BPE-knockout: Pruning Pre-existing BPE Tokenisers with Backwards-compatible Morphological Semi-supervision

Thomas Bauwens and **Pieter Delobelle**
Department of Computer Science, KU Leuven
`firstname.lastname@kuleuven.be`

Abstract

Byte-pair encoding (BPE) has become the default subword tokeniser in language models (LMs), allowing the representation of an infinite space of text with a finite set of units. Yet, BPE training is unsupervised, receiving no explicit information about a language's morphology. This results in a subword vocabulary wherein many units are a concatenation of partial morphemes, preventing their formation as tokens. This, in turn, causes consistent intra-word patterns to be displayed inconsistently to downstream models, and bloats the vocabulary, hence requiring unnecessary embedding storage. In this paper, we address this issue by identifying blameworthy BPE merges and removing the resulting subwords from the BPE vocabulary, without impeding further use of merges that relied on them. We find that our method, BPE-KNOCKOUT, is effective at making BPE's segmentation positions adhere better to derivational and compound boundaries in English, Dutch and German, and improves token-based tasks in Dutch RoBERTa models, indicating that a tokeniser's adherence to morphology impacts downstream models. We demonstrate the latter not only by training LMs from scratch, but also by continuing the pre-training of existing LMs. This proves promising, showing that suboptimal tokenisers can be remedied whilst salvaging training cost of downstream LMs.

1 Introduction

Subword tokenisation has become a mainstay in natural language processing (NLP) in recent years, riding on the success of translation models (TMs) (Sennrich et al., 2016) and language models (LMs) (Devlin et al., 2019) that incorporate it as the first step to process text. Rather than keeping a theoretically infinite dictionary of words in the considered language, a subword tokeniser segments each word into smaller parts (*subword tokens*) such that each segment is part of a known and finite set (the *subword vocabulary* V of *subword types*).

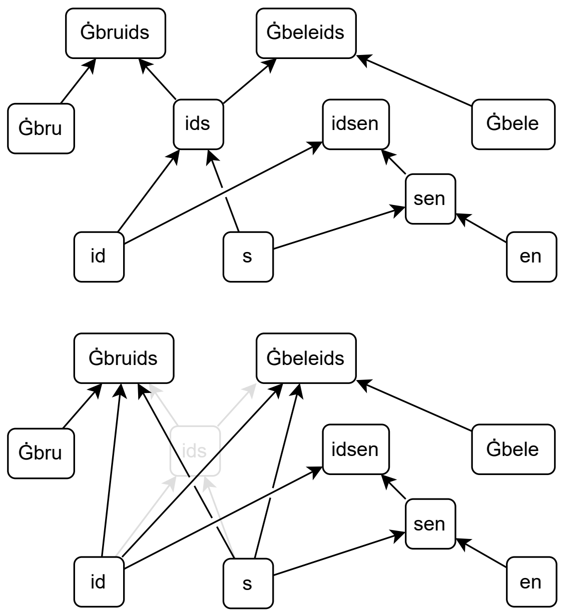


Figure 1 – Visualisation of the changes in part of the merge graph for RobBERT's BPE tokeniser after knockout of type *ids*, with merge *id+s*. (Note that as shown in §3.1, this graph is not sufficient to reconstruct the corresponding merges of $\mathcal{M}$. See Figure 11 for that.)

An early subword tokeniser was proposed by Creutz and Lagus (2002), and in the last decade, there has been a growing set of alternatives (Schuster and Nakajima, 2012; Varjokallio et al., 2013; Sennrich et al., 2016; Kudo, 2018; Wu and Zhao, 2018; Grönroos et al., 2020; He et al., 2020; Vilar and Federico, 2021). Subword tokenisation rose to fame when Sennrich et al. (2016) proposed translating a sequence of subwords (rather than words) into a sequence of subwords (rather than words) in the attention-based machine translation architecture of Bahdanau et al. (2015). In particular, Sennrich et al. proposed adapting Gage (1994)'s hierarchical compression algorithm *byte-pair encoding (BPE)* for building the subword vocabulary (later termed *vocabularisation* by Xu et al. (2021)), as well as for applying the learnt vocabulary in a tokeniser.

A recap of the BPE algorithm follows later. For now, it suffices to know that BPE vocabularisation builds types in the vocabulary with a bottom-up approach. This has its pathologies (Bauwens, 2023): e.g., some subwords are formed by adding few characters at a time, similar to the *chaining effect* in hierarchical clustering (Manning et al., 2008). Many of the stepping stones are never used afterwards, either to learn other types or even as tokens during application (Bostrom and Durrett, 2020), which means it is pointless to reserve storage for them in downstream models (Cognetta et al., 2024).

Furthermore, BPE vocabularisation and segmentation are both unsupervised, informed by nothing more than absolute frequencies in the training corpus. The benefit is that massive, automatically gathered corpora can be used. The downside is that no morphological constraints are imposed on the tokeniser, and indeed, BPE segmentation boundaries often deviate from morphological boundaries deduced from underlying morphemes (Huck et al., 2017; Zhou, 2018; Bostrom and Durrett, 2020).

In particular, BPE tends to abridge morphemes. In the best case, it fuses *full* morphemes into one token, but the problem is often worse: Ataman et al. (2017) show an example wherein the letters of the passivating Turkish morpheme *-ıl-* are pulled apart into the adjacent tokens, which obscures it enough to lose the passive voice in translation.

We hypothesise that recalling these boundaries in token sequences improves downstream performance of a model trained on them. Not only will this avoid evaporation of morphemes by subsuming their characters into other tokens (as above), but it will also provide the most reliable intra-word patterns possible in the model's input (namely, those dictated by the language of the corpus): as Tan et al. (2020) point out, the most consistent surface segmentation of danced, dancing and entered are the morphologically sound danc ed, danc ing and enter ed, using the same token for the same semantics. Patterns found in any other segmentation (e.g.: dance d, dancing, ente red) are necessarily more noisy, making the language harder to model.

That said, *only* recall of morphological boundaries is not sufficient; otherwise, character-level tokenisation would do. The advantage of subword tokenisation is that there are many more units to which a static embedding is assigned. Having more embeddings means having more storage space for pooling any knowledge obtained about a subword across all the contexts it appears in.

Ideally, all words containing the same morpheme would contribute towards informing a single embedding for that morpheme, rather than fragmenting knowledge about it across the vocabulary. This way, when an unseen word arrives with that morpheme, the model could draw on all of its past occurrences, rather than a subset. Furthermore, having only a single subword type for a morpheme could facilitate equity across languages, allowing a model to store the same amount of meanings given the same vocabulary capacity, even if its language produces many more surface word forms.[1]

In summary, BPE tokenisers have a bloated vocabulary and a lack of morphology. In light of this, we present the following contributions:

- We formalise BPE's learnt subword and merge relations in a deduplicated graph (§3.1).
- We address the bloated-vocabulary problem with a method to prune subword types from these graphs, which we call *knockout* (§3.3).
- We introduce a metric to detect BPE merges that cause a consistent mismatch between tokenisations and morphological decompositions, which we call *blame* (§3.4).
- We evaluate the combination of these two solutions, *BPE-knockout*, on three non-isolating languages (English, Dutch and German), and find significant gains in both precision and recall of derivational and compound boundaries.
- Finally, we pre-train several RoBERTa models with BPE and BPE-knockout, and show that BPE-knockout improves token-level classification tasks, particulary for models *already pre-trained* with BPE.

2 Related Work

Byte-Pair Encoding (BPE) Gage (1994)'s BPE consists of compressing a sequence of bytes by *iteratively selecting the most frequent pair of bytes* (x, y), *replacing it by a single unused byte* z, and recording this as the *merge* $x + y \to z$ stored in an ordered list M, so that decompression consists

[1]E.g.: the lexeme of the English verb TO CHOOSE only contains 4 word types {*choose*, *chooses*, *chose*, *chosen*}, whereas the equivalent French verb CHOISIR has many more – even just the indicative present tense has 5, {*choisis*, *choisit*, *choisissons*, *choisissez*, *choisissent*}. Given a fixed vocabulary size, a word-level tokeniser would fill up its capacity tens of times faster in French, and hence can't store the same amount of distinct verbs as in English, making the model arbitrarily less knowledgeable about the world. In a subword tokeniser, storing the essence of both lexemes can be done with 1 type, resp. *cho-* and *choisi-*, despite their vast size difference.

of reversing the merges from end to start. Sennrich et al. (2016) adapt BPE compression to text with two changes: they apply the algorithm to characters instead of bytes (later reverted by e.g. Radford et al. (2019) and Wang et al. (2019)), and they disallow merges involving a space character.

Whereas Gage intended BPE for compressing the sequence over which the merges were learnt (the "training corpus"), Sennrich et al. assumed these merges to also be applicable to unseen character sequences: their BPE tokeniser collects merges in a list M, and *replays them in order* so as to compress any input sequence, even if it contains never-before-seen words or has a different word distribution than the training corpus. This continues until no more merges can be applied; the remaining sequence of tokens is the BPE segmentation.

Wu and Zhao (2018) point out that many variants on BPE can be created by simply swapping out the metric which is maximised to select the next subword pair to be merged. In BPE, it is absolute frequency; they propose other metrics like "description length gain" (Kit and Wilks, 1999). Note that although this generalises the BPE algorithm, it does not formalise it mathematically as in §3.1.

Morphology As the application of a BPE tokeniser is deterministic (like its training), the same word always results in the same token sequence. Since LMs can't see the characters underlying a token, the token patterns they learn from should be as reliable as possible. Morphology could provide this reliability, but it is unlikely that the output of a BPE tokeniser accords with it: indeed, even when a morphological constraint is put on its data during training time, the same constraint is still violated at application time (Huck et al., 2017).
Therefore, token sequences often obscure the underlying morphemes, making it difficult for a model to discern when two very distinct token sequences actually have very similar character subsequences or morphology. Provilkov et al. (2020) propose *BPE-dropout* to remedy this: by providing a model with multiple equivalent segmentations of the same word at application time, it can "triangulate" its understanding of the underlying characters. This also improves support for homographs with unequal morphological decompositions, since now all of them could be seen rather than just one (or none) with a deterministic BPE tokeniser. Additionally, as BPE-dropout works by temporarily disabling some merges, it leads to smaller tokens (and longer input sequences) on average, and might hence recall more morphological boundaries.

The beneficial effect of adhering to morphological boundaries on downstream tasks was observed by Park et al. (2021), finding that causal LMs had consistently lower perplexities when pretokenising BPE's input with a morphological analyser. In masked LMs, Hofmann et al. (2021) found that even just isolating hand-picked derivative affices improved topic and sentiment classification, tracing back the effect to poor segmentation of word stems due to merges with surrounding affices.

Redundancy It is well-known that several units of natural language, in particular word types, have a Zipfian frequency distribution (Zipf, 1949): when ranked by frequency, a word type's rank r is inversely proportional to its frequency f. The order of magnitude of f and r, namely $\log f$ and $\log r$, then fall on a straight line.
Unsurprisingly, BPE's *sub*word types also vary by orders of magnitude in their frequency; however, Bostrom and Durrett (2020) observe that unlike for word types, the Zipfian distribution breaks down nearing the low-frequency tail of the vocabulary. Rather than the frequency tapering off gradually, it drops sharply, towards subword types appearing very rarely or never. This suggests that these subwords are purposeless, both because they are barely used and because they don't follow Zipf's law.

Salesky et al. (2018) make a similar observation in a different context: they notice that when a model is given access to a progressively larger BPE vocabulary during training (increasing $|V|$ by 10 000 every few epochs), the model seems to lose sight of a quarter of the previously available subword types with each expansion. Half of the forgotten subword types also no longer appear in the tokeniser's output, yet they still belong to V and hence require memory for storing embeddings. The authors suggest dropping those subwords from the vocabulary, for which we devise an algorithm in §3.3. Large vocabularies indeed lead to several practical complications, see Stahlberg (2020).

Most similar to the present paper is concurrent work by Cognetta et al. (2024) on *trimmed BPE*: they mark BPE types whose frequency falls under a given threshold as disabled, and then, when tokenising a word, all BPE merges are applied as usual, but the final segmentation is post-processed by reverting the merge of any token of a disabled type. The latter is repeated recursively until no such token is present. We compare our approach to this sort of vocabulary reduction in §3.5.

Semi-supervision All subword tokenisers mentioned in the introduction are trained completely unsupervised from raw text corpora, with the exception of later members of the Morfessor family like FlatCat (Grönroos et al., 2014) and EM+Prune (Grönroos et al., 2020) which both allow – but don't require – supplementing the unlabelled data with pre-segmented words to show desirable segmentations. The latter are provided by datasats such as those of the Morpho Challenge (Kurimo et al., 2010) and the CELEX project (Baayen et al., 1995), which is still available as part of WebCelex[2] and datasets like the Dutch e-Lex (Taalunie, 2014).

