenisations to skew towards fewer, longer tokens. Furthermore, GRaMPa is single-pass, guaranteeing significantly better computational complexity than previous approaches relying on rejection sampling. We show experimentally that language models trained with GRaMPa outperform existing regularising tokenisers in a data-scarce setting on token-level tasks such as dependency parsing, especially with spelling errors present.

1 Introduction

Before subword tokenisation became the *de facto* standard in NLP, models operated on word-level tokens (Sutskever et al., 2014): i.e., text was split into words, and those words were each converted to one integer identifier serving as lookup indices in an embedding matrix. Tokenising at this level, it is impossible to tell based on the identifiers whether the underlying strings contain similar characters or not, and this absence of surface information implies

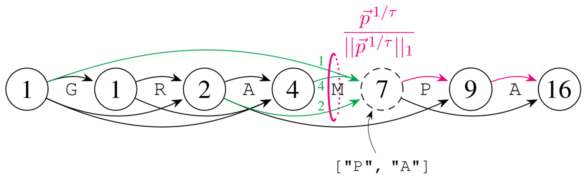


Figure 1 – Directed acyclic graph representing the valid segmentations of the string $s =$ "GRAMPA". Each of the $|s| + 1$ nodes stores the amount of valid paths from itself to the first node. Each arc corresponds to a subword in the vocabulary. So far, the GRaMPa tokeniser has taken the path in pink, giving the tokens P A. It can now move along any of the arcs in green, choosing the tokens M, AM and GRAM with respective probabilities $4/7$ and $2/7$ and $1/7$ when $\tau = 1$, in which case all 16 valid segmentations of s are equally likely to be sampled.

that the corresponding embeddings necessarily represent concepts rather than characters.

With the shift to subword tokenisation (Sennrich et al., 2016), this is no longer necessarily true. A word is segmented losslessly into a sequence of tokens, and those tokens are each mapped to their identifiers. The same identifier can now appear in many different words, in which case the words are guaranteed to share the corresponding substring. As the same identifier is used regardless of the meaning the substring contributes to each word, it's unclear how much semantic information the corresponding embedding contributes; this is supported by the finding that popular subword tokenisers mostly fail to match morphological boundaries (Zhou, 2018; Bostrom and Durrett, 2020; Bauwens and Delobelle, 2024), in which case tokens smaller than a word represent surface-form units rather than meaning-bearing units. For example: Gemma 2's BPE tokeniser (Gemma Team et al., 2024) segments the word *horseshoe* into *horses* and *hoe*.

Subword tokenisers are typically deterministic, sticking to a single segmentation for each word. Given the above framing, this makes little sense: if we want a model to properly understand the tokens it's given, we need to emphasise during training that our opaque identifiers represent characters in the first place, not concepts. Hofmann et al. (2021) and Clark et al. (2022) propose that language models are meant to be *dual-route*, *memorising* some meanings and *composing* others. Deterministic subword tokenisation does allow a model to memorise sequences of identifiers as if they were a single identifier, but doesn't generalise to unseen words. By lack of meaningful tokenisation – exacerbated in multilingual tokenisers (Minixhofer et al., 2023) – such words are presented as unseen sequences of identifiers that merely encode the surface string; to learn to compose their meaning, (part of) training may need to explicitly treat input as mere sequences of characters, without preferring one token sequence that represents it over another. Even for seen words, composing meaningless tokens becomes useful in noisy text (Provilkov et al., 2020).

The promise of *subword regularisation* as coined by Kudo (2018) was exactly this: make tokenisers stochastic, so that for any string s, all its valid segmentations are sampled during training. The tokens of a "valid segmentation" satisfy two constraints: (1) concatenating back into s, and (2) all being part of a given finite vocabulary V. Both make sampling difficult. The first implies having to support $N_V(s) \leq 2^{|s|-1}$ outcomes,[1] easily leading to intractable time or space complexity. The second is usually managed by basing the sampler on the vocabulary's deterministic tokeniser: stochastic ULM (Kudo, 2018) uses the same Viterbi grid as deterministic ULM, except now tracking the $k > 1$ best segmentations; BPE-dropout (Provilkov et al., 2020) uses the same merge rules as BPE, except disabling some at random; MaxMatch-dropout (Hiraoka, 2022) greedily acquires tokens left-to-right like MaxMatch (Bauwens, 2023; Uzan et al., 2024), except ignoring some at random. Since the original inference is naturally constrained by the vocabulary, augmenting it stochastically is a straightforward way to satisfy the second constraint.

Cognetta et al. (2024) recently observed that these approaches all yield extremely skewed distributions, so that still only a handful of segmentations occur in practice. They propose an algorithm that returns every valid segmentation with equal probability (see § 2). Yet, this implicitly assumes that any sampled set of segmentations is equally informative for model training. We argue against this. For example, say we sample 5 segmentations for the string *tokenisation*. We hypothesise that a model will learn more from examples using

{*t oken isat ion*; *tok eni sation*; *to ken is ati on*; *t o ken isat io n*; *toke ni sa ti on*}

than from examples using

{*to k e n i s a t i on*, *t ok e n i s a t io n*, *t o ke n i s a ti o n*, *t o k en i s at i o n*, *t o k e ni sa t i o n*}

because segmentations in the latter set have a certain redundancy to them. Conversely, in the former set, the longer tokens cause embeddings to be more specialised and hence more semantically laden, a larger variety of previously unassociated tokens is brought into the same context with every segmentation, and the model is forced to reason about the meaning of the same word with a variable amount of embeddings to store its results in.

Problematically, although both sets are equally probable in uniform sampling ($P = N_V(s)^{-5}$), segmentations and sets that *look like* the latter set are more probable than the former because *there exist more of those*. That is, redundancy is more likely when all segmentations are equally likely.

In short: rather than aiming for a diversity of *segmentations*, it may be better to aim for diversity of the *properties* that distinguish them (e.g. the amount of tokens) by skewing away from uniformity, which already improved Kudo's downstream results when allowing a larger set of segmentations. Further, uniform sampling doubles the average amount of tokens produced (m in Table 7), quadrupling the cost of transformer attention (Vaswani et al., 2017); Cognetta et al.'s algorithm unfortunately does not easily allow for skewing towards longer tokens, and itself needs an indeterminate amount of processing time. In this paper, we hence:

- Analyse the distributional characteristics of uniform sampling (§ 3.1);
- Propose a single-pass quadratic tokenisation algorithm, **GRaMPa**, that samples random segmentations of a string such that every segmentation is equally likely and every token is in the vocabulary, based on counting paths through the segmentation graph (§ 4);
- Show how to skew this sampling algorithm away from uniformity with a single temperature hyperparameter (§ 4.3), and how to softly enforce a minimum token length (§ 4.4);
- Train DeBERTa models on GRaMPa's tokenisations for several combinations of hyperparameters and subword vocabularies in a data-scarce setting, showing improvement on dependency parsing (§ 6.1), especially when introducing spelling errors after training (§ 6.2).

[1] $|s| - 1$ inter-character positions can be a boundary or not.

2 Related Work

Segmentation DAG Kudo (2018)'s ULM introduces a procedure for creating a subword vocabulary and an associated unconditional[2] probability distribution across it. It does inference by constructing a Viterbi grid derived from the directed acyclic graph (DAG) representing all valid segmentations of an input string s under the vocabulary V. The DAG has $|s|+1$ nodes (indexed from 0 to $|s|$) with an arc from node i to node j if and only if the token $s_{i:j}$ (indexed *exclusively*, i.e. $s_{i:j} = s_i s_{i+1} \dots s_{j-1}$) exists in V. Figure 1 shows an example of such a DAG. There is a 1:1 correspondence between valid segmentations of the string $s_{i_1:i_2}$ and paths in this graph from node i_1 to node i_2, and hence the set of all paths through the full graph is isomorphic to the set of all valid segmentations of s. ULM also assigns a weight to each arc (i,j) equal to the unigram probability $P(s_{i:j})$. This was generalised by He et al. (2020) to the conditional probability $P(s_{i:j} \mid s_{0:i})$.

In ULM, the k most probable paths through this DAG, i.e. the k segmentations whose tokens have the largest joint probability, can be obtained in $\Theta(k^2|s|^2)$ time with a Viterbi grid taking only $\Theta(k|s|)$ space. The tokeniser can then sample those proportionally to their probabilities (optionally after a transformation like Eq. 4), ignoring all other valid segmentations. To consider more segmentations, one could use a k-worst sampler and/or increase k, but sampling across all valid segmentations requires $k = O(2^{|s|-1})$ time and space.

Rejection sampling One way to sample uniformly across all *valid* segmentations is to repeatedly sample uniformly across *all* segmentations, rejecting the results until every token is in V. Unconstrained sampling can be done by choosing segmentation boundaries (e.g. § 3.2) or token lengths like Hiraoka and Iwakura (2024).[3]

Cognetta et al. (2024) do rejection sampling on already-valid segmentations to make their distribution uniform. They first construct the segmentation DAG described above in $\Theta(|s|^2)$ time. To generate a segmentation, they start at node 0 and iteratively sample an outgoing arc *without preference* to get to the next node, until node $|s|$. If node i has $\delta_o(i)$ outgoing arcs, then each path $\boldsymbol{i} = [i_0, i_1, \dots, i_m]$ resulting from these local arc samples is a Markov chain (Markov, 1907, §5) with probability

$$P_{\text{Markov}}(\boldsymbol{i}) = \prod_{i \in \boldsymbol{i}} \frac{1}{\delta_o(i)}. \qquad (1)$$

To make sure every path is emitted with equal probability, samples are only kept with probability $P_{\text{accept}}(\boldsymbol{i}) = \varepsilon / P_{\text{Markov}}(\boldsymbol{i})$, where ε is a small constant, chosen as big as possible[4] without ever making $P_{\text{accept}}(\boldsymbol{i}) > 1$, meaning $\varepsilon \leq \min_{\boldsymbol{i}'} P_{\text{Markov}}(\boldsymbol{i}')$. Together, the probability of being emitted is that of being sampled and then being accepted, which is $P_{\text{Markov}}(\boldsymbol{i}) \cdot P_{\text{accept}}(\boldsymbol{i}) = \varepsilon$ regardless of the path $\boldsymbol{i}$.

We prove in § A.1 that applying the same Markovian sampling to ULM's DAG produces a *different* distribution than ULM's joint. In § B, we fix that.

Multiplexing Cognetta et al. also *multiplex* between deterministic and stochastic tokenisation with a global hyperparameter p. In the dual-route paradigm of Hofmann et al. (cfr. § 1), p can be seen as the proportion of compositional learning that happens in training, the rest being memorisation.

Shorter input Models such as CANINE (Clark et al., 2022), Charformer (Tay et al., 2022) and hourglass transformers (Nawrot et al., 2023) all have provisions in place to reduce the amount of tokens L passed to the bulk of the model, since attention has time complexity $\Theta(L^2)$. Task scores do start to decrease when reducing the amount of tokens too much (Schmidt et al., 2024).

Input augmentation As shown by Provilkov et al. (2020), one benefit of subword regularisation is increased robustness to perturbed input (e.g. text with spelling errors). There exist other input augmentations that provide robustness to other perturbations: for example, the Morpheus system by Tan et al. (2020) identifies nouns, verbs and adjectives and adversarially perturbs their inflection, such that the model gets used to correct word forms with incorrect placements. To apply this augmentation requires access to a part-of-speech tagger trained on supervised data. This augmentation is explicitly lossy, unlike subword regularisation.

Sampling for marginalisation Considering all $O(2^{|s|-1})$ segmentations is also done for *marginalising* the loss of downstream models across segmentations of their text input. He et al. (2020) achieve

[2]This can be seen as an N-gram language model for the trivial context-independent case $N = 1$, and the resulting tokeniser is named after this unigram language model.

[3]Do note however that these authors don't sample uniformly, and circumvent having to reject samples by generating token embeddings on-the-fly using a character-BiLSTM.

[4]A safe choice is $\varepsilon = \prod_{i=0}^{|s|-1} \frac{1}{\delta_o(i)}$ because no path can visit more nodes than all nodes.

this exactly with a Viterbi sum across all segmentations in $\Theta(|s|^2)$ time by using characters as context. With tokens as context, it is only possible to take an approximative sum over fewer segmentations; Cao and Rimell (2021) thus use *importance sampling*, where the goal is to average estimates of the marginal across several highly probable segmentations. They sample non-uniformly according to ULM probabilities, while Chirkova et al. (2023) use the model being trained to come up with a segmentation and its probability. Interestingly, although marginalisation and regularisation are not the same, Song et al. (2024) find that the former only works when combined with the latter.

3 Distributional properties of segmentations

3.1 Uniform sampling causes binomiality

As argued in § 1, the underlying goal of sampling segmentations is actually to get a uniform distribution of their *properties*. Yet, basic properties like the amount of tokens m in a segmentation, the length of the individual tokens ℓ, and quantities derived from m and ℓ, are *not uniformly distributed* given a uniform distribution over segmentations.

Sampling a segmentation uniformly is equivalent[5] to doing $|s|-1$ Bernoulli experiments, one on each inter-character boundary, with $p = \frac{1}{2}$ of putting a boundary there. The amount of tokens is 1 more than the amount of segmentation boundaries, and since the latter is binomially distributed, we get $m \sim 1 + \text{Binom}(|s|-1, \frac{1}{2})$.

Furthermore, token length ℓ is nearly geometrically distributed with $p = \frac{1}{2}$ in uniformly sampled segmentations, which we prove in § A.3. When $\ell \sim \text{Geom}(\frac{1}{2})$, tokens with length 1 appear twice as much as tokens with length 2, which appear twice as much as those with length 3, etc.

Figure 2 (in § E) illustrates these distributions for a uniform sample across segmentations of one word. It also shows the distribution of derived quantities across a corpus, namely the characters-per-token ratio $R = |s|/m$ (Dagan et al., 2024) and the ratio $\mathcal{S} = \frac{m-1}{|s|-1}$ which we dub *segmentality*,[6] being exactly 0 for word-level tokenisers and exactly 1 for character-level tokenisers, unlike $\frac{1}{R}$.

[5]We assume that the vocabulary is infinitely large here, to get upper bounds in closed form.

[6]The distribution of $R \in [1, |s|]$ can't be expressed explicitly. We do know its expected value approaches 2 (§ A.4). We also know $\mathcal{S} \in [0, 1]$ and $\mathcal{S} \sim \frac{1}{|s|-1}\,\text{Binom}(|s|-1, \frac{1}{2})$.

3.2 Rejection sampling is hard to skew

One way of sampling segmentations uniformly with rejection sampling goes as follows: first pick a random integer between 0 and $2^{|s|-1}-1$, then convert it to bits, and then overlay these as boundaries on the $|s|-1$ inter-character positions (see also § C). Keep retrying until all tokens exist in V. Since this is equivalent to choosing $|s|-1$ bits with $p = \frac{1}{2}$ chance of a `1`, it is possible to skew this rejection sampler towards larger tokens by reducing p, the proclivity to segment. Unfortunately, the amount of retries T is geometrically distributed: let $N_V(s)$ be the amount of valid segmentations of s given V, then for $p = \frac{1}{2}$, the success rate of a sample is $p_a = N_V(s)/2^{|s|-1}$ and the expected amount of tries is $\mathbb{E}[T] = 1/p_a = 2^{|s|-1}/N_V(s)$. This is vulnerable to adversarial attacks: a malicious user could input a nonsense string s which can only be split into characters or very small tokens, and hence $N_V(s) \approx 1$ and $\mathbb{E}[T] \approx 2^{|s|-1}$, causing the tokeniser to hang while it looks for a needle in a haystack. Decreasing p only makes this worse.

The graph-based approach of Cognetta et al. (2024) is not vulnerable to attacks,[7] but is difficult to consistently skew away from uniformity. This is because (1) the uniformity comes not from the graph but from the rejection step afterwards, and (2) the probabilities inside the graph, $\frac{1}{\delta_o(i)}$ in Eq. 1, are entirely local with no awareness of the rest of the graph. There is no signal that allows skewing them with a consistent global effect.

4 GRaMPa tokeniser

We now introduce **GRaMPa**, a **G**raph-based segmentation **Ra**ndomiser that walks along a **M**arkov chain of **Pa**th counts through the graph described in § 2, and does so in one pass (i.e. without rejection). We first show how this tokeniser samples segmentations of a string uniformly, and then demonstrate how to skew it to correct for the effects in § 3.1.

4.1 Construction

GRaMPa starts by constructing a Viterbi grid containing $|s|+1$ cells, representing the DAG nodes (see Figure 1). Each cell j stores a path counter c_j and an initially empty list b_j of *backpointers* representing the DAG arcs, with one backpointer to each cell $i < j$ for which $s_{i:j} \in V$. To fill the grid,

[7]In fact, a nonsense string would have a graph with very few arcs, a very high ε, and hence any sampled segmentation would have a very high acceptance rate.

Algorithm 1 *GRaMPa* Markov graph construction

```
1: function MAKEGRAPH($s$, $V$, $f_\tau$, $\ell_{\min}$)
2:     $c \leftarrow$ ARRAY$(|s|+1, 0)$
3:     $b \leftarrow$ ARRAY$(|s|+1,$ [ ]$)$
4:     $p \leftarrow$ ARRAY$(|s|+1,$ [ ]$)$
5:     $c_0 \leftarrow 1$
6:     for $i \in 0 \ldots |s|-1$ do
7:         for $j \in i+1 \ldots |s|$ do
8:             if $j - i \geq \ell_{\min}$ or $|b_j| = 0$ then
9:                 if $s_{i:j} \in V$ then
10:                    $c_j \leftarrow c_j + c_i$
11:                    $b_j \leftarrow$ APPEND$(b_j, i)$
12:                    $p_j \leftarrow$ APPEND$(p_j, c_i)$
13:        for $k \in 0 \ldots |p_i|-1$ do
14:            $p_{i,k} \leftarrow p_{i,k}/c_i$
15:        $p_i \leftarrow f_\tau(p_i)$
16:    return $(b, p)$
```

GRaMPa walks from node $i = 0$ to node $i = |s|$. When the walk arrives in node i, it is assumed[8] that (1) the current counter c_i gives the exact amount of paths that exist through the DAG from node 0 to node i, and (2) the current list of backpointers b_i is complete. These two facts are then exploited:

1. For the nodes $j > i$ that can be reached from i, it is true that any path that arrives in i can be extended to arrive in j. Hence, we increment c_j by c_i and keep a backpointer to i in b_j.
2. The fraction of paths that arrive in i through one of its backpointers $b_{i,k}$ is given by $c_{b_{i,k}}/c_i$.

