Theoretical Analysis of Byte-Pair Encoding

László Kozma
Faculty of Computer Science, TU Dresden, Germany

Johannes Voderholzer
TomTom Location Technology Germany GmbH, Berlin, Germany

Abstract

Byte-Pair Encoding (BPE) is a widely used method for subword tokenization, with origins in grammar-based text compression. It is employed in a variety of language processing tasks such as machine translation or large language model (LLM) pretraining, to create a token dictionary of a prescribed size. Most evaluations of BPE to date are empirical, and the reasons for its good practical performance are not well understood.

In this paper we focus on the optimization problem underlying BPE: finding a *pair encoding* that achieves optimal compression utility. We show that this problem is APX-complete, indicating that it is unlikely to admit a polynomial-time approximation scheme. This answers, in a stronger form, a question recently raised by Zouhar et al. [42].

On the positive side, we show that BPE approximates the compression utility of the optimal pair encoding to a worst-case factor between 0.333 and 0.625. Our results aim to explain the ongoing success of BPE and are, to our knowledge, the first rigorous guarantees on its compression utility that hold for all inputs.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms; Mathematics of computing → Combinatorial algorithms

Keywords and phrases byte pair encoding, approximation, hardness, tokenization

Digital Object Identifier 10.4230/LIPIcs.ESA.2026.80

Related Version *Full Paper*: `https://arxiv.org/abs/2411.08671`

Acknowledgements Work partly done while both authors were at Freie Universität Berlin.

1 Introduction

A common step in the modern natural language processing (NLP) application pipeline is *tokenization*: given an input text, the task is to partition it into *tokens*, i.e., frequently occurring consecutive groups of symbols. The main goal is to identify semantically meaningful units (words or subwords) in order to facilitate higher level tasks (e.g., translation or text generation) [29, 1]. As this goal is difficult to directly optimize for, tokenization is usually solved heuristically, or formulated as a different but closely related task: *data compression*. Indeed, the dictionary-encoding of tokens reduces text length; the amount of compression is easy to measure, and was found to be a good predictor of the quality of tokenization for downstream tasks, e.g., for translation accuracy [14] or language model performance [16]. It is thus natural to study tokenization with the proxy optimization goal of *compression utility*.

Byte-Pair Encoding (BPE), introduced by Gage in 1994 [12], is a commonly used heuristic for tokenization. It proceeds by repeatedly identifying the *most frequently occurring* pair of symbols and replacing all occurrences of this pair with a new symbol, thereby shortening the text.[1] The new symbols, together with the pairs they replace, are stored in a lookup-table, which allows the reconstruction of the original text. In typical applications, the number

[1] The special cases of overlapping copies of a pair and ties in frequency will be discussed later.


© László Kozma and Johannes Voderholzer;
licensed under Creative Commons License CC-BY 4.0

of new symbols (and thus the size of the lookup-table) is fixed upfront, and the goal is to achieve the best compression within this budget. The symbols of the resulting (shortened) text correspond to the tokens of the input. Figure 1 shows an example of the encoding of a text by BPE.

BPE has become a de-facto standard in NLP applications, widely employed in machine translation [35, 41, 8, 14, 17] and in the preprocessing stage of training large language models [6, 27, 38, 32, 25, 40, 26][2]. Besides its effectiveness, the popularity of BPE is likely due to its simplicity and computational efficiency, when compared with more sophisticated or linguistically motivated methods (e.g., see [5, 34]). A careful implementation of BPE has a total runtime that is linear in the input length, for an arbitrary number of replacement rounds. In addition, a BPE-encoded representation can support efficient random access and pattern-matching on the original text, which is important in some applications [36].

Given the popularity and good empirical performance of BPE, there is a surprising lack of rigorous guarantees for the quality of its output. In this paper we study the problem of compressing a text (a string s over some alphabet Σ) by successive encoding of pairs (strings of length two). Adopting the framework of approximation algorithms [39] we study how well BPE, as a natural greedy heuristic, approximates this problem. Our optimization goal is to maximize *compression utility*, i.e., the reduction in text length, within k pair-encoding rounds, where s and k are given as the input (we precisely define this problem – *optimal pair encoding* – later in this section). In applications of BPE the value of k is informed both by domain knowledge (the size of the dictionary we aim to meaningfully extract) and system constraints (the afforded memory footprint), it is thus natural to fix k upfront.

Our problem definition closely resembles the one recently introduced by Zouhar et al. [42] for the same task. This abstract setting presents a challenging algorithm design problem and allows a clean theoretical analysis of BPE. Note however, that we necessarily ignore some practical aspects and optimizations of BPE-variants (e.g., special treatment of whitespace and punctuation or language-specific rules [32, 1]).

An algorithm $\mathcal{A}$ for optimal pair encoding has *approximation ratio* $\alpha \leq 1$, if the compression utility of $\mathcal{A}$ is at least α times the optimum for all inputs (s, k). The greedy step of BPE is locally optimal, and thus, for $k = 1$ it achieves optimal compression. For $k > 1$, however, simple examples show that BPE may not achieve optimal compression (see Figure 1).

As our main complexity-result, we show that optimal pair encoding is *APX-complete*. This means (informally) that no polynomial-time algorithm can approximate it to a factor arbitrarily close to 1, unless P=NP. On the positive side, we show that BPE achieves an approximation ratio α, with $0.333 < \alpha \leq 0.625$. We note that previously no constant-approximation guarantee was known for BPE or other algorithms. The question of whether optimal pair encoding is NP-complete was raised recently by Zouhar et al. [42]; our result settles this question in a stronger form.

Before precisely stating our results, we review some further related work and give a formal definition of the problem and the algorithms that we study.

Related work. BPE has its origins in text compression, in particular, the class of *grammar-based* compression methods or *macro schemes*, e.g., see [37, 22, 7, 28] for surveys. (The encoding obtained by BPE can be seen as a restricted kind of context-free grammar or string straight-line program.) A compression method closely related to BPE is Re-Pair [24]. Re-Pair differs from BPE in that its number of replacement rounds k is not fixed; instead, it

[2] See also `https://github.com/google/sentencepiece` and `https://github.com/openai/tiktoken`.

$$\texttt{aabaaaba} \to \texttt{XbXaba} \to \texttt{YXaba} \to \texttt{Zaba}$$
$$\texttt{aabaaaba} \to \texttt{aXaaXa} \to \texttt{YaYa} \to \texttt{ZZ}$$

Figure 1 Input $s = \texttt{aabaaaba}$ encoded by BPE merge sequence ($\texttt{aa} \to \texttt{X}$, $\texttt{Xb} \to \texttt{Y}$, $\texttt{YX} \to \texttt{Z}$) (above). An optimal encoding by the merge sequence ($\texttt{ab} \to \texttt{X}$, $\texttt{aX} \to \texttt{Y}$, $\texttt{Ya} \to \texttt{Z}$) (below). Here, the difference is due to an unlucky tie-breaking; we show later that BPE can be far from optimal even if there are no ties.

performs replacements as long as they achieve a saving (i.e., as long as some pair appears in at least two disjoint copies). Re-Pair is widely used, e.g., in bioinformatics, and several variants and practical improvements of it have been proposed [23, 15, 13, 11].

The central question of grammar-based compression is to find a minimal grammar that generates a given text. This task is known to be NP-hard, as well as hard to approximate [37, 7] (by some constant factor). The best known approximation ratio for the grammar-based compression of an input of length n is $O(\log n)$ [33, 19]. The approximation ratio of Re-Pair is at most $O((n/\log n)^{2/3})$ and at least $\Omega(\log n/\log\log n)$ [7, 3, 18]. Note that these results relate the size of the obtained grammar to that of the minimal grammar, where the latter can be of a more general form than what Re-Pair (or BPE) can produce.

Navarro and Russo [30] show a different kind of theoretical guarantee for Re-Pair, namely that its cost approximates the order-t entropy of the input, for a certain range of t and alphabet size. Furuya et al. [11] bound the gap between runs of Re-Pair with different tie-breaking. These results relate to a different measure of optimality, compressed length instead of reduction, are thus not directly comparable to ours (we discuss these criteria in detail later); nevertheless, a construction from [11] turns out to be useful in our context.