3 BPE-knockout

As hinted at in the introduction, we want to rid the BPE vocabulary of excessive types. Since types depend on each other, the impact of doing this is not immediately obvious, and hence it is best to construct an equivalent representation of a BPE tokeniser that makes this clear. Once this is set up, we can define which types are "excessive" based on morphological resources.

3.1 Formalising BPE

Let there be an alphabet Σ of non-empty strings. BPE vocabularisation initialises V by Σ, and constructs an ordered list M of merges $x + y \to xy$, with $x, y \in V$ before merging and $xy \in V$ after.

By itself, M has a flat structure, but it is possible to transform it into a directed graph (digraph). As demonstrated by Delobelle et al. (2022) and in our Figure 1, subword types in V *could* serve as vertices in such a graph, with each merge adding two parent-child relationships. However, this intuitive model will not suffice to replace M: to fully encode the ith merge $x+y \to z$, we must conserve i and the tuple (x, y). Let the collection of all such objects be $\mathcal{M} = \{(i, (x_i, y_i))\}_{i=1}^{|M|}$. Now we can see why such a simple graph doesn't capture $\mathcal{M}$:

1. All merge priorities i are missing.
2. The ordering of the set of incoming arcs $\{x, y\}$ is unknown.
3. In case multiple merges form the same subword type (which isn't the case in BPE by default, see below), it is unknown which of the >2 arcs belong to the same merge.

[2] `http://celex.mpi.nl/`

It is sufficient to use $\mathcal{M}$ itself as the set of vertices, as a merge rule $x + y \to xy$ is implicitly referred to by any amount of *later* rules (i.e. $xy + \ldots \to \ldots$ and $\ldots + xy \to \ldots$) and by 0 or more *earlier* merge rules (i.e. $\ldots \to x$ and $\ldots \to y$, which exist if $x \notin \Sigma$ and $y \notin \Sigma$ respectively).

However, there is redundancy in such a graph: if multiple merges in $\mathcal{M}$ (and hence vertices) produce xy, then all such vertices will have the same outgoing arcs. We solve this by reorganising the digraph to *alternate* between merge objects of $\mathcal{M}$ and types of V as vertices; every merge vertex is connected to the vertex of the type it produces, and every type vertex is connected to the vertices of merges it participates in. Figure 11 illustrates this.

For use in an algorithm, the entire digraph can be described from the point of view of each type vertex $\mathfrak{t} \in V$ via its incoming arcs $\mathcal{M}_i(\mathfrak{t}) \subset \mathcal{M}$ and its outgoing arcs $\mathcal{M}_o(\mathfrak{t}) \subset \mathcal{M}$.

3.2 Byte-tuple encoding

The BPE graph above has two invariants: the first is by design, and for the second, the proof by Bauwens (2023) is reproduced in §A.1.

1. Every merge has exactly two parents: $\forall (i, m) \in \mathcal{M} : |m| = 2$
2. Every type is formed by exactly one merge: $\forall \mathfrak{t} \in V : |\mathcal{M}_i(\mathfrak{t})| = 1$

The first invariant mirrors Gage (1994)'s algorithm and is useful to bound the complexity of each BPE iteration to search the most frequent pair of types. Yet, it isn't strictly necessary: a merge $(i, (a, b, c))$ to form a type abc is perfectly compatible with the existing BPE application step as long as it is created during vocabularisation.

Indeed, it is trivial to extend "byte-pair encoding", which deletes 1 space per applied merge, to *byte-tuple encoding (BTE)*, which deletes $\geq$1 spaces per merge. This generalisation is not novel: Kit and Wilks (1999) already provide support for it in the algorithm they describe.

Using BTE will become inevitable in the next section, but it has other benefits: as mentioned above, Bostrom and Durrett (2020) notice that BPE fills its vocabulary with types that become obsolete soon after being added, since they are only needed as stepping stones to form larger, more useful types. Whilst BPE is forced to do this (it can only concatenate one pair of types at a time to build bigger ones), BTE is free to concatenate many types in a single merge, no longer requiring stepping stones.

3.3 Knockout

Now that we have BTE, we could extend BPE's search for the most frequent *pair* to find the most frequent N*-tuple* instead. However, this leads to combinatoric explosion in the search space of possible merges, which has size $|V|^N$. Not only would it take excessive time to do the counting, but more importantly, given the same amount of words, a higher N inherently leads to more sparsity and thus the counts can't be trusted as being representative, reminiscent of the limitations of N-gram models (Jurafsky and Martin, 2009).

We propose, instead of *adding* merges, *removing* them from an existing, binary BPE tokeniser $(V, \mathcal{M}_i, \mathcal{M}_o)$. Doing this naively raises one issue: by deleting a type $\mathtt{t}$ from the graph, it becomes impossible to ever apply the merges in $\mathcal{M}_o(\mathtt{t})$, since they all require the presence of $\mathtt{t}$, and due to the second invariant, the merges in $\mathcal{M}_o(\mathtt{t})$ are the *only* merges that produce the types they produce. This means that all those types are rendered obsolete too, when we only wanted to stop one type $\mathtt{t}$. This cascades recursively through the graph. We need extra provisions to prevent this unintended cascade.

Algorithm 1 *Knockout*: removing a type from the BPE merge graph.

```
 1: function KNOCKOUT($V$, $M_i$, $M_o$, $\mathtt{t}$)
 2:     if $|M_i(\mathtt{t})| = 0$ then
 3:         return $(V, M_i, M_o)$
 4:     $\{m_{\text{old}}\} \leftarrow M_i(\mathtt{t})$
 5:     $p_1, \ldots, p_n \leftarrow \text{PARTS}(m_{\text{old}})$
 6:     for $i \in 1 \ldots n$ do
 7:         $M_o(p_i) \leftarrow M_o(p_i) \setminus \{m_{\text{old}}\}$
 8:     for all $(q, (t_1, \ldots, \mathtt{t}, \ldots, t_m)) \in M_o(\mathtt{t})$ do
 9:         $m_{\text{new}} \leftarrow (q, (t_1, \ldots, p_1, \ldots, p_n, \ldots, t_m))$
10:         $M_i(t_1 \ldots \mathtt{t} \ldots t_m) \leftarrow \{m_{\text{new}}\}$
11:         for $i \in 1 \ldots n$ do
12:             $M_o(p_i) \leftarrow M_o(p_i) \cup \{m_{\text{new}}\}$
13:     $V \leftarrow V \setminus \{\mathtt{t}\}$
14:     $M_i(\mathtt{t}) \leftarrow \{\}$
15:     $M_o(\mathtt{t}) \leftarrow \{\}$
16:     return $(V, M_i, M_o)$
```

Hence, we provide an alternative procedure for removing types from the BPE merge graph, *knockout*, with limited effect. It abridges the removed type by reassigning its outgoing arcs to its parent types. This is formalised in Algorithm 1. As a visual aid, Figure 1 shows (part of) the merge graph surrounding the type $\mathtt{t} =$ *ids* inside the Dutch BPE tokeniser whose output was used by Delobelle et al. (2020) to train RobBERT, a Dutch RoBERTa$_{\text{base}}$ model. The bottom panel of the figure shows what happens when it is knocked out: notice, for example, that the resulting BTE tokeniser now forms the type *Ġbruids* (from *bruid*, "bride", and the possessive interfix *s*)[3] using a triplet merge *Ġbru+id+s* rather than a pair merge *Ġbru+ids*.

3.4 Blame

We now need some kind of signal to point out which types are beneficially removed; intuitively (but see §A.2), removing subword types randomly will lead to suboptimal outcomes since many subwords are useful additions to the vocabulary.

Because the second invariant holds before and after knockout, *we can synonymise a type with the unique merge that forms it*. We can hence ask which merges, rather than which types, should disappear. We judge a merge as follows: after BPE has been applied to a sequence of characters, every position where they were concatenated was affected by exactly one merge. We can judge the validity of this concatenation with a morphological reference corpus wherein words are pre-split along morpheme boundaries. If the reference segmentation dictates that two characters should be kept separated because they belong to different morphemes, it is bad for a merge to delete the space in between.[4]

Indeed, given a BPE tokeniser, we can apply it to the words in such a corpus and check for false-negative split positions (i.e.: spaces between characters that have been deleted, but should have been kept). There is one merge to blame for each of those. We keep a tally for each merge $m \in \mathcal{M}$ of how many times $N(m)$ it was applied and how many times $B(m)$ it was blamed for an erroneous contraction. The *blame ratio*

$$R(m) = B(m)/N(m) \tag{1}$$

then gives the degree to which a merge is unwanted. As a simple heuristic (but see §A.3), merges with $R(m) \geq \frac{1}{2}$ cause more harm than good, so we select those for knockout. The order in which they are knocked out, does not matter.[5]

[3]Ġ is the "start-of-word" character inherited from GPT-2's byte-based BPE (Radford et al., 2019).

[4]Note that it is sufficient for a merge to glue *one character* of each morpheme together, even before each morpheme's characters have been grouped into a token. Indeed, once the boundary merges, such grouping becomes impossible.

[5]We encourage the reader to verify this on a small graph.

As pointed out by Creutz and Lagus (2005), morphology can be learnt from either word types (like Morfessor CatML) or word tokens (like Morfessor CatMAP), i.e. ignoring a word's frequency in running text or not. To avoid neglecting rare words, we stick with word types (but see §A.5).

3.5 Comparison to trimmed BPE

As mentioned in §2, an alternative way of pruning types from the BPE graph was studied by Cognetta et al. (2024). Now that we have introduced knockout and blame, a comparison is appropriate.

There exist cases where knockout and trimming produce the same segmentation: indeed, for leaves in the BPE graph ($\mathtt{t} \in V$ for which $|\mathcal{M}_o(\mathtt{t})| = 0$), it is equivalent to *not do* the merge (knockout) or to *do and undo* it (trimming). For general $\mathtt{t} \in V$, however, this equivalence doesn't hold: the reason is that trimming post-processes the *segmentation*, whilst knockout post-processes the *tokeniser*. That is, for any word, trimmed BPE applies the same merges as BPE and then preserves a subset by truncating that word's merge tree, whilst knockout allows merges outside of the original set and can hence changes the tree structure, as shown in Figure 7. Furthermore, if the type of a high-priority merge (i.e. low in the merge tree, like *dt* in Figure 7) is trimmed, this will only affect the segmentation if *all* types built on top of it are also trimmed; meanwhile, when such a merge (like *d+t*) is knocked out, it becomes completely forbidden and guarantees a different tree, except in the limited contexts of the tuple merges it is absorbed into.

The latter is related to the different measuring object of the two approaches: Cognetta et al. (2024) judge a type *directly* by measuring its *frequency*, whereas we use the unique merge that forms a type as a *proxy* for it and measure one of its properties, *morphological blame*, instead.

4 Experiments

To evaluate tokenisers resulting from blame-guided knockout (henceforth called *BPE-knockout*), we first check whether it indeed produces closer matches with morphological boundaries, after which we verify our hypothesis by pre-training and fine-tuning LMs downstream of it.

4.1 Better morphologicality of BPE tokens

A tokeniser splits a word into tokens. As suggested by Kurimo et al. (2006), the points at which it splits this word can be compared to the points at which a reference lexicon of morphological decompositions splits it (as was already explained for calculating $B(m)$ in Eq. 1), to then compute the binary classification metrics of precision (Pr), recall (Re) and F_1, micro-averaged across all entries in the lexicon. Each entry of n characters contributes $n-1$ binary tests, with a split point being a positive.

Reference We use the CELEX lexicon as gold standard, giving us decompositions for English, Dutch and German lemmata. Note that although *computing merge blame can be done using any flat segmentation as reference*, CELEX assigns to each lemma a PoS-tagged morpheme tree. For example, the Dutch compound *reanimatietechniek* ("resuscitation technique") receives the analysis

```
((((re)[V|.V],(animeer)[V])[V],(atie)[N|V.])[N],
    ((technisch)[A],(iek)[N|A.])[N])[N]
```

where the square brackets denote each morpheme's tag and the parentheses indicate the hierarchy of how the word is formed. (See Figure 8 for a visual aid.) Note that these morphemes are "hidden" and cannot be concatenated to get the original word, although they can be linked to the corresponding substrings with an efficient Viterbi algorithm. Doing so for the above example gives

```
((((re)[V|.V],(anim)[V])[V],(atie)[N|V.])[N],
    ((techn)[A],(iek)[N|A.])[N])[N]
```

which, removing all markup, gives the flat segmentation `re anim atie techn iek`. This contributes 4 positives and 13 negatives. Following Table 1, we can expect a CELEX lemma to contribute around 1 positive on average (see Figure 10 for the full morpheme distribution).