Once finished, the grid cells are equivalent to this *first-order Markov model* over the DAG nodes:

$$P(\text{node}_{t-1} = i \mid \text{node}_t = j) = \frac{c_i}{c_j}\,\mathbb{1}\{s_{i:j} \in V\}. \quad (2)$$

Constructing the grid takes $\Theta(|s|^2)$ in space and time. Unlike ULM, GRaMPa needs to keep the entire segmentation DAG in memory.

4.2 Sampling

Starting at node $|s|$, GRaMPa looks backwards and iteratively *samples a preceding node using the fraction of paths that came in through that node as its probability*, until it reaches node 0 (see again Figure 1). Like all first-order Markov models, it only sees the current and next node, yet the resulting segmentation is globally valid and uniform, having probability $\frac{1}{N_V(s)}$. The intuition behind why this samples paths uniformly is that if we *had* a way to pick a random path through the graph uniformly, we would on average expect that path to arrive at a node twice as much through one arc as another if twice as many paths arrived through that one arc as the other. A formal proof is given in § A.2.

Sampling this way takes exactly one $O(|s|)$ pass. This is contrary to rejection sampling, which has *no limit* on the worst-case amount of passes it takes before one is accepted, and as we show in § A.5, the expected runtime of Cognetta et al. (2024)'s rejection sampler scales *superexponentially* with $|s|$ in strings where all segmentations are valid (which is all strings in the limit of an infinite vocabulary).

Algorithm 2 *GRaMPa* Markov graph sampling

```
1: function SAMPLEGRAPH($s$, $b$, $p$)
2:     $t \leftarrow$ [ ]
3:     $i \leftarrow |s|$
4:     $j \leftarrow |s|$
5:     while $j \neq 0$ do
6:         $k \leftarrow$ SAMPLE$(p_j)$
7:         $i \leftarrow b_{j,k}$
8:         $t \leftarrow$ PREPEND$(t, s_{i:j})$
9:         $j \leftarrow i$
10:    return $t$
```

4.3 Skewing

In § 3, we saw that the most common amounts of tokens produced by uniform sampling ($m \approx \frac{|s|}{2}$) are unfavourably large. Because GRaMPa's graph produces uniform samples by choosing inbound arcs proportional to the amount of paths that come in through them, we hypothesise a causal relationship between the two: *the fraction of excessively long paths carried by an arc is bigger when it carries more paths*. To skew the uniform distribution to more desirable segmentations, then, we should choose arcs with *higher* path counts *less*.

We can do this by applying a smoothing transformation across the Markov probabilities $\vec{p}$ of the arcs coming into each node, and preferably one that allows controlling the amount of smoothing with a hyperparameter. One candidate for this is softmax with temperature $\tau \in \mathbb{R}_0^+$, defined as

$$\text{softmax}_\tau(\vec{p}) = \frac{e^{\vec{p}/\tau}}{||e^{\vec{p}/\tau}||_1} \quad (3)$$

where all the operations are elementwise. Problematically, there is no τ that represents the baseline,

[8] More formally, this is an inductive hypothesis.

since every value for τ changes at least one element of $\vec{p}$. *Power normalisation*, defined as

$$\mathrm{PN}_\alpha(\vec{p}) = \frac{\vec{p}^\alpha}{||\vec{p}^\alpha||_1} = \frac{e^{\ln(\vec{p}^\alpha)}}{||e^{\ln(\vec{p}^\alpha)}||_1} = \frac{e^{\alpha\ln(\vec{p})}}{||e^{\alpha\ln(\vec{p})}||_1} = \mathrm{softmax}_{1/\alpha}(\ln(\vec{p})) \tag{4}$$

does not have this problem since $\mathrm{PN}_1(\vec{p}) = \vec{p}$. For $\alpha \in (0, 1)$, or equivalently $1/\alpha = \tau \in (1, \infty)$, this makes arcs with lower path counts appear more similar to those with higher path counts. They become indistinguishable for $\tau \to \pm\infty$, where the DAG is essentially unweighted. Finally, for $\alpha \in (-\infty, 0)$ and $\tau \in (-\infty, 0)$, arcs with lower path counts are explicitly chosen *more* than those with higher path counts. Changing this local τ has a smooth global effect (see Figure 5 in §E).

Note that although ULM (Kudo, 2018) also works with a DAG and also applies a power normalisation, it does so over a different set of probabilities: GRaMPa applies it to the $|s|$ sets of *token* probabilities that reach each node, whereas ULM applies it to the single set of k best *path* probabilities *after* sampling them from the graph.

4.4 Soft token length minimum

We also propose a way to have each token in a segmentation be at least ℓ characters long, but don't enforce this when it makes the segmentation impossible. While constructing the grid, node i could always start checking for reachable nodes at $j = i+\ell$ rather than $j = i + 1$. If at that moment there existed a node $j < i + \ell$ where no arcs had arrived yet, no more opportunities would arise to add an inbound arc to it. Meanwhile, arcs could still come from that node j. This means that GRaMPa could arrive at j while sampling backwards, but have no arc to escape from it.

Instead, we not only check $j = i + \ell$ for reachability, but also those $j < i + \ell$ with no inbound arcs yet. This way, if such a j is reachable at all, it will end up with exactly one arc, namely the longest arc shorter than ℓ reaching it. If now all characters are in the vocabulary (which is possible with a byte-mapping preprocessor), there can be no unreachable nodes, meaning the in-degree and out-degree of every node is guaranteed to be $\geq$1. This prevents any dead-end sampling paths and ensures at least one valid segmentation to exist.

The final algorithm for constructing the Markov graph, including a skewing transformation f_τ and a minimum token length $\ell_{\min}$, is given in Algorithm 1. The sampler is in Algorithm 2.

Algorithm 3 Multiplexed *GRaMPa* tokenisation

```
1: function TOKENISE($s$, $f_\tau$, $\ell_{\min}$, $p$, $T$)
2:     if RAND $< p$ then
3:         $V \leftarrow$ GETVOCAB($T$)
4:         $(b, p) \leftarrow$ MAKEGRAPH($s, V, f_\tau, \ell_{\min}$)
5:         return SAMPLEGRAPH($s, b, p$)
6:     else
7:         return $T(s)$
```

4.5 Directionality

As described, GRaMPa constructs the DAG and path counts from left to right, and samples from right to left (R2L). The reverse, i.e. starting construction on the right and sampling from the left (L2R), also yields a uniform sampler when $\tau = 1$. When skewed with $\tau \neq 1$, these two implementations diverge. One effect is that wherever sampling stops (the start of the string for R2L, the end for L2R), a smaller token is expected since the largest possible token size shrinks as sampling progresses.

4.6 Multiplexing

Finally, as mentioned in §1 and §2, downstream performance might be improved further by promoting a mixture of memorisation and composition. This can be achieved by *multiplexing* two tokenisers, using GRaMPa to stochastically tokenise each word with a probability p and an existing deterministic tokeniser T otherwise. The multiplexing rate $p \in [0, 1]$ is then a hyperparameter of the tokeniser. Algorithm 3 brings everything together.

5 Experiments

As previous work suggests that subword regularisation most significantly improves downstream performance in low-resource scenarios (Kudo, 2018; Cognetta et al. 2024; Song et al., 2024), we limit both the model and dataset size in all experiments.

5.1 Model and data setup

Model We pre-train and fine-tune DeBERTa (He et al., 2021) masked language models (MLMs) with $L = 6$ layers and embedding size $H = 512$, following the results of Turc et al. (2019) showing diminishing returns for bigger values. We also follow them in using $H/64$ attention heads and $4H$ hidden feed-forward neurons per layer.

We limit the context length to 1024 tokens and use 1024 relative positional embeddings, meaning all tokens further apart than $k = \frac{1024}{2} = 512$ see each other as if only 512 tokens apart. More hyper-parameters are given in § D.

Pre-training We pre-train with the single-term loss function of RoBERTa (Liu et al., 2019) with a 15% token masking rate, on English data from the SlimPajama corpus (Soboleva et al., 2023). To simulate a data-scarce setting, we only use the first[9] 50k examples. We *pack* tokens of consecutive examples until they fill the context length (Zhao et al., 2024) to avoid wastefully processing pad tokens. We build on the HuggingFace `transformers` library (Wolf et al., 2020) for training (see § 7).

Fine-tuning We fine-tune on a subset of sentence-level tasks in GLUE (Wang et al., 2019), and on three token-level tasks: named-entity recognition (NER) on CoNLL-2003 (Tjong Kim Sang and De Meulder, 2003), and part-of-speech tagging (PoS) and dependency parsing (DP) on $\text{UD}_{\text{en\_ewt}}$ (Silveira et al., 2014). For the sequence-level tasks, we mean-pool the token embeddings and apply a head consisting of an $H \times H$ dense+tanh layer followed by a dense+softmax layer. We use a simpler dense+softmax head for PoS and NER (which need less reasoning), applied respectively to the last and first token of a word (Ács et al., 2021). For DP, we use a biaffine head (Dozat and Manning, 2017).

We also test increased robustness to spelling errors (as shown by Provilkov et al. (2020) in machine translation, even with no augmentation to the pre-training data) with three extra fine-tuning experiments for DP: training with or without spelling errors, and testing with or without spelling errors. We perturb each space-surrounded string with 15% chance: then, like Provilkov et al., we select one random character in the string, and then with equal odds we (1) drop it, or pick a random lowercase letter to (2) insert before it or (3) substitute it with.

5.2 Tokeniser setup

Preprocessing Each tokeniser is preceded by two preprocessors $\mathcal{P}_1$ and $\mathcal{P}_2$ that progressively split the input text into smaller pretokens. $\mathcal{P}_1$ applies NFKC normalisation, then splits on (and removes) whitespace, and isolates non-hyphen punctuation. Then, $\mathcal{P}_2$ isolates English contractions, maps the resulting strings to their UTF-8 bytes represented as characters, prefixes each string with a boundary marker, and finally isolates digits and hyphens.

Vocabularies We construct two vocabularies of size $|V| = 32768$ with the SentencePiece package (Kudo and Richardson, 2018): one with ULM and one with BPE. We vocabularise on the first 3 000 000 examples of SlimPajama.

We don't tune the vocabulary size, because this would lead us too far. How much the segmentation graphs change for smaller vocabularies is quantifiable and visualised in Figure 9 of § E.

Baselines Since Kudo (2018) and Provilkov et al. (2020) show that stochastic ULM and BPE-dropout outperform the deterministic variants, we use these subword-regularised tokenisers as strong baselines. For BPE-dropout, we use a dropout $p_d = 0.1$ as tuned on machine translation by Provilkov et al. For ULM, we sample across the 64-best segmentations like Kudo, and choose $\alpha = 0.15$ for smoothing ULM's segmentation probabilities with Eq. 4, as it causes the *regularisation rate*, i.e. the fraction of segmentations that differ from the maximally likely one, to be balanced at 50% (see Figure 6).

GRaMPas We experiment with 3 different temperatures (§ 4.3): $\tau = 1$ to represent the uniform case, $\tau = +5$ for moderate skew, and $\tau = -10$ for heavy skew.[10] Figure 3 and Figure 4 show the extent to which m and $\mathcal{S}$ change for these temperatures. We combine each of these with a soft length minimum (§ 4.4) of $\ell_{\text{min}} = 1$ and $\ell_{\text{min}} = 2$. We use the L2R implementation (§ 4.5) as the last morpheme in English words tends to be the smallest.

Multiplexing During model pre-training *and* fine-tuning, each pretoken of $\mathcal{P}_1$ is tokenised (after applying $\mathcal{P}_2$) by GRaMPa with probability p and otherwise with the deterministic tokeniser from which GRaMPa's vocabulary was obtained (ULM with $k = 1$ or BPE with $p_d = 0$). We keep p fixed for both cases, and match it to the regularisation rate of the ULM baseline, so $p = 0.5$.

Although this is about 5× the regularisation rate of the BPE-dropout baseline (which is $p \approx 0.11$ for $p_d = 0.1$, according to Figure 6 in § E), Cognetta et al. showed that with a BPE vocabulary, raising the rate of uniform sampling from $p = 0.1$ to $p = 0.25$ *increased* performance. This suggests

[9]Note that SlimPajama comes shuffled by default, so there should be no difference between taking the first 50k examples or a random sample without replacement of 50k examples.

[10]Counterintuitively, *local uniformity* causes *global skew* and vice versa. Hence, because temperature in Eq. 4 is local, the skew rises for τ being $0^+ \to +\infty$, is identical at $+\infty$ and $-\infty$, and rises even more for $-\infty \to 0^-$ (see Figure 8).

				Token-level						Sequence-level						
				PoS	NER	DP				SST-2	QQP	MRPC	RTE	WNLI		
				Acc	F_1	UAS	LAS	UCM	LCM	Acc	F_1	Acc	Acc	Acc	$\bar{\%}$	$\bar{\Delta}$
BPE	BPE-dropout		$p_d = 0.1$	**95.3**	81.1	81.6	77.2	49.4	39.4	**89.9**	74.4	69.1	53.4	46.5	68.85	
	GRaMPa	$\ell_{\text{min}} = 1$	$\tau = 1.0$	92.5	78.8	79.7	73.8	44.7	34.2	83.5	72.0	69.9	53.4	50.7	66.66	−2.19
			$\tau = 5.0$	93.2	80.1	81.9	77.1	48.7	38.3	86.0	69.6	**70.6**	54.2	50.7	68.21	−0.64
			$\tau = -10.0$	93.3	78.8	79.9	74.9	46.6	37.0	84.1	69.1	68.4	43.0	50.7	65.97	−2.88
		$\ell_{\text{min}} = 2$	$\tau = 1.0$	94.6	81.0	85.4	81.4	53.9	44.8	85.2	75.4	69.6	52.0	49.3	70.25	+1.40
			$\tau = 5.0$	94.1	**81.5**	81.0	76.4	47.3	38.4	83.1	73.2	69.1	52.3	45.1	67.42	−1.43
			$\tau = -10.0$	93.1	79.8	78.8	73.7	46.5	36.2	83.0	73.2	70.1	**54.9**	50.7	67.27	−1.58
ULM	ULM	$k = 64$	$\alpha = 0.15$	92.7	79.3	82.1	77.3	49.1	39.8	82.6	68.1	69.9	53.4	50.7	67.71	
	GRaMPa	$\ell_{\text{min}} = 1$	$\tau = 1.0$	93.1	76.8	83.1	78.4	48.7	39.6	86.4	70.7	68.9	53.4	49.3	68.03	+0.32
			$\tau = 5.0$	93.2	79.3	83.9	79.6	50.1	41.0	84.7	71.5	69.9	52.7	43.7	68.16	+0.44
			$\tau = -10.0$	92.7	77.6	84.0	79.8	50.3	41.6	84.7	72.2	68.6	50.5	**53.5**	68.69	+0.98
		$\ell_{\text{min}} = 2$	$\tau = 1.0$	94.7	80.9	86.0	**82.4**	54.5	45.4	87.4	74.4	69.4	50.5	47.9	70.31	+2.60
			$\tau = 5.0$	94.6	79.0	**86.1**	82.3	**55.6**	**46.5**	86.4	75.0	70.3	53.1	46.5	70.48	+2.77
			$\tau = -10.0$	94.3	80.6	85.7	82.2	54.5	45.6	83.7	**77.6**	69.4	51.3	50.7	**70.51**	**+2.80**

Table 1 – Fine-tuning results. Each row corresponds to a separate encoder-only transformer with DeBERTa architecture pre-trained with an MLM objective on a 50k subset of the SlimPajama dataset, and then fine-tuned separately on each of the downstream tasks, totalling $14 \times 8 = 112$ fine-tuning runs (see § D.2 for details). Each row first shows the vocabulary, inference algorithm, and hyperparameter values of the model's tokeniser. We report overall accuracy for PoS, span F_1 for NER, and binary accuracy or F_1 otherwise; for DP, we report the unlabelled and labelled attachment score of relations, as well as unlabelled and labelled complete match rates of dependency trees (Li et al., 2024). All metrics range from 0% to 100%. Higher is better. $\bar{\Delta}$ is the average deviation across all tasks from the baseline model with the same vocabulary.

				UAS				LAS				UCM				LCM					
				tr/te	tr/te	tr/te	tr/te	tr/te	tr/te	tr/te	tr/te	tr/te	tr/te	tr/te	tr/te	tr/te	tr/te	tr/te	tr/te	$\bar{\%}$	$\bar{\Delta}$
BPE	BPE-dropout		$p_d = 0.1$	74.1	75.4	80.4	81.6	67.3	69.6	76.4	77.2	40.3	43.5	48.6	49.4	31.0	34.3	41.6	39.4	58.13	
	GRaMPa	$\ell_{\text{min}} = 1$	$\tau = 1.0$	76.0	76.5	79.2	79.7	68.6	70.0	73.9	73.8	41.8	42.4	45.1	44.7	31.3	32.6	35.2	34.2	56.56	−1.57
			$\tau = 5.0$	77.1	79.5	83.0	81.9	70.0	73.5	78.3	77.1	41.7	45.1	49.5	48.7	30.7	35.0	39.7	38.3	59.32	+1.19
			$\tau = -10.0$	77.4	79.1	83.5	79.9	70.4	72.8	79.3	74.9	42.5	46.3	51.0	46.6	31.7	35.4	41.9	37.0	59.35	+1.21
		$\ell_{\text{min}} = 2$	$\tau = 1.0$	73.6	74.4	**84.9**	85.4	66.4	68.2	80.7	81.4	41.2	41.7	52.0	53.9	31.0	33.6	43.6	44.8	59.81	+1.68
			$\tau = 5.0$	73.2	76.5	78.6	81.0	66.0	70.8	73.5	76.4	39.9	44.2	46.0	47.3	29.4	35.2	37.0	38.4	57.09	−1.04
			$\tau = -10.0$	73.0	75.0	83.9	78.8	65.6	68.6	79.6	73.7	40.2	42.1	52.0	46.5	30.0	31.8	43.1	36.2	57.50	−0.63
ULM	ULM	$k = 64$	$\alpha = 0.15$	77.1	79.1	81.3	82.1	69.8	72.9	75.9	77.3	42.3	44.8	46.6	49.1	30.9	34.3	36.2	39.8	58.72	
	GRaMPa	$\ell_{\text{min}} = 1$	$\tau = 1.0$	76.3	77.2	82.1	83.1	69.2	70.7	77.4	78.4	41.8	44.0	48.4	48.7	31.5	34.0	38.7	39.6	58.83	+0.11
			$\tau = 5.0$	78.0	80.1	83.2	83.9	71.0	74.5	78.6	79.6	**43.9**	45.9	49.4	50.1	33.1	36.1	39.4	41.0	60.48	+1.76
			$\tau = -10.0$	76.2	79.6	83.3	84.0	69.1	74.0	79.1	79.8	40.0	45.9	50.1	50.3	30.4	36.6	41.4	41.6	60.09	+1.36
		$\ell_{\text{min}} = 2$	$\tau = 1.0$	76.2	79.8	84.6	86.0	69.3	74.1	80.7	**82.4**	41.6	47.3	52.5	54.5	31.2	37.6	43.5	45.4	61.66	+2.94
			$\tau = 5.0$	**78.8**	81.6	84.8	**86.1**	**71.8**	76.3	**81.1**	82.3	43.3	**48.1**	**53.4**	**55.6**	32.5	37.6	**44.4**	**46.5**	**62.76**	**+4.04**
			$\tau = -10.0$	78.3	**81.9**	83.4	85.7	71.5	**76.7**	78.9	82.2	43.8	47.5	51.0	54.5	**33.2**	**38.1**	41.6	45.6	62.11	+3.39

Table 2 – Fine-tuning results on DP when applying LD1 typos to 15% of all words in the different splits of the dataset. A red line under te means typos were applied to the validation and test set, while tr means typos were applied to the train set.

that raising it even higher may improve downstream performance even further. We hypothesise that it is the low *quality* of BPE-dropout's regularisation that makes the optimal dropout rate one with a low *quantity* of regularisation.

