Length-Maximization Tokenization for Language Models: An Approximation Algorithm

Anonymous Author
Anonymous Institution

Abstract

We introduce a novel tokenization approach for language models that focuses on maximizing token length through a length-maximization framework. Unlike traditional frequency-based methods such as Byte Pair Encoding (BPE), our approach employs clique partitioning techniques from graph theory to optimize tokenization. We formulate the tokenization problem as a combinatorial optimization task, aiming to partition sentences to maximize the length of shared prefixes between tokens. Due to the computational intractability of finding an exact solution, we develop an efficient approximation algorithm leveraging clique detection. Our tokenizer, Length-MAX Tokenizer, outperforms existing tokenization strategies in both tokenization efficiency and generalization across diverse text corpora. Extensive evaluations on large-scale datasets demonstrate improvements in tokenization quality, model inference efficiency, and language model performance. These results suggest that our approach is a promising alternative for future language model applications.

1 Introduction

Tokenization is a critical preprocessing step in natural language NLP, where text is segmented into smaller units called tokens [15]. The choice of tokenizer significantly impacts both the efficiency and performance of NLP models, particularly in terms of computational resource utilization and the handling of long-term dependencies in sequences [17, 26]. Traditional tokenization methods, such as BPE [28], iteratively merge the most frequent pairs of symbols to reduce vocabulary size. While effective in reducing the number of unique tokens, BPE and similar frequency-based methods often produce a large number of short tokens [23]. This can lead to increased sequence lengths, higher computational costs, and potential degradation in model performance due to elongated dependency chains [7, 22].

Figure 1: Length-MAX Tokenizer, aiming to represent text with fewer tokens, maximizing the average token length.

In this work, we introduce the *Length-MAX Tokenizer*, a novel tokenization algorithm designed to maximize the average token length, thereby reducing the number of tokens required to represent text. Our approach balances token frequency and length by selecting tokens based on the product of their frequency and length. This strategy aims to create longer, more meaningful tokens that can improve model efficiency and performance [10].

We formalize the tokenization task as a combinatorial optimization problem, seeking a token vocabulary of fixed size that maximizes the expected token length across the corpus. Specifically, we aim to find tokens that are common prefixes among sentences, allowing us to cover more text with fewer tokens. The optimization problem is proven to be NP-hard [13], making exact solutions computationally infeasible for large datasets. To address this challenge, we develop an efficient approximation algorithm based on graph

Preliminary work. Under review by AISTATS 2025. Do not distribute.

partitioning techniques, leveraging clique detection to identify groups of sentences that share long common prefixes [21].

Our main contributions are as follows:

- We propose the Length-MAX Tokenizer, a new method that maximizes average token length to improve tokenization efficiency.
- We formalize the tokenization problem as a combinatorial optimization task and prove its NP-hardness.
- We design an efficient approximation algorithm utilizing graph partitioning and clique detection to generate the token vocabulary.
- We conduct extensive experiments comparing our tokenizer with popular methods such as BPE [28], WordPiece [27], and SentencePiece [17], demonstrating that our approach reduces the total number of tokens needed to represent text and improves model performance in preliminary experiments with GPT-2 [24].

2 Related Work

2.1 Universal Tokenizer

Tokenization is a fundamental process in NLP that involves segmenting text into tokens, the smallest units meaningful for processing [27]. Traditional tokenization methods often rely on language-specific rules [12], which can pose challenges such as handling vocabulary size limitations and unknown words, especially in morphologically rich languages [19].

Byte Pair Encoding [28] is an unsupervised subword segmentation algorithm that starts with a base vocabulary of individual characters and iteratively merges the most frequent pairs of symbols to form new subword units. This method effectively handles out-of-vocabulary words by representing them as sequences of subword units, enhancing models' ability to generalize to unseen words [23].

WordPiece [27] was initially developed for Japanese and Korean voice search. Similar to BPE, WordPiece constructs a vocabulary of subword units by iteratively merging symbol pairs. However, it uses a probabilistic approach to select merges that maximize the likelihood of the training data, resulting in a vocabulary that captures the most salient subword patterns [27].

SentencePiece [17] is a language-independent subword tokenizer and detokenizer. Unlike BPE and WordPiece, SentencePiece treats the input text as a raw sequence without predefined token boundaries, enabling it to model languages lacking clear word segmentation (e.g., Japanese and Chinese). It implements subword segmentation using either BPE or unigram language models within a unified framework.