Whole-word boundaries We motivate the choice of CELEX, with its languages and hierarchical format, by introducing a new metric: whole-word boundary recall.

English, Dutch and German range from a low to high morpheme-to-word ratio, also called the *index of synthesis (IoS)* (Lieber, 2009), with English inflecting, deriving and compounding the least, and German the most.[6] Morphemes due to inflection and derivation are *bound* (and can't appear on their own), whereas those due to compounding are *free*. CELEX labels for bound morphemes include a "|" and a code to indicate if it is a prefix ($\mathtt{|.}y$), a

[6]For completeness: English and Dutch are predominantly analytic languages with equal amounts of derivation, although closed compounding is much more commonplace in Dutch. German is synthetic due to its fusional suffices.

suffix (| x .), or an interfix (| x . y). By concatenating prefices and suffices to their respective sibling nodes in the tree, any remaining splits are compound boundaries, as in

```
((reanimatie)[N],(techniek)[N])[N]
```

It is useful to evaluate recall on this reduced set of split positions, because recalling the boundaries between compound constituents is a necessary condition for whole-word recognition by the subword tokeniser. Merging such a boundary is guaranteed to confuse an LM, because it produces at least one token that would not be produced when tokenising the constituents separately.[7] For example, consider the closed English compound *horseshoe*. An English BPE tokeniser (like ours, see below) segments the separate constituents as *Ġhorse* and *Ġshoe*. Yet, the first merge applied in the compound is *e+s*, the whole-word boundary, resulting in two unrecognisable tokens *Ġhorses hoe* (with equally unrecognisable merge trees, making dropout ineffective).

Weights In some applications, erroneously splitting a highly frequent word is worse than erroneously splitting a very rare word. To better reflect this in the metrics, we additionally report frequency-weighted Pr, Re and F_1: for each word in the CELEX lexicon, we retrieve the absolute frequency f from its language's part in OSCAR[8] (Ortiz Suárez et al., 2019), a corpus of web texts grouped by language, defaulting to 1 if the word isn't present. We then multiply that word's positive and negative contributions by f, as if the lexicon consists of word tokens rather than word types.[9]

Baseline tokenisers For each language, we train one BPE tokeniser on its part in OSCAR (see §B.1 for preprocessing details). We follow the tokeniser training of RoBERTa (Liu et al., 2019) with a byte-based alphabet and $|V| = 40\,000$ like RobBERT. Knockout based on CELEX blame is applied to each baseline, and both versions are evaluated.

Dropout As a second point of comparison, we evaluate the three baseline tokenisers with a random dropout rate of[10] $p = 5\%$, averaging each of Pr, Re and F_1 across 10 repetitions.

[7]A similar hypothesis is proposed by Rogers et al. (2021) for why BERT struggles to interpret numbers.

[8]https://huggingface.co/datasets/oscar

[9]For example: *reanimatietechniek* appears 26 times in OSCAR, so it contributes 26 times as many positives ($26 \cdot 4 = 104$) and negatives ($26 \cdot 13 = 338$) to the weighted metrics.

[10]This is half the recommended $p = 10\%$ by Provilkov et al. (2020). Our more conservative rate is motivated in §A.6.

We hypothesise these variants to have better recall, as mentioned in §2. Since BPE-dropout disables merges *temporarily* and *data-agnostically*, it serves as a nice midpoint between BPE and BPE-knockout, the latter *permanently* disabling merges while *informed* by morphology.

Holdout In computing blame (Eq. 1) to apply knockout to the baseline tokenisers, the same dataset (CELEX) is used as for evaluation. When the goal is to measure how well a predictive model *generalises*, exposing it to the test set alongside the training set is evidently bad practice. In our case, however, the goal of BPE-knockout is to *memorise* as much morphological information from the data it is given, meaning it *is* useful to train and test on the same data;[11] furthermore, there certainly exist merges whose application is confined to one specific word, so by restricting that word to only one of the two aforementioned sets, BPE-knockout will either never be tested on it or never be given the chance to correct BPE's errors.

For transparency, we also report the same metrics as above except using a random 50-50 holdout.

Hypothesis We hypothesise that recall of split positions will increase after knockout, given the observation by Provilkov et al. (2020) that (temporarily) dropping merges tends to increase the amount of splits made per word (but see §A.4). Due to past works showing that BPE recalls barely 20% of English morpheme boundaries (Zhou, 2018; Bostrom and Durrett, 2020), we also foresee an increase in precision, despite the increased amount of splits and hence more opportunity for incorrect ones.

Language	Morphologies	Index of synthesis (IoS)
English	38 108	1.94
German	44 437	2.31
Dutch	96 540	2.48

Table 1 – Amount of entries with unique morphological decompositions in CELEX (StrucLab in the database) per language, and the morpheme-to-word ratio.

4.2 Effect on language modelling

As Dutch has the most available data in CELEX, we pre-train two RoBERTa models from scratch on the Dutch part of OSCAR, one using the Dutch BPE tokeniser before applying knockout and one after. We then evaluate these on the same downstream

[11]It is not obvious that memorisation is easy. As an example, we again point to the constraints put on BPE's training data by Huck et al. (2017) being violated at application time.

language modelling tasks as Delobelle et al. (2022) do for RobBERT-2022,[12] including Salazar et al. (2020)'s pseudo-perplexity (PPPL). We train both with a fixed budget of 30k training batches.

BPE-knockout tokenisers have the benefit of being inherently *backwards-compatible* with all models that were trained on their output prior to knockout. Hence, as a third model in our comparison, we take the model trained on BPE, apply knockout to its tokeniser, and then resume pre-training for an added 30% of the budget. As a control, we also pre-train the original model 30% more.

Setup We follow Izsak et al. (2021)'s recommendations for training LMs on a budget, see §B.2.

Hypothesis We hypothesise that model loss will be lower after pre-training with BPE-knockout, given that it reduces the amount of types that could be behind a mask. Furthermore, if the morphological evaluation shows improvement, our main hypothesis suggests that pre-training loss should drop faster since the tokeniser presents clearer input patterns to the model, and that using knowledge from morphology should help token-based downstream tasks since certain highly informative token boundaries are prevented from disappearing.

5 Results

5.1 Morphology

Binary metrics Table 2 shows the evaluation results for the four tokeniser variants compared in each of the three languages.[17] Table 11 shows the relative offsets w.r.t. each language's BPE baseline.

For morphemic boundaries in word types, BPE's baseline is always superseded by BPE-knockout's, with Pr rising by an absolute +10%, Re by +25% and F_1 by +15%. For word tokens, Re and F_1 each see a striking +50%-60%, whereas Pr improves more inconsistently (resp. by +45% in English, +5% in German, and +25% in Dutch).

For whole-word boundaries, Re improves about +10% in word types and again +50%-70% in word tokens. Unsurprisingly (see § A.7), better morpheme recall means worse precision identifying whole-word boundaries, with small drops for word types and drops up to -10% for word tokens; nevertheless, F_1 still rises due to the gains in Re.

Applying $p = 5\%$ dropout to BPE causes a slight rise in Re, yet drops in both Pr and F_1, with most remarkably an absolute drop of -31% morphemic precision in English word tokens. BPE-dropout is thus only useful for regularisation, *not* for improved adherence to morphology.

Applying 50% holdout to BPE-knockout reduces its gains by much less than 50% on word types, showing that memorisation of single types is not the primary driver of BPE-knockout's gains. Simultaneously, holdout reduces the gains in Re by an absolute -30% to -50% in word tokens, confirming that BPE undersegments very common lemmata and that a few knockouts can already cause major gains in a corpus. Despite these moderated gains, note that *even after 50% holdout, BPE-knockout boosts morphemic F_1 in word tokens by +30%* in English and Dutch, and +20% in German.

Finally, note that precision tends to increase with a language's CELEX size and IoS (Table 1). Size likely doesn't matter, since using only 50% of all Dutch data still allows better precision than using all German data (but see §A.8). Rather, tokenisers in a language with higher IoS are more likely to hit a boundary even when splitting a word at a random position, making them more precise by default.

Knockout distribution Lastly, we analyse which subwords are knocked out. If knockout mainly takes place at the end of the subword vocabulary, then BPE-knockout would be equivalent to choosing a smaller $|V|$, in line with Ding et al. (2019).

However, for all three languages, the distribution of merges with $R(m) \geq \frac{1}{2}$ is very uniform across the vocabulary: e.g., all adjacent quartiles lie apart by between 9000 and 11 000 types, typical for a discrete uniform distribution across $|V| = 40\,000$ values (see Figure 6 for detailed histograms).

5.2 Language modelling

Table 3 shows fine-tuning metrics for all RoBERTa LMs at various points in pre-training; see Figure 12 for the validation loss curves.

From scratch As hypothesised, loss decreases quicker using BPE-knockout, although the BPE model still obtains a lower test-set PPPL than the

[12]To eliminate all possible variability in the training setup, we chose to redo Delobelle et al. (2020)'s pre-training from scratch. We hence deviate slightly from RobBERT-2020.

[13]https://universaldependencies.org/treebanks/nl_lassysmall/index.html

[14]https://www.clips.uantwerpen.be/conll2002/ner/

[15]https://github.com/benjaminvdb/DBRD

[16]https://github.com/gijswijnholds/sick_nl

[17]Keep in mind that the four orange-red columns are overly pessimistic, since we aim to isolate *all* morphemes: this includes affices, which count towards false positives in the right half of the table. §A.7 explores the ceiling on precision.

				morphemic						whole-word					
				word types			word tokens			word types			word tokens		
			$\lvert V\rvert$	Pr	Re	F_1	Pr	Re	F_1	Pr	Re	F_1	Pr	Re	F_1
English	BPE	base	40 000	43.0	49.6	46.1	40.7	4.0	7.3	**13.3**	80.2	**22.8**	11.2	9.5	10.3
		dropout	40 000	41.3	50.3	45.3	9.7	5.9	7.1	12.6	80.5	21.8	2.4	11.9	3.9
	BPE-knockout	base	36 814	**53.2**	**75.1**	**62.3**	**84.5**	**59.2**	**69.6**	12.1	**89.0**	21.2	**12.7**	**76.5**	**21.8**
		holdout	37 952	50.0	67.8	57.5	69.2	28.4	40.3	11.7	83.0	20.6	8.4	32.8	13.3
German	BPE	base	40 000	45.0	54.0	49.1	54.3	8.4	14.5	**19.8**	67.5	30.6	**24.3**	14.7	18.3
		dropout	40 000	44.0	54.5	48.7	30.4	10.7	15.7	19.3	67.7	30.0	11.7	17.3	13.8
	BPE-knockout	base	35 919	**55.3**	**79.8**	**65.3**	**59.5**	**69.2**	**64.0**	19.8	**81.1**	**31.8**	16.6	**76.0**	**27.2**
		holdout	37 309	53.2	73.3	61.7	51.4	25.9	34.4	19.5	76.4	31.1	13.1	28.4	17.9
Dutch	BPE	base	40 000	52.6	55.3	53.9	54.3	11.0	18.4	**38.3**	69.7	49.5	**36.2**	29.5	32.5
		dropout	40 000	51.4	55.6	53.4	38.1	12.7	19.0	37.3	69.8	48.7	22.6	31.4	26.3
	BPE-knockout	base	35 525	**61.7**	**78.2**	**68.9**	**81.6**	**64.1**	**71.8**	37.6	**82.5**	**51.7**	25.8	**81.2**	**39.1**
		holdout	36 763	60.8	75.8	67.5	73.0	35.1	47.4	37.7	81.1	51.4	25.8	48.8	33.7

Table 2 – Evaluation (in %) of BPE and BPE-knockout on morphemic and whole-word split points in CELEX, with word token frequencies from OSCAR. *Note:* the four redmost columns (whole-word Pr and F_1) don't adjust for the fact that a tokeniser with morphemic $F_1 = 1.0$ can't have a whole-word Pr $= 1.0$, see §A.7.

					Sequence-level						Token-level					
		PPPL			SA			NLI			NER			PoS		
		30k	35k	39k	30k	35k	39k	30k	35k	39k	30k	35k	39k	30k	35k	39k
Dutch	BPE	**3.88**	**3.74**	**3.66**	**82.10**	81.56	81.43	**83.16**	82.55	**83.53**	80.04	83.18	83.98	93.50	85.91	93.78
	BPE-knockout	4.67			82.01			82.98			**86.21**			91.19		
	BPE → BPE-knockout	200.37	4.62	4.35	81.34	**82.42**	**81.74**	81.74	**82.59**	83.14	80.58	**86.77**	**87.51**	**94.82**	**96.03**	**96.01**

Table 3 – Evaluation of RoBERTa models trained on Dutch OSCAR with different tokenisers. From left to right: pseudo-perplexity, sentiment analysis accuracy on DBRD[13](van der Burgh and Verberne, 2019), natural-language inference accuracy on SickNL[14](Wijnholds and Moortgat, 2021), named-entity recognition F_1 on CoNLL-2002[15](Tjong Kim Sang, 2002), and part-of-speech tagging accuracy on LassyUD[16](van Noord et al., 2013).