A note about Rényi efficiency Recent work by Zouhar et al. (2023) proposed tuning tokeniser hyperparameters for use in LMs by maximising Rényi efficiency E_α of the vocabulary. Cognetta et al. (2024, bis) found two somewhat contrived tokenisers whose hyperparameters caused E_α to rise monotonically with worse modelling results, making E_α an unsuitable objective to optimise them properly. We argue below that this holds for all our hyperparameters – a real-life case of needing a tuning criterion orthogonal to Rényi efficiency.

6 Results

Table 1 shows results on downstream tasks. Table 2 shows results for DP after perturbing its texts. Appendix § E contains both tables in relative form.

6.1 Fine-tuning

Baselines As predicted, BPE-dropout shows itself to be a strong baseline. We hypothesise that this may be due to it being 5× more deterministic than all other tested tokenisers, with a regularisation rate of ∼10% rather than 50%. This consistency makes memorisation easier,[11] perhaps causing the large gap with the ULM baseline on some tasks. On PoS and NER, both highly memorisable, BPE-dropout is hardly ever outperformed, whereas it commonly is for DP, which isn't memorisable at the token level. Especially notable is the unconventionally higher regularisation rate of the tokenisers that beat BPE-dropout, affirming our central hypothesis that *not all subword regularisation is of equal quality*: a

[11]One interesting though expensive experiment would be to make all tokenisers fully deterministic during the training phase of fine-tuning and/or the testing phase, rather than varying the segmentation in $p = 50\%$ of words.

BPE-dropout tokeniser with our regularisation rate of $p = 0.5$ would have $p_d \approx 0.37$ (see Figure 6 in §E) which Provilkov et al. show to be suboptimal, and yet on DP, using the same vocabulary, GRaMPa with that regularisation rate outperforms even the optimal BPE-dropout setting of $p_d = 0.1$.

Vocabularies ULM-based GRaMPa models are overall better than the BPE-based ones: they improve more over their own baseline, tend to have a higher average score, and on DP specifically, this holds true even without controlling for τ and ℓ_{min}.

Temperature Depending on the vocabulary, the response to increasing τ varies. There's no clear trend in the BPE-based models, whereas with the ULM vocabulary, higher τ is always better. This is counter to our hypothesis w.r.t. temperature, but in either case, the uniform model ($\ell_{\text{min}} = 1, \tau = 1.0$) is (one of) the worst, affirming our hypothesis that *skewing away from uniformity is better*.

Minimum length There is a much clearer effect with either vocabulary that higher ℓ_{min} is better. The reason is likely that in the uniform case, single-character tokens dominate 50% of the token distribution (cfr. §3.1). Raising τ merely attenuates this domination of low-information tokens, whereas raising ℓ_{min} fully excludes them (when possible) in favour of more semantically laden tokens.

6.2 Robustness

With only 10% of words perturbed, the GRaMPa models already diverge from the baselines. Again the vocabularies behave distinctly: BPE tokens seem less fit for dealing with typos than ULM tokens, perhaps since ULM has a larger variety of small subwords (see Figure 7 in §E). All GRaMPa models with ULM outperform the baseline, which is *stronger* than BPE-dropout here. The most robust of those had *longer* tokens (Table 7) and *less* segmentational entropy (Table 8), yet BPE-dropout did too, making these insufficient predictors.

6.3 Rényi efficiency revisited

We now see that we could not have tuned GRaMPa's τ, ULM's α, or the multiplexing p using Rényi efficiency E_α. We know performance is not monotonic in α (Kudo, 2018) and not in τ either (§6.1), yet increasing τ and α causes a monotonic skew away from uniformity in segmentations and thus away from *non*-uniformity in token properties (cfr. §3.1). The result is a monotonic increase in E_α (see Figure 10 and Figure 11b in Appendix §E). Additionally, for the typical BPE-dropout setting of $p_d = 0.1$, E_α is markedly higher (see Figure 11a) than for ULM, whereas its models aren't strictly better; thus, E_α values aren't absolute even for identical vocabulary sizes, and cannot be compared between tokeniser architectures. This is an issue for multiplexing, because when multiplexing two or more tokenisers, the setting with highest E_α will simply be such that the tokeniser with highest individual E_α is chosen 100% of the time, i.e. $p = 0$ or $p = 1$. Hence, tuning p using E_α results in no multiplexing at all, whereas its benefits were outlined above. For example, tuning p between deterministic ULM and GRaMPa results in no determinism at all (Figure 12b). On the off-chance that two tokenisers do hover around the same baseline E_α (like in Figure 12a), it is mere coincidence, and E_α being concave in p is meaningless.

7 Releases

A software implementation of GRaMPa is available on GitHub as part of the HuggingFace-compatible **Tokeniser Toolkit (`TkTkT`)** Python package:

`https://github.com/bauwenst/TkTkT`.

Model training experiments were set up using the **Language Modelling Tasks as Objects (`LaMoTO`)** Python package, partly developed for this work:

`https://github.com/bauwenst/LaMoTO`.

All tables and graphs were generated using the **Figures as Objects (`Fiject`)** Python package:

`https://github.com/bauwenst/fiject`.

Lastly, the scripts needed to reproduce this paper are found at `https://github.com/bauwenst/Experiments_GRaMPa`.

8 Conclusion

We introduce GRaMPa, a stochastic tokenisation algorithm that can sample valid tokenisations of a string s uniformly in $\Theta(|s|^2)$ time, thereby significantly improving on rejection sampling. We argue that stochastic tokenisers should take into account not only the distribution of *segmentations*, but also the distribution of the segmentations' *properties* like token length, which is systematically too low when sampling uniformly. GRaMPa can be skewed to prefer longer tokens using the temperature τ and soft minimum length ℓ_{min} hyperparameters. We show that this results in improved downstream performance on language modelling tasks, due to the model seeing a mix of segmentation properties.

Limitations

Having shown the effectiveness of regularising monolingual (English) MLMs through skewed uniform tokenisation sampling, a natural extension would be to test its effectiveness in multilingual MLMs and translation models. We suspect that because multilingual tokenisers are even worse at providing meaningful token boundaries than monolingual ones, they already treat strings as little more than characters (like GRaMPa), and thus teach the model to compose better.

Due to computational constraints, we did not tune the vocabulary size $|V| = 32768$ nor the BPE/GRaMPa multiplexing rate $p = 0.5$, which we based on ULM's regularisation rate at $\alpha = 0.15$ (chosen heuristically between the recommended values of 0.1 and 0.2, and likely is not the optimal value either). Therefore, GRaMPa's performance could likely be improved with a more extensive hyperparameter search, but so could the stochastic ULM baseline.

We found that increasing ℓ_{min} from 1 to 2 usually had a positive impact. This begs the question at which ℓ_{min} this upwards trend stops/reverses.

For practitioners who want to use GRaMPa, we realise that we have introduced multiple hyperparameters. We recommend the hyperparameter setting $(V_{\text{ULM}}, \tau = 5.0,\ \ell_{\text{min}} = 2)$. We recommend a multiplexing rate p lower than 0.5 for tasks involving memorisation (e.g. PoS) and higher for tasks involving fine-grained word comprehension.

We did not train in a high-resource setting – only with 6-layer DeBERTa models on a 50k truncated corpus. Nevertheless, the latter is still a realistic setting, and it also served to prove that there exists *at least one* setting in which GRaMPa is useful. We leave the investigation of how GRaMPa (or another skewing mechanism for uniform sampling) scales to larger models and datasets to future work.

We did not test what happens when the *stochastic* tokenisers are made *deterministic* at inference time, e.g. by setting $p = 0$. As confirmed by Bauwens and Delobelle (2024), using a different tokeniser during fine-tuning than during pre-training can boost performance, and indeed, it is conceivable that the embeddings for that deterministic inference are of higher quality due to the regularisation at train time. For fair comparison, the same could be done to the baseline tokenisers at inference, such that effectively, two identical BPE tokenisers and two identical ULM tokenisers are compared but one with and one without regularised token embeddings.

In the same line of reasoning, we could have also tested the converse, fine-tuning the models that were pre-trained with a *deterministic* tokeniser using all the different *stochastic* tokenisers. Since GRaMPa becomes more deterministic with increasing ℓ_{min} and τ, in the limit, it is just another deterministic tokenisation algorithm, and thus it is conceivable that swapping out e.g. BPE for a BPE-based GRaMPa may have the same effect as Bauwens and Delobelle (2024) saw for BPE-knockout.

We also did not pre-train any models with $p = 0$ or $p = 1$, which would have led to an additional 14 models. In particular, since GRaMPa performs best when multiplexed with deterministic ULM and strongly outperformed a stochastic ULM baseline, one could wonder if this actually implies that ULM is best used with no regularisation. We suspect not, since it was precisely the beneficial effect of ULM's regularisation as found by Kudo (2018) that popularised ULM and subword regularisation in the first place.

Although both tables with fine-tuning results show visible trends, they were generated using only 5 hyperparameter grid samples per model-task pair. To be more confident about the reported best, and to really get the best out of each model, we could have done more runs.

Finally, our theoretical results sometimes assume an infinite vocabulary. Whilst tokenisers used in some state-of-the-art models have vocabularies that may easily exceed hundreds of thousands of subwords in size, an infinite vocabulary does not model any of the intricate constraints that exist in the segmentation graphs produced by finite vocabularies.

Author Contributions

DK conceived the path-based Markov model for single-pass uniform sampling, and the proof for non-deterministic complexity of previous work. TB conceived the argument and methods for skewing, implemented GRaMPa in Python, executed the experiments, and wrote the bulk of the manuscript. MdL helped iterating on the manuscript.

Acknowledgements

We would like to thank Marco Cognetta for his open and cordial correspondence, as well as one of the anonymous reviewers for his/her particularly humane and useful feedback.

TB is funded by a Bijzonder Onderzoeksfonds (BOF) internal fund at KU Leuven, namely the C1 project fund with reference C14/23/096.

Most of the computational resources and services used in this work were provided by the VSC (Flemish Supercomputer Centre), funded by the Research Foundation Flanders (FWO) and the Flemish Government department EWI.

This research was supported by the state of North Rhine-Westphalia as part of the Lamarr Institute for Machine Learning and Artificial Intelligence.

References

Roy Bar-Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. 2006. The Second PASCAL Recognising Textual Entailment Challenge. In *Proceedings of the second PASCAL challenges workshop on recognising textual entailment*, volume 1.

Thomas Bauwens. 2023. BPE-knockout: Systematic review of BPE tokenisers and their flaws with application in Dutch morphology. Master's thesis, KU Leuven.

Thomas Bauwens and Pieter Delobelle. 2024. BPE-knockout: Pruning Pre-existing BPE Tokenisers with Backwards-compatible Morphological Semi-supervision. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 5810–5832, Mexico City, Mexico. Association for Computational Linguistics.

Luisa Bentivogli, Bernardo Magnini, Ido Dagan, Hoa Trang Dang, and Danilo Giampiccolo. 2009. The Fifth PASCAL Recognizing Textual Entailment Challenge. In *Proceedings of the Second Text Analysis Conference, TAC 2009, Gaithersburg, Maryland, USA, November 16-17, 2009*. NIST.

Kaj Bostrom and Greg Durrett. 2020. Byte Pair Encoding is Suboptimal for Language Model Pretraining. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4617–4624, Online. Association for Computational Linguistics.

Kris Cao and Laura Rimell. 2021. You should evaluate your language model on marginal likelihood over tokenisations. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 2104–2114, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Min-Te Chao and William Edward Strawderman. 1972. Negative moments of positive random variables. *Journal of the American Statistical Association*, 67(338):429–431.

Bobbie Chern, Persi Diaconis, Daniel M. Kane, and Robert C. Rhoades. 2014. Closed expressions for averages of set partition statistics. *Research in the Mathematical Sciences*, 1(1):2.

Nadezhda Chirkova, Germán Kruszewski, Jos Rozen, and Marc Dymetman. 2023. Should you marginalize over possible tokenizations? In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–12, Toronto, Canada. Association for Computational Linguistics.

Jonathan H. Clark, Dan Garrette, Iulia Turc, and John Wieting. 2022. Canine: Pre-training an Efficient Tokenization-Free Encoder for Language Representation. *Transactions of the Association for Computational Linguistics*, 10:73–91. Place: Cambridge, MA Publisher: MIT Press.

Marco Cognetta, Vilém Zouhar, and Naoaki Okazaki. 2024a. Distributional properties of subword regularization. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 10753–10763, Miami, Florida, USA. 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.

Gautier Dagan, Gabriel Synnaeve, and Baptiste Roziere. 2024. Getting the most out of your tokenizer for pre-training and domain adaptation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 9784–9805. PMLR.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2006. The pascal recognising textual entailment challenge. In *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, pages 177–190, Berlin, Heidelberg. Springer Berlin Heidelberg.

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

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.

Timothy Dozat and Christopher D. Manning. 2017. Deep biaffine attention for neural dependency parsing. In *International Conference on Learning Representations*.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, Johan Ferret, Peter Liu, Pouya Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, Anton Tsitsulin, Nino Vieillard, Piotr Stanczyk, Sertan Girgin, Nikola Momchev, Matt Hoffman, Shantanu Thakoor, Jean-Bastien Grill, Behnam Neyshabur, Olivier Bachem, Alanna Walton, Aliaksei Severyn, Alicia Parrish, Aliya Ahmad, Allen Hutchison, Alvin Abdagic, Amanda Carl, Amy Shen, Andy Brock, Andy Coenen, Anthony Laforge, Antonia Paterson, Ben Bastian, Bilal Piot, Bo Wu, Brandon Royal, Charlie Chen, Chintu Kumar, Chris Perry, Chris Welty, Christopher A. Choquette-Choo, Danila Sinopalnikov, David Weinberger, Dimple Vijaykumar, Dominika Rogozińska, Dustin Herbison, Elisa Bandy, Emma Wang, Eric Noland, Erica Moreira, Evan Senter, Evgenii Eltyshev, Francesco Visin, Gabriel Rasskin, Gary Wei, Glenn Cameron, Gus Martins, Hadi Hashemi, Hanna Klimczak-Plucińska, Harleen Batra, Harsh Dhand, Ivan Nardini, Jacinda Mein, Jack Zhou, James Svensson, Jeff Stanway, Jetha Chan, Jin Peng Zhou, Joana Carrasqueira, Joana Iljazi, Jocelyn Becker, Joe Fernandez, Joost van Amersfoort, Josh Gordon, Josh Lipschultz, Josh Newlan, Ju-yeong Ji, Kareem Mohamed, Kartikeya Badola, Kat Black, Katie Millican, Keelin McDonell, Kelvin Nguyen, Kiranbir Sodhia, Kish Greene, Lars Lowe Sjoesund, Lauren Usui, Laurent Sifre, Lena Heuermann, Leticia Lago, Lilly McNealus, Livio Baldini Soares, Logan Kilpatrick, Lucas Dixon, Luciano Martins, Machel Reid, Manvinder Singh, Mark Iverson, Martin Görner, Mat Velloso, Mateo Wirth, Matt Davidow, Matt Miller, Matthew Rahtz, Matthew Watson, Meg Risdal, Mehran Kazemi, Michael Moynihan, Ming Zhang, Minsuk Kahng, Minwoo Park, Mofi Rahman, Mohit Khatwani, Natalie Dao, Nenshad Bardoliwalla, Nesh Devanathan, Neta Dumai, Nilay Chauhan, Oscar Wahltinez, Pankil Botarda, Parker Barnes, Paul Barham, Paul Michel, Pengchong Jin, Petko Georgiev, Phil Culliton, Pradeep Kuppala, Ramona Comanescu, Ramona Merhej, Reena Jana, Reza Ardeshir Rokni, Rishabh Agarwal, Ryan Mullins, Samaneh Saadat, Sara Mc Carthy, Sarah Cogan, Sarah Perrin, Sébastien M. R. Arnold, Sebastian Krause, Shengyang Dai, Shruti Garg, Shruti Sheth, Sue Ronstrom, Susan Chan, Timothy Jordan, Ting Yu, Tom Eccles, Tom Hennigan, Tomas Kocisky, Tulsee Doshi, Vihan Jain, Vikas Yadav, Vilobh Meshram, Vishal Dharmadhikari, Warren Barkley, Wei Wei, Wenming Ye, Woohyun Han, Woosuk Kwon, Xiang Xu, Zhe Shen, Zhitao Gong, Zichuan Wei, Victor Cotruta, Phoebe Kirk, Anand Rao, Minh Giang, Ludovic Peran, Tris Warkentin, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, D. Sculley, Jeanine Banks, Anca Dragan, Slav Petrov, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Sebastian Borgeaud, Noah Fiedel, Armand Joulin, Kathleen Kenealy, Robert Dadashi, and Alek Andreev. 2024. Gemma 2: Improving Open Language Models at a Practical Size. ArXiv:2408.00118.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. 2007. The Third PASCAL Recognising Textual Entailment Challenge. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, RTE '07, page 1–9, USA. Association for Computational Linguistics.