Closest to our work is the recent paper of Zouhar et al. [42], which initiated the formal study of BPE and optimal pair encoding that we also largely follow in this paper, apart from small notational differences and a more general problem formulation. Using the theory of submodular functions, they relate the approximation ratio of BPE to a certain parameter (total backward curvature) that depends on the unknown optimum. Zouhar et al. observe an empirical bound on this quantity, however, without giving any worst-case guarantees.

Problem definition. We consider strings (sequences of symbols) over some alphabet Σ and denote concatenation of strings a and b by $a \cdot b$, omitting the $\cdot$ when clear from context. We denote the *length* of a string s by $|s|$, the i-th symbol of s by $s[i]$, and the substring $s[i] \cdot s[i+1] \cdots s[j]$ by $s[i:j]$. We thus have $s = s[1:|s|]$.

A *replacement rule* is a function $\mathsf{replace}_{x\to y}$ that transforms a string s by replacing *all* occurrences of the string x in s with the string y. Formally, $\mathsf{replace}_{x\to y}(s) = s$ if s does not contain x, and otherwise $\mathsf{replace}_{x\to y}(s) = s[1:i-1] \cdot y \cdot \mathsf{replace}_{x\to y}(s[i+|x|:|s|])$, where i is the smallest index for which $s[i:i+|x|-1] = x$.

A sequence of replacement rules $\mathcal{R} = (\mathcal{R}_1, \ldots, \mathcal{R}_k)$ with $\mathcal{R}_i = \mathsf{replace}_{a_i b_i \to c_i}$, where a_i, b_i, c_i are symbols, is called a *merge sequence* of length k. Denoting $s' = (\mathcal{R}_k \circ \cdots \circ \mathcal{R}_1)(s)$, where $\circ$ is the function composition, we refer to $|s'|$ as the *compressed length*, and $|s| - |s'|$ as the *utility* of $\mathcal{R}$ for s. In words, s' is obtained from s by applying the sequence of replacement rules $\mathcal{R}_1, \ldots, \mathcal{R}_k$. We refer to the i-th step, i.e., the application of $\mathcal{R}_i$ as the i-th *merge*. We sometimes use the term *full merge* to emphasize that no copies of $a_i b_i$ remain after the operation.

$$\texttt{abcd}\,|\,\texttt{bc}\,|\,\texttt{bcda}\,|\,\texttt{cd}\,|\,\texttt{cdab}\,|\,\texttt{da}\,|\,\texttt{dabc}\,|\,\texttt{ab} \to \texttt{XY}\,|\,\texttt{Z}\,|\,\texttt{ZT}\,|\,\texttt{Y}\,|\,\texttt{YX}\,|\,\texttt{T}\,|\,\texttt{TZ}\,|\,\texttt{X}$$
$$\texttt{abcd}\,|\,\texttt{bc}\,|\,\texttt{bcda}\,|\,\texttt{cd}\,|\,\texttt{cdab}\,|\,\texttt{da}\,|\,\texttt{dabc}\,|\,\texttt{ab} \to \texttt{XZ}\,|\,\texttt{Y}\,|\,\texttt{YT}\,|\,\texttt{Z}\,|\,\texttt{ZX}\,|\,\texttt{T}\,|\,\texttt{dXc}\,|\,\texttt{X}$$

Figure 2 Input $s = \texttt{abcd}\,|\,\texttt{bc}\,|\,\texttt{bcda}\,|\,\texttt{cd}\,|\,\texttt{cdab}\,|\,\texttt{da}\,|\,\texttt{dabc}\,|\,\texttt{ab}$, where | denotes a distinct symbol for each occurrence. An optimal OPE encoding of s (above) with utility $\mathsf{OPT}(s,4) = 12$. An optimal OMS encoding of s (below) with utility $\mathsf{OPT}^m(s,4) = 11$. The OMS solution is obtained via the merge sequence ($\texttt{ab} \to \texttt{X},\ \texttt{bc} \to \texttt{Y},\ \texttt{cd} \to \texttt{Z},\ \texttt{da} \to \texttt{T}$).

Given the resulting encoded (compressed) string s', we can recover s by applying the sequence of reverse transformations $\mathcal{R}' = (\mathcal{R}'_k, \ldots, \mathcal{R}'_1)$ to s', with $\mathcal{R}'_i = \mathsf{replace}_{c_i \to a_i b_i}$. The symbols c_i, for all $i \in [k]$, are assumed to be new, i.e., not appearing in s or in $(\mathcal{R}_j \circ \cdots \circ \mathcal{R}_1)(s)$ for $j < i$. Indeed, if c_i already appears in the string, then the replacement $a_i b_i \to c_i$ may not be unambiguously reversible.

We can now formulate our main optimization problems. Given a string s and an integer $k > 0$, find a merge sequence $\mathcal{R}$ of length k, of maximal utility for s (or equivalently, of minimal compressed length). We denote this optimal utility as $\mathsf{OPT}^m(s,k)$, and call the task of computing it the **optimal merge sequence (OMS) problem**.[3] Note that maximizing compression utility and minimizing compressed length are equivalent for exact computation, but not necessarily for approximability.

We also define a more general optimization problem where we do not require to replace *every occurrence* of a pair. Formally, a *partial replacement rule* $\mathcal{R}^*_i = \mathsf{replace}^*_{a_i b_i \to c_i}$ can be any function that satisfies $\mathsf{replace}_{c_i \to a_i b_i}(\mathsf{replace}^*_{a_i b_i \to c_i}(s)) = s$ for all s. In words, $\mathcal{R}^*_i$ replaces *some* occurrences of $a_i b_i$ with c_i. A sequence of partial replacement functions $\mathcal{R}^* = (\mathcal{R}^*_1, \ldots, \mathcal{R}^*_k)$ is a *partial merge sequence*. Denoting $s' = (\mathcal{R}^*_k \circ \cdots \circ \mathcal{R}^*_1)(s)$, we define utility and compressed length of $\mathcal{R}^*$ analogously to merge sequences. Notice that s can be recovered from s' identically to the case of merge sequences.

The **optimal pair encoding (OPE) problem** asks, given a string s and an integer $k > 0$, to find a partial merge sequence $\mathcal{R}^*$ of length k, of maximal utility for s. We denote this optimal utility as $\mathsf{OPT}(s,k)$.

While the OMS problem is perhaps more natural, OPE is more general, and as shown in Figure 2, it can indeed be stronger (i.e., it is sometimes worth not merging every occurrence of a pair). Most of our results in this paper apply to both problems.

Byte-pair encoding (BPE). BPE solves both problems (OPE and OMS) as follows. Starting with the input string s, it performs k *locally optimal* full merge steps, always choosing a pair whose replacement maximizes compression utility. Formally, for input (s,k), it outputs $\mathcal{R} = (\mathcal{R}_1, \ldots, \mathcal{R}_k)$, where $\mathcal{R}_i = \mathsf{replace}_{a_i b_i \to c_i}$. Denoting $s^{(0)} = s$, and $s^{(i)} = \mathcal{R}_i(s^{(i-1)})$ for $i \in [k]$, each c_i is a new symbol, i.e., not occurring in $s^{(j)}$ with $j < i$, and for $i = 1, \ldots, k$, the pair $a_i b_i$ is chosen so that $|\mathcal{R}_i(s^{(i-1)})|$ is minimal.

[3] Apart from slightly different notation that is more convenient for our arguments, the OMS problem is identical to the problem defined by Zouhar et al. [42].

With careful data structuring, identifying $a_i b_i$ and performing $\mathcal{R}_i$ can be done in linear total time over all k merge steps, e.g., see [36]. In this paper, we ignore such implementation details and focus on the total utility of BPE, i.e., $|s| - |s'|$, where $s' = s^{(k)}$. We denote this quantity as $\mathsf{BPE}(s, k)$. Notice that $\mathsf{BPE}(s, k) \leq \mathsf{OPT}^m(s, k) \leq \mathsf{OPT}(s, k)$ clearly holds.