BPE-knockout model and overall outperforms it in fine-tuning (except in NER, where there is quite a large gap in the opposite direction). Two possible explanations might be that (i) all LMs use the same hyperparameters as a control, which might end up benefitting one tokeniser over the other, and (ii) the BPE-knockout model has a slightly steeper slope at the pre-training cut-off of 30k batches, indicating that it might be further from convergence and hence have more training potential left.

Adapted from existing LM After applying knockout to the BPE model's tokeniser, perplexity spikes due to the novel segmentations shown to the model. Yet, remarkably, after being shown only 5k extra batches, this adapted model's PPPL already drops below that of the model that has always seen BPE-knockout segmentations. What's more: despite PPPL remaining higher than for BPE, the adapted model performs better on token-level tasks by several percentage points, achieving scores comparable to SOTA models (Delobelle et al., 2020) yet on a lower budget. Perhaps most surprisingly is that *even without any further pre-training, fine-tuning with a different tokeniser can already produce superior results*, in line with Hofmann et al. (2021)'s conclusion and reminiscent of zero-shot learning.

All of this shows that it is not only *feasible* to adapt existing LMs to a new tokeniser, but that it might even be *better* than using a novel tokeniser from the start, highlighting exciting new research opportunities.

A speculative hypothesis for why this adapted model works so well, is that perhaps BPE's linguistically inconsistent behaviour might actually have a regularising effect: hence, the bulk of pre-training is more challenging, producing a stronger model that can then maximally capitalise on the clearer segmentation of BPE-knockout during fine-tuning.

6 Releases

We release all code on GitHub at `https://github.com/bauwenst/BPE-knockout`. In addition, we release the final checkpoints for the 3 Dutch LMs on HuggingFace as part of a collection, `Bauwens/BPE-knockout`.

7 Conclusion

In this paper, we showed that a BPE tokeniser can be made significantly more adherent to morphological boundaries by disabling merges that abridge them excessively. We showed that this adherence to morphology benefits downstream LM tasks, and that it is even possible to adjust the tokeniser of LMs after their pre-training in a cost-effective way.

8 Limitations

8.1 Dataset

No inflection As shown in Table 1, the morphological decompositions in CELEX have an IoS, i.e. an average amount of morphemes per word, of roughly 2. This says nothing about the *grammatical role* of those morphemes, however: a language can have a high IoS whilst having few suffices for conjugation or declension, making it more analytic than synthetic. Dutch and English behave this way, so their tokenisers are mainly attempting to recognise word stems and derivational affices, not inflectional morphemes.

Even for German, which *does* have inflection, our experiments do *not* cover false-negative merges of fusional morphemes, because CELEX only decomposes *lemmata*, which are uninflected. Hence, no plural words, no conjugated verbs and (for German) no other cases than the nominative are present in CELEX, which means that not all merges that are morphologically harmful in a corpus of running text are counted towards blame. This likely doesn't affect the intrinsic experiments (§4.1) since the test set also consists of only lemmata, but it does limit the improvement of our LMs (§ 4.2) since those encounter more inflected forms than lemmata.

No indication of lexicalisation Sometimes, the meaning of a word cannot be deduced from its morphemes; the meaning is *lexicalised*, i.e. taken up as a new, standalone unit in a speaker's vocabulary. It would hence also be confusing to split such a word into the morphemes it originates from and then use the embeddings associated with them.

An example of such non-compositionality is the Dutch verb *muggenziften* ("to nitpick", but literally "to sift mosquitoes"), which is analysed by CELEX as if it were compositional:

```
((mug)[N],(e)[V|N.V],(zift)[V])[V]
```

This then informs BPE-knockout that the verb should be split. In fact, merges that form lexicalised words from their constituents (like *muggen* and *ziften*) are *most* at risk of being blamed, because those merges are likely applied nowhere else and so $B(m)$ increases whenever $N(m)$ does in Eq. 1.

8.2 Resulting vocabulary

No new merges Although our method removes pathological types, it does not add new, better types. In some cases, BPE spontaneously applies different merges to subsume the newly available tokens into bigger, better tokens: BPE tokenises the Dutch noun *twintiger* ("twenty-year-old", from *twintig*, "twenty") as *Ġtwint iger*, whilst BPE-knockout prohibits the merge *ig+er* and subsequently uses the *ig* token in a later merge to output *Ġtwintig er*.

In other examples, this might not happen: the Dutch adjective *subtropisch* ("subtropical") is tokenised by BPE as *Ġsubt rop isch*, whereas it is just decomposed further as *Ġsub t rop isch* by BPE-knockout. Learning a new merge *t+rop* would be useful here (with the embedding of a subword *trop* more suitable for storing relevant knowledge).

No iteration In Figure 1, we saw the example of *Ġbru+ids* turning into *Ġbru+id+s* after knockout. However, note that this didn't actually fix the erroneous concatenation of *id* and *s* in any of the words known to the tokeniser; the string *bruids-* ("bridal") will indeed be tokenised the same way as before, just in fewer steps (due to the triplet merge).

This problem could be solved by repeating the process of computing blame and applying knockout. In a sense, knockout makes a binary merge context-sensitive: we know that the merge *id+s* is problematic as it is blamed in the majority of its applications, but instead of never again merging *id* with *s*, the decision to do so is spread across several merges with several other tokens. In *Ġbru+id+s*, the merge should not happen, whilst in *Ġg+id+s* ("guide") it should. Iteratively applying knockout would reveal this: the triplet merge *Ġbru+id+s* would be knocked out next, whilst the triplet merge *Ġg+id+s* would stay.

This approach would leave the token sequence *Ġbru*, *id*, *s* unmerged. Optionally, after the first round of knockout, a new binary merge *Ġbru+id* could be inserted to turn the triplet *Ġbru+id+s* back into a binary merge, *Ġbruid+s*. If knockout is then run a second time, the merge *Ġbruid+s* would be flagged as bad, and we would end up with the most recognisable tokens *Ġbruid* ("bride") and *s* ("of").

We leave this iterative application of the BPE-knockout process as future work.

Chain effect BPE's stepping stones (single-use types that only exist to reach bigger types) bloat the vocabulary, but cannot be detected by our blame metric as they don't abridge morpheme boundaries.

An example is the formation of the Dutch noun *bibliotheken* ("libraries"), for which three of the intermediate tokens are exclusively dedicated to forming only that word (see Figure 9 for details).

Author Contributions

TB conceived the BPE-knockout algorithm, implemented it in Python, designed the morphological experiments, and wrote the bulk of the manuscript. PD designed the language modelling experiments, managed all pretraining and finetuning runs, and analysed the results.

Acknowledgements

We would like to thank prof. dr. Bettina Berendt and dr. ir. Thomas Winters for assessing the manuscript from which this paper was derived, and prof. dr. Miryam de Lhoneux for her detailed comments reviewing an earlier version of the paper.

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

PD was supported by the Research Foundation - Flanders (FWO) under EOS No. 30992574 (VeriLearn) and received funding from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme.

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.

All visualisations in this paper (graphs, histograms and tables) were made with the Fiject package (`https://github.com/bauwenst/fiject`).

References

Duygu Ataman, Matteo Negri, Marco Turchi, and Marcello Federico. 2017. Linguistically Motivated Vocabulary Reduction for Neural Machine Translation from Turkish to English. *The Prague Bulletin of Mathematical Linguistics*, 108(1):331–342.

Rolf Harald Baayen, Richard Piepenbrock, and Léon Gulikers. 1995. The CELEX lexical database.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. ICLR. ArXiv:1409.0473 [cs, stat] version: 7.

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

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.

Marco Cognetta, Tatsuya Hiraoka, Naoaki Okazaki, Rico Sennrich, and Yuval Pinter. 2024. An Analysis of BPE Vocabulary Trimming in Neural Machine Translation. ArXiv:2404.00397 [cs].

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

Mathias Creutz and Krista Lagus. 2005. Inducing the Morphological Lexicon of a Natural Language from Unannotated Text. In *Proceedings of the International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning*, volume 1, pages 51–59.

Pieter Delobelle, Thomas Winters, and Bettina Berendt. 2020. RobBERT: a Dutch RoBERTa-based Language Model. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3255–3265, Online. Association for Computational Linguistics.

Pieter Delobelle, Thomas Winters, and Bettina Berendt. 2022. RobBERT-2022: Updating a Dutch Language Model to Account for Evolving Language Use. ArXiv:2211.08192 [cs].

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.

Shuoyang Ding, Adithya Renduchintala, and Kevin Duh. 2019. A Call for Prudent Choice of Subword Merge Operations in Neural Machine Translation. In *Proceedings of Machine Translation Summit XVII: Research Track*, pages 204–213, Dublin, Ireland. European Association for Machine Translation.

Philip Gage. 1994. A New Algorithm for Data Compression. *C Users Journal*, 12(2):23–38.

Stig-Arne Grönroos, Sami Virpioja, and Mikko Kurimo. 2020. Morfessor EM+Prune: Improved Subword Segmentation with Expectation Maximization and Pruning. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 3944–3953, Marseille, France. European Language Resources Association.

Stig-Arne Grönroos, Sami Virpioja, Peter Smit, and Mikko Kurimo. 2014. Morfessor FlatCat: An

HMM-Based Method for Unsupervised and Semi-Supervised Learning of Morphology. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 1177–1185, Dublin, Ireland. Dublin City University and Association for Computational Linguistics.

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.

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.

Matthias Huck, Simon Riess, and Alexander Fraser. 2017. Target-side Word Segmentation Strategies for Neural Machine Translation. In *Proceedings of the Second Conference on Machine Translation*, pages 56–67, Copenhagen, Denmark. Association for Computational Linguistics.

Peter Izsak, Moshe Berchansky, and Omer Levy. 2021. How to Train BERT with an Academic Budget. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10644–10652, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Dan Jurafsky and James H. Martin. 2009. *Speech and Language Processing*. Pearson Prentice Hall, Upper Saddle River, N.J.

Chunyu Kit and Yorick Wilks. 1999. Unsupervised Learning of Word Boundary with Description Length Gain. In *EACL 1999: CoNLL-99 Computational Natural Language Learning*.

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

Mikko Kurimo, Mathias Creutz, Matti Varjokallio, Ebru Arisoy, and Murat Saraclar. 2006. Unsupervised segmentation of words into morphemes – Challenge 2005 An Introduction and Evaluation Report. In *Proceedings of the PASCAL Challenge Workshop on Unsupervised segmentation of words into morphemes*, pages 1–11.

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

Rochelle Lieber. 2009. *Introducing Morphology*. Cambridge University Press, New York.

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].

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, Massachusetts.

Pedro Javier Ortiz Suárez, Benoît Sagot, and Laurent Romary. 2019. Asynchronous Pipeline for Processing Huge Corpora on Medium to Low Resource Infrastructures. Leibniz-Institut für Deutsche Sprache.

Hyunji Hayley Park, Katherine J. Zhang, Coleman Haley, Kenneth Steimel, Han Liu, and Lane Schwartz. 2021. Morphology Matters: A Multilingual Language Modeling Analysis. *Transactions of the Association for Computational Linguistics*, 9:261–276. Place: Cambridge, MA Publisher: MIT Press.

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.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. *OpenAI Blog*, 1(8).

Anna Rogers, Olga Kovaleva, and Anna Rumshisky. 2021. A Primer in BERTology: What We Know About How BERT Works. *Transactions of the Association for Computational Linguistics*, 8:842–866.

Julian Salazar, Davis Liang, Toan Q. Nguyen, and Katrin Kirchhoff. 2020. Masked Language Model Scoring. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2699–2712, Online. Association for Computational Linguistics.

Elizabeth Salesky, Andrew Runge, Alex Coda, Jan Niehues, and Graham Neubig. 2018. Optimizing Segmentation Granularity for Neural Machine Translation. ArXiv:1810.08641 [cs].

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and Korean voice search. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152. ISSN: 2379-190X.

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.

Felix Stahlberg. 2020. Neural Machine Translation: A Review. *Journal of Artificial Intelligence Research*, 69:343–418.

Taalunie. 2014. e-Lex 1.1.1 voor Taal- en Spraaktechnologie. Technical report, Instituut voor de Nederlandse Taal, NTU/NWO.

Samson Tan, Shafiq Joty, Lav Varshney, and Min-Yen Kan. 2020. Mind Your Inflections! Improving NLP for Non-Standard Englishes with Base-Inflection Encoding. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5647–5663, Online. Association for Computational Linguistics.