Godfrey Harold Hardy and Srinivasa Ramanujan. 1917. Asymptotic Formulæ for the Distribution of Integers of Various Types. *Proceedings of the London Mathematical Society*, 2(1):112–132.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. DeBERTa: Decoding-enhanced BERT with Disentangled Attention.

Xuanli He, Gholamreza Haffari, and Mohammad Norouzi. 2020. Dynamic Programming Encoding for Subword Segmentation in Neural Machine Translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3042–3051, Online. Association for Computational Linguistics.

Carl Friedrich Hindenburg. 1779. *Infinitinomii dignitatum exponentis indeterminati historia leges ac formulae*. Dieterich.

Tatsuya Hiraoka. 2022. MaxMatch-Dropout: Subword Regularization for WordPiece. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 4864–4872, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Tatsuya Hiraoka and Tomoya Iwakura. 2024. Tokenization Preference for Human and Machine Learning Model: An Annotation Study. ArXiv:2304.10813 [cs].

Valentin Hofmann, Janet Pierrehumbert, and Hinrich Schütze. 2021. Superbizarre Is Not Superb: Derivational Morphology Improves BERT's Interpretation of Complex Words. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3594–3608, Online. Association for Computational Linguistics.

Shankar Iyer, Nikhil Dandekar, and Kornél Csernai. 2017. First Quora Dataset Release: Question Pairs — Quora. Online; accessed 15 October 2024.

Dayal Singh Kalra and Maissam Barkeshli. 2024. Why Warmup the Learning Rate? Underlying Mechanisms and Improvements. ArXiv:2406.09405 version: 1.

Donald Ervin Knuth. 2014. *The Art of Computer Programming, Volume 4A: Combinatorial Algorithms, Part 1*. Pearson Education.

Taku Kudo. 2018. Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, Melbourne, Australia. Association for Computational Linguistics.

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

Misha Lavrov. 2018. A procedure for sampling paths in a directed acyclic graph — Mathematics Stack Exchange. Online; accessed 7 October 2024.

Hector J. Levesque, Ernest Davis, and Leora Morgenstern. 2012. The Winograd schema challenge. In *Proceedings of the Thirteenth International Conference on Principles of Knowledge Representation and Reasoning*, KR'12, page 552–561. AAAI Press.

Yeshu Li, Danyal Saeed, Xinhua Zhang, Brian D. Ziebart, and Kevin Gimpel. 2024. Moment distributionally robust tree structured prediction. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, pages 12237–12252, Red Hook, NY, USA. Curran Associates Inc.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. ArXiv:1907.11692 [cs].

Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization.

Andrey Andreyevich Markov. 1907. Распространеніе закона большихъ чиселъ на величины, зависящія другъ отъ друга. *Izvestiya Fiziko-matematicheskogo obschestva pri Kazanskom universitete*, 15(4):135–156.

Benjamin Minixhofer, Jonas Pfeiffer, and Ivan Vulić. 2023. CompoundPiece: Evaluating and Improving Decompounding Performance of Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 343–359, Singapore. Association for Computational Linguistics.

Piotr Nawrot, Jan Chorowski, Adrian Lancucki, and Edoardo Maria Ponti. 2023. Efficient Transformers with Dynamic Token Pooling. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6403–6417, Toronto, Canada. Association for Computational Linguistics.

A. Yavuz Oruç. 2016. On number of partitions of an integer into a fixed number of positive integers. *Journal of Number Theory*, 159:355–369.

Ivan Provilkov, Dmitrii Emelianenko, and Elena Voita. 2020. BPE-Dropout: Simple and Effective Subword Regularization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1882–1892, Online. Association for Computational Linguistics.

Craig W Schmidt, Varshini Reddy, Haoran Zhang, Alec Alameddine, Omri Uzan, Yuval Pinter, and Chris Tanner. 2024. Tokenization Is More Than Compression. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 678–702, Miami, Florida, USA. Association for Computational Linguistics.

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.

Natalia Silveira, Timothy Dozat, Marie-Catherine De Marneffe, Samuel R Bowman, Miriam Connor, John Bauer, and Christopher D Manning. 2014. A Gold Standard Dependency Corpus for English. In *LREC*, pages 2897–2904.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. 2023. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama — Cerebras Blog.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Haiyue Song, Francois Meyer, Raj Dabre, Hideki Tanaka, Chenhui Chu, and Sadao Kurohashi. 2024. SubMerge: Merging Equivalent Subword Tokenizations for Subword Regularized Models in Neural Machine Translation. In *Proceedings of the 25th Annual Conference of the European Association for Machine Translation (Volume 1)*, pages 147–163, Sheffield, UK. European Association for Machine Translation (EAMT).

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

Samson Tan, Shafiq Joty, Min-Yen Kan, and Richard Socher. 2020. It's Morphin' Time! Combating Linguistic Discrimination with Inflectional Perturbations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2920–2935, Online. Association for Computational Linguistics.

Yi Tay, Vinh Q. Tran, Sebastian Ruder, Jai Gupta, Hyung Won Chung, Dara Bahri, Zhen Qin, Simon Baumgartner, Cong Yu, and Donald Metzler. 2022. Charformer: Fast Character Transformers via Gradient-based Subword Tokenization.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: On the importance of pre-training compact models. ArXiv:1908.08962 [cs].

Omri Uzan, Craig W. Schmidt, Chris Tanner, and Yuval Pinter. 2024. Greed is all you need: An evaluation of tokenizer inference methods. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 813–822, Bangkok, Thailand. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. ArXiv:1804.07461.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Yu Zhao, Yuanbin Qu, Konrad Staniszewski, Szymon Tworkowski, Wei Liu, Piotr Miłoś, Yuxiang Wu, and Pasquale Minervini. 2024. Analysing the impact of sequence composition on language model pre-training. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7897–7912, Bangkok, Thailand. Association for Computational Linguistics.

Giulio Zhou. 2018. Morphological Zero-Shot Neural Machine Translation. Master's thesis, University of Edinburgh.

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.

Judit Ács, Ákos Kádár, and Andras Kornai. 2021. Subword Pooling Makes a Difference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 2284–2295, Online. Association for Computational Linguistics.

A Proofs

A.1 Segmentation graphs weighted with unigram probabilities can't be sampled Markovianly *and* proportionally to the joint unigram probability

Proof. Let the n_i tokens that can extend a given string s from character $i = 0 \ldots n-1$ onward be called $\{t_{i,j}\}_{j=1}^{n_i}$ with associated probabilities $\{p_{i,j}\}_{j=1}^{n_i}$ that need local renormalisation with denominator $Z_i = \sum_{j=1}^{n_i} p_{i,j}$ to be sampled.

Now, imagine sampling the path that always chooses the first outgoing token: it starts with $t_{0,1}$ with a length we'll call ℓ_1, then chooses $t_{\ell_1,1}$ with length ℓ_2, then chooses $t_{\ell_1+\ell_2,1}$ with length ℓ_3, then $t_{\ell_1+\ell_2+\ell_3,1}$, ... until the lengths sum to $|s|$. The probability of the first step is $p_{0,1}/Z_0$, of the second is $p_{\ell_1,1}/Z_{\ell_1}$, of the third is $p_{\ell_1+\ell_2,1}/Z_{\ell_1+\ell_2}$ and so on. In total, this path has a probability

$$P^{(1)}_{\text{Markov}} = \frac{p_{0,1} \cdot p_{\ell_1,1} \cdot p_{\ell_1+\ell_2,1} \cdots}{Z_0 \cdot Z_{\ell_1} \cdot Z_{\ell_1+\ell_2} \cdots} \tag{5}$$

of being sampled Markovianly. We now note that the sequence of tokens $\{t_{0,1}; t_{\ell_1,1}; t_{\ell_1+\ell_2,1} \ldots\}$ has a joint unigram probability equal to

$$P^{(1)}_{\text{LM}} = p_{0,1} \cdot p_{\ell_1,1} \cdot p_{\ell_1+\ell_2,1} \cdots \tag{6}$$

which is exactly the numerator in Eq. 5. This is promising: if $P_{\text{Markov}} \propto P_{\text{LM}}$ with the same proportionality constant for all paths (which would have to be the sum of $P^{(i)}_{\text{LM}}$ across all segmentations i) then we have proved the opposite case.

Yet, imagine a second path that starts with a token of length $k_1 \neq \ell_1$, then a token of length k_2 such that $k_1 + k_2 = \ell_1 + \ell_2$, and beyond that always takes the first token, overlapping with the first path. This second path $\{t_{0,*}; t_{k_1,*}; t_{\ell_1+\ell_2,1} \ldots\}$ has probability

$$P^{(2)}_{\text{Markov}} = \frac{p_{0,*} \cdot p_{k_1,*} \cdot p_{\ell_1+\ell_2,1} \cdots}{Z_0 \cdot Z_{k_1} \cdot Z_{\ell_1+\ell_2} \cdots} \tag{7}$$

which is *also* proportional to the joint unigram probability of the segmentation, namely

$$P^{(2)}_{\text{LM}} = p_{0,*} \cdot p_{k_1,*} \cdot p_{\ell_1+\ell_2,1} \cdots \tag{8}$$

yet the denominators of Eq. 5 and Eq. 7 differ in their second factor Z_{ℓ_1} and Z_{k_1} respectively, which are constructed from an arbitrarily different subset of the vocabulary and hence $Z_{\ell_1} \neq Z_{k_1}$ in general, so the denominators differ. ■

This result also makes intuitive sense: the segmentation graph can only return segmentations of one specific string, and hence it samples across a reduced set of token sequences that together make up 100% of the possible segmentations. Meanwhile, the joint probability (which is used by e.g. the ULM tokeniser to find the most likely segmentation) actually models *all* token sequences, not just those that concatenate to form the given string. These are two different joint distributions.

A.2 Unskewed GRaMPa samples segmentations uniformly

We prove the theorem for a forward-constructed grid and hence backward decoding (R2L). An analogous proof exists for the L2R implementation.

Proof. Given a string s of n characters, let the path count at node $i = 0 \ldots n$ in the lattice be called $c_i \in \mathbb{N}$, where $c_0 = 1$ and c_n is the total amount $N_V(s)$ of paths through the lattice and hence the amount of valid segmentations of s. We now prove that GRaMPa samples the $N_V(s) \leq 2^{n-1}$ segmentations uniformly, all having equal probability $1/N_V(s)$ of occurring in the decoding process.

When GRaMPa finds itself in node i while decoding, it knows that c_i paths arrive there and that they came from the preceding nodes $0 \ldots i-1$ according to

$$c_i = \sum_{j=0}^{i-1} c_j \cdot \mathbb{1}\{s_{j:i} \in V\}. \tag{9}$$

It then samples a preceding node j from the reachable nodes with probability c_j/c_i (assuming no skewing transformation). If $j = 0$, decoding ends. Otherwise, another $k < j$ is sampled with probability c_k/c_j. In the latter case, the joint probability of both samples is

$$\frac{c_k}{\cancel{c_j}} \cdot \frac{\cancel{c_j}}{c_i} = \frac{c_k}{c_i}. \tag{10}$$

We now extend this argument: let the sequence of all visited nodes be called $\{i_0, i_1, i_2, \ldots, i_\ell\}$, with $i_0 = 0$ and $i_\ell = n$, representing a segmentation of ℓ tokens obtained through ℓ samples. The probability of this segmentation is then

$$\begin{aligned} P = \prod_{k=1}^{\ell} \frac{c_{i_{k-1}}}{c_{i_k}} &= \frac{c_{i_0}}{\cancel{c_{i_1}}} \cdot \frac{\cancel{c_{i_1}}}{\cancel{c_{i_2}}} \cdot \ldots \cdot \frac{\cancel{c_{i_{\ell-2}}}}{\cancel{c_{i_{\ell-1}}}} \cdot \frac{\cancel{c_{i_{\ell-1}}}}{c_{i_\ell}} \\ &= \frac{c_{i_0}}{c_{i_\ell}} = \frac{c_0}{c_n} = \frac{1}{N_V(s)} \end{aligned} \tag{11}$$

which telescopes regardless of ℓ. ■

A.3 Token lengths are distributed (almost) geometrically across segmentations

Given is a string s of n characters. Assuming an infinite vocabulary, there exist 2^{n-1} segmentations for this string. Given a token length $\ell \in \{1, ..., n\}$, we want to count how many tokens[12] have this length across all 2^{n-1} segmentations.[13] Let this number be called $M(\ell, n)$. We will first find an explicit expression for $M(\ell, n)$, and then show that the fraction of all tokens across all segmentations that is taken up by tokens of length ℓ, i.e.

$$P(\ell \mid n) = \frac{M(\ell, n)}{\sum_{\tilde{\ell}=1}^{n} M(\tilde{\ell}, n)} \tag{12}$$

approximates a (truncated) geometric distribution for Bernoulli experiments with $p = \frac{1}{2}$. In other words: *tokens of length* 1 *appear twice as much as tokens of length* 2*, which appear twice as much as tokens of length* 3*, and so on.* This then implies that if we sample uniformly across segmentations, we will see geometrically distributed token lengths.

Proof. We want to find the amount of tokens of length ℓ that appear across all segmentations. This is equivalent to counting how many tokens of length ℓ appear in each segmentation, and summing those counts.

If there are k tokens of length ℓ in a particular segmentation, it contributes k to the total count. Now, duplicate each such segmentation k times, and call each one a "representative" of one of the tokens of length ℓ they contain. Put all representatives for the same token (i.e. the same pair of boundary indices) into a separate set. The sum of the sizes of these sets is still $M(\ell, n)$. By now generating these sets, we have turned *counting tokens* into *counting segmentations*, which we already have an expression for ($2^{|s|-1}$).

Assume $\ell \neq n$ for now. Consider the $n-1$ inter-character positions. A token of length ℓ is defined by its starting boundary i and its ending boundary $i+\ell$. Excluding the token whose starting boundary lies before the string and the token whose ending boundary lies after the string, we find all tokens of length ℓ by sliding a window from inter-character position $i = 1$ to $i = n - 1 - \ell$, at which point the ending boundary falls on the last inter-character position $i + \ell = n - 1$.

Remember that segmentations are generated by choosing whether or not a boundary goes on each of the $n-1$ inter-character positions. When a token of length ℓ starts at position i, that eliminates $\ell + 1$ choices: the two ends i and $i+\ell$ receive a boundary, and all $\ell - 1$ positions in between explicitly have no boundary. Hence, $n - 1 - (\ell + 1) = n - 2 - \ell$ binary choices remain to be made. All $2^{n-2-\ell}$ variations of these boundaries together form the set of representatives for token $(i, i + \ell)$. Since the window had $n - 1 - \ell$ different starting points, the tokens it forms account for

$$(n - 1 - \ell) \cdot 2^{n-2-\ell} \tag{13}$$

representatives in total. We still have to account for the two[14] tokens we left out, which respectively start and end on a non-inter-character position. Hence, for these two tokens, only ℓ choices are eliminated rather than $\ell + 1$, leaving $n - 1 - \ell$ choices open and hence both have $2^{n-1-\ell}$ representatives, giving

$$2 \cdot 2^{n-1-\ell} = 4 \cdot 2^{n-2-\ell} \tag{14}$$

total. Adding Eq. 13 and Eq. 14, we finally get

$$M(\ell, n) = \underbrace{(n - 1 - \ell)\, 2^{n-2-\ell}}_{\text{window}} + \underbrace{4 \cdot 2^{n-2-\ell}}_{\text{edges}} \tag{15}$$

given our assumption that $\ell \leq n - 1$. Note that at $\ell = n - 1$, the window has 0 positions and hence only the second term contributes anything.

For the case where $\ell = n$, i.e. the one token that spans the entire string, the window argument breaks down: Eq. 13 suggests that a negative amount of representatives is added by the window, whilst Eq. 14 alleges that 2^{-1} representatives are added at both edges. Since the second term coincidentally still adds the correct amount (namely 1), we just need to ignore the first term for this one case, which we can do by clipping it to 0. We get

$$\begin{aligned} M(\ell, n) &= \max\left\{0; (n-1-\ell)\, 2^{n-2-\ell}\right\} + 4 \cdot 2^{n-2-\ell} \\ &= \max\{0; n-1-\ell\}\, 2^{n-2-\ell} + 4 \cdot 2^{n-2-\ell} \\ &= (4 + \max\{0; n - 1 - \ell\})\, 2^{n-2-\ell} \end{aligned} \tag{16}$$

[12]Not types. E.g.: for $s =$ "ababc" and the segmentations {ab/ab/c, a/b/ab/c}, we count 7 tokens total, despite using 2 types in the first, 4 types in the second, and 4 types together.

[13]This problem is related to counting how many sets of size ℓ appear across all partitions of a set of n elements (Chern et al., 2014), except in our case, we are restricted to forming subsets from adjacent elements only.

[14]There exist at least two tokens of all lengths $\ell \neq n$.

which is exact for all token sizes $\ell \in \{1, ..., n\}$. The fact that this is strictly increasing given a constant n and decreasing ℓ matches our intuition that smaller tokens appear more across segmentations.