We remark that a number of choices allow for small variation in the definition of BPE (and partly of OPT^m): (1) when choosing a pair to replace, in case of a tie in utility, we pick the pair that appears first; (2) the utility of a chosen pair may be smaller than its number of occurrences in case of overlaps (e.g., the pair `aa` appears twice in `aaa`, but only one of its copies can be replaced) – one could pick by number of occurrences, instead of utility; and (3) in case of such overlapping pairs, we do the replacements left-to-right, e.g., `aaa` $\to$ `Xa`, whereas `aaa` $\to$ `aX` would also be a valid choice.

Overall, the effect of these design decisions appears negligible. Our results hold unchanged regardless of the tie-breaking strategy for (1). As for (2) and (3), the implementation we chose appears better motivated than the alternatives, but our results can easily be adapted to the other variants; see also [24, 42] for discussion.

Our results. As defined, OPE and OMS are natural string compression problems (maximizing compression utility), and BPE is a straightforward greedy heuristic for both. Surprisingly, no worst-case guarantee is known for BPE or for any other algorithm solving OPE or OMS.

Zouhar et al. [42] raised the question of whether the OMS decision problem is NP-hard. Our first result, shown in § 2 answers this question in a stronger form. We show that both OMS and OPE are in fact APX-complete, ruling out the existence of a *polynomial time approximation scheme* (PTAS), unless P=NP.

▶ **Theorem 1.** *OPE and OMS are APX-complete.*

The fact that the number k of merge-steps is part of the input is crucial; for fixed values of k a polynomial-time exact algorithm is easy to derive. The APX-hardness also holds for the problem of minimizing compressed length, as well as for some other variants of the problem, as discussed in § 2.

As for BPE, we analyze its approximation ratio for compression utility, showing in § 3:

▶ **Theorem 2.** *BPE approximates OPE with a ratio of* α*, where* $1/3 \leq \alpha \leq 0.625$*.*

Closing this gap is an intriguing open question. Note that the result also implies an approximation of OMS with the same or better ratio, as well as that OPT and OPT^m are within a constant factor of each other. Unlike the hardness result, this guarantee does not transfer to the dual optimization problem of minimizing compressed length. In particular, we show in § 4 that BPE cannot achieve a constant approximation for this measure.

▶ **Theorem 3.** *The BPE approximation ratio for OPE or OMS compression length is* $\Omega(n)$*.*

While our main focus is on the BPE algorithm, we find the OPE optimization problem of independent interest. We give in § 5 a simple algorithm we call EvenOdd, that achieves an approximation ratio of 0.5. We stress that despite this guarantee, on typical inputs BPE appears to behave better than EvenOdd and the latter should be seen as a proof of concept.

▶ **Theorem 4.** *EvenOdd is a linear time* 0.5*-approximation for OPE.*

The following four sections are dedicated to the proofs of Theorems 1–4. In § 6 we conclude with a list of open questions.

2 Hardness of Approximation

In this section we show that the optimal merge sequence (OMS) problem is APX-complete, proving part of Theorem 1. The APX-completeness of OPE is based on similar ideas and is completed in Appendix A. As we show in § 3, both OPE and OMS are in APX (i.e., admit a constant-factor approximation), it just remains therefore to show APX-hardness. We do so via an L-reduction (linear reduction, see [31]) from maximum cut in cubic graphs.

In maximum cut, given an undirected graph $G = (V, E)$, the task is to partition the vertex set V into two parts, such as to maximize the number of edges between the two parts. Maximum cut is NP-complete [20], and its optimization version is APX-complete [31] even when restricted to input graphs in which every vertex has degree exactly three [2].

Our reduction is from such (cubic, undirected, unweighted) instances $G = (V, E)$ of maximum cut.

Assume $V = \{v_1, \ldots, v_n\}$ and associate a symbol ℓ_i to each vertex $v_i \in V$. We construct a string s by concatenating, for each edge $\{v_i, v_j\} \in E$ in arbitrary order, a string s_{ij}, with $i < j$. Then, we append for each vertex $v_i \in V$ four copies of a string s_i. Finally, we append $20n$ copies of a padding string s_0. These strings are defined as follows, with product signs indicating concatenation:

$$\begin{aligned} s_{ij} &= \#\ell_i\#\#\ell_j\#\,|\,\#\ell_j\#\#\ell_i\#\,|, \\ s_i &= \ell_i\#\ell_i\,|, \\ s_0 &= \#\#\,|, \\ s &= \prod_{\substack{\{v_i,v_j\}\in E \\ i<j}} s_{ij} \cdot \prod_{v_i \in V} s_i s_i s_i s_i \cdot \prod_{t=1}^{20n} s_0. \end{aligned}$$

Here, $\#$ is a single fixed symbol, and $|$ denotes a distinct symbol for each occurrence. Setting $k = n+1$, we take $\mathsf{OPT}^m(s, k)$ to be the optimal utility of the resulting OMS instance. Notice that the construction takes polynomial time.

We first show that G has a cut of size at least c if and only if $\mathsf{OPT}^m(s, k) \geq 30n + c$, i.e., if there is a merge sequence of length k, of at least the given utility for s. This already implies the NP-hardness of the problem and sets the stage for proving hardness of approximation.

($\Rightarrow$) Consider a cut of G, i.e., a partitioning $(S, V \setminus S)$ of V with c edges across the cut. We construct a merge sequence of length k (recall that a *merge* replaces every occurrence of a pair with a new symbol). We first merge $\#\ell_i$ for every $v_i \in S$, and $\ell_i\#$ for every $v_i \in V \setminus S$. Notice that each merge step achieves utility 10 (6 from the s_{ij} corresponding to the three edges incident to v_i, and 4 from the s_i strings), for a total of $10n$. Then, we merge $\#\#$, which achieves utility $c + 20n$. To see this, notice that, after the first n merges, one of the two $\#\#$ pairs remains in a string s_{ij} exactly if the corresponding edge $\{v_i, v_j\}$ is in the cut, and otherwise no such pair remains in s_{ij}. The term $20n$ is from the padding string s_0. This adds up to a utility of $30n + c$ as claimed.

($\Leftarrow$) Consider a merge sequence $\mathcal{R}$ of length k, of utility $30n + c$ for s. We call the merge sequence *well-formed*, if it includes merging the pair $\#\#$, and for each $v_i \in V$ exactly one of the pairs $\ell_i\#$ and $\#\ell_i$; moreover, the merge $\#\#$ appears last. We show that it is sufficient to consider well-formed merge sequences.

▷ Claim 5. If $\mathcal{R}$ is not well-formed, then we can find (in polynomial time) a well-formed merge sequence $\mathcal{R}'$ of the same length with utility at least as large as the utility of $\mathcal{R}$ on s.

Proof. Observe that any new symbol introduced in a merge encodes a substring of s. Since pairs other than ## occur at most 10 times in s, a string different from ## can also occur at most this many times. Thus, any merge other than ## can have utility at most 10.

Recall that *any* well-formed merge sequence has utility at least $30n$, by the earlier correspondence to an arbitrary cut, possibly of size zero. So, if $\mathcal{R}$ does not merge ##, then its total utility is at most $10(n+1) < 30n$ and we can replace $\mathcal{R}$ by an arbitrary well-formed merge sequence of larger utility. Assume therefore that ## is merged by $\mathcal{R}$.

Suppose $\mathcal{R}$ is not well-formed, and that there exist t indices (for some $t > 0$) $i_1, \ldots, i_t \in [n]$ such that none of the two pairs $\ell_{i_j}\#$, $\#\ell_{i_j}$ are merged by $\mathcal{R}$, for all $j \in [t]$. Instead, there must be exactly t "bad" merges, which can be of the types listed below. We show that removing the t bad pairs from $\mathcal{R}$ and adding the missing pairs $\ell_{i_j}\#$, for all $j \in [t]$, we obtain a merge sequence $\mathcal{R}'$ of larger utility on s.