Erik F. Tjong Kim Sang. 2002. Introduction to the CoNLL-2002 Shared Task: Language-Independent Named Entity Recognition. In *COLING-02: The 6th Conference on Natural Language Learning 2002 (CoNLL-2002)*.

Benjamin van der Burgh and Suzan Verberne. 2019. The merits of Universal Language Model Fine-tuning for Small Datasets – a case with Dutch book reviews. ArXiv:1910.00896 [cs].

Gertjan van Noord, Gosse Bouma, Frank Van Eynde, Daniël de Kok, Jelmer van der Linde, Ineke Schuurman, Erik Tjong Kim Sang, and Vincent Vandeghinste. 2013. Large Scale Syntactic Annotation of Written Dutch: Lassy. In Peter Spyns and Jan Odijk, editors, *Essential Speech and Language Technology for Dutch: Results by the STEVIN programme*, pages 147–164. Springer, Berlin, Heidelberg.

Matti Varjokallio, Mikko Kurimo, and Sami Virpioja. 2013. Learning a subword vocabulary based on unigram likelihood. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 7–12.

David Vilar and Marcello Federico. 2021. A Statistical Extension of Byte-Pair Encoding. In *Proceedings of the 18th International Conference on Spoken Language Translation (IWSLT 2021)*, pages 263–275, Bangkok, Thailand (online). Association for Computational Linguistics.

Changhan Wang, Kyunghyun Cho, and Jiatao Gu. 2019. Neural Machine Translation with Byte-Level Subwords. ArXiv:1909.03341 [cs].

Gijs Wijnholds and Michael Moortgat. 2021. SICK-NL: A Dataset for Dutch Natural Language Inference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 1474–1479, Online. Association for Computational Linguistics.

Yingting Wu and Hai Zhao. 2018. Finding Better Subword Segmentation for Neural Machine Translation. In *Chinese Computational Linguistics and Natural Language Processing Based on Naturally Annotated Big Data*, Lecture Notes in Computer Science, pages 53–64, Cham. Springer International Publishing.

Jingjing Xu, Hao Zhou, Chun Gan, Zaixiang Zheng, and Lei Li. 2021. Vocabulary Learning via Optimal Transport for Neural Machine Translation. 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 7361–7373, Online. Association for Computational Linguistics.

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

George Kingsley Zipf. 1949. *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology*. Addison-Wesley Press, Cambridge, Massachusetts.

A Substantiations

This appendix substantiates some of the appeals to intuition made in the main body of the paper.

A.1 Proof of the second BPE invariant

Consider BPE's vocabularisation phase: a corpus is split into words, every word is split into character tokens, and then merges are learnt by iteratively selecting the type pair $(x, y) \in V \times V$ that appears most commonly as a pair of adjacent tokens. Importantly, when the merge $x + y \to xy$ is learnt, it is not only added to $\mathcal{M}$, but it is also immediately applied throughout the corpus.

We now prove that the same type can never be learnt by two different merges within the same vocabularisation phase, or equivalently, that it is impossible to learn a second merge that results in it after a first has been learnt. For this, it suffices to prove the stronger statement that *after a merge has been applied throughout the corpus during vocabularisation, there cannot exist a sequence of >1 tokens in the corpus whose concatenation is that merge's result*; if this is true, then all possible merges that could form the resulting type – even tuple merges of more than two tokens – will forever have 0 frequency and thus there will always be a better merge to be added to $\mathcal{M}$ instead.

Proof. Call the type of interest xy (e.g. $x =$ *"re"* and $y =$ *"d"*) for which we know a merge $x + y \to xy =$ *"red"* occurs at some point.

Consider all occurrences of the string xy in the training corpus after splitting it into words but before applying any merges. For each word, three possible cases can occur: it doesn't contain xy, it is equal to xy, or it strictly contains xy and is hence a string of the form $wxyz$ with $|wz| > 0$ (in our example: *<u>red</u>ness*, *laye<u>red</u>*, *he<u>red</u>ity*, ...).

1. Words that don't contain xy will, at no point in vocabularisation, produce a token subsequence that concatenates to xy. Neglect these.
2. Words[18] of the form $wxyz$ bifurcate into two sets. Specifically, consider all merges that take place before $x + y \to xy$:
   - If one such merge occurs in a word $wxyz$ such that some characters of w and x, or some characters of y and z, are merged into an abridging token, then it is impossible for subsequent merges to ever end up with the isolated token xy, since this would require losing the characters amassed from w and/or z later on. These too can hence be neglected.
   - If this situation never occurs, then w and z can be safely ignored until xy is merged into a single type. Hence, the subsequence of tokens corresponding to the substring xy will be the same for all such words at all times, and indeed, always match the tokens into which the isolated word xy itself is currently split.

Because the latter subset behaves uniformly, executing the merge $x+y \to xy$ will either form xy in all of its words, or none of them. If the latter, then the pair (x, y) must have 0 occurrences, meaning the merge would have never been selected, which contradicts that it was. Hence the former is true; therefore, after applying the merge, the corpus consists only of words whose tokens can never merge to xy, and words containing the token xy. ■

A.2 Randomly knocking out merges is bad

We saw in Table 2 that Re, Pr and F_1 all increased due to knockout informed by blame (Eq. 1). It isn't obvious that this hinges on the selection metric; perhaps any amount of knockout after vocabularising a BPE tokeniser, or the mere mechanism of merging N-tuples, causes this positive change.

Hence, we do intrinsic evaluation (morphemic, unweighted, Dutch, no-holdout Re, Pr, F_1) for two other methods of selecting merges to knock out:

- Randomly choosing $p\%$ of merges (with results averaged over 10 repetitions);
- Choosing the last $p\%$ of merges;

Although the blame metric eliminates between 10% and 20% of merges, we let p range between 0 and 50 to get a better idea of the trend.

Figure 2 shows line plots of both selection methods. As expected, recall increases with more deleted merges, as with usage of blame. However, unlike knockout informed by blame, both selection methods have a consistent *decrease* in precision.

We found the same results when knocking out the same amounts of *leaves* from the latest merges (i.e. types that don't partake in any merge), probably since leaves are so prevalent (90% of the last 50% of merges, see Figure 3).

[18]Technically, "occurrences" is more appropriate, since a single word could contain the substring xy more than once, and each of those should be treated as an independent case of the $wxyz$ pattern. This is just a matter of phrasing, though.

A.3 $R(m) \geq 1/2$ is a good blame threshold

Although it makes intuitive sense that merges are undesirable if they cause a false negative more often than not when applied (exceeding a blame ratio threshold of $\frac{1}{2}$), it could also be true that such a merge is absolutely essential in that minority of cases where it *is* desirable. Conversely, some merges might be blamed a minority of the time, but *lead to* bad merges the majority of the time. To verify if $R(m) \geq \frac{1}{2}$ is a good middle ground, we evaluate the Dutch BPE-knockout tokeniser for blame thresholds between $R(m) \geq 1\%$ (very punishing) and $R(m) \geq 99\%$ (very tolerant). We also track the amount of knocked-out merges. The two panels of Figure 5 show the results.

The amount of knocked-out merges decreases linearly (and monotonously) with an increasing threshold. The Pr, Re and F_1 curves, however, all have a concave progression, attaining a maximum at different thresholds. Recall peaks at a very low threshold (because knocking out more merges leads to more segmentation, see §A.4), precision peaks at 50%, and F_1 somewhere in between.

Indeed, although recall is vital (it directly measures our goal of getting rid of false negatives), it is also easy to increase by just segmenting more granularly. Meanwhile, ensuring that we segment sparingly, i.e. have tokens that are big enough to have more than an alphabet's worth of meaningful embeddings, is much more delicate, which precision measures. The fact that a threshold of 50% maximises precision is hence a good justification for the heuristic in the main body of the paper.

A.4 Knockout produces more tokens per word on average

Given the same training corpus, a BPE tokeniser with a smaller vocabulary size always produces at least as many tokens as one with a larger vocabulary size, because applying the latter is equivalent to applying the former *plus* additional merges. Knockout reduces the vocabulary size, but not necessarily by pruning only the latest merges (as shown above), so it's not obvious how it affects the amount of tokens produced per word.

To see how knockout could actually *increase* the amount of merges that take place and hence *decrease* the amount of produced tokens, consider the Dutch word *masterthesis* ("master's thesis"), a compound of *master* and *thesis*. RobBERT's BPE tokeniser splits it into 4 tokens (after adding the start-of-word symbol "Ġ"):

```
Ġmast ert hes is
```

Yet, it produces only 3 tokens when segmenting the two distinct substrings separately:

`Ġmaster` and `the sis`

The token `Ġmaster` couldn't be formed in the compound because the merge *er* + *t* → *ert* has priority over the merge *Ġmast* + *er* → *Ġmaster*, hence "trapping" the *er* token. By removing *ert* from the vocabulary, the compound is segmented with the same 3 tokens

```
Ġmaster the sis
```

which is *fewer* tokens even though there are also *fewer* types in the vocabulary.

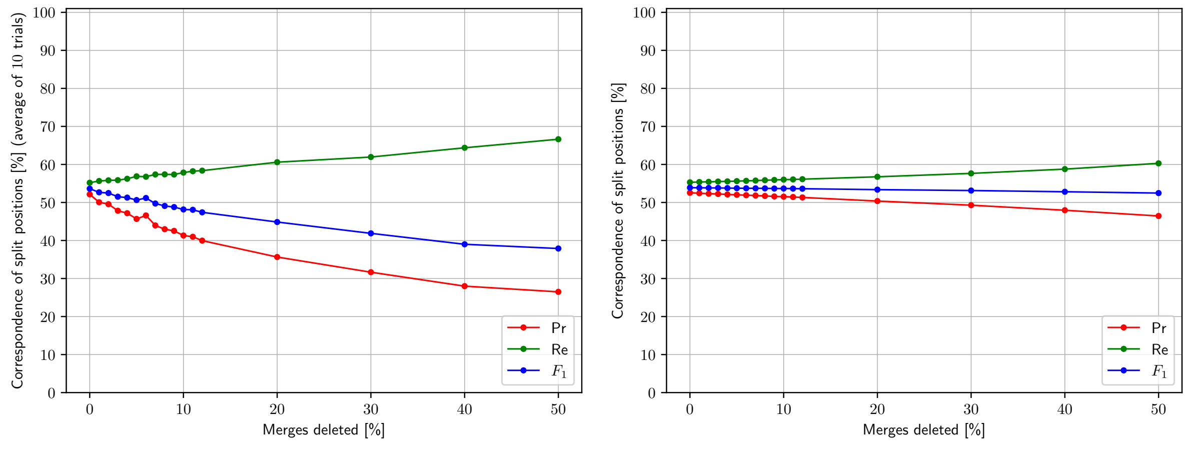
Since there must be a suitable alternative available, this effect takes place only sparsely in practice. For Dutch CELEX lemmata, fewer than 1% have a smaller amount of tokens after knockout, whereas more than 25% see an increase. Figure 4 shows the full distribution, confirming the hypothesis of overall gain in tokens.

A.5 Weighted blame isn't strictly better

To construct the BPE-knockout tokenisers, we computed blame ratio over CELEX lemmata without accounting for their frequency in running text, whilst to evaluate them, we did do so in half of the columns of Table 2.

We could also compute blame this way, which could change whether $R(m) \geq \frac{1}{2}$ or not. As an example, take the three Dutch nouns *bruidsjurk* ("bridal gown"), *beleidsmaker* ("policymaker") and *gids* ("guide"), all applying the merge $m =$ *id+s* at some point. If this is the whole lexicon, then m merges across a morpheme boundary in the first $B(m) = 2$ examples of the $N(m) = 3$, and is thus blamed in $R(m) = \frac{2}{3} \geq \frac{1}{2}$ of them. Assume now that the word *gids* appears 30 times in a corpus, whilst the others appear 10 times each. Multiplying each word's contributions by its frequency, we see $B(m) = 10 \cdot 1 + 10 \cdot 1 + 30 \cdot 0 = 20$ and $N(m) = 10 + 10 + 30 = 50$, giving $R(m) = \frac{20}{50} < \frac{1}{2}$.

Table 5 shows that unsurprisingly, accounting for weighted blame optimises the weighted metrics at the cost of the unweighted ones. It is hence not strictly better, but could be used in applications where it is more desirable to split the majority of word tokens correctly rather than the majority of word types. The latter is more suitable when rare words are important, for example.



(a) Random-merge selection

(b) Latest-merge selection

Figure 2 – Evaluation over Dutch CELEX after removing $p\%$ of all merges according to a selection criterion.