Tokenizer-Free Models Recent advancements have explored tokenizer-free approaches to address the limitations of traditional tokenization. [8] proposed a method that bridges the gap between character-level and subword-level tokenization, allowing models to process raw text without explicit tokenization. Models like **ByT5** [33] and **CANINE** [10] operate directly on byte sequences, eliminating the need for tokenization and enabling more uniform processing across languages. [30] introduced **Charformer**, combining the efficiency of subword tokenization with the expressiveness of character-level models through gradient-based subword tokenization.

2.2 Impact of Tokenization

Tokenization methods significantly influence the performance of language models by affecting the representation of input text. Factors such as average token length [31] and whether tokens cross word boundaries [25] play crucial roles in determining model efficiency and effectiveness.

Subword-level tokenization methods, such as BPE, WordPiece, and SentencePiece, produce tokens representing common subword units that may cross word boundaries. [32] analyzed the impact of different tokenization strategies on Turkish language models, highlighting that average token length and token granularity significantly affect model performance. They found that subword tokenization balances the trade-off between handling rare words and maintaining manageable sequence lengths.

Furthermore, tokenization strategies influence vocabulary size and the representation of rare or OOV words [12]. Fine-grained tokenization methods may lead to smaller vocabularies but longer sequences, while coarse-grained methods result in larger vocabularies and shorter sequences. Choosing an optimal tokenization strategy involves considering specific language characteristics and the target application [3].

The choice of tokenizer can significantly impact the training and performance of LLMs. [1] investigated tokenizer choice for LLM training, emphasizing that it can be crucial rather than negligible. Similarly, [26] examined the monolingual performance of multilingual

language models, highlighting the importance of tokenizer quality.

Understanding the impact of tokenization on factors such as average token length and the use of tokens that cross word boundaries is essential for optimizing language model performance. Research has demonstrated that token length and granularity significantly influence model efficiency and effectiveness [32, 2]. In designing tokenizers that aim to maximize token length, such as our proposed `length_max` tokenizer, longer tokens can capture more semantic information and potentially reduce the overall sequence length, thereby improving computational efficiency [6, 1]. By tailoring the tokenization approach to the specific characteristics of a language and the requirements of a task, we can enhance model performance and efficiency [29].

3 Methods

3.1 Clique Partitioning Tokenization

We propose a novel tokenization method called *Clique Partitioning*, which leverages graph theory concepts to partition a corpus into optimally related subsets corresponding to tokens. Unlike traditional approaches such as BPE, our method constructs a graph where nodes represent characters, and edges represent relationships based on character co-occurrence in the corpus. The goal is to partition the graph into a minimal number of cliques, selecting the largest covering clique as the token in each iteration.

3.1.1 Greedy Algorithm

Let S denote the set of sentences in the corpus, and let $T = \{t_1, t_2, \ldots, t_m\}$ be the current set of tokens, initially consisting of single characters. Each token t_l is associated with a subset of sentences $V_l \subset S$ containing t_l.

Our method employs a greedy algorithm to iteratively refine the token set T by introducing new tokens that improve tokenization efficiency. At each iteration, we aim to find a new token t_{m+1} by identifying a subset $V_l' \subset V_l$ that maximizes the objective function:

$$F(V_l') = |V_l'| \times \min_{s_i, s_j \in V_l'} E_{ij}$$

where E_{ij} measures the relatedness between sentences s_i and s_j. This function balances the coverage of the token and the strength of relationships among sentences containing it.

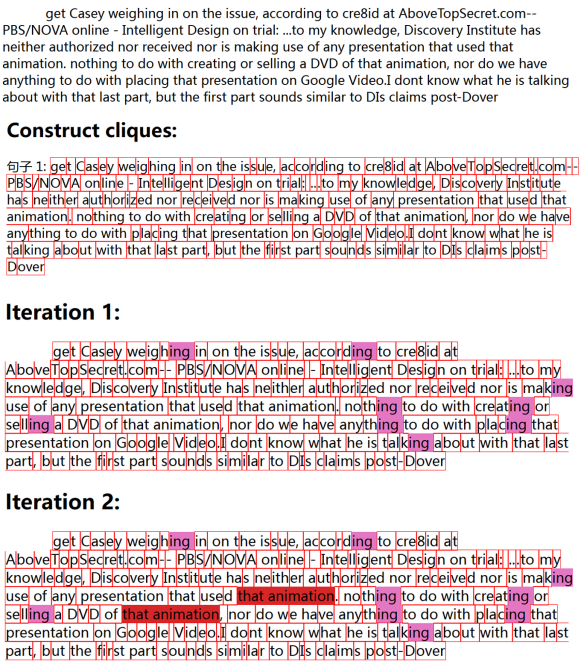

Figure 2: Clique Partitioning Motivation: Each character is a node; the corpus is partitioned into cliques, and the largest clique is selected as the token at each iteration

3.1.2 Problem Approximation

Solving the optimization problem directly is computationally intractable due to its combinatorial nature. To approximate the solution, we leverage graph-based methods by exploiting properties of the relatedness measure E_{ij} (e.g., symmetry and triangle inequality).