Since $\mathcal{R}'$ contains the set of merges of a well-formed sequence, each newly added pair contributes a utility of at least 7. This is because the pair appears once in each of the three relevant strings s_{ij}, in copies that do not overlap with ##, and thus contribute to the utility. (Note that we did not yet assume anything about when the ##-merge happens.) An additional utility of 4 comes from the four copies of s_i. As we add the pairs to the end of the merge sequence, they do not reduce the utilities of other merges.

To finish the proof, we argue that the removal of each "bad" merge from $\mathcal{R}$ decreases utility by at most 6; as the remaining merges are unaffected, we obtain an overall increase in utility of at least one for each bad merge. Bad merges that deviate from a well-formed merge sequence can be of the following types:

1. A merge involving the symbol $|$: this yields a utility of one, by construction.
2. A merge involving a new symbol (not part of the original s): such a merge corresponds to a substring of s of length at least three. Of all such substrings, those involving $|$ occur at most once, $\#\ell_i\#$ occurs 6 times, $\#\#\ell_i$ and $\ell_i\#\#$ occur 3 times, and $\ell_i\#\ell_i$ occurs 4 times, for all i; there are no further cases (longer substrings also cannot occur more than 6 times). Thus any such merge can yield utility of at most 6.
3. A merge of $\ell_i\#$ after $\#\ell_i$ has already been merged, or a merge of $\#\ell_i$ after $\ell_i\#$ has already been merged: assuming that bad merges of the first two types have been removed already, this yields utility at most 6. To see this, observe that for all such pairs, initially there are 6 copies in s_{ij} strings and 4 copies in s_i strings, and the 4 copies in s_i have been destroyed by merging the other pair.

Finally, we argue that merging ## last cannot decrease utility. Indeed, any reduction in utility, i.e., a copy of ## that is destroyed by an earlier $\#\ell_i$- or $\ell_i\#$-merge made possible by postponing ## to the end, is offset by the additional utility of that merge. ◁

Suppose that $\mathcal{R}$ is well-formed (otherwise, transform it according to Claim 5). From $\mathcal{R}$, compute a cut $(S, V \setminus S)$ of G by letting S be the set of vertices $v_i \in V$ where merge $\#\ell_i$ is in $\mathcal{R}$. Consider the string s_{ij} for an edge $\{v_i, v_j\}$. Notice that we can achieve a total utility of 5 for this string if and only if one of v_i and v_j was placed in S and the other was not, i.e., if the edge contributes to the constructed cut. Otherwise, both $\#\ell_i$ and $\#\ell_j$, or both $\ell_i\#$ and $\ell_j\#$ are merged, and a utility of at most 4 is achievable for s_{ij}, as both ## pairs overlap with the two other pairs.

Of the total utility of at least $30n + c$ we obtain $20n$ in the s_0-strings, $4n$ in the s_i-strings, with $6n + c$ remaining for the s_{ij}-strings corresponding to the $3n/2$ edges of G. By the above calculations these can contribute exactly $4 \cdot 3n/2 = 6n$ plus the size of the constructed cut, which must therefore be at least c. This concludes the reduction for NP-hardness.

We next turn the argument into an L-reduction [31], thus showing the APX-hardness of OMS. Let OPT^c denote the maximum cut size for graph G, and let $\mathsf{OPT}^m = \mathsf{OPT}^m(s,k)$ be the maximum utility for the OMS instance resulting from the above reduction. It remains to show that there exist positive constants α and β (independent of the input G) so that, for all sufficiently large G:

1. $\mathsf{OPT}^m \leq \alpha \cdot \mathsf{OPT}^c$, and
2. for every feasible solution of the OMS instance (s,k) of utility u we can obtain, in polynomial time, a cut of G of size c, with $|\,\mathsf{OPT}^c - c\,| \leq \beta \cdot |\,\mathsf{OPT}^m - u\,|$.

Part 1. is immediate: since G is cubic with n vertices, it has $3n/2$ edges, and we can add up the lengths of the different parts as $|s| = 3n/2 \cdot 14 + n \cdot 4 \cdot 4 + 20n \cdot 3 = 97n$, which upper bounds OPT^m; on the other hand, from the well-known lower bound on the maximum cut (half the number of edges) we have $\mathsf{OPT}^c \geq 3n/4$, implying the inequality for $\alpha = 130$.

For Part 2., consider a merge sequence $\mathcal{R}$ of utility u for (s,k). If $\mathcal{R}$ is not well-formed, we use the transformation of Claim 5 to obtain a well-formed merge sequence $\mathcal{R}'$ for (s,k) of utility $u' > u$. Otherwise let $u' = u$ and $\mathcal{R}' = \mathcal{R}$. Now, as $\mathcal{R}'$ is well-formed, we interpret it as a cut of size c in G, where, by the earlier reduction, $c = u' - 30n$. Moreover, by applying this argument to the optimal OMS solution, we obtain a cut of size $\mathsf{OPT}^m - 30n$. As we have shown earlier that $\mathsf{OPT}^m \geq 30n + c'$ for every cut c', our cut must be maximal and $\mathsf{OPT}^c = \mathsf{OPT}^m - 30n$. We thus have $|\,\mathsf{OPT}^c - c\,| = |\,\mathsf{OPT}^m - u'\,| \leq |\,\mathsf{OPT}^m - u\,|$, proving the inequality for $\beta = 1$.

This completes the APX-hardness of OMS. In Appendix A we extend the argument to show the APX-hardness of the OPE problem.

Two small remarks are in order. First, we note that the APX-hardness also applies to the compressed length minimization version of OMS (and OPE); in the above hard instance the input length is $|s| = 97n$, and $\mathsf{OPT}^m(s,k) \geq 30n$. Thus, a $(1+\varepsilon)$-approximation for $|s'| = |s| - \mathsf{OPT}^m(s,k)$ would also yield a $(1-\varepsilon')$-approximation for OPT^m, with $\varepsilon' = ((97-30)/30)\,\varepsilon \leq 2.24\,\varepsilon$.

Second, we could restrict the OMS (and OPE) problems such as to only allow merging pairs of original input symbols, and not pairs involving new symbols. (Equivalently, all resulting tokens would be of length at most two; note that the BPE solution does not conform to this restriction.) Since in the above reduction optimal merges involve only input symbols, our APX-hardness results immediately transfer to these restricted variants.

3 Approximation ratio of Byte-Pair Encoding

In this section we study the approximation ratio of BPE for the OPE problem, showing constant upper and lower bounds for this ratio, proving Theorem 2.

Upper bound. Consider the string $s = \texttt{abaacaaba}\,|\,\texttt{aca}$, with a merge sequence of length $k = 4$. Suppose BPE first performs the merge $\texttt{aa} \to \texttt{X}$, resulting in the string $\texttt{abXcXba}\,|\,\texttt{aca}$. As now each pair occurs at most once, subsequent merges can only shorten the string by 1, for a total utility of 5. Consider now the alternative merge sequence ($\texttt{ac} \to \texttt{X}$, $\texttt{Xa} \to \texttt{Y}$, $\texttt{ab} \to \texttt{Z}$, $\texttt{Za} \to \texttt{T}$) resulting in $\texttt{TYT}\,|\,\texttt{Y}$, with utility 8, and a ratio of $5/8 = 0.625$.

To enforce that BPE merges $\texttt{aa}$ first, we can concatenate t copies of $s \cdot \#$, and a final $\texttt{aa}$. The utility of BPE is now (without relying on favorable tie-breaking) $5t+1$, whereas the alternative merge sequence yields $8t$, for a ratio arbitrarily close to 0.625. Notice that while we claimed the result for OPE, the upper bound (more strongly) applies to OMS (i.e.,

for the ratio $\mathsf{BPE}(s,k)/\mathsf{OPT}^m(s,k)$), as the alternative encoding above is given by a full merge sequence. Determining whether a construction with worse ratio exists for BPE is an intriguing open question.