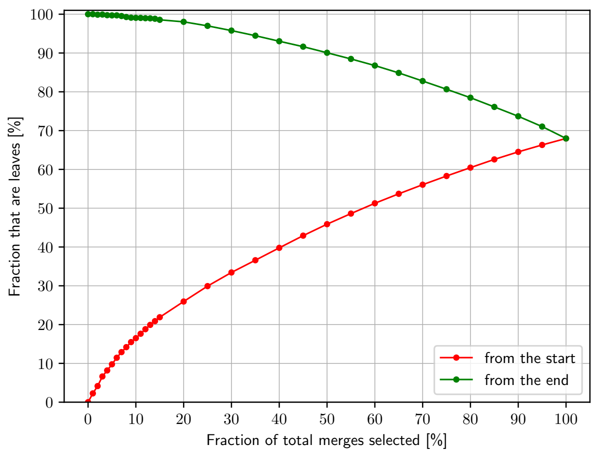
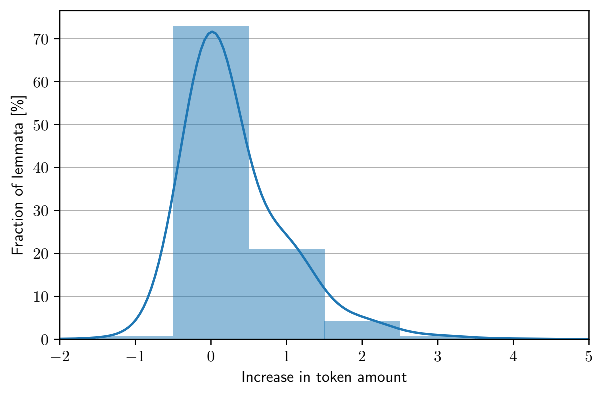
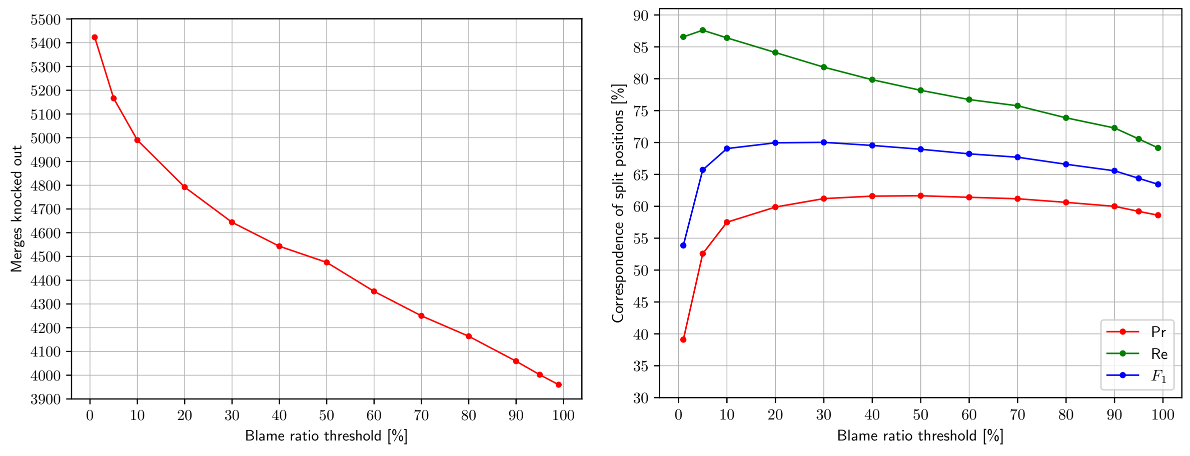


Figure 3 – Fraction of selected merges that make leaves (i.e. the resulting type isn't merged further) as function of selection size, in the Dutch tokeniser before knockout.



−2	−1	0	1	2	3	4	5
73	678	70352	20311	4151	784	148	35
0.08%	0.70%	72.88%	21.04%	4.30%	0.81%	0.15%	0.04%

Figure 4 – Increase in produced amount of tokens after knockout for Dutch CELEX lemmata.



(a) Merges deleted by knockout

(b) Evaluation over Dutch CELEX after knockout

Figure 5 – Effect of varying the blame ratio threshold for knockout in the Dutch tokeniser.

		p	$\|V\|$	morphemic						whole-word					
				word types			word tokens			word types			word tokens		
				Pr	Re	F_1	Pr	Re	F_1	Pr	Re	F_1	Pr	Re	F_1
Dutch	BPE-dropout	0%	40 000	**52.6**	55.3	**53.9**	**54.3**	11.0	18.4	**38.3**	69.7	**49.5**	**36.2**	29.5	**32.5**
		1%	40 000	52.4	55.4	53.8	50.4	11.3	18.5	38.1	69.7	49.3	32.5	29.8	31.1
		2.5%	40 000	52.0	55.5	53.7	40.5	11.9	18.2	37.9	69.8	49.1	28.6	30.5	29.5
		5%	40 000	51.4	55.6	53.4	34.4	12.6	18.3	37.3	69.8	48.7	21.9	31.5	25.8
		10%	40 000	50.0	**56.0**	52.8	26.4	**14.7**	**18.8**	36.2	**70.0**	47.7	15.9	**33.9**	21.6

Table 4 – Evaluation of BPE-dropout for increasing dropout rates p, averaged over 10 runs.

			$\|V\|$	morphemic						whole-word					
				word types			word tokens			word types			word tokens		
				Pr	Re	F_1	Pr	Re	F_1	Pr	Re	F_1	Pr	Re	F_1
Dutch	BPE	base	40 000	52.6	55.3	53.9	54.3	11.0	18.4	**38.3**	69.7	49.5	**36.2**	29.5	32.5
	BPE-knockout	base	35 525	**61.7**	**78.2**	**68.9**	81.6	**64.1**	71.8	37.6	**82.5**	**51.7**	25.8	**81.2**	39.1
		weighted	35 585	61.1	76.4	67.9	**85.9**	64.1	**73.4**	37.3	80.6	51.0	27.0	80.5	**40.4**

Table 5 – Comparison of knockout with unweighted vs. weighted blame.

			$\|V\|$	morphemic						whole-word					
				word types			word tokens			word types			word tokens		
				Pr	Re	F_1	Pr	Re	F_1	Pr	Re	F_1	Pr	Re	F_1
Dutch	BPE	base	40 000	52.6	55.3	53.9	54.3	11.0	18.4	**38.3**	69.7	49.5	**36.2**	29.5	32.5
	BPE-knockout	base	35 525	**61.7**	**78.2**	**68.9**	**81.6**	**64.1**	**71.8**	37.6	**82.5**	**51.7**	25.8	**81.2**	**39.1**
		90-10	35 734	61.0	75.7	67.6	73.9	35.7	48.2	37.7	80.7	51.4	25.0	45.0	32.1
		80-20	35 985	60.9	76.0	67.6	73.5	34.8	47.3	37.5	80.9	51.2	23.5	45.2	30.9
		70-30	36 259	60.9	75.9	67.6	74.3	35.4	47.9	37.6	81.0	51.4	25.0	47.2	32.7
		60-40	36 552	60.8	75.7	67.4	74.1	35.5	48.0	37.6	80.8	51.3	24.1	45.8	31.6
		50-50	36 827	60.7	75.4	67.2	73.9	34.9	47.4	37.7	81.0	51.5	24.1	45.1	31.4

Table 6 – Comparison of different holdout splits.

		$\|V\|$	morphemic						whole-word					
			word types			word tokens			word types			word tokens		
			Pr	Re	F_1	Pr	Re	F_1	Pr	Re	F_1	Pr	Re	F_1
English	ideal	15 374	100	100	100	100	100	100	19.1	100	32.1	11.6	100	20.9
German	ideal	16 923	100	100	100	100	100	100	35.2	100	52.1	25.4	100	40.4
Dutch	ideal	18 216	100	100	100	100	100	100	57.8	100	73.3	25.0	100	39.9

Table 7 – Best possible evaluation values for morphologically perfect tokenisers.

A.6 $p = 5\%$ dropout is a fairer baseline than the usual $p = 10\%$ dropout

BPE-dropout disables the application of any instance of any merge with probability p. Provilkov et al. (2020) find that $p = 10\%$ gives the best BLEU in machine translation, and hence recommend using it (including in their own follow-up experiments). For comparison against BPE-knockout, however, we now motivate that it is fairer to compare against the more conservative $p = 5\%$.

Retrospective effective dropout rate Like BPE-dropout, BPE-knockout also disables the application of certain merge instances, namely all merges that would form a knocked-out subword type. Because we already track the amount of times $N(m)$ that a merge m was applied (in order to compute blame in Eq. 1), we can count in retrospect what fraction of merge instances that did happen will become impossible after knockout:

$$p_{\text{eff,ret}} = \frac{\sum\limits_{m\in\mathcal{M}^\dagger} N(m)}{\sum\limits_{m\in\mathcal{M}} N(m)} \tag{2}$$

where $\mathcal{M}^\dagger = \{m \in \mathcal{M} \mid R(m) \geq \frac{1}{2}\}$ is the set of all knocked-out merges. This could be seen as an estimate for the effective dropout rate of applying BPE-knockout; it is retrospective because it doesn't actually reflect the state of the tokeniser af-

		$p_{\text{eff,ret}}$	$\lvert\mathcal{M}^\dagger\rvert/\lvert\mathcal{M}\rvert$
English	BPE-knockout	4.681%	8.017%
German	BPE-knockout	5.174%	10.270%
Dutch	BPE-knockout	5.025%	11.261%

Table 8 – Retrospective effective dropout rate $p_{\text{eff,ret}}$ and vocabulary pruning rate $\lvert\mathcal{M}^\dagger\rvert/\lvert\mathcal{M}\rvert$ after knockout.

ter knockout, since $N(m)$ is accumulated in every word's original merge tree, prior to those merge trees changing due to knockout (see Figure 7).

As shown in Table 8, this dropout rate is ~5% for each of the tested languages (even though ~10% of merges are pruned from the BPE graph).

Precision hit with p Naturally, when the dropout rate increases, fewer merges are performed, more tokens are produced, and hence more split points are predicted. We should hence expect a monotonous rise in recall and a montonous drop in precision for BPE-dropout with increasing p, but it is not obvious how fast both will take place (given that p's range is from 0% to 100%).

Table 4 shows how a Dutch BPE-dropout tokeniser's morphological evaluation evolves when we vary $p \in [0\%, 10\%]$, a tenth of its full range, where the upper bound of 10% is the usual recommended value. Notice that when measured in a corpus, precision is extremely sensitive to small changes in p, in such a way that 4% is lost when adding a mere 1% of dropout, and *less than half of BPE's precision* is conserved for the recommended $p = 10\%$. Hence, when compared to having no dropout at all, using this value for p gives an overly pessimistic view of BPE-dropout's adherence to morphology.

A.7 Whole-word precision has a low ceiling for morphologically perfect tokenisers

As explained in §3.4, a merge is blamed when it abridges *any* morpheme boundary in CELEX, not just whole-word boundaries. The goal of knocking out blameworthy merges is hence to approach a tokeniser with lower and lower blame, with the optimum being a tokeniser for which $\forall m \in \mathcal{M} : B(m) = 0$. In other words: a tokeniser whose output is indistinguishable from the morphemic segmentation in CELEX.

This ideal tokeniser will necessarily have a perfect score in the left half of Table 2, but will also necessarily have an *imperfect* score in the right half. Whole-word boundaries are a subset of all morpheme boundaries (which follows from their construction in §4.1), so a tokeniser that puts a split position at every morpheme boundary will split on whole-word boundaries *and others*, counting towards false positives and therefore taking away from perfect precision.

Table 7 shows the result of evaluating CELEX's morpheme boundaries as if they are produced by a tokeniser. As described above, this gives a perfect score in the left half of the table, and also a perfect recall for whole-word boundaries because the segmentation positions are a superset of those. The remaining four columns are again orange-red as in Table 2; as it turns out, the BPE-knockout tokenisers are only 10%-20% removed from this ceiling at most, not over 50%.

A.8 Holdout performs worse due to which data it sees, not how much

Table 2 showed how computing blame on 50% of the data and evaluating on the other 50% produced markedly worse recall than when using the full dataset for both. The cause could be either underfitting due to lack of data (only 50%), or not being given the opportunity to memorise the test set. We argued the latter in §4.1: confining words to only one of the two sets prohibits us from measuring BPE-knockout's improvement on leaf merges.

Table 6 shows that intrinsic performance barely changes when varying the holdout split from 50-50 to 90-10 to increase the amount of data seen whilst still hiding the test set. This confirms that the bottleneck is not the former, but the latter.