We now want to plug this expression into Eq. 12. To do so, we find a closed expression for the sum in the denominator. To start, we note the following: imagine we didn't have the max in the above equation, i.e. we made the approximation

$$\begin{aligned} M(\ell, n) &\approx (4 + n - 1 - \ell)\, 2^{n-2-\ell} \\ &= (n + 3 - \ell)\, 2^{n-2-\ell}. \end{aligned} \tag{17}$$

The only ℓ for which the latter is not exactly equal to $M(\ell, n)$ is $\ell = n$, where it is $3 \cdot 2^{-2} = 0.75$ rather than 1. Since Eq. 12 sums exponentially increasing values, this difference of 0.25 becomes negligible and we use this simpler expression for $M(\ell, n)$ instead. The summation becomes

$$\sum_{\ell=1}^{n} M(\ell, n) \approx \sum_{\ell=1}^{n} (n+3-\ell)\, 2^{n-2-\ell} \tag{18}$$

$$= 2^{n-2} \sum_{\ell=1}^{n} (n + 3 - \ell)\, 2^{-\ell} \tag{19}$$

$$= 2^{n-2} \left((n + 3) \sum_{\ell=1}^{n} 2^{-\ell} - \sum_{\ell=1}^{n} \ell\, 2^{-\ell} \right) \tag{20}$$

The first summation is a geometric series of the form $q, q^2, q^3, ...$ with $q = 2^{-1}$ amounting to

$$\sum_{k=1}^{n} q^k = q\, \frac{1 - q^n}{1 - q} = 1 - 2^{-n}. \tag{21}$$

The second summation follows from taking the derivative of Eq. 21 w.r.t. q and some rearranging:

$$\begin{aligned} \sum_{k=1}^{n} k\, q^k &= \frac{q}{1 - q}(1 - (n + 1) \cdot q^n) \\ &\quad + \left(\frac{q}{1 - q}\right)^2 (1 - q^n) \\ &= (1-(n+1)\, 2^{-n}) + (1-2^{-n}) \\ &= 2 - (n + 2)\, 2^{-n}. \end{aligned} \tag{22}$$

Putting these together,

$$\begin{aligned} &\sum_{\ell=1}^{n} M(\ell, n) \\ &\approx 2^{n-2}\big((n + 3)(1-2^{-n})-(2-(n+2)\, 2^{-n})\big) \\ &= 2^{n-2}\big((n+3)-(n+3)2^{-n}-2+(n+2)\, 2^{-n}\big) \\ &= \big((n + 1) - 2^{-n}\big)2^{n-2}. \end{aligned} \tag{23}$$

We don't simplify further, because this and Eq. 17 both slot nicely into Eq. 12:

$$P(\ell \mid n) \approx \frac{(n + 3 - \ell)\, 2^{n-2-\ell}}{((n + 1) - 2^{-n})2^{n-2}} \tag{24}$$

$$= \frac{n + 3 - \ell}{n + 1 - 2^{-n}}\, 2^{-\ell} \tag{25}$$

$$\approx \frac{n + 3 - \ell}{n + 1}\, 2^{-\ell} \tag{26}$$

$$= \left(1 - \frac{\ell - 2}{n + 1}\right) 2^{-\ell}. \tag{27}$$

Despite the two approximations made, this is still very close to being normalised across the token lengths $\ell \in \{1, ..., n\}$, and for very long strings ($n \to \infty$), it is the geometric distribution of $p = \frac{1}{2}$:

$$\lim_{n\to\infty} P(\ell \mid n) \approx 2^{-\ell} = \left(\frac{1}{2}\right)^{\ell-1} \frac{1}{2} \tag{28}$$

$$= \left(1 - \frac{1}{2}\right)^{\ell-1} \frac{1}{2} \tag{29}$$

$$= (1 - p)^{\ell-1} p \tag{30}$$

The first factor in Eq. 27 is only greater than 1 for tokens of length $\ell = 1$, meaning that slightly *more* probability mass is allocated to the very smallest tokens than a geometric distribution would. (This is also necessary to keep the distribution normalised when $n < \infty$.) The $\ell = 2$ tokens follow the geometric distribution exactly. Tokens with $\ell \geq 3$ slightly undershoot the geometric distribution. ■

This result follows intuition too. If we sample segmentations uniformly with no vocabulary constraint, there is a $p = 50\%$ chance of a token boundary occurring between any two characters. We walk through the string from left to right and perform such i.i.d. Bernoulli experiments until we get a token boundary. The amount of experiments it takes before this happens is geometrically distributed, and is also equal to the length of the token delimited on the right by that boundary. The length of the second token follows the exact same procedure except starting on the current boundary.

Assuming a string is long enough (large n), the fact that the last token cannot extend past the last character has negligible influence on the length distribution of each token, and hence token lengths are geometrically distributed.

A.4 Expected value of characters-per-token approaches 2 in uniform sampling

Let s be a string whose segmentations are sampled uniformly, resulting in a random amount of tokens m. Define the characters-per-token ratio

$$R = \frac{|s|}{m}. \tag{31}$$

We know $m \sim 1 + \text{Binom}(|s| - 1, \frac{1}{2})$ as shown in §3.1, assuming an infinite vocabulary. We now prove that $\lim_{|s|\to\infty} \mathbb{E}[R] = 2$.

Proof. Let $X = m - 1$, i.e. the random part of m, then R is of the form

$$R = (n+1) \cdot \frac{1}{1+X} \tag{32}$$

with $n = |s| - 1$ and $X \sim \text{Binom}(n, p)$, making

$$\mathbb{E}[R] = (n+1)\,\mathbb{E}\left[\frac{1}{1+X}\right]. \tag{33}$$

The latter factor is known (Chao and Strawderman, 1972) to be

$$\mathbb{E}\left[\frac{1}{1+X}\right] = \frac{1-(1-p)^{n+1}}{(n+1)p}. \tag{34}$$

for binomial variables X, so substituting back into Eq. 33 we get

$$\begin{aligned}\mathbb{E}[R] &= \cancel{(n+1)}\,\frac{1-(1-p)^{n+1}}{\cancel{(n+1)}\,p} \\ &= \frac{1}{p}\left(1-(1-p)^{n+1}\right)\end{aligned} \tag{35}$$

and since we know $n = |s| - 1$ and $p = \frac{1}{2}$,

$$\mathbb{E}[R] = 2 \cdot \left(1 - \frac{1}{2^{|s|}}\right) \tag{36}$$

where the second factor is 1 when $|s| \to \infty$. ■

A.5 Non-deterministic complexity of Cognetta et al. (2024) rejection sampling

Cognetta et al. (2024) present a non-deterministic algorithm to uniformly sample string segmentations using rejection sampling by Lavrov (2018). A description of this algorithm is given in §2.

For a string of n characters, their algorithm requires $\Theta(n^2)$ in space and time to construct the directed acyclic graph (DAG) $\mathcal{G}$ representing valid segmentations. Then, each sampling pass takes $O(n)$ time and succeeds with probability $P_{\text{accept}}(\mathcal{G}, \boldsymbol{i}) = \epsilon(\mathcal{G})/P_{\text{Markov}}(\mathcal{G}, \boldsymbol{i})$, where $\epsilon(\mathcal{G})$ is a lower bound on $P_{\text{accept}}(\mathcal{G}, \cdot)$ for the worst-case path. This lower bound can be computed as

$$\epsilon(\mathcal{G}) = \prod_{j=0}^{|s|-1} \frac{1}{\delta_o(j)}, \tag{37}$$

which is the probability $P_{\text{Markov}}(\mathcal{G}, \boldsymbol{i}_{wc})$ of sampling the path $\boldsymbol{i}_{wc}$ that visits every node.

In general, $P_{\text{accept}}(\mathcal{G}, \boldsymbol{i})$ depends both on $\mathcal{G}$ and on the path $\boldsymbol{i}$ currently being sampled. For a fixed $\mathcal{G}$, we can consider rejection sampling as a sequence of independent trials $X_j \sim \text{Bernoulli}(P_{\text{accept}}(\mathcal{G}, \boldsymbol{i}_j))$, where the sampled path $\boldsymbol{i}_j$ is drawn from a non-uniform discrete distribution over paths $\boldsymbol{i}$. Each path $\boldsymbol{i}_j$ is drawn with probability $P_{\text{Markov}}(\mathcal{G}, \boldsymbol{i}_j)$. The sequence of trials is a generalisation of a geometric process with a success rate p, where now p is itself variable across trials. Nevertheless, we still want to know the expected[15] number of trials. Suppose that we know the expected success rate $\mathbb{E}[P] = \mathbb{E}_{\boldsymbol{i}\subset\mathcal{G}}[P_{\text{accept}}(\mathcal{G}, \cdot)]$. It can be shown that the expected number of trials until the first success T is given by $1/\mathbb{E}[P]$.

Proof. Let T be the number of trials until the first success. Then we can decompose $\mathbb{E}[T]$ as

$$\begin{aligned}\mathbb{E}[T] &= P_{\text{accept}}(\mathcal{G}, \boldsymbol{i}_1)\,\mathbb{E}[T \mid T = 1] \\ &\quad + (1 - P_{\text{accept}}(\mathcal{G}, \boldsymbol{i}_1))\,\mathbb{E}[T \mid T > 1]),\end{aligned} \tag{38}$$

corresponding to two cases: if the first sampled path is accepted, then $T = 1$, and if it is not accepted, in which case $T > 1$. Then, taking the expectation over paths for the first trial $\mathbb{E}_{\boldsymbol{i}_1\subset\mathcal{G}}$, we obtain

$$\begin{aligned}\mathbb{E}_{\boldsymbol{i}_1\subset\mathcal{G}}[\mathbb{E}[T]] &= \mathbb{E}_{\boldsymbol{i}_1\subset\mathcal{G}}\big[P_{\text{accept}}(\mathcal{G}, \boldsymbol{i}_1)\,\mathbb{E}[T \mid T = 1] \\ &\quad + (1 - P_{\text{accept}}(\mathcal{G}, \boldsymbol{i}_1))\,\mathbb{E}[T \mid T > 1])\big] \\ \mathbb{E}[T] &= \mathbb{E}[P]\mathbb{E}[T \mid T = 1] \\ &\quad + (1 - \mathbb{E}[P])\,\mathbb{E}[T \mid T > 1]) \\ \mathbb{E}[T] &= \mathbb{E}[P] + (1 - \mathbb{E}[P])(1 + \mathbb{E}[T]),\end{aligned} \tag{39}$$

which simplifies to

$$0 = 1 - \mathbb{E}[T]\,\mathbb{E}[P] \tag{40}$$

and thus $\mathbb{E}[T] = 1/\mathbb{E}[P]$. ■

The total expected time complexity is then

$$\Theta(n^2) + \mathbb{E}[T]O(n) = \Theta(n^2) + \frac{O(n)}{\mathbb{E}[P]}. \tag{41}$$

[15] I.e.: the average, across all paths in $\mathcal{G}$.

To compute the expectation $\mathbb{E}[P]$ over both DAGs and paths, we must specify a distribution over DAGs. We do not attempt a full analysis, but instead consider only the special case of the fully connected DAG on $n+1$ nodes $\mathcal{G}_{\text{full}}$ corresponding to an infinite vocabulary. This is a DAG for which the j-th node has edges to all subsequent nodes from $j+1$ to $n+1$. In this case, we get

$$\mathbb{E}[P] := \epsilon(\mathcal{G}_{\text{full}})\,\mathbb{E}\left[\frac{1}{P_{\text{Markov}}(\mathcal{G}_{\text{full}}, \boldsymbol{i})}\right], \tag{42}$$

since ϵ is a graph-dependent constant that does not depend on the path. It is given by the probability of the least likely path, which for $\mathcal{G}_{\text{full}}$ is the path going through all nodes. At the j-th node, there are $\delta_o(j) = n - j$ outgoing edges, of which exactly one goes to the $j+1$-th node. Therefore,

$$\epsilon(\mathcal{G}_{\text{full}}) = \prod_{j=0}^{n-1} \frac{1}{\delta_o(j)} = \prod_{j=0}^{n-1} \frac{1}{n-j} = \frac{1}{n!}. \tag{43}$$

The expected value of $1/P_{\text{Markov}}(\mathcal{G}_{\text{full}}, \boldsymbol{i})$ is the size of the domain of paths, which we now prove.

Proof. For any discrete stochastic variable X with probability mass $p_X(x)$, domain $\mathcal{D}$ and support $\mathcal{D}' = \{x \in \mathcal{D} \mid p(x) > 0\}$, and any function $g : \mathcal{D} \to \mathcal{D}$, the *law of the unconscious statistician* says

$$\mathbb{E}[g(X)] = \sum_{x \in \mathcal{D}'} g(x)\, p_X(x). \tag{44}$$

When we choose $1/p_X(x)$ as $g(x)$, we then get

$$\begin{aligned} \mathbb{E}\left[\frac{1}{p_X(X)}\right] &= \sum_{x\in\mathcal{D}'} \frac{1}{p_X(X)}\, p_X(x) \\ &= \sum_{x\in\mathcal{D}'} 1 = |\mathcal{D}'|. \end{aligned} \tag{45}$$

The expected value in the right-hand side of Eq. 42 ranges over the distribution of paths *before* the rejection process, i.e. paths $\boldsymbol{i}$ distributed according to $P_{\text{Markov}}(\boldsymbol{i})$. Hence, we can apply this result. ■

Since all segmentations are possible in $\mathcal{G}_{\text{full}}$, we get

$$\mathbb{E}_{\boldsymbol{i}\subset\mathcal{G}_{\text{full}}}\left[\frac{1}{P_{\text{Markov}}(\mathcal{G}_{\text{full}}, \boldsymbol{i})}\right] = 2^{n-1}. \tag{46}$$

Plugging Eq. 43 and Eq. 46 into Eq. 42, we obtain the expected success rate $\mathbb{E}[P] = 2^{n-1}/n!$ for a fully connected DAG. The expected number of samples until a success is thus

$$\mathbb{E}[T] = \frac{n!}{2^{n-1}}. \tag{47}$$

Finally, inserting this result into Eq. 41 yields the expected time complexity of tokenising a string of length n with Cognetta et al. (2024)'s rejection sampler given an infinite vocabulary:

$$\Theta(n^2) + O\left(\frac{n!}{2^{n-1}}\right) \cdot O(n) = O\left(\frac{(n+1)!}{2^{n-1}}\right). \tag{48}$$

In contrast, GRaMPa requires $\Theta(n^2)$ in space and time to construct the DAG and $O(n)$ for a single pass that is guaranteed to return a valid segmentation. The mean- and worst-case runtimes therefore scale like $\Theta(n^2)$, regardless of the vocabulary.

B Uniform rejection sampling for any unigram distribution

In §A.1, we proved that sampling Markovianly from a segmentation graph weighted by token probabilities that only satisfy global normalisation, i.e.

$$\sum_{t\in V} P(t) = 1 \tag{49}$$

gave rise to a differently shaped distribution than the joint distribution in a unigram language model, the latter being the product of the probabilities of the resulting tokens $\boldsymbol{t} = [t_1, t_2, t_3, \ldots]$

$$P_{\text{LM}}(\boldsymbol{t}) = \prod_{i=1}^{|\boldsymbol{t}|} P(t_i). \tag{50}$$

The fundamental reason for this mismatch was that we couldn't get the local renormalisation constants Z_i to match up in Eq. 5 and Eq. 7,

$$P_{\text{Markov}}(\boldsymbol{t}) = \prod_{i=1}^{|\boldsymbol{t}|} \frac{P(t_i)}{Z_i}. \tag{51}$$

Rejection sampling can resolve this mismatch by introducing a binary acceptance variable A and sampling from $P_{\text{Markov}}(\boldsymbol{t} \mid A = 1)$, which we do by sampling a $\boldsymbol{t}$ and then randomly deciding which of the two subsets ($A = 1$ or $A = 0$) we are in on this try. If we are in $A = 1$, then we can return it.

Proof. According to Bayes's law,

$$\begin{aligned} P(\boldsymbol{t} \mid A = 1) &= \frac{P(A = 1 \mid \boldsymbol{t})P(\boldsymbol{t})}{P(A = 1)} \\ &= c_1 \cdot P(A = 1 \mid \boldsymbol{t})P(\boldsymbol{t}) \end{aligned} \tag{52}$$

for some $c_1 \in [0, 1]$ independent of $\boldsymbol{t}$. Define the acceptance rate depending on $\boldsymbol{t}$'s Markov path:

$$P_{\text{Markov}}(A = 1 \mid \boldsymbol{t}) = \frac{1}{c_2} \prod_{i=1}^{|\boldsymbol{t}|} Z_i \tag{53}$$

for some c_2 independent of $\boldsymbol{t}$. This then means

$$\begin{aligned} P_{\text{Markov}}(\boldsymbol{t} \mid A{=}1) &= c_1 \cdot P_{\text{Markov}}(A{=}1 \mid \boldsymbol{t}) P_{\text{Markov}}(\boldsymbol{t}) \\ &= \frac{c_1}{c_2} \prod_{i=1}^{|\boldsymbol{t}|} \frac{P(t_i) \cancel{Z_i}}{\cancel{Z_i}} \\ &= \frac{c_1}{c_2} P_{\text{LM}}(\boldsymbol{t}) \propto P_{\text{LM}}(\boldsymbol{t}) \end{aligned} \tag{54}$$

and thus segmentations are returned from this sampler at the same relative proportions as the language model joint unigram probability would dictate. ■

Since $1 \geq Z_i \geq P(t_i) > 0$, the product in Eq. 53 is very small. Because this equation gives the acceptance rate, it being small implies many rejections and many retries. To avoid this, we should make c_2 as small as possible whilst respecting the fact that $0 \leq P_{\text{Markov}}(A = 1 \mid \boldsymbol{t}) \leq 1$. It is above 0 as long as $c_2 > 0$. It is under 1 as long as

$$\forall \boldsymbol{t} : \frac{1}{c_2} \prod_{i=1}^{|\boldsymbol{t}|} Z_i \leq 1 \iff \forall \boldsymbol{t} : c_2 \geq \prod_{i=1}^{|\boldsymbol{t}|} Z_i \tag{55}$$

so c_2 must be at least as big as the biggest product, i.e.

$$c_2 \geq \max_{\boldsymbol{t}} \prod_{i=1}^{|\boldsymbol{t}|} Z_i \tag{56}$$

and since we want c_2 to be as small as possible,

$$c_2 = \max_{\boldsymbol{t}} \prod_{i=1}^{|\boldsymbol{t}|} Z_i. \tag{57}$$

In other words: c_2 in Eq. 53 is the largest product of normalising denominators that can be encountered when walking through the segmentation graph, along one of the $N_V(s) \leq 2^{|s|-1}$ paths $\boldsymbol{t}$.

Luckily, this can be computed in $\Theta(|s|^2)$ time with a Viterbi maximiser like the one used by ULM, except rather than taking and comparing products of token probabilities, we take and compare products of denominators Z_i.

C Length-ordered composition keys (LOCKs) to sort segmentations

In order to visualise a skew towards segmentations with fewer, larger tokens in a tokeniser's output distribution for a string s of n characters, we need to be able to place a given segmentation $[i_0, i_1, i_2, \ldots, i_\ell]$ on a horizontal axis *without* explicitly generating and sorting the exponentially many segmentations on the axis – because it takes an intractably large amount of space and time, respectively $O(2^{n-1})$ and $O(2^n + 2^{n-1} \log 2^{n-1}) = O(n\, 2^n)$ – and *in such a way that* segmentations with fewer, larger tokens are concentrated on one side of the axis and those with more, smaller tokens appear on the other. We can do this with a suitable function that maps each segmentation to a unique integer between 0 and $2^{n-1} - 1$.