Lower bound. We now show our main theoretical guarantee for BPE, a lower bound of 1/3 on its approximation ratio. We show this bound (more strongly) with respect to OPE. Consider an optimal (possibly partial) merge sequence $\mathcal{R}^* = (\mathcal{R}^*_1, \dots, \mathcal{R}^*_k)$, where $\mathcal{R}^*_i = \mathsf{replace}^*_{a_i b_i \to c_i}$ for $i \in [k]$. Let s' be the string resulting from applying $\mathcal{R}^*$ to input s. Recall that $\mathsf{OPT}(s,k) = |s| - |s'|$.

In order to relate $\mathsf{BPE}(s,k)$ and $\mathsf{OPT}(s,k)$, we first introduce a simpler quantity that upper bounds both.

For a string s of length n, a set of t indices $\{i_1, \dots, i_t\} \subseteq [n-1]$ is a *pair packing*, if the pairs starting at the given indices of s do not overlap and are all equal as strings. Formally, $|i_j - i_k| \geq 2$ for $j,k \in [t]$, and $s[i_1 : i_1 + 1] = \cdots = s[i_t : i_t + 1]$. A *$k$-packing* is the union of k disjoint pair packings, i.e., no index appears in more than one of the pair packings. We refer to a largest possible k-packing as the *optimal k-packing* and denote its size by $P_k(s)$. We remark that $P_k(s)$ can be computed in polynomial time, but this is not needed in our current study.

▶ **Lemma 6.** *For s' obtained from s by applying an arbitrary (partial) merge sequence $\mathcal{R}^*$ of length k, we have $|s| - |s'| \leq P_k(s)$.*

Proof. We create a k-packing from s and $\mathcal{R}^*$, of size $|s| - |s'|$. To do this, it will be useful to map every occurrence of a symbol during the application of $\mathcal{R}^*$, to the substring of s which is encoded by the symbol. For a string c let $\mathsf{decode}(c,0) = c$, and for $i \in [k]$ let $\mathsf{decode}(c,i) = \mathsf{decode}(\mathsf{replace}_{c_i \to a_i b_i}(c), i-1)$. Observe that every occurrence of c_i after the application of $\mathcal{R}^*_i$ encodes a unique copy of $\mathsf{decode}(c_i,i)$ in s, these copies do not overlap, and $\mathsf{decode}(c_i,i) = \mathsf{decode}(a_i, i-1) \cdot \mathsf{decode}(b_i, i-1)$.

We *charge* each replacement $a_i b_i \to c_i$, i.e., each unit of the compression utility, in the order of their application in $\mathcal{R}^*$, to a certain index of s. More precisely, charge the replacement $a_i b_i \to c_i$ to the index in s of the last symbol of $\mathsf{decode}(a_i, i-1)$ in the location encoded by the current copy of a_i. Intuitively, we charge the merging of a pair to the location in s where the "gluing" happens.

Observe that this index cannot be charged again in any step, since after replacement $a_i b_i \to c_i$, it points to a symbol "inside" $\mathsf{decode}(c_i,i)$, that is not the last one, and future merges cannot cause it to become a last symbol. The indices charged due to $\mathcal{R}^*_i$ are at pairwise distance at least 2, for all $i \in [k]$ (as otherwise two copies of $\mathsf{decode}(c_i,i)$ would overlap in s), and the pairs starting at the charged indices are equal as strings, as all consist of the last symbol of $\mathsf{decode}(a_i, i-1)$ and first symbol of $\mathsf{decode}(b_i, i-1)$. It follows that the set of indices charged during the entire process is a k-packing, thus, the total utility $|s| - |s'|$ is at most the size $P_k(s)$ of the optimal k-packing. See Figure 3 for illustration. ◀

Notice that as a corollary of Lemma 6 we have $\mathsf{OPT}(s,k) \leq P_k(s)$.

In the remainder of the section we *lower bound* the BPE utility in terms of the quantity we defined, which, together with Lemma 6 will immediately imply the claimed lower bound on the approximation ratio of BPE.

▶ **Lemma 7.** *For all strings s and $k \geq 0$ we have $\mathsf{BPE}(s,k) \geq P_k(s)/3$.*

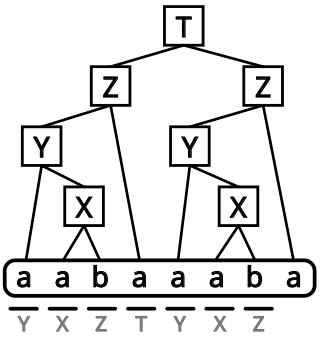

Figure 3 Sequence of merges `aabaaaba` → `aXaaXa` → `YaYa` → `ZZ` → `T` shown as a tree (above) and corresponding k-packing (below).

Proof. As before, $s^{(0)} = s$ and let $s^{(k)} = s'$ denote the encoding obtained by BPE, with $s^{(i)}$ resulting from $s^{(i-1)}$ (for $i \in [k]$) by a greedy merge step, as described in the definition of BPE.

Let a_ib_i be the pair merged by BPE when going from $s^{(i-1)}$ to $s^{(i)}$, for $i \in [k]$, and let $\mathcal{B}$ be a k-packing derived from the BPE merge sequence according to the process in Lemma 6. Let $\mathcal{B}_1 \cup \cdots \cup \mathcal{B}_k$ be the partitioning of $\mathcal{B}$ into the k pair packings corresponding to the k merge steps of BPE. (When BPE does a replacement $a_ib_i \to c_i$ in $s^{(i-1)}$, then the corresponding entry in $\mathcal{B}_i$ is the index of the last symbol of the substring in s encoded by a_i.) Note that $|\mathcal{B}| = \mathsf{BPE}(s, k)$.

Let $\mathcal{P}$ be an optimal k-packing partitioned into k pair packings $\mathcal{P}_1 \cup \cdots \cup \mathcal{P}_k$, where $\mathcal{P}_i$ contains starting indices of the pair x_iy_i in s for $i \in [k]$. Note that $|\mathcal{P}| = P_k(s)$. Also note that pairs x_iy_i, x_jy_j from different pair packings are not necessarily distinct. For BPE, however, pairs a_ib_i, a_jb_j from different steps are in fact, distinct.

We iterate i from 1 to k, relating for each i the number of entries in $\mathcal{B}_i$ and the number of entries in $\mathcal{P}_i$. In each step we delete at most $3|\mathcal{B}_i|$ elements from $\mathcal{P}$. The set $\mathcal{B}$ is not modified during the process. In the end, $\mathcal{P}$ will be empty, implying the claim.

We maintain two invariants:
$\mathbf{I_1}$: After step i, the sets $\mathcal{P}_j$ for $j \le i$ are empty.
$\mathbf{I_2}$: After step i, no index $x \in \mathcal{P}$ is a neighbor of an index $y \in \mathcal{B}_j$, for $j \le i$.
Two indices x and y are neighbors if $|x - y| = 1$.

Intuitively, $\mathbf{I_1}$ ensures that all elements of $\mathcal{P}$ are accounted for, and $\mathbf{I_2}$ ensures that pairs $s[x : x + 1]$ corresponding to $x \in \mathcal{P}$ can still be merged by BPE, as the two symbols forming the pair did not take part in merges yet. Initially both invariants are trivially true.

Consider step i. For all elements $x \in \mathcal{B}_i$, delete x and its neighbors $x - 1$ and $x + 1$ from $\mathcal{P}$. Note that we delete at most $3|\mathcal{B}_i|$ elements in this way. Also note that this establishes $\mathbf{I_2}$ for the current step. It remains to verify $\mathbf{I_1}$. We distinguish two cases.