We construct a graph where nodes represent sentences, and edges connect pairs of sentences with relatedness above a threshold e. Finding the largest clique in this graph corresponds to identifying the largest subset V_l' where sentences are strongly related. This reduces the problem to the well-studied clique-finding problem in graph theory [21].

Algorithms for finding cliques have been extensively studied [11, 4], and techniques from token graphs [11] and token sliding problems [5] provide valuable insights for our approximation method.

3.1.3 Limitations of the Greedy Algorithm

While the greedy algorithm is efficient, it does not guarantee an optimal tokenization. We illustrate this with examples showing cases where the algorithm fails to find the globally optimal partition, particularly as the number of tokens K increases.

Example with $K = 3$ Consider a set of sentences S partitioned into four subsets:

$$S = S_1 \cup S_2 \cup S_3 \cup S_4$$

where:

- $S_1 = \{s \in S \mid s \text{ contains } t_1\}$
- $S_2 = \{s \in S \mid s \text{ contains } t_2\}$
- $S_3 = \{s \in S \mid s \text{ contains } t_3 \text{ but not } t_3t_4\}$
- $S_4 = \{s \in S \mid s \text{ contains } t_3t_4\}$

Assume:

- $|S_=|S_2| = 2|S_3| = 2|S_4| = m$
- Tokens t_1, t_2, t_3 are distinct and non-overlapping.
- Token lengths satisfy $|t_1| = |t_2| = |t_3| = l$, and $|t_4| = 1.1l$.

The optimal tokenization for $K = 3$ uses tokens $\{t_1, t_2, t_3\}$, grouping S_3 and S_4 together. However, the greedy algorithm may prematurely create a token for t_4, resulting in suboptimal partitions such as:

- $\{S_4,\ S_1 \cup S_3,\ S_2\}$
- $\{S_4,\ S_2 \cup S_3,\ S_1\}$

This occurs because the algorithm focuses on local optimization, missing the globally optimal combination of S_3 and S_4.

Example with $K = 4$ Similarly, consider S partitioned into five subsets:

$$S = S_1 \cup S_2 \cup S_3 \cup S_4 \cup S_5$$

with:

- $S_1 = \{s \in S \mid s \text{ contains } t_1\}$
- $S_2 = \{s \in S \mid s \text{ contains } t_1t_2\}$
- $S_3 = \{s \in S \mid s \text{ contains } t_3\}$
- $S_4 = \{s \in S \mid s \text{ contains } t_4\}$
- $S_5 = \{s \in S \mid s \text{ contains } t_5\}$

All tokens t_i have equal length, and subsets S_i are of equal size. The optimal tokenization combines S_1 and S_2 under a single token, but the greedy algorithm may fail to find this, instead grouping S_4 and S_5 together.

3.1.4 Algorithm Enhancements

To mitigate the limitations of the greedy approach, we propose iterative refinement strategies.

Sequential Improvement We iteratively examine pairs of subsets in the current partition. For each pair (S_i, S_j), we consider merging them and finding an optimal re-partitioning. If this yields a better partition, we update the subsets accordingly. This process helps the algorithm escape local optima and move toward a global optimum [34, 16].

Preprocessing for Homogeneity Recognizing that sentences can be heterogeneous, we introduce a preprocessing step to promote homogeneity:

1. Begin tokenization from the start of each sentence.
2. Apply longest-token matching multiple times to capture significant patterns.
3. Limit iterations to prevent over-segmentation.
4. Apply the Clique Partitioning algorithm to the preprocessed corpus to update the token vocabulary.

By extracting longer, informative tokens early, we increase the likelihood of identifying cohesive subsets, enhancing the overall tokenization quality.

4 Experiment

4.1 Implementation

4.1.1 Algorithm for new token

Common tokenization methods typically follow these steps: 1. Prepare the training corpus 2. Extract an initial vocabulary from the corpus, usually consisting of all the individual characters, and tokenize the corpus into tokens using the initial vocabulary 3. Iteratively find and merge the most frequent adjacent token pairs 4. End the iteration based on a predefined vocabulary size [28, 17, 27].

In all, the process is driven by frequency of token pairs.