C.1 Bit-ordered

One example of a function that does this, works like as follows. Since a segmentation is entirely defined by the $n - 1$ binary choices of whether to put a token boundary at each inter-character position (1) or not (0), encoding those decisions into a binary string gives a unique $(n-1)$-bit number that can be converted to a decimal integer between 0 and $2^{n-1} - 1$. The problem with this function is that it orders segmentations very poorly: for example, the two very distinct segmentations *a/bcdef* and *ab/c/d/e/f* correspond to respectively $10000_2 = 16$ and $01111_2 = 15$, which are next to each other.

C.2 Length-ordered

We propose instead a function f to bijectively map segmentations to integers in $\{0, 1, \ldots, 2^n - 1\}$ such that the results are ordered first by the *token amount*, then by the *ordered token lengths*, and haphazardly (but uniquely) after that (since it is desirable for the order to not prefer one segmentation over another if they have the same list of token lengths).

We represent the segmentations as ordered lists of token lengths: e.g., the segmentations $\tau_1 =$ *a/bcdef* and $\tau_2 =$ *ab/c/d/e/f* become $t_1 = (1, 5)$ and $t_2 = (2, 1, 1, 1)$, respectively.

Formally, we ensure three ordering rules between any two distinct segmentations τ_1 and τ_2. Let t_1 and t_2 be the ordered lists of their token lengths. Then $f(\tau_1) < f(\tau_2)$ if and only if

1. $|t_1| < |t_2|$, or
2. $|t_1| = |t_2|$, $\text{sort}(t_1) < \text{sort}(t_2)$, or
3. $|t_1| = |t_2|$, $\text{sort}(t_1) = \text{sort}(t_2)$, $g(t_1) < g(t_2)$

where sorting is ascending, lists of equal length are compared element-by-element until a pair is not equal, and g is a function that doesn't generate unique outputs for *all* lists, but does generate unique outputs for lists containing the same elements. That is, $g(t)$ is a bijection between the permutations of $\mathrm{sort}(t)$ and

$$\mathrm{MN}(t) = \binom{|t|}{c_1, c_2, \ldots, c_k} = \frac{|t|!}{c_1!c_2!\cdots c_k!} \tag{58}$$

unequal integers, where $c_1, c_2, \ldots, c_k$ are the counts of the k unique elements of t.[16] Though not necessary, we require that those unequal integers are $0\ldots\mathrm{MN}(t)-1$.

The above three rules define a total order and could be enforced by sorting with a three-tiered sorting key (k_1, k_2, k_3). To compute f *without* explicitly sorting using this key, we need to compute how many segmentations *would* be ordered under (k_1, k_2, k_3). In other words: how many have a first key under k_1 (how many segmentations exist with fewer tokens) and how many with the same first key (same amount of tokens) have a second key under k_2 (how many permutations exist for sorted lists that also sum to n but are elementwise smaller), and we'll need an implementation of g. More formally, for the set $\mathcal{G}$ of all possible token length lists of the string,

$$f_1(t) = \#\{u \in \mathcal{G} : |u| < |t|\} \tag{59}$$
$$f_2(t) = \#\{u \in \mathcal{G} : |u| = |t| \wedge \mathrm{sort}(u) < \mathrm{sort}(t)\} \tag{60}$$
$$f_3(t) = \#\{u \in \mathcal{G} : |u| = |t| \wedge \mathrm{sort}(u) = \mathrm{sort}(t) \wedge g(u) < g(t)\} \tag{61}$$

should be computed as part of

$$f(\tau) = f_1(t) + f_2(t) + f_3(t). \tag{62}$$

We call the resulting identifier a *length-ordered composition key (LOCK)*, because it defines sorting keys over the *integer compositions* of the number n (i.e. the ordered lists of integers greater than 0 that sum to n). Below, we show that we can compute the functions f_1, f_2 and f_3 efficiently.

C.2.1 f_1: Compositions with fewer integers

Given a composition of length $|t|$ for n, we need to know the amount of compositions for n that have $m = 1, 2, \ldots, |t|-1$ integers. Mapping back to segmentations for a moment: having m tokens means having chosen $m-1$ token boundaries among the $n-1$ possible locations, and there are $\binom{n-1}{m-1}$ ways to do this. Hence, the first term in f is

$$\begin{aligned} f_1(t) &= \sum_{m=1}^{|t|-1} \binom{n-1}{m-1} \\ &= 2^{n-1} - \binom{n-1}{|t|-1} {}_2F_1(1, |t|-n; |t|; -1) \end{aligned} \tag{63}$$

where ${}_2F_1(a, b; c; z)$ is the Gaussian hypergeometric function.

C.2.2 f_2: Permutations of lower-order partitions

Sorted compositions are called *integer partitions*. They represent equivalence classes across the integer compositions. For segmentations, this means *ab/c/def* and *a/bcd/ef*, having token lengths [2,1,3] and [1,3,2] respectively, belong to the same equivalence class represented by the partition [1,2,3], unlike *a/bcde/f* which still has three tokens but instead belongs to the lower-order class represented by the partition [1,1,4].

The function $f_2(t)$ is the number of *compositions* of n into $|t|$ integers that, after sorting into ascending order with sort, are strictly smaller than $\mathrm{sort}(t)$, comparing elements left to right. The reason we choose to order partitions this way and no other way (e.g. comparing right to left, or some other procedure) is that we want the left side of the LOCK axis to represent segmentations that become more common when GRaMPa is skewed more. When $\tau \to \pm\infty$, all nodes in the segmentation DAG have a uniform distribution across their arcs, exactly like P_{Markov} in Eq. 1, and it can be proved that if you pick the most probable path from each of the equivalence classes above, ordering them from most to least probable is the same as ordering them by comparing their elements left-to-right.

We are not aware of a closed-form expression for $f_2(t)$, so we proceed as follows: *if* we knew the partitions representing the equivalence classes preceding the class to which t belongs, we could apply Eq. 58 to get the size of each class and sum the results. For knowing partitions to be feasible, two things must be true: the amount of partitions to generate must never explode (even for the last-ranked partition), and the work required *per generated partition* also must never explode.

The amount of integer partitions of n scales with $O\left(\frac{1}{n}\exp(\sqrt{n})\right)$ (Hardy and Ramanujan, 1917), but fortunately, it is still quite moderate for $n < 45$,

[16]The denominator avoids permutations of equal elements.

which is more than enough for tokenising words.[17]

An iterative algorithm to generate integer partitions of length $|t|$ in exactly the order we want, and one at a time (allowing early stopping), was described by Hindenburg (1779, pp. 74–76) and rephrased by Knuth (2014, section 7.2.1.4, Algorithm H and Theorem H). For clarity, however, we reimplement it recursively in Algorithm 4. The time complexity of this algorithm scales as the amount of partitions it outputs, namely $O\left(\frac{1}{n-|t|}\exp(\sqrt{n-|t|})\right)$ (Oruç, 2016). In practice the algorithm performs well. E.g.: it takes 2 ms on a single Intel i7 core to find all sorted lists of token lengths (partitions) of $k = 20$ tokens that sum to $n = 40$ characters, out of the 68.9 billion unsorted such lists (compositions).

Algorithm 4 Generating the integer k-partitions (sorted k-compositions) of n in lexicographic order

```
function PARTITIONSK($n$, $k$, PREFIX= [ ])
    if $k = 0$ then
        yield PREFIX
    else
        if $k = 1$ then
            $L \leftarrow n$
        else if LENGTH(PREFIX) $> 0$ then
            $L \leftarrow$ LAST(PREFIX)
        else
            $L \leftarrow 1$
        $U \leftarrow \lfloor n/k \rfloor$
        for $s \in L, \dots, U$ do
            for $p \in$ PARTITIONSK($n-s$, $k-1$,
                    APPEND(PREFIX, $s$)) do
                yield $p$
```

C.2.3 f_3 : Lower-order permutations of the same partition

Finally, let's define $g(t)$, a bijection from a permutation of a list $\mathrm{sort}(t)$ with possible duplicate elements to a unique integer identifier.

Start at permutation identifier 0. First do a pass through the list counting how many of each unique element is present. Then walk through a second time, from left to right, keeping track of how many of each unique element is yet to be encountered on this walk. Then, at each position i with value $t[i]$, find which of the remaining elements are lower than $t[i]$, and for each such element e, we use Eq. 58 to count how many permutations *could have been* formed with the remaining elements if, rather than having $t[i]$ at position i, we had e there and still had $t[i]$ available. Because $e < t[i]$, we know for sure that all permutations that have e in position i are ordered below the current permutation and hence occupy that many permutation identifiers already, so we bump the current identifier by at least that amount. At the end of the walk, whatever number we arrive at is unique for the given permutation. Call this number $g(t)$. A pseudocode implementation is given in Algorithm 5.

For $f_3(t)$, we wanted to know the amount of permutations whose identifier precedes t. Since the identifiers are integers starting at 0 with no gaps in between, the amount of identifiers under $g(t)$ is just $f_3(t) = g(t)$.

Algorithm 5 Counting multiset permutations preceding t in lexicographic order

```
function $g(t)$
    $n \leftarrow |t|$
    ▷ CTR is a dict mapping $t[i]$ to their counts
    CTR $\leftarrow$ COUNTUNIQUE($t$)
    $(a_1, \dots, a_k) \leftarrow$ SORTASC(CTR.KEYS)
    $g \leftarrow 0$
    for $i \in \{1, \dots, n\}$ do
        for $a \in (a_1, \dots, a_k)$ do
            if $a < t[i]$ and CTR$(a) > 0$ then
                CTR$(a) \leftarrow$ CTR$(a) - 1$
                ▷ MN is given by Eq. 58
                $g \leftarrow g +$ MN$(n-i$, CTR.VALS$)$
                CTR$(a) \leftarrow$ CTR$(a) + 1$
        CTR$(t[i]) \leftarrow$ CTR$(t[i]) - 1$
    return $g$
```

[17] For some sense of scale: $n = 37$ has just over 20 000 integer partitions, of which we only need a fraction since we only care about those with length $|t|$. Meanwhile, this n has $2^{37-1} = 68.7$ billion integer compositions.

D Experimental Setup

D.1 Hardware

Vocabularisation with SentencePiece was done on an Intel Xeon Platinum 8360Y (2.4 GHz) with peak memory usage being about 50 GiB for BPE and 260 GiB for ULM.

Pre-training the 14 DeBERTa models to 512 batches was done with one GPU each. Half were trained on an NVIDIA A100 SXM4 (80 GiB), the other half on an NVIDIA H100 HBM3 (80 GiB). *Each pre-training run* took about a day on the A100 and about half a day on the H100. Experimental runs amounted to about 4 extra days on one A100.

Fine-tuning was also done on a mixture of A100s and H100s. The precise mixture is shown in Table 3, although we didn't track steps 1–4 in § D.2 (the 5 shortened hyperparameter tuning runs preceding each reported fine-tuning run). Considering one H100 as equivalent to two A100s, the *total compute* spent was at least 20 A100 GPU days.

	A100		H100		A100-eq. total	
	hours	days	hours	days	hours	days
pretraining	259.48	10.81	79.22	3.30	417.92	17.41
finetuning	27.35	1.14	19.00	0.79	65.35	2.72
all	286.82	11.95	98.22	4.09	483.27	20.14

Table 3 – Minimal GPU time spent on this paper.

D.2 Hyperparameters

Pre-training We use the pre-training hyperparameters in Table 4. The masking rate is taken from BERT (Devlin et al., 2019). The context length was chosen to be twice that of RoBERTa (Liu et al., 2019) because of the finding that uniform sampling doubles the amount of tokens produced by the tokeniser. The device batch size was chosen because after including the model and optimiser, it held the VRAM usage of the GPUs at a constant 94% given packed examples. The effective batch size (i.e. the amount of examples between gradient descents) was chosen equal to that of DeBERTa (He et al., 2021) and because it is the best-performing size in the RoBERTa paper. The validation interval and validation set size were chosen based on timing benchmarks (§ D.3) such that respectively (1) no more than 4 hours would pass without evaluation and checkpointing, and (2) evaluation would take no more than about 10% of the total compute.

The learning rate schedule was chosen to *not* decrease after reaching its peak because reducing the size of gradient updates is exactly the purpose of an adaptive momentum-based optimiser, which is AdamW (Loshchilov and Hutter, 2019) with default hyperparameters in our case. We use much fewer warmup batches than the aforementioned papers because we expect to train for no more than two days; we take this number from Kalra and Barkeshli (2024) who found it to be the minimal number needed for stable training.

The maximum number of batches in the stopping criterion, namely 512, was selected due to a server outage in pre-training, causing some runs to terminate early at that value.

Hyperparameter	**Value**
Device batch size	$2^7 = 128$ ex
Effective batch size	$2^{11} = 2048$ ex/bs
Validation set size	$2^{14} = 16384$ ex
Context length	$2^{10} = 1024$ tk/ex
Validation interval	$2^6 = 64$ bs
Warmup batches	$2^8 = 256$ bs
Peak learning rate	1×10^{-3}
Learning rate schedule	Fixed w/ warmup
Stopping criterion	Convergence or 512 bs
Masking rate	15%
Tied embeddings	yes
AdamW decay rate	0.01
AdamW (β_1, β_2)	(0.9, 0.999)
Layers	6
Embedding size	512
FFNN size per layer	2048
Att. heads per layer	8
DeBERTa k	512

$\Rightarrow$ Base model size: 39 362 560 parameters

Table 4 – DeBERTa pre-training hyperparameters

Hyperparameter	**Value**
Warmup batches	{50, 100, 500, 1000}
Effective batch size	{16, 32, 64, 128, 256, 512}
Learning rate	{1e-6, 5e-6, 1e-5, 5e-5, 1e-4, 5e-4, 1e-3}
AdamW decay rate	{0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.10}

Table 5 – DeBERTa fine-tuning hyperparameter domains

Fine-tuning All fine-tuning experiments proceeded as follows, using the following method for hyperparameter search: for a given task $\mathcal{T}$ and a pre-trained model $\mathcal{M}_0$,

1. Take 5 samples of the form $h = (w, b, r, \lambda)$ from the grid in Table 5.

2. For each sample h: fine-tune $\mathcal{M}_0$ on $\mathcal{T}$ with hyperparameters h, evaluating and checkpointing after every $\frac{2^{14}}{5}$ examples or 1 epoch (whichever is smaller), stopping if either (1) validation loss $\mathcal{L}_v$ hasn't decreased for 5 evaluations in a row or (2) 2^{14} examples have been seen. Finally, keep the checkpoint with lowest $\mathcal{L}_v$ and call it $\mathcal{M}_1^h$.

3. For each sample h: evaluate $\mathcal{M}_1^h$ on a task-dependent downstream metric $m_\mathcal{T}$.

4. Select the best sample h^* according to the best value of $m_\mathcal{T}$, which in our case is always better when bigger.

5. Fine-tune $\mathcal{M}_0$ on $\mathcal{T}$ with hyperparameters h^*, evaluating and checkpointing after every 2^{14} examples or 1 epoch (whichever is smaller), stopping if either (1) $m_\mathcal{T}$ hasn't decreased for 5 evaluations in a row or (2) 2^{19} examples have been seen. Call the resulting fine-tuned model $\mathcal{M}_2$.

6. Evaluate $\mathcal{M}_2$ on the test set and report that.

D.3 Benchmarks

During pre-training, processing 1 effective batch (from tokenisation to gradient descent) took the models 2m30s with the baseline tokenisers (20 minutes per evaluation, 160 minutes between evaluations) and 3m45s with the GRaMPa tokenisers (30 minutes per evaluation, 240 minutes between evaluations).

D.4 Dataset sizes

Table 6 shows the sizes of the different partitions of the datasets used in the experiments. SlimPajama was truncated to the first 50k in training.

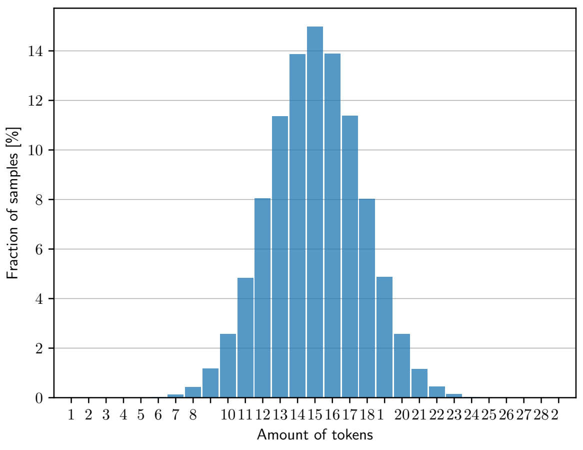
Since the test set for all the GLUE datasets has no publicly available labels, we replace it by taking a random sample (stratified by label value) out of the train set equal in size to the validation set. This is better than splitting the validation set, since it is usually the case in GLUE that the train and test set are abundant whilst the validation set is very lean.

Dataset	Train	Validation	Test
SlimPajama[18]	50000	20000	–
CoNLL-2003	14041	3250	3453
UD$_{\text{en\_ewt}}$	12543	2002	2077
QQP	323416	40430	40430
SST-2	66477	872	872
MRPC	3260	408	408
RTE	2213	277	277
WNLI	564	71	71

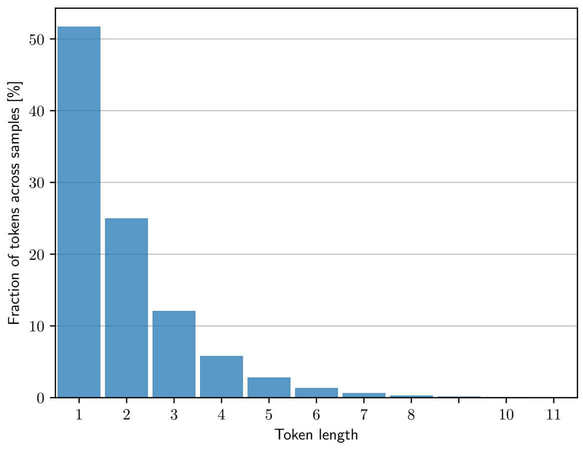
Table 6 – Dataset sizes used in our fine-tuning experiments. The non-GLUE datasets are due to respectively Soboleva et al. (2023) (SlimPajama), Tjong Kim Sang and De Meulder (2003) (CoNLL-2003) and Silveira et al. (2014) (UD$_{\text{en\_ewt}}$). The GLUE tasks are due to respectively Iyer et al. (2017) (QQP), Socher et al. (2013) (SST-2), Dolan and Brockett (2005) (MRPC), Dagan et al. (2006); Bar-Haim et al. (2006); Giampiccolo et al. (2007); Bentivogli et al. (2009) (RTE), and Levesque et al. (2012) (WNLI).