Case 1. If $a_ib_i = x_\ell y_\ell$, for some $i \le \ell \le k$, then swap x_iy_i with $x_\ell y_\ell$ and $\mathcal{P}_i$ with $\mathcal{P}_\ell$ (this is only necessary if $\ell \ne i$; also note that the swap does not affect $\mathbf{I_1}$, $\mathbf{I_2}$). Now, if $\mathcal{P}_i$ is nonempty (after the deletions and swap), then take any element $x \in \mathcal{P}_i$. Notice that BPE could have also merged $s[x : x + 1]$ in the i-th step, in addition to the pairs indexed by $\mathcal{B}_i$. This is because the pair equals a_ib_i, it does not neighbor any pair merged by BPE in the present or past rounds (by $\mathbf{I_2}$), and the pair itself was not merged before, as otherwise x would have been deleted. This contradicts the definition of BPE, showing $\mathcal{P}_i = \emptyset$, and hence, $\mathbf{I_1}$.

Case 2. If $a_ib_i \neq x_\ell y_\ell$ for $i \leq \ell \leq k$. Notice that in this case, $\mathcal{P} \cap \mathcal{B}_i = \emptyset$ even before the deletion (by $\mathbf{I_1}$), thus we could have only deleted the neighbors of $\mathcal{B}_i$ from $\mathcal{P}$ in the i-th step. If, before the deletion, $|\mathcal{P}_i| > |\mathcal{B}_i|$, then BPE could have merged, instead of the pairs a_ib_i indexed by $\mathcal{B}_i$, the pairs x_iy_i indexed by $\mathcal{P}_i$, contradicting the definition of BPE. This is because pairs $s[x : x + 1]$ for $x \in \mathcal{P}_i$ do not neighbor any pair merged by BPE in the present or past rounds (by $\mathbf{I_2}$), and were not merged before (as otherwise x would have been deleted). Moreover, as $\mathcal{P}_i$ forms a pair packing, the pairs indexed by $\mathcal{P}_i$ are not neighboring each other and could thus have been merged in one round of BPE.

Thus, $|\mathcal{P}_i| \leq |\mathcal{B}_i|$ and $\mathcal{P} \cap \mathcal{B}_i = \emptyset$ (before the deletion). As we cannot delete entries of $\mathcal{B}_i$ from $\mathcal{P}$, we instead use our remaining budget of $|\mathcal{B}_i|$ deletions to delete the entries of $\mathcal{P}_i$, establishing $\mathbf{I_1}$.

After the k-th step, $\mathbf{I_1}$ implies that $\mathcal{P}$ is empty, proving the lemma. ◀

4 Lower bound for compressed length

In this section we show that for the dual optimization problem of OPE of minimizing compressed length (instead of maximizing compression utility), the approximation ratio of BPE is *unbounded*, and in fact, linear in the input size. This proves Theorem 3.

The construction is essentially the same as the example used in [11, Thm. 3] to prove a different property of Re-Pair.

Consider the string

$$s = \prod_{i=1}^{t} x_i\mathtt{a}\mathtt{a}y_i \cdot \prod_{i=1}^{t} |\, x_i\mathtt{a} \cdot \prod_{i=1}^{t} |\, \mathtt{a}y_i,$$

where $t > 2$, each occurrence of $|$ is a distinct symbol, and $\mathtt{a}$, x_i, y_i are symbols for $i \in [t]$.

If we merge every occurrence of $x_i\mathtt{a}$ and $\mathtt{a}y_i$, the resulting string has length $2t+2t+2t = 6t$ after $2t$ steps. So after $2t + (6t - 1) = 8t - 1$ total steps, we have reduced s to a single symbol.

On the other hand, BPE will always start with $\mathtt{aa}$, since it occurs t times, while every other pair occurs at most twice. After merging $\mathtt{aa}$, no pair occurs more than once in the string, so every subsequent step has a utility of one. After merging $\mathtt{aa}$, the length of the string is $3t + 3t + 3t = 9t$. So after $8t - 1$ total steps, the resulting string will be of length $9t - (8t - 2) = t + 2$.

Since $|s| = 10t$, it holds that $t + 2 = \frac{|s|}{10} + 2 \in \Omega(|s|)$.

Notice that since all merges were full, the approximation lower bound also holds for the OMS problem (in the compression length version).

5 Approximating Optimal Pair Encoding

In this section we give a simple algorithm that achieves a 0.5-approximation for the OPE problem, proving Theorem 4.

For a pair xy, let $\mathsf{freq}_{xy}(s)$ denote the number of occurrences of xy in s. Note that we allow overlaps, e.g., $\mathsf{freq}_{\mathtt{aa}}(\mathtt{aaa}) = 2$. Let $F_k(s)$ denote the total number of occurrences of the k most frequent pairs in s (breaking ties arbitrarily, as this does not affect the total value). Let $x_1y_1, \ldots, x_ky_k$ be such a set of most frequent pairs, and let $I = \{i_1, \ldots, i_N\}$, where $N = F_k(s)$, denote the set of indices where they occur, i.e., $s[i_j : i_j + 1] \in \{x_iy_i \mid i \in [k]\}$ for all $j \in [N]$.

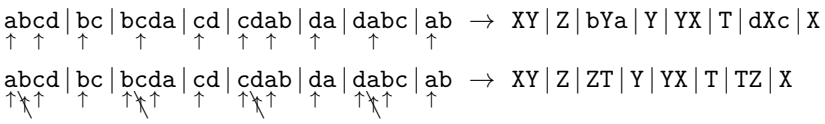

Figure 4 Encoding $s = \texttt{abcd}\,|\,\texttt{bc}\,|\,\texttt{bcda}\,|\,\texttt{cd}\,|\,\texttt{cdab}\,|\,\texttt{da}\,|\,\texttt{dabc}\,|\,\texttt{ab}$, where | denotes a distinct symbol for each occurrence. The BPE encoding (above) with utility $\mathsf{BPE}(s, 4) = 10$ via the merge sequence ($\texttt{ab} \to \texttt{X}$, $\texttt{cd} \to \texttt{Y}$, $\texttt{bc} \to \texttt{Z}$, $\texttt{da} \to \texttt{T}$). Arrows show the k-packing solution derived from the BPE merge sequence. The EvenOdd encoding (below) with $k = 4$ and utility 12. Arrows show the indices of most frequent pairs, with crossed ones removed.

We can efficiently, i.e., in time $O(|s|)$, compute a set $I = \{i_1, \ldots, i_N\}$ by enumerating all pairs occurring in s and storing the indices of the k most frequent of them. Assume that $i_1, \ldots, i_N$ are sorted increasingly. We then sparsify I, by removing neighboring indices, through a single pass through s. More precisely, for each index of the form $i_{2\ell}$, remove $i_{2\ell}$ if $i_{2\ell} = i_{2\ell-1} + 1$ or $i_{2\ell} = i_{2\ell+1} - 1$. Let I' denote the indices in I remaining after this process. Observe that I' has at least $N/2$ elements. As there are no neighboring indices in I', the occurrences of pairs in s indexed by I' do not overlap, can therefore be merged simultaneously. Notice that only k distinct pairs appear, so we obtain a valid OPE solution of utility $|I'|$. We call this the EvenOdd algorithm. The overall running time is clearly $O(|s|)$.

Since $|I'| \geq F_k(s)/2$, and clearly, $F_k(s) \geq P_k(s) \geq \mathsf{OPT}(s, k)$, the theorem follows.

On some inputs, EvenOdd can have larger utility than BPE. Indeed, consider the earlier example $s = \texttt{abcd}\,|\,\texttt{bc}\,|\,\texttt{bcda}\,|\,\texttt{cd}\,|\,\texttt{cdab}\,|\,\texttt{da}\,|\,\texttt{dabc}\,|\,\texttt{ab}$. With $k = 4$ merge rounds, BPE achieves utility 10 due to the unfavorable tie-breaking, but as seen earlier, the maximum utility of *any* OMS algorithm (regardless of tie-breaking) is at most 11. EvenOdd takes the most frequent 4 pairs $\{\texttt{ab}, \texttt{bc}, \texttt{cd}, \texttt{da}\}$ with 16 occurrences in s. After sparsification, 12 non-neighboring pairs remain and are merged, matching the OPE optimum utility of 12 on this instance; see Figure 4.