B Experimental Setup

B.1 Corpus preprocessing

Sennrich et al. (2016) train their BPE tokeniser over a dictionary of words and their counts. To obtain this from each of the three languages' part of OSCAR, we proceed as follows:

1. Limit to first 30 million entries (∼paragraphs).
2. Add whitespace around all *non-hyphen* punctuation. For example, the Dutch sentence

   > *BPE-knockout snoeit vocabularia; zelfs tokenisers van modellen die reeds geconvergeerd zijn, kunnen zo verbeteren zonder verlies van pre-training (wat energie-efficiëntie bevordert).*

   becomes

   > *BPE-knockout snoeit vocabularia ; zelfs tokenisers van modellen die reeds geconvergeerd zijn , kunnen zo verbeteren zonder verlies van pre-training (wat energie-efficiëntie bevordert).*
3. Split on (and drop all) whitespace.
4. Count the amount of times every unique string in the result appears.
5. Drop all strings with frequency <10. (This reduces the amount of strings to operate on.)
6. Drop all strings containing >60 characters (which are likely garbage strings from the web) or any character from a foreign alphabet (which were misclassified by OSCAR).
7. Byte-encode each string with UTF-8. (In the above example, the only character that becomes longer than one byte is "ë".)
8. Prefix each string by the start-of-word byte (printed as Ġ for historical reasons).
9. Split each string on hyphens.

The BPE training algorithm[19] is then run on the remaining byte-based word count dictionary.

B.2 RoBERTa hyperparameters

To speed up pre-training, we limit the sequence length to 128 tokens during the entire process, as proposed by Izsak et al. (2021); this is also how 90% of BERT's pre-training proceeded (Devlin et al., 2019). As we are only comparing models we trained ourselves, any debt incurred due to this limited context length is equal across all models. Table 9 gives other pre-training hyperparameters.

[19] `https://github.com/huggingface/tokenizers`

Hyperparameter	**Value**
Parameters[20]	117M
Sequence length	128
Batch size	4096
Precision	half (`fp16`)
Learning rate	10^{-4} (fixed schedule)
AdamW β	0.9, 0.999
AdamW ϵ	10^{-8}
Weight decay	0.1

Table 9 – RoBERTa hyperparameters for pre-training

We use a similar setup for fine-tuning, where we do a hyperparameter search with 10 random initialisations for the batch size, learning rate and weight decay. These additional hyperparameters are found in Table 10. We *select* the best model of these 10 by evaluating on a held-out validation set, and we *report* its performance on a held-out test set.

Hyperparameter	**Value**
Batch size	$\{16, 32, 64, 128, 256, 512\}$
Learning rate	$[10^{-6}, 10^{-3}]$
Weight decay	$[0.01, 0.1]$
Max epochs	3 (10 for SA)

Table 10 – Changed and additional RoBERTa hyperparameter ranges for fine-tuning.

B.3 Hardware

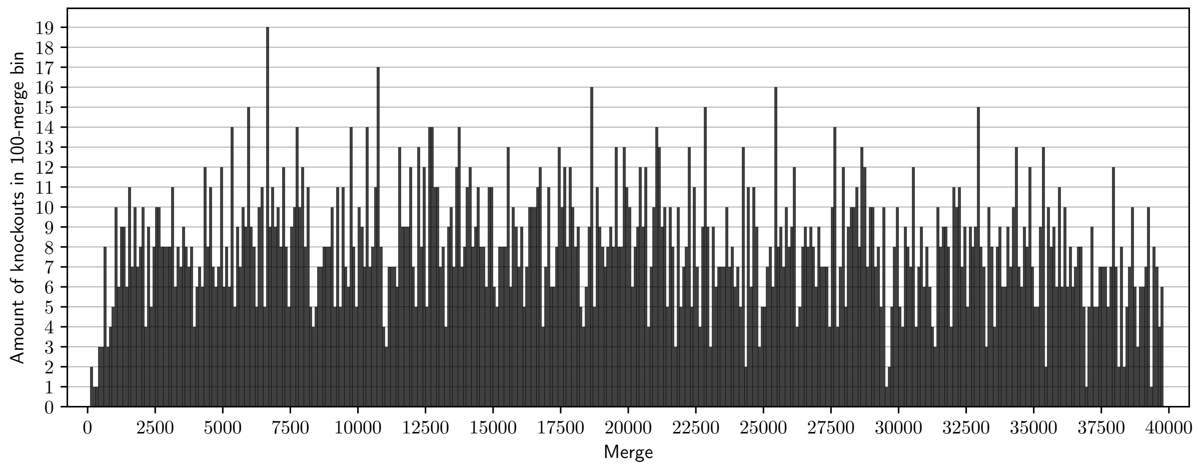
The intrinsic evaluation (generating word counts for OSCAR, training BPE tokenisers and testing them morphologically) was performed on a consumer machine with an Intel i7-4790 3.6 GHz CPU and 8 GiB of RAM.

The extrinsic evaluation (training LMs) was performed on (i) a server with 2 Intel Xeon Gold 6230R 2.10GHz CPUs using at most 40 threads and 2 NVIDIA GeForce RTX 3080 Ti GPUs for pre-training of all models and (ii) a server with 2 AMD EPYC 7502 32-Core CPUs and 4 NVIDIA RTX A5000 GPUs.

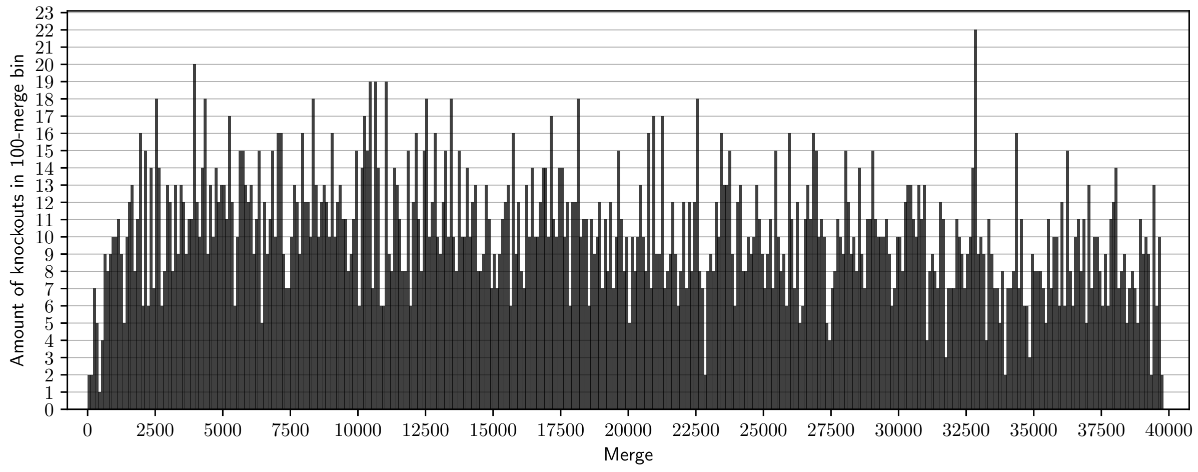
Collectively, all experiments (preliminary and final) took 4886 GPU hours to complete, of which 557 hours were needed to generate the results printed in the body of this paper (and hence also to reproduce them).

[20] BERT$_{\text{base}}$ (Devlin et al., 2019) has $|V| = 30$k embeddings and 110M parameters, RobBERT-2020 (Delobelle et al., 2020) has 40k and 117M parameters, and RoBERTa$_{\text{base}}$ (Liu et al., 2019) has 50k and 125M.

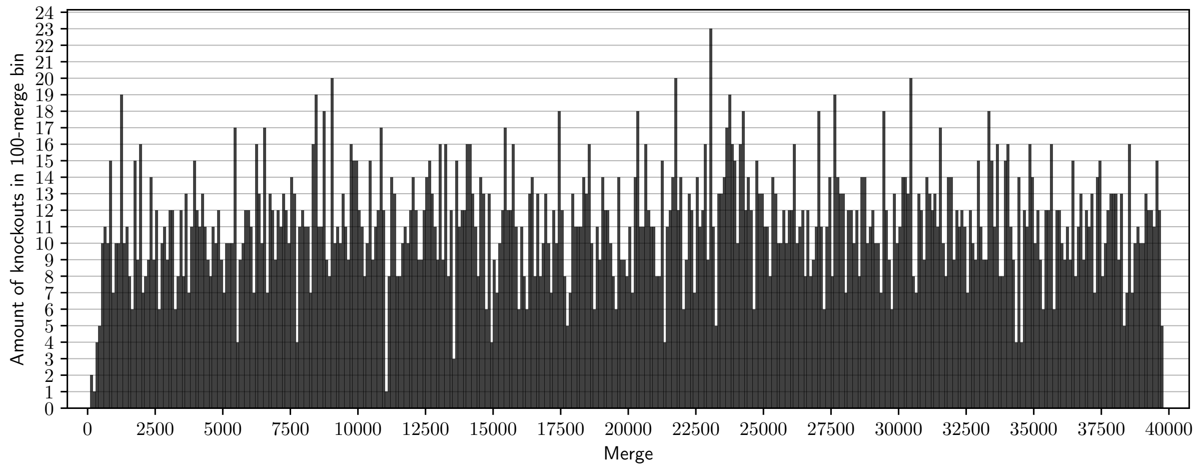
C Supplementary Figures



(a) English

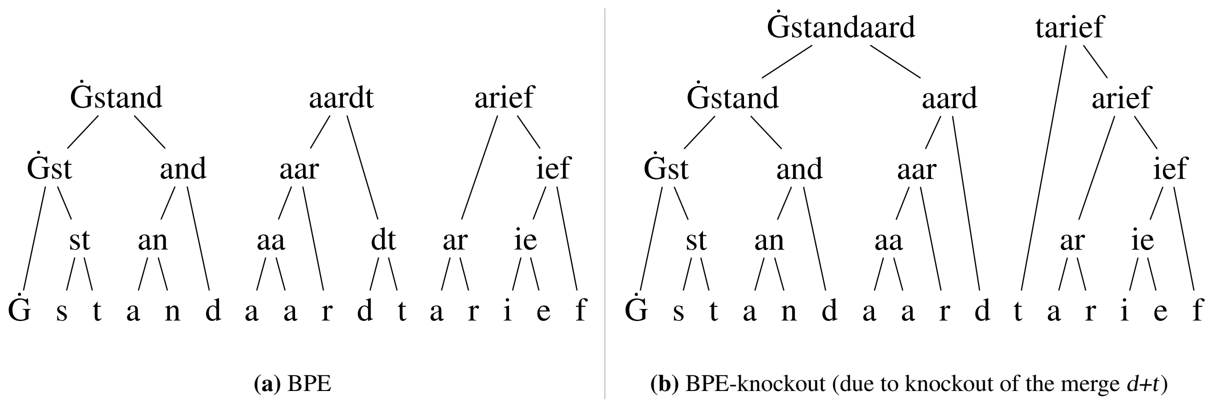


(b) German



(c) Dutch

Figure 6 – Distribution of the position of knocked-out types in the vocabulary of the Dutch BPE tokeniser.



(a) BPE **(b)** BPE-knockout (due to knockout of the merge *d+t*)

Figure 7 – BPE tokenisation of the Dutch compound *standaardtarief* (from *standaard*, "standard", and *tarief*, "fee").

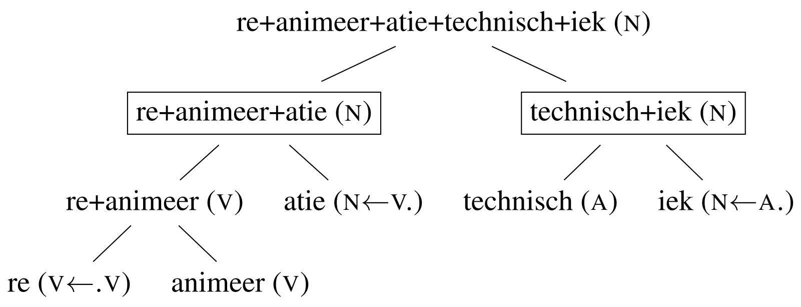


Figure 8 – Schematic representation of the CELEX StrucLab tag for the Dutch lemma *reanimatietechniek*:
`((((re)[V|.V],(animeer)[V])[V],(atie)[N|V.])[N],((technisch)[A],(iek)[N|A.])[N])[N]`

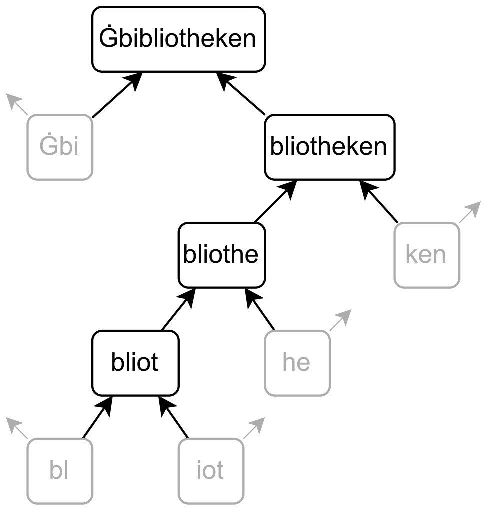


Figure 9 – Limitation: multiple single-use types needed to form the Dutch type *Ġbibliotheken*.