[18]The training set for vocabularisation was 3 000 000 examples rather than the 50 000 for pre-training. The validation set was used only to compute the Rényi entropy and the regularisation rate of the BPE and ULM tokenisers.

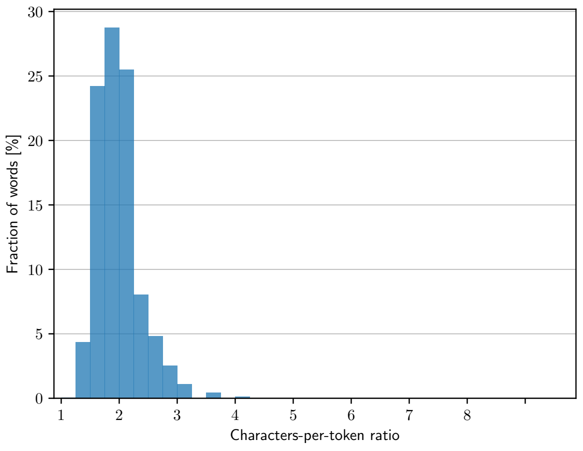
E Supplementary figures



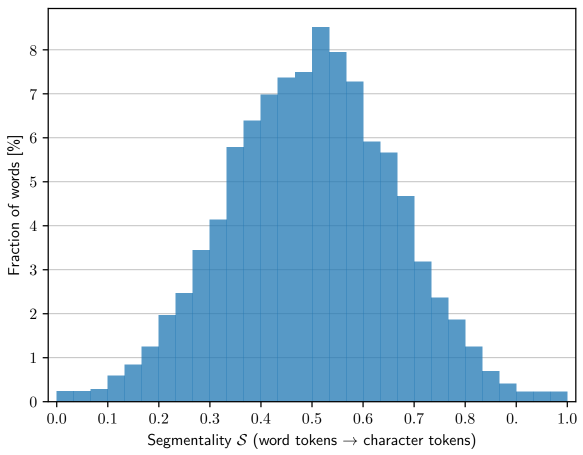
(a) Amount of tokens $m \sim 1 + \mathrm{Binom}(|s| - 1, \frac{1}{2})$



(b) Token length $\ell \sim \mathrm{Geom}(\frac{1}{2})$



(c) Characters-per-token ratio R



(d) Segmentality $\mathcal{S}$

Figure 2 – Different distributional views for a uniform segmentation distribution with infinite vocabulary. The top two plots aggregate 100 000 samples from the segmentations of a string with $|s| = 29$. The bottom plots were generated across the words in the 20k SlimPajama validation set.

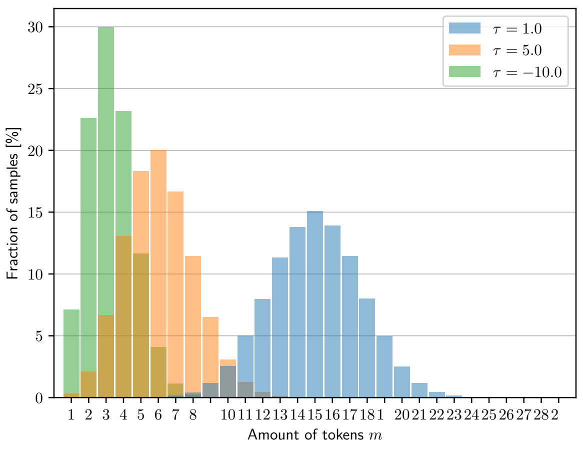


Figure 3 – Same histogram as Figure 2a for different skewing temperatures of GRaMPa.

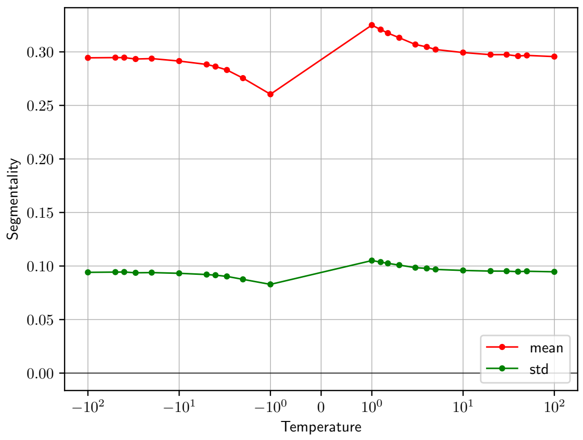
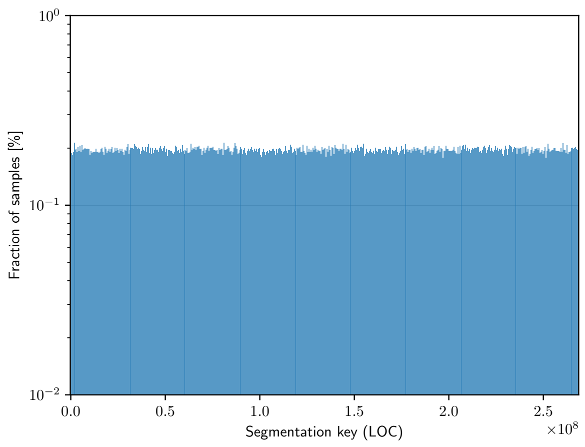
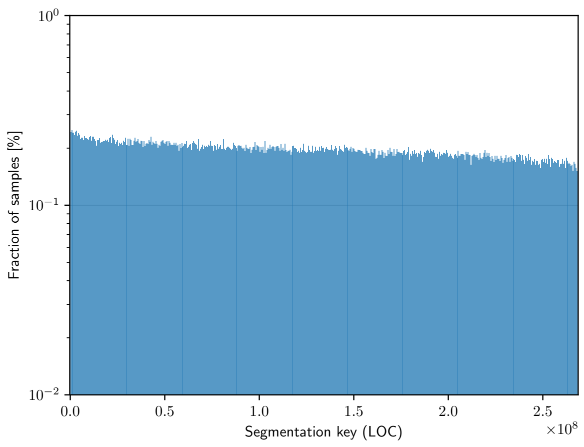


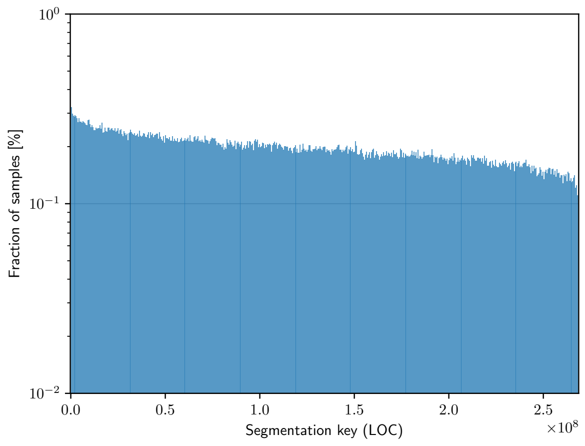
Figure 4 – Evolution of the mean and standard deviation of *segmentality* $\mathcal{S}$ (see § 3.1) w.r.t. GRaMPa temperature τ, measured across the 20k SlimPajama validation set.



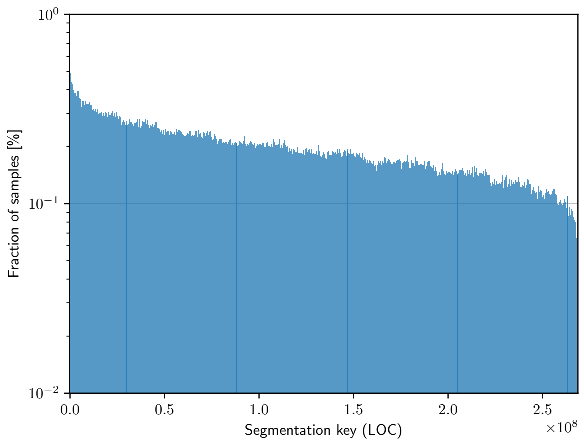
(a) $\tau = 1.0$



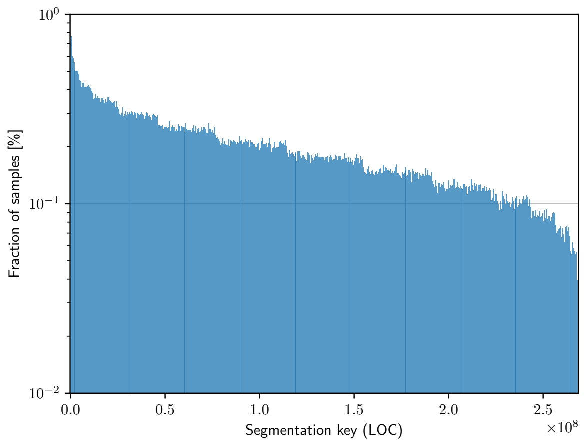
(b) $\tau = 1.025$



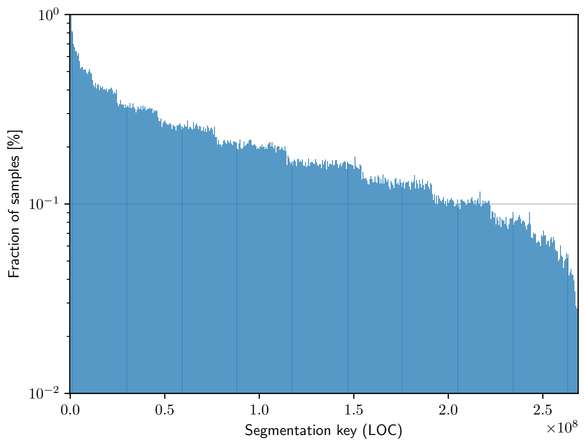
(c) $\tau = 1.05$



(d) $\tau = 1.1$

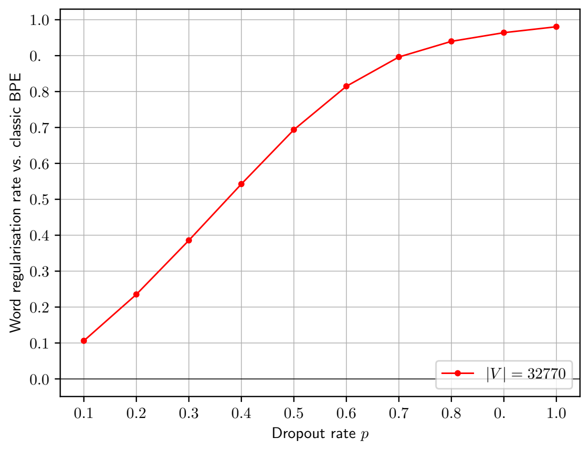


(e) $\tau = 1.15$

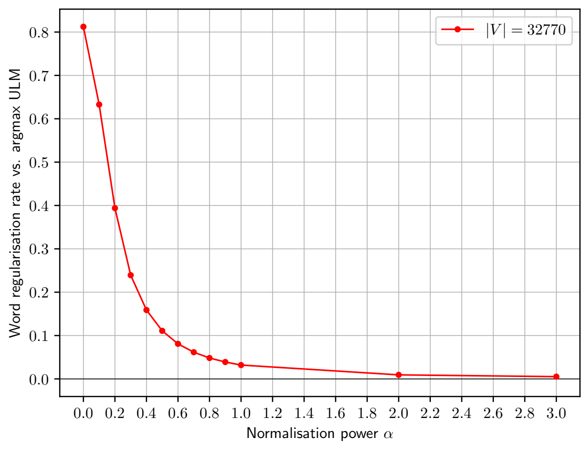


(f) $\tau = 1.2$

Figure 5 – Progressively stronger skew in the frequency distribution of segmentations with increasing GRaMPa temperature τ. The horizontal axis enumerates the domain of possible segmentations of a string. The histograms were generated by taking 100 000 samples from GRaMPa each time from a word of length $|s| = 29$, and then binning the $2^{28} = 268\,435\,456$ LOC keys for the possible segmentations (see § C) into adjacent spans of 512 bins. Since an infinite vocabulary was used, the histograms hold for any string of the same length.



(a) BPE-dropout



(b) Stochastic ULM with $k = 64$

Figure 6 – Regularisation rate (i.e. fraction of produced segmentations that differ from the one deterministic segmentation) of BPE and ULM w.r.t. their temperature-like hyperparameter in the SlimPajama 20k validation set.

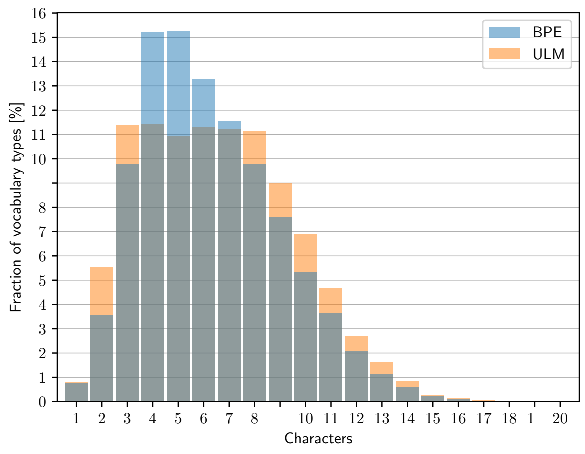


Figure 7 – Distribution of the length of the 32k subword types in our two SlimPajama-3M vocabularies.

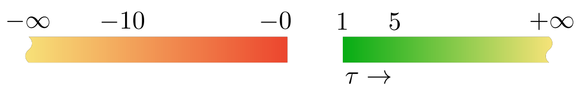
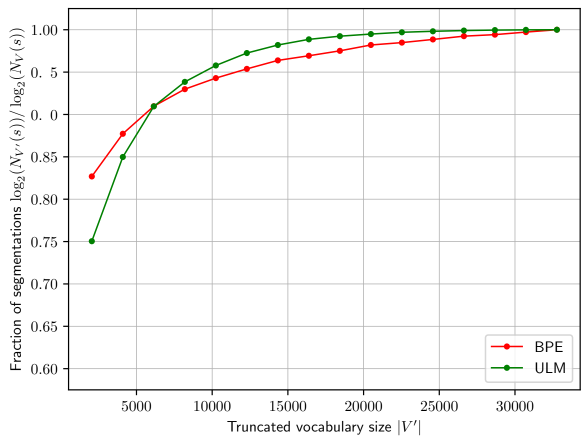
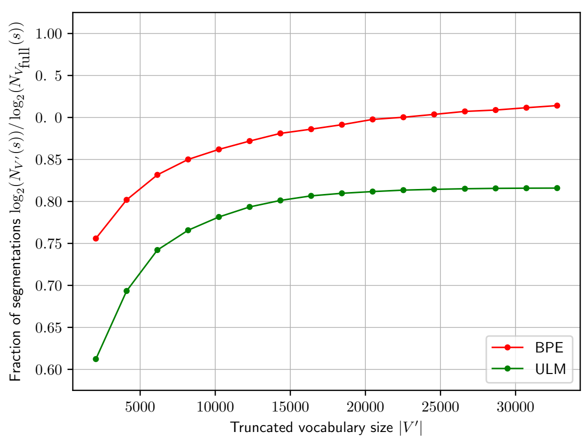


Figure 8 – Skewing away from uniformity ($\tau = 1$) towards segmentations with longer tokens (redder) overlayed on the GRaMPa temperature $\tau \in \mathbb{R}$ axis. The gap from $1 \rightarrow 0^+$ covers all skewing in the opposite direction (i.e. shorter tokens becoming even more common).

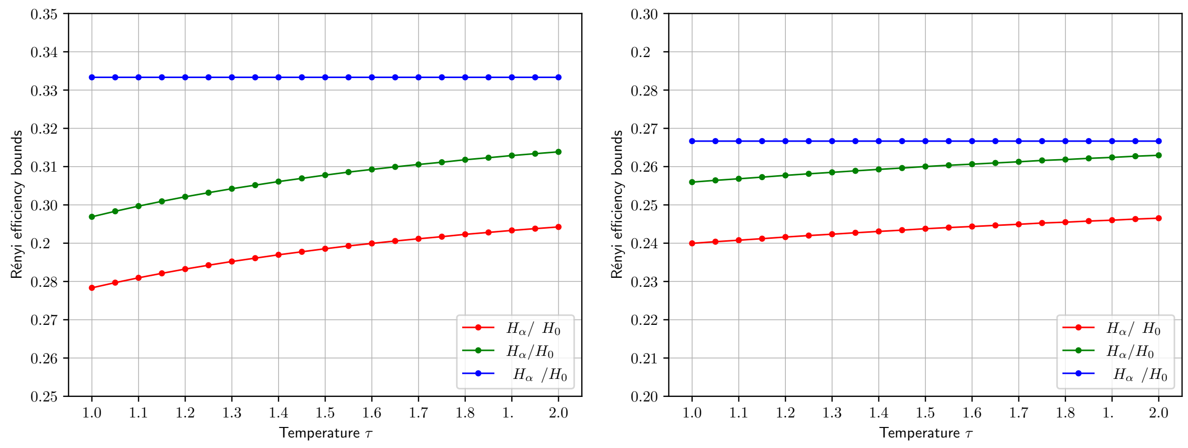


(a) relative to using our $|V| = 32\text{k}$ vocabularies



(b) relative to using an infinite vocabulary

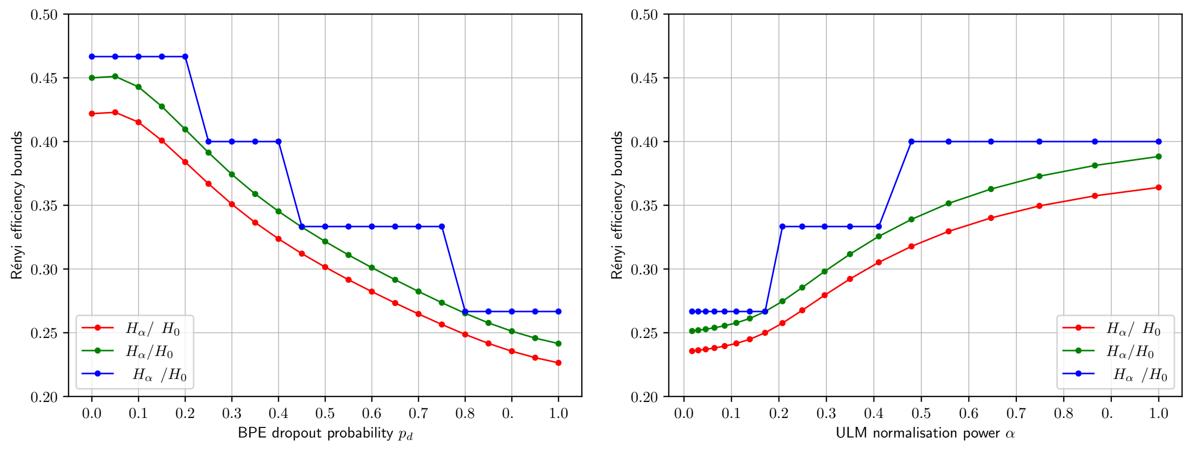
Figure 9 – Fraction of the maximal log-amount of segmentations that are available in the GRaMPa segmentation graph w.r.t. increasing vocabulary size $|V'|$, averaged over all words in the SlimPajama 20k validation set.