We remark that the EvenOdd algorithm only merges input symbols (and not newly introduced symbols), and thus, the decoded string of each symbol is of length at most two. Note that, instead of removing even-ranked indices, we could, in polynomial time, find the minimal set of indices whose removal makes the remainder non-overlapping (by a simple greedy strategy). In the worst case, however, no algorithm in this class can improve the approximation ratio 0.5.

Indeed, consider an input string s consisting of $2n$ identical symbols, followed by $2(k-1)$ pairwise distinct symbols. Setting $k = \log_2 n + 1$, the optimal pair encoding of s will collapse the first $2n$ symbols to one, with utility $2n - 1$. An algorithm that can only merge input pairs can achieve a maximum utility of n on the first part of the string (in one merge), and an additional $k - 1$ on the last part, yielding an approximation ratio of $(n + \log_2 n)/(2n - 1) = 0.5 + o(1)$. In this example, the optimal encoding coincides with the one found by BPE, also showing a factor $2 - o(1)$ gap between BPE and input-symbol-only algorithms, to the advantage of the former.

6 Conclusion and open questions

In this paper we studied the complexity of optimal pair encoding: the task of compressing a string by replacing pairs with new symbols, such as to maximize the overall reduction in length. We showed that the problem is APX-complete and that BPE, a popular greedy

heuristic used in state-of-the-art LLM pretraining, achieves a constant-factor approximation. Finding the best approximation ratio for the OPE or OMS problems, by BPE or by other polynomial-time algorithms, i.e., closing the gaps between our bounds in Theorem 2, resp., Theorems 1 and 4 are the central remaining questions. In particular: is there an efficient algorithm for OPE with approximation ratio above 0.5? Note that analyzing natural greedy strategies for some other string problems turned out to be very difficult. For instance, the famous *greedy superstring conjecture* [4] concerns a very intuitive string merging process, with worst-case approximation ratio conjectured to be 2 but only proven to be below ≈ 3.396 [9, 10].

While the APX-hardness (Theorem 1) extends to the compression length, the approximation guarantee (Theorem 2) does not. The polynomial-time approximability of compression length (for OPE or OMS, by any algorithm) is left open. In fact, as we lack a constant-factor approximation for this problem (BPE does not achieve it, as shown in Theorem 3), we can only claim its APX-hardness, not APX-completeness.

Our hardness result (Theorem 1) relies on an alphabet whose size increases with the input. In recent follow-up work, Kastreva et al. [21] rule out the existence of a PTAS even with constant-size alphabets, when compressing a *collection* of strings. Whether the guarantees of BPE can be strengthened in the small-alphabet case remains an interesting question.

References

1 Mehdi Ali, Michael Fromm, Klaudia Thellmann, Richard Rutmann, Max Lübbering, Johannes Leveling, Katrin Klug, Jan Ebert, Niclas Doll, Jasper Schulze Buschhoff, et al. Tokenizer choice for LLM training: Negligible or crucial? *arXiv preprint*, 2023. arXiv:2310.08754.

2 Paola Alimonti and Viggo Kann. Some APX-completeness results for cubic graphs. *Theor. Comput. Sci.*, 237(1-2):123–134, 2000. doi:10.1016/S0304-3975(98)00158-3.

3 Hideo Bannai, Momoko Hirayama, Danny Hucke, Shunsuke Inenaga, Artur Jeż, Markus Lohrey, and Carl Philipp Reh. The smallest grammar problem revisited. *IEEE Transactions on Information Theory*, 67(1):317–328, 2020. doi:10.1109/TIT.2020.3038147.

4 Avrim Blum, Tao Jiang, Ming Li, John Tromp, and Mihalis Yannakakis. Linear approximation of shortest superstrings. *J. ACM*, 41(4):630–647, July 1994. doi:10.1145/179812.179818.

5 Kaj Bostrom and Greg Durrett. Byte pair encoding is suboptimal for language model pretraining. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4617–4624, 2020. doi:10.18653/V1/2020.FINDINGS-EMNLP.414.

6 Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

7 Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and Abhi Shelat. The smallest grammar problem. *IEEE Trans. Inf. Theory*, 51(7):2554–2576, 2005. doi:10.1109/TIT.2005.850116.

8 Miguel Domingo, Mercedes García-Martínez, Alexandre Helle, Francisco Casacuberta, and Manuel Herranz. How much does tokenization affect neural machine translation? In *International Conference on Computational Linguistics and Intelligent Text Processing*, pages 545–554. Springer, 2019. doi:10.1007/978-3-031-24337-0_38.

9 Matthias Englert, Nicolaos Matsakis, and Pavel Veselý. Improved approximation guarantees for shortest superstrings using cycle classification by overlap to length ratios. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 317–330. ACM, 2022. doi:10.1145/3519935.3520001.

10 Matthias Englert, Nicolaos Matsakis, and Pavel Veselý. Approximation guarantees for shortest superstrings: Simpler and better. In Satoru Iwata and Naonori Kakimura, editors, *34th International Symposium on Algorithms and Computation, ISAAC 2023, Kyoto, Japan, December 3-6, 2023*, volume 283 of *LIPIcs*, pages 29:1–29:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ISAAC.2023.29.

11 Isamu Furuya, Takuya Takagi, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Takuya Kida. MR-RePair: Grammar compression based on maximal repeats. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *Data Compression Conference, DCC 2019, Snowbird, UT, USA, March 26-29, 2019*, pages 508–517. IEEE, 2019. doi:10.1109/DCC.2019.00059.

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

13 Travis Gagie, Tomohiro I, Giovanni Manzini, Gonzalo Navarro, Hiroshi Sakamoto, and Yoshimasa Takabatake. Rpair: rescaling RePair with rsync. In *International Symposium on String Processing and Information Retrieval*, pages 35–44. Springer, 2019. doi:10.1007/978-3-030-32686-9_3.

14 Matthias Gallé. Investigating the effectiveness of BPE: The power of shorter sequences. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pages 1375–1381, 2019. doi:10.18653/V1/D19-1141.

15 Michał Gańczorz and Artur Jeż. Improvements on Re-Pair grammar compressor. In *2017 Data Compression Conference (DCC)*, pages 181–190. IEEE, 2017.

16 Omer Goldman, Avi Caciularu, Matan Eyal, Kris Cao, Idan Szpektor, and Reut Tsarfaty. Unpacking tokenization: Evaluating text compression and its correlation with model performance, 2024. doi:10.48550/arXiv.2403.06265.

17 Ximena Gutierrez-Vasques, Christian Bentz, and Tanja Samardžić. Languages through the looking glass of BPE compression. *Computational Linguistics*, 49(4):943–1001, 2023. doi:10.1162/COLI_A_00489.

18 Danny Hucke, Artur Jez, and Markus Lohrey. Approximation ratio of repair, 2017. arXiv:1703.06061.

19 Artur Jeż. A really simple approximation of smallest grammar. In *Symposium on Combinatorial Pattern Matching*, pages 182–191. Springer, 2014.

20 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

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

22 John C Kieffer and En-Hui Yang. Grammar-based codes: A new class of universal lossless source codes. *IEEE Transactions on Information Theory*, 46(3):737–754, 2000. doi:10.1109/18.841160.

23 Justin Kim, Rahul Varki, Marco Oliva, and Christina Boucher. Recursive RePair: Increasing the scalability of repair by decreasing memory usage. *bioRxiv*, pages 2024–07, 2024.

24 N Jesper Larsson and Alistair Moffat. Off-line dictionary-based compression. *Proceedings of the IEEE*, 88(11):1722–1732, 2000. doi:10.1109/5.892708.

25 Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. Bloom: A 176b-parameter open-access multilingual language model, 2023.

26 Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint*, 2024. arXiv:2412.19437.

27 Zhuang Liu, Wayne Lin, Ya Shi, and Jun Zhao. A robustly optimized bert pre-training approach with post-training. In *China National Conference on Chinese Computational Linguistics*, pages 471–484. Springer, 2021. doi:10.1007/978-3-030-84186-7_31.