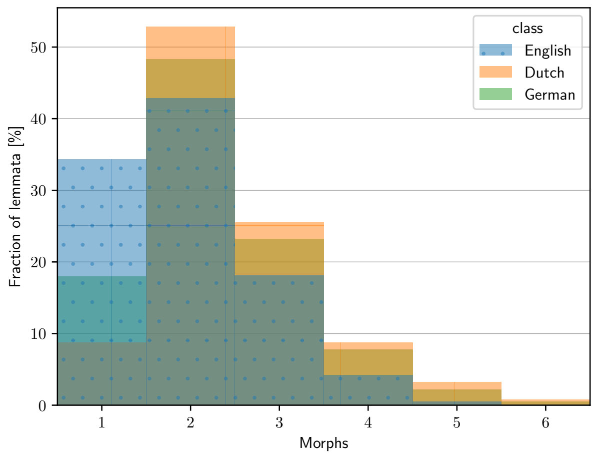


Figure 10 – Fraction of words with a certain amount of morphemes, grouped by language.

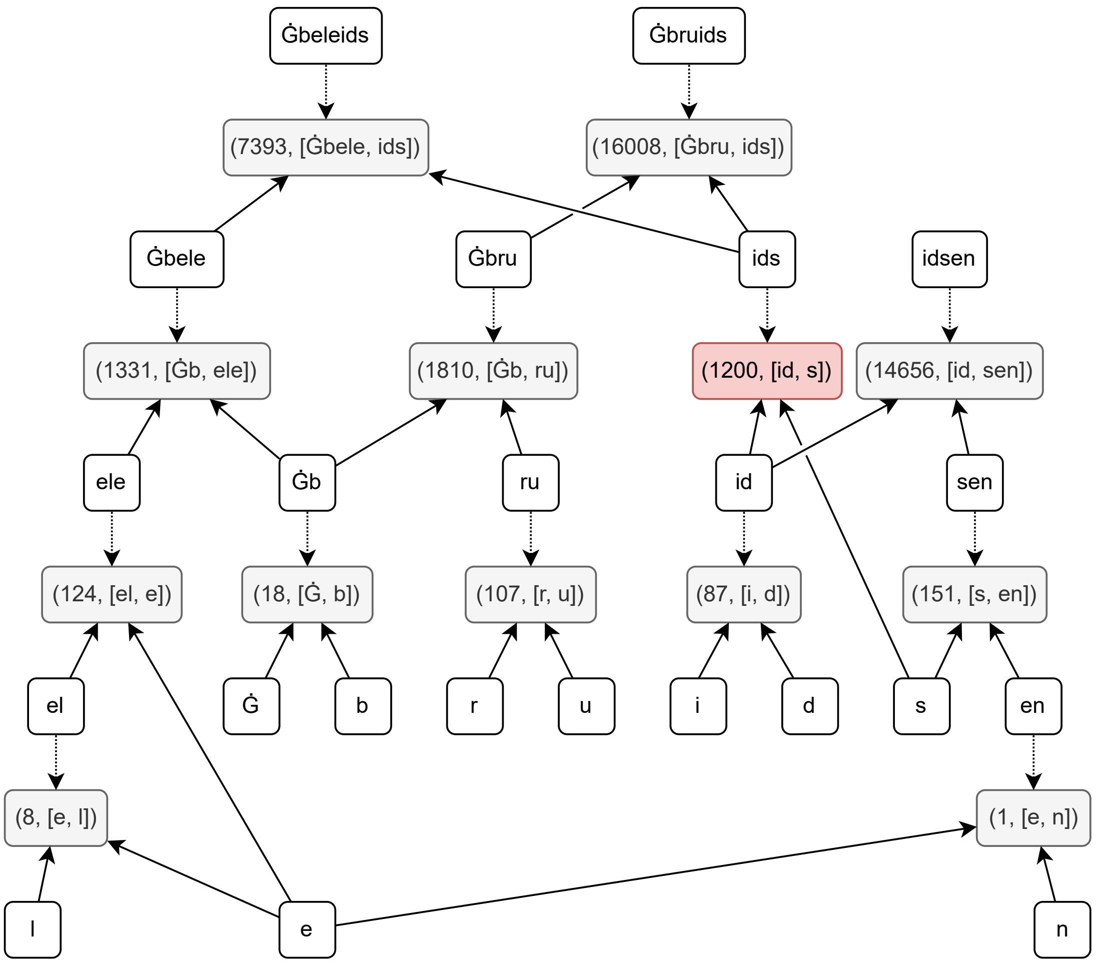


Figure 11 – Formal graph representation of the top panel in Figure 1, fully expanded down to the alphabet. Grey boxes are merges in $\mathcal{M}$, white boxes are types in V. The set of solid arrows emanating from a type $\mathfrak{t}$ together represent $\mathcal{M}_o(\mathfrak{t})$, whilst the dotted arrows represent $\mathcal{M}_i(\mathfrak{t})$. Note how there are always exactly two solid arrows arriving in a merge vertex (the first invariant) and exactly one dotted arrow (the second invariant).

				morphemic						whole-word					
				word types			word tokens			word types			word tokens		
			$\lvert V\rvert$	Pr	Re	F_1	Pr	Re	F_1	Pr	Re	F_1	Pr	Re	F_1
English	BPE	base	40 000	43.01	49.64	46.09	40.70	4.04	7.35	**13.28**	80.16	**22.79**	11.15	9.51	10.26
		dropout	40 000	-1.7	+0.6	-0.8	-31.0	+1.8	-0.2	-0.6	+0.3	-0.9	-8.8	+2.4	-6.3
	BPE-knockout	base	36 814	**+10.2**	**+25.4**	**+16.2**	**+43.8**	**+55.1**	**+62.3**	-1.2	**+8.8**	-1.5	**+1.6**	**+67.0**	**+11.5**
		holdout	37 952	+7.0	+18.2	+11.5	+28.5	+24.3	+32.9	-1.5	+2.8	-2.2	-2.8	+23.3	+3.1
German	BPE	base	40 000	45.02	54.00	49.10	54.33	8.36	14.49	**19.79**	67.47	30.60	**24.26**	14.72	18.32
		dropout	40 000	-1.0	+0.5	-0.4	-23.9	+2.3	+1.2	-0.5	+0.2	-0.6	-12.6	+2.6	-4.6
	BPE-knockout	base	35 919	**+10.3**	**+25.8**	**+16.2**	**+5.1**	**+60.8**	**+49.5**	-0.0	**+13.6**	**+1.2**	-7.7	**+61.3**	**+8.9**
		holdout	37 309	+8.2	+19.3	+12.6	-3.0	+17.5	+19.9	-0.3	+8.9	+0.5	-11.2	+13.7	-0.4
Dutch	BPE	base	40 000	52.59	55.32	53.92	54.26	11.04	18.35	**38.33**	69.70	49.46	**36.18**	29.51	32.51
		dropout	40 000	-1.2	+0.3	-0.5	-16.2	+1.6	+0.6	-1.0	+0.1	-0.8	-13.6	+1.9	-6.3
	BPE-knockout	base	35 525	**+9.1**	**+22.9**	**+15.0**	**+27.3**	**+53.1**	**+53.5**	-0.7	**+12.8**	**+2.2**	-10.4	**+51.7**	**+6.6**
		holdout	36 763	+8.2	+20.5	+13.6	+18.8	+24.1	+29.1	-0.7	+11.4	+2.0	-10.4	+19.3	+1.2

(a) Absolute differences [%]

				morphemic						whole-word					
				word types			word tokens			word types			word tokens		
			$\lvert V\rvert$	Pr	Re	F_1	Pr	Re	F_1	Pr	Re	F_1	Pr	Re	F_1
English	BPE	base	40 000	43.01	49.64	46.09	40.70	4.04	7.35	**13.28**	80.16	**22.79**	11.15	9.51	10.26
		dropout	40 000	×0.96	×1.01	×0.98	×0.24	×1.45	×0.97	×0.95	×1.00	×0.96	×0.21	×1.25	×0.38
	BPE-knockout	base	36 814	×**1.24**	×**1.51**	×**1.35**	×**2.08**	×**14.66**	×**9.47**	×0.91	×**1.11**	×0.93	×**1.14**	×**8.05**	×**2.12**
		holdout	37 952	×1.16	×1.37	×1.25	×1.70	×7.03	×5.48	×0.88	×1.04	×0.90	×0.75	×3.45	×1.30
German	BPE	base	40 000	45.02	54.00	49.10	54.33	8.36	14.49	**19.79**	67.47	30.60	**24.26**	14.72	18.32
		dropout	40 000	×0.98	×1.01	×0.99	×0.56	×1.28	×1.09	×0.97	×1.00	×0.98	×0.48	×1.18	×0.75
	BPE-knockout	base	35 919	×**1.23**	×**1.48**	×**1.33**	×**1.09**	×**8.28**	×**4.42**	×1.00	×**1.20**	×**1.04**	×0.68	×**5.17**	×**1.48**
		holdout	37 309	×1.18	×1.36	×1.26	×0.95	×3.09	×2.38	×0.99	×1.13	×1.02	×0.54	×1.93	×0.98
Dutch	BPE	base	40 000	52.59	55.32	53.92	54.26	11.04	18.35	**38.33**	69.70	49.46	**36.18**	29.51	32.51
		dropout	40 000	×0.98	×1.01	×0.99	×0.70	×1.15	×1.03	×0.97	×1.00	×0.98	×0.62	×1.07	×0.81
	BPE-knockout	base	35 525	×**1.17**	×**1.41**	×**1.28**	×**1.50**	×**5.81**	×**3.91**	×0.98	×**1.18**	×**1.04**	×0.71	×**2.75**	×**1.20**
		holdout	36 763	×1.16	×1.37	×1.25	×1.35	×3.18	×2.58	×0.98	×1.16	×1.04	×0.71	×1.65	×1.04

(b) Ratios

Table 11 – Intrinsic evaluations of Table 2, but relative to the BPE baselines (grey).

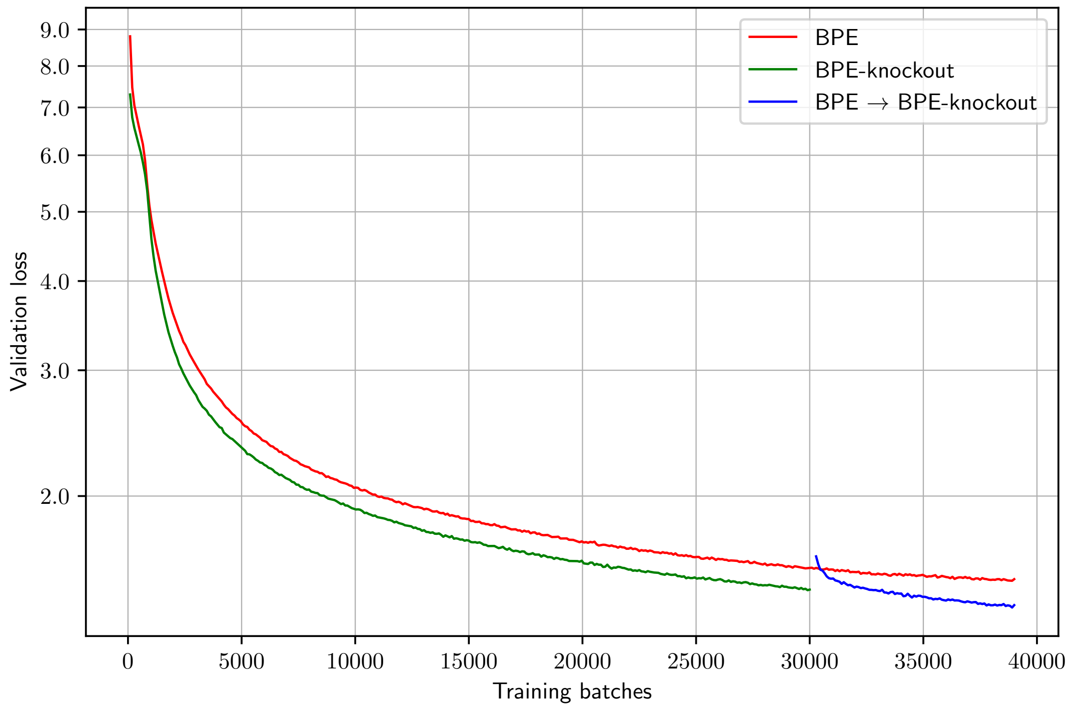


Figure 12 – Masked language modelling loss over a held-out validation set of 10k examples for Dutch RoBERTa models with different tokenisers.