(a) $\ell_{\text{min}} = 1$ and BPE vocabulary

(b) $\ell_{\text{min}} = 1$ and ULM vocabulary

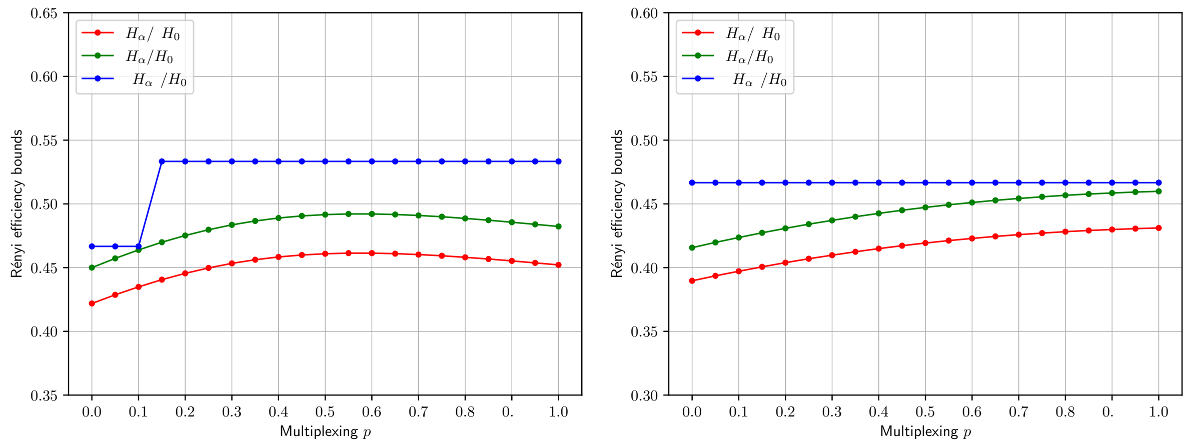
Figure 10 – Rényi efficiency E_α ($\alpha = 2.5$) w.r.t. non-multiplexed ($p = 1.0$) GRaMPa temperature τ.



(a) BPE-dropout rate p_d

(b) ULM($k = 64$) normalisation power α

Figure 11 – Rényi efficiency E_α ($\alpha = 2.5$) w.r.t. non-multiplexed non-GRaMPa tokeniser hyperparameters.



(a) $\ell_{\text{min}} = 2$ and $\tau = 1.0$, multiplexed with BPE

(b) $\ell_{\text{min}} = 2$ and $\tau = 1.0$, multiplexed with ULM($k = 1$)

Figure 12 – Rényi efficiency E_α ($\alpha = 2.5$) w.r.t. GRaMPa multiplexing rate p.

				m		$\mathcal{S}$		ℓ		R	
				mean	std	mean	std	mean	std	mean	std
BPE	BPE		$p_d = 0.0$	1.27	1.08	0.05	0.18	4.19	2.62	**4.65**	2.58
			$p_d = 0.1$	1.41	1.16	0.10	0.24	3.79	2.57	4.39	**2.60**
	GRaMPa	$\ell_{\text{min}} = 1$	$\tau = 1.0$	3.42	2.21	0.54	0.32	1.56	0.86	1.74	**0.80**
			$\tau = 5.0$	2.85	1.89	0.44	0.33	1.87	1.29	2.19	1.39
			$\tau = -10.0$	2.59	1.75	0.40	0.34	2.06	1.58	2.49	1.79
		$\ell_{\text{min}} = 2$	$\tau = 1.0$	2.25	1.50	0.25	0.23	2.37	1.23	2.58	1.21
			$\tau = 5.0$	2.14	1.45	0.23	0.23	2.50	1.48	2.78	1.54
			$\tau = -10.0$	2.09	1.43	0.22	0.23	2.56	1.61	2.88	1.70
ULM	ULM	$k = 1$	$\alpha = 1.0$	**1.24**	**0.78**	**0.04**	**0.13**	**4.30**	**2.64**	4.64	2.58
		$k = 64$	$\alpha = 0.15$	2.17	1.45	0.29	0.32	2.45	2.03	3.11	2.28
	GRaMPa	$\ell_{\text{min}} = 1$	$\tau = 1.0$	**3.55**	**2.30**	**0.57**	0.33	**1.50**	**0.85**	**1.72**	0.89
			$\tau = 5.0$	2.82	2.00	0.44	0.36	1.89	1.46	2.39	1.74
			$\tau = -10.0$	2.46	1.83	0.38	**0.37**	2.17	1.87	2.86	2.21
		$\ell_{\text{min}} = 2$	$\tau = 1.0$	1.86	1.51	0.16	0.24	2.87	1.84	3.41	1.92
			$\tau = 5.0$	1.78	1.45	0.15	0.23	3.00	2.03	3.57	2.10
			$\tau = -10.0$	1.74	1.42	0.14	0.23	3.06	2.12	3.65	2.19

Table 7 – Mean and standard deviation for segmentation properties of the tokenisers used in this paper (see § 3.1) on the 20k SlimPajama validation set. m is averaged across (space-separated) words, ℓ is averaged across produced tokens, and both $R = |s|/m$ and $\mathcal{S} = (m-1)/(|s|-1)$ are macro-averaged across words. m and $\mathcal{S}$ can be viewed as unnormalised and normalised fertility. ℓ can be viewed as the micro-average version of R. ***Note:*** Unlike Table 1, the GRaMPa tokenisers shown here are ***without*** multiplexing with a deterministic tokeniser. The values as seen by the models of Table 1 lie halfway between each deterministic tokeniser's mean and each GRaMPa tokeniser's mean (since they were multiplexed with $p = 0.5$). For example, when multiplexing GRaMPa($\ell_{\text{min}} = 1, \tau = 5.0$) with ULM, its mean m is $\frac{1}{2}(1.24 + 2.82) = 2.03$.

				$H_1^{\cancel{m}}/H_0^{\cancel{m}}$		H_1/H_0		$\text{RR}^{\cancel{m}}$		$\max(C, U)$		C		U	
				mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
BPE	BPE		$p_d = 0.0$	0.00	0.00	0.00	0.00	0.00	0.00	16.33	17.38	16.24	17.45	1.00	0.00
			$p_d = 0.1$	48.36	34.49	22.39	16.42	11.13	7.16	44.62	38.38	44.33	38.70	3.32	1.51
	GRaMPa	$\ell_{\text{min}} = 1$	$\tau = 1.0$	97.12	3.20	96.75	3.20	79.48	17.78	94.69	9.85	87.20	25.15	25.17	28.56
			$\tau = 5.0$	92.68	5.84	91.42	6.40	74.29	15.84	89.40	15.86	83.10	27.44	21.53	23.36
			$\tau = -10.0$	88.55	8.39	85.95	10.31	69.34	13.39	84.31	21.11	79.39	29.72	18.15	18.43
		$\ell_{\text{min}} = 2$	$\tau = 1.0$	32.65	24.99	44.89	21.95	58.21	30.40	34.41	14.19	32.11	16.26	6.71	8.66
			$\tau = 5.0$	31.94	24.29	43.80	21.24	55.81	28.63	34.20	14.15	32.07	16.32	6.50	7.89
			$\tau = -10.0$	31.30	23.69	42.81	20.65	54.09	27.38	33.99	14.21	32.02	16.39	6.29	7.26
ULM	ULM	$k = 1$	$\alpha = 1.0$	0.00	0.00	0.00	0.00	0.00	0.00	18.93	17.39	18.87	17.46	1.00	0.00
		$k = 64$	$\alpha = 0.15$	83.56	12.38	73.64	14.63	51.29	16.52	82.35	25.73	80.33	30.03	12.00	11.14
	GRaMPa	$\ell_{\text{min}} = 1$	$\tau = 1.0$	97.46	3.06	97.10	3.06	76.87	17.80	95.59	9.14	89.99	22.41	21.09	26.52
			$\tau = 5.0$	92.33	7.24	88.72	8.92	66.45	15.37	89.90	16.67	85.43	25.72	17.36	20.60
			$\tau = -10.0$	87.50	12.19	79.87	16.21	57.73	14.54	83.95	23.64	80.77	29.36	13.81	15.36
		$\ell_{\text{min}} = 2$	$\tau = 1.0$	12.71	20.15	20.20	22.90	30.93	34.13	25.30	16.40	24.35	16.99	3.19	5.15
			$\tau = 5.0$	12.53	19.80	19.60	22.16	28.84	31.54	25.26	16.38	24.35	16.99	3.14	4.84
			$\tau = -10.0$	12.36	19.49	19.03	21.50	27.45	29.97	25.20	16.37	24.34	17.00	3.08	4.54

Table 8 – Mean and standard deviation for properties of *the distribution of* segmentations produced by the tokenisers used in this paper (again without multiplexing, as in Table 7). To generate segmentation distributions, we use the first 10% (2k) of examples in our SlimPajama validation set, split those into space-separated words, and then for each tokeniser, we filter out the words that have only one possible segmentation and sample the tokeniser $M = 100$ times for the others. For each word's segmentation distribution, we compute the *Shannon efficiency* (i.e. the percentage of the maximal possible Shannon entropy H_0 that is achieved by the actual Shannon entropy H_1, where $H_0 = \log_2(\min\{M, N_V(s)\})$ corrects for the fact that M samples may not be able to cover the entire domain and hence even in the uniform case have entropy $H_1 = \log_2(M) < \log_2(N_V(s)) = H_0'$) of respectively the distribution *without* the most likely segmentation ($H_1^{\cancel{m}}/H_0^{\cancel{m}}$) and the entire distribution (H_1/H_0). We also compute the *regularisation rate* w.r.t. that most likely segmentation (i.e. the percentage of segmentations that are *not* that segmentation) $\text{RR}^{\cancel{m}}$. Finally, we count the amount of unique segmentations u produced for a string and then compute their *coverage* $C = u/N_V(s)$ of the domain, the *uniqueness* $U = u/M$ of the samples, and the maximum of the two (which measures uniqueness for distributions that could have never been covered fully with M samples, and coverage for those that could, since $\max\{C, U\} = \max\{u/N_V(s), u/M\} = u/\min\{N_V(s), M\}$).

				Token-level						Sequence-level						
				PoS	NER	DP				SST-2	QQP	MRPC	RTE	WNLI		
				Acc	F_1	UAS	LAS	UCM	LCM	Acc	F_1	Acc	Acc	Acc	$\bar{\%}$	$\bar{\Delta}$
BPE	BPE-dropout		$p_d = 0.1$	**95.27**	81.13	81.64	77.18	49.40	39.38	**89.91**	74.42	69.12	53.43	46.48	68.85	
	GRaMPa	$\ell_{\min} = 1$	$\tau = 1.0$	–2.74	–2.32	–1.98	–3.34	–4.67	–5.20	–6.42	–2.39	+0.74	+0.00	**+4.23**	66.66	–2.19
			$\tau = 5.0$	–2.08	–1.06	+0.26	–0.12	–0.72	–1.06	–3.90	–4.83	**+1.47**	+0.72	**+4.23**	68.21	–0.64
			$\tau = -10.0$	–1.96	–2.32	–1.70	–2.33	–2.84	–2.41	–5.85	–5.32	–0.74	–10.47	**+4.23**	65.97	–2.88
		$\ell_{\min} = 2$	$\tau = 1.0$	–0.63	–0.14	**+3.75**	**+4.25**	**+4.53**	**+5.44**	–4.70	**+0.99**	+0.49	–1.44	+2.82	70.25	+1.40
			$\tau = 5.0$	–1.18	**+0.32**	–0.59	–0.74	–2.07	–0.96	–6.77	–1.24	+0.00	–1.08	–1.41	67.42	–1.43
			$\tau = -10.0$	–2.14	–1.30	–2.82	–3.52	–2.94	–3.23	–6.88	–1.24	+0.98	**+1.44**	**+4.23**	67.27	–1.58
ULM	ULM	$k = 64$	$\alpha = 0.15$	92.74	79.26	82.10	77.25	49.11	39.77	82.57	68.08	69.85	**53.43**	50.70	67.71	
	GRaMPa	$\ell_{\min} = 1$	$\tau = 1.0$	+0.33	–2.45	+0.98	+1.19	–0.39	–0.19	+3.78	+2.60	–0.98	**+0.00**	–1.41	68.03	+0.32
			$\tau = 5.0$	+0.50	+0.06	+1.81	+2.35	+1.01	+1.25	+2.18	+3.47	+0.00	–0.72	–7.04	68.16	+0.44
			$\tau = -10.0$	–0.03	–1.69	+1.86	+2.53	+1.20	+1.83	+2.18	+4.16	–1.23	–2.89	**+2.82**	68.69	+0.98
		$\ell_{\min} = 2$	$\tau = 1.0$	**+1.93**	**+1.66**	+3.87	**+5.15**	+5.44	+5.58	**+4.82**	+6.35	–0.49	–2.89	–2.82	70.31	+2.60
			$\tau = 5.0$	+1.87	–0.26	**+3.98**	+5.06	**+6.45**	**+6.74**	+3.78	+6.88	**+0.49**	–0.36	–4.23	70.48	+2.77
			$\tau = -10.0$	+1.52	+1.39	+3.62	+4.93	+5.39	+5.87	+1.15	**+9.53**	–0.49	–2.17	+0.00	**70.51**	**+2.80**

Table 9 – Results of Table 1 expressed relative to the two baselines (in grey).

				UAS				LAS				UCM				LCM					
				tr/te	tr/te	tr/te	tr/te	tr/te	tr/te	tr/te	tr/te	tr/te	tr/te	tr/te	tr/te	tr/te	tr/te	tr/te	tr/te	$\bar{\%}$	$\bar{\Delta}$
BPE	BPE-dropout		$p_d = 0.1$	74.10	75.37	80.44	81.64	67.32	69.62	76.41	77.18	40.25	43.48	48.63	49.40	31.01	34.33	41.55	39.38	58.13	
	GRaMPa	$\ell_{\min} = 1$	$\tau = 1.0$	+1.91	+1.10	–1.23	–1.98	+1.24	+0.42	–2.51	–3.34	+1.54	–1.11	–3.56	–4.67	+0.29	–1.69	–6.36	–5.20	56.56	–1.57
			$\tau = 5.0$	+2.98	**+4.11**	+2.61	+0.26	+2.69	**+3.86**	+1.86	–0.12	+1.49	+1.64	+0.87	–0.72	–0.29	+0.63	–1.83	–1.06	59.32	+1.19
			$\tau = -10.0$	**+3.26**	+3.75	+3.09	–1.70	**+3.06**	+3.22	+2.85	–2.33	**+2.21**	**+2.79**	+2.36	–2.84	**+0.67**	**+1.11**	+0.34	–2.41	59.35	+1.21
		$\ell_{\min} = 2$	$\tau = 1.0$	–0.48	–0.96	**+4.46**	**+3.75**	–0.90	–1.43	**+4.34**	**+4.25**	+0.96	–1.73	**+3.42**	**+4.53**	+0.00	–0.72	**+2.02**	**+5.44**	59.81	+1.68
			$\tau = 5.0$	–0.92	+1.11	–1.88	–0.59	–1.36	+1.14	–2.92	–0.74	–0.34	+0.72	–2.60	–2.07	–1.64	+0.87	–4.53	–0.96	57.09	–1.04
			$\tau = -10.0$	–1.07	–0.37	+3.46	–2.82	–1.74	–1.07	+3.24	–3.52	–0.10	–1.40	+3.37	–2.94	–0.96	–2.50	+1.54	–3.23	57.50	–0.63
ULM	ULM	$k = 64$	$\alpha = 0.15$	77.11	79.13	81.34	82.10	69.78	72.88	75.94	77.25	42.32	44.82	46.61	49.11	30.91	34.33	36.16	39.77	58.72	
	GRaMPa	$\ell_{\min} = 1$	$\tau = 1.0$	–0.80	–1.90	+0.81	+0.98	–0.56	–2.17	+1.45	+1.19	–0.53	–0.82	+1.83	–0.39	+0.63	–0.34	+2.50	–0.19	58.83	+0.11
			$\tau = 5.0$	+0.84	+0.97	+1.82	+1.81	+1.17	+1.62	+2.65	+2.35	**+1.54**	+1.06	+2.84	+1.01	+2.21	+1.73	+3.27	+1.25	60.48	+1.76
			$\tau = -10.0$	–0.88	+0.46	+1.94	+1.86	–0.64	+1.09	+3.13	+2.53	–2.31	+1.11	+3.51	+1.20	–0.53	+2.26	+5.25	+1.83	60.09	+1.36
		$\ell_{\min} = 2$	$\tau = 1.0$	–0.96	+0.72	+3.30	+3.87	–0.52	+1.17	+4.79	**+5.15**	–0.72	+2.46	+5.87	+5.44	+0.24	+3.27	+7.32	+5.58	61.66	+2.94
			$\tau = 5.0$	**+1.72**	+2.44	**+3.50**	**+3.98**	**+2.00**	+3.37	**+5.13**	+5.06	+0.96	**+3.32**	**+6.84**	**+6.45**	+1.59	+3.27	**+8.23**	**+6.74**	**62.76**	**+4.04**
			$\tau = -10.0$	+1.15	**+2.77**	+2.02	+3.62	+1.72	**+3.77**	+2.98	+4.93	+1.49	+2.65	+4.38	+5.39	**+2.26**	**+3.80**	+5.44	+5.87	62.11	+3.39

Table 10 – Results of Table 2 expressed relative to the two baselines (in grey).