28 Markus Lohrey. Algorithmics on slp-compressed strings: A survey. *Groups-Complexity-Cryptology*, 4(2):241–299, 2012. doi:10.1515/GCC-2012-0016.

29 Sabrina J Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Gallé, Arun Raja, Chenglei Si, Wilson Y Lee, Benoît Sagot, et al. Between words and characters: A brief history of open-vocabulary modeling and tokenization in nlp. *arXiv preprint*, 2021. arXiv:2112.10508.

30 Gonzalo Navarro and Luís Russo. Re-pair achieves high-order entropy. In *Proceedings of the Data Compression Conference*, page 537, 2008. doi:10.1109/DCC.2008.79.

31 Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. Syst. Sci.*, 43(3):425–440, 1991. doi:10.1016/0022-0000(91)90023-X.

32 Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

33 Wojciech Rytter. Application of Lempel–Ziv factorization to the approximation of grammar-based compression. *Theoretical Computer Science*, 302(1-3):211–222, 2003. doi:10.1016/S0304-3975(02)00777-6.

34 Craig W Schmidt, Varshini Reddy, Haoran Zhang, Alec Alameddine, Omri Uzan, Yuval Pinter, and Chris Tanner. Tokenization is more than compression. *arXiv preprint*, 2024. doi:10.48550/arXiv.2402.18376.

35 Rico Sennrich. Neural machine translation of rare words with subword units. *arXiv preprint*, 2015. arXiv:1508.07909.

36 Yusuke Shibata, Takuya Kida, Shuichi Fukamachi, Masayuki Takeda, Ayumi Shinohara, Takeshi Shinohara, and Setsuo Arikawa. Byte pair encoding: A text compression scheme that accelerates pattern matching. *Technical Report DOI-TR-161, Department of Informatics, Kyushu University*, 1999.

37 James A Storer and Thomas G Szymanski. Data compression via textual substitution. *Journal of the ACM (JACM)*, 29(4):928–951, 1982. doi:10.1145/322344.322346.

38 Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint*, 2023. arXiv:2302.13971.

39 David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. URL: http://www.cambridge.org/de/knowledge/isbn/item5759340/?site_locale=de_DE.

40 Jiayang Wu, Wensheng Gan, Zefeng Chen, Shicheng Wan, and S Yu Philip. Multimodal large language models: A survey. In *2023 IEEE International Conference on Big Data (BigData)*, pages 2247–2256. IEEE, 2023. doi:10.1109/BIGDATA59044.2023.10386743.

41 Jingjing Xu, Hao Zhou, Chun Gan, Zaixiang Zheng, and Lei Li. 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, 2021. doi:10.18653/V1/2021.ACL-LONG.571.

42 Vilém Zouhar, Clara Meister, Juan Gastaldi, Li Du, Tim Vieira, Mrinmaya Sachan, and Ryan Cotterell. A formal perspective on byte-pair encoding. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 598–614, 2023. arXiv:2306.16837.

A APX-hardness of OPE

In § 2 we showed that the OMS problem is APX-complete. We now show the same for the the more general OPE problem (where partial merges are also allowed).

We follow the same reduction as in § 2, just adding a necessary extra step in the proof. Recall that given a cubic, undirected, unweighted graph G, we construct a string s and an integer k. We now take (s, k) to be an OPE instance. We claim that G has a cut of size at least c if and only if $\mathsf{OPT}(s, k) \geq 30n + c$, where n is the number of vertices in G.

The forward direction is identical to the proof in § 2, as the constructed OMS merge sequence also serves as a partial merge sequence for OPE.

For the reverse direction, consider a partial merge sequence $\mathcal{R}^*$ of length $k = n + 1$, of utility $30n + c$ for s. Claim 5 showed that a merge sequence $\mathcal{R}$ that is not *well-formed* can be transformed into a well-formed one, while increasing its utility. We now extend the argument to a partial merge sequence $\mathcal{R}^*$. We show that we can transform $\mathcal{R}^*$ into a *full merge sequence* (with no partial merges) that is well-formed, and the remainder of the proof in § 2 can go through unchanged.

▷ Claim 8. If $\mathcal{R}^*$ is a partial merge sequence, then we can find (in polynomial time) a well-formed (full) merge sequence $\mathcal{R}$ of the same length that achieves greater or equal utility on s.

Proof. As pairs other than ## occur at most 10 times in s, we assume $\mathcal{R}^*$ (at least partially) merges ##, for otherwise the total utility would be at most $10(n + 1) < 30n$, and any well-formed merge sequence $\mathcal{R}$ would have larger utility.

Substrings of s containing | occur at most once, and substrings of length 3 occur at most 6 times; it follows that any (partial) merge involving | or a newly created symbol has utility at most 6. Suppose there are t such partial merges in $\mathcal{R}^*$. Remove them from $\mathcal{R}^*$, and notice that all remaining merges in $\mathcal{R}^*$ can only involve pairs of the form ##, ℓ_i#, and #ℓ_i. Add to the end of $\mathcal{R}^*$, t full merges for pairs ℓ_i#, where no other merge involves ℓ_i. (As there are $n + 1$ merges initially in $\mathcal{R}^*$, and n symbols ℓ_i, there must be t such pairs.) Each added merge has utility at least 7 (one from each copy of s_i, and one from each copy of s_{ij} or s_{ij} that does not intersect another merged pair), so the total utility of $\mathcal{R}^*$ has increased.

We now only have merges of the form ##, ℓ_i#, #ℓ_i in $\mathcal{R}^*$, but some may be partial, and for some ℓ_i, we may not have any of the two merges. If there are duplicate merges in $\mathcal{R}^*$, i.e., more than one merge of the same pair, then we extend the first such merge to also perform the replacements of the other merges of the same pair. After this, we can remove the later duplicate merges (as they no longer have any utility), and replace them by some full merge ℓ_j# where no other merge involves ℓ_j.

We then move ## to be the last merge in $\mathcal{R}^*$. Notice that a partial merge sequence where all merges involve only input symbols (as the case here) can be freely re-arranged without changing utility. After we have moved ## to the end of $\mathcal{R}^*$, we can turn it into a full merge, as this can only increase the utility of $\mathcal{R}^*$.

We next fix the cases where both pairs #ℓ_i and ℓ_i# are (partially) merged in $\mathcal{R}^*$, for some i. Suppose #ℓ_i appears first (the other case is symmetric). We remove, one by one, all the replacements of ℓ_i#. In each copy of s_i there could be one such replacement in ℓ_i#ℓ_i, which we can compensate by an additional replacement of #ℓ_i. In s_{ij} (or s_{ji}) strings there can be overall at most 6 replacements of ℓ_i#, which we account as a loss. The total loss due to removing ℓ_i# is thus at most 6. Now we add instead of ℓ_i# in $\mathcal{R}^*$, a new merge ℓ_q#, where no other merge involves ℓ_q. This new merge has utility at least 7 (as argued before), without affecting other merges, and compensating the loss of 6. After doing this for each pair, we end up with $\mathcal{R}^*$ containing, for each i, exactly one of the merges #ℓ_i and ℓ_i#.

Finally, we proceed inductively through $\mathcal{R}^*$ from the end to the beginning and turn each merge into a full merge. Suppose we are at the i-th merge step and for all $j > i$, the j-th merge is full. Suppose the i-th merge is for $\#\ell_q$ (the case $\ell_q\#$ is entirely symmetric, and thus omitted). We turn $\#\ell_q$ into a full merge. Within a copy of s_q, this can only increase utility, as $\#\ell_q$ does not overlap with any other merged pair. Within some $s_{qq'}$ or $s_{q'q}$, a new replacement of $\#\ell_q$ may only interfere with a later $\#\#$ merge, but in this case we just exchange one replacement by another, leaving the total utility unchanged. At the end of the process all merges are full and we have turned $\mathcal{R}^*$ into a well-formed merge sequence that we call $\mathcal{R}$. ◁