In contrast to frequency-driven methods [6, 23], our method, utilizing relationships between words, focuses more on maximizing coverage of the new token in the corpus, presuming maximizing the average token length in the tokenized corpus. The algorithm can be summarized as follows:

Algorithm 1 Clique Partition Tokenizer

Require: Training corpus S, desired vocabulary size K
Ensure: Token vocabulary T
Initialization:
Initialize token vocabulary T with all unique characters in S
while $|T| < K$ **do**
  Construct graph G**:**
  Nodes represent tokens in the current vocabulary T
  For each sentence $s \in S$:
  for all tokens t_i in s **do**
    for all tokens t_j following t_i in s **do**
      Add edge between nodes t_i and t_j in G
    end for
  end for
  Identify independent cliques in G
  Find the largest clique Q in G
  Create new token:
  $t_{\text{new}} \leftarrow$ concatenation of tokens in the prefix of Q
  Add t_{new} to vocabulary T
  Update corpus S**:**
  for all occurrences of tokens forming Q in S **do**
    Replace sequential tokens in Q with t_{new}
  end for
  Update graph G**:**
  Replace nodes corresponding to Q with node t_{new}
  Reconstruct G, causing the largest clique to partition into two
end whilereturn Token vocabulary T

We modified an open-sourced tokenization tool by HuggingFace to implement our algorithm. Core implementations include the construction of the token relation graph and updating the graph by partitioning the largest clique. Algorithms of the two methods are shown below, and code can be found at `https://github.com/DWinnter/tokenizers_clique_partition`.

Now that the algorithm for finding the new token is clear, we can start to use it to train our tokenizer. Following the common practice, we will start from the vocabulary containing only single characters, and iteratively find the new token with the maximum coverage in the whole corpus.

4.1.2 Cross-Word

We observed that current tokenization methods like BPE, WordPiece, and SentencePiece employ pre-tokenization [18, 26], treating words as independent and ignoring inter-word relationships [29]. This approach can miss obvious multi-word sequences [6, 1]; for instance, it might not recognize that `the United States` is a frequent and meaningful phrase that should be treated as a single token rather than three separate tokens. Our method avoids pre-tokenization, training directly on the raw corpus to maximize average token length. This enables capturing common phrases as single tokens, leading to more coherent and semantically meaningful tokens [9].

To merge tokens for maximal corpus coverage, we use a length-oriented matching pattern:

$$\text{Regex} = \texttt{token}_1|\texttt{token}_2|\ldots|\texttt{token}_n \tag{1}$$

where $\texttt{token}_i$ is the i-th token in the vocabulary, ordered by decreasing length ($|\texttt{token}_i| \geq |\texttt{token}_{i+1}|$). Each time a new token is added, we re-tokenize the corpus using the updated vocabulary to maximize token coverage, ensuring the token table remains aligned with the corpus [12].

4.2 Corpus

The corpus used to train the model and tokenizer is FineWeb(10B), a large-scale web crawl dataset [20]. This dataset contains approximately 10 billion words (text size $\approx$ 50GB) of high-quality English web content, and covers a wide range of topics and writing styles like normal chats, news, medical reports, and Shakespearean poems. This dataset was chosen for its size, quality, and diversity, which are crucial factors in training a robust language model and developing an effective tokenization method [35].

Resources of this dataset can be found at `https://huggingface.co/datasets/HuggingFaceFW/fineweb`.

4.3 Evaluation Metric

4.3.1 Tokenize Ratio and Generalization

Tokenize Ratio A ratio is widely used in tokenization evaluation [9], which is defined as:

$$\text{Tokenize Ratio} = \frac{\text{number of tokens}}{\text{number of words}}$$

A lower ratio indicates that the method is able to find longer subword units, which is thought to lead to more efficient processing. We not only trained our clique partition tokenizer on the training corpus, but also re-trained other current popular tokenizers, including SentencePiece [17], BPE [28], and WordPiece [27], on the same corpus to compare effectiveness of algorithm.

Generalization on Diverse Text To evaluate the generalization capability of our tokenization method, we conducted experiments on various datasets outside the training corpus. This assessment helps us understand how well our method performs on unseen text and different domains. We used the following datasets for this evaluation, each of which is a collection of text from sources different from the training corpus:

- News articles from CNN
- Scientific papers in computer science
- Professional content in the medical field
- Daily conversations from Reddit
- Shakespeare's works

For each dataset, we computed the tokenize ratio and compared it to that of other popular tokenizers. The result shown in figure 7 indicates that our method, Clique, outperforms other tokenizers by 15.26% on average tokenize ratio on the training corpus.

Testing on multi-domain test corpora different from the training corpus also demonstrates that the vocabulary trained by our Clique method has good generalization across different tasks, outperforming the best among popular methods by 16.31% on daily conversations, 7.97% on medical records, 16.31% on news reports, 1.74% on computer science papers, and 7.58% on Shakespeare's poems. Results are shown in graph 4.

The reason why performance on CS papers is not as good as other corpora is that the GPT-2 corpus consists mainly of news reports and plain English text, lacking in LaTeX-style coding corpus [20]. But our clique partition method still outperforms others.

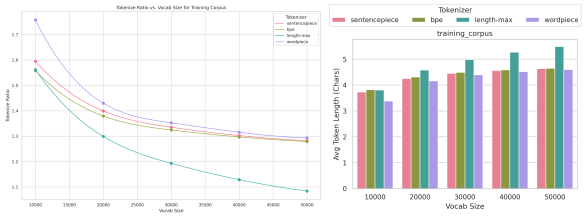

Figure 3: Tokenize Ratios Across Different Tokenizers and Vocabulary Sizes on FineWeb10B Corpus [20], the Corpus Used to Train GPT-2 and Tokenizer

Modifications to the Original Algorithm In experiments, we kept seeking further improvement of the performance of our Length-Max tokenization method, we implemented two modifications to the original algorithm.

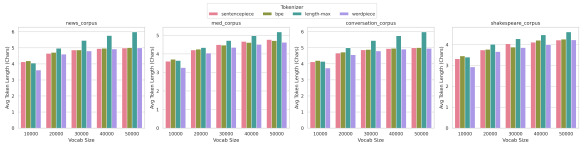

Figure 4: Average Token Length (chars), reciprocal to tokenize ratio Across Different Tokenizers and Vocabulary Sizes on Different Corpora, Types Varied

- Modification 1 (Iterate Beyond Beginnings):

  Upon analyzing the corpus, we found that sentence beginnings often contain repetitive patterns (e.g., common phrases in news articles or standard greetings in conversations). During the clique partitioning process, these repetitive tokens can dominate the initial token output, potentially overshadowing more important content from the main body of the text. To address this issue, Modification 1 involves delaying the token output for the initial iterations (specifically, for 10% of the target vocabulary size). By postponing tokenization of these frequent beginning patterns, we aim to focus on capturing more significant information from the main content, thereby improving overall token utilization.

- Modification 2 (Merge Split Cliques):

Figure 5: Reconstruct the graph by merging split cliques

When a maximum clique is obtained, that part of characters is seen as one entire token. How-

ever, the rest part of the corpus is partitioned in situation where the largest-covering clique is regarded as separate nodes. A reconstruction of the partitioned corpus may lead to larger cliques as shown in Figure 5, in which case more characters are involved in the clique of next ineration. By re-partition at every iteration, we ensure that all parts of the text are greedily processed, enhancing the tokenizer's coverage.

After implementing these modifications, we conducted experiments to compare their performance with the raw clique partition method. The results are shown in Figure 6. The gap between the modified methods and the original method widens as the vocabulary size grows, suggesting that the modifications become more effective with larger vocabularies [2].

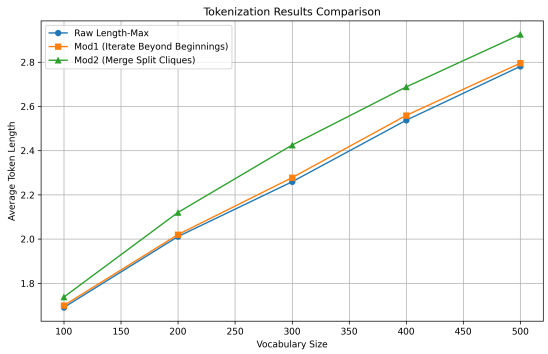

Figure 6: A small scale comparison of Average Token Length for the modifications

The improvements in average token length suggest that these modified methods are better at capturing more meaningful linguistic units and greedy searching, which may lead to more efficient text representation for downstream tasks [9, 32].

4.3.2 Language Modeling Task

The value of a tokenization method can't be evaluated without considering its impact on language modeling tasks [10, 33, 1, 6]. Due to the changes in the token vocabulary, we needed to train a completely new model from scratch, as previous models are not compatible with different tokenization systems [25, 14]. We chose the largest open-source model for training from scratch currently available: GPT-2, for our experiments. Given computational resource constraints, we retrained the 124M parameter version of GPT-2.

We followed the standard procedure of training GPT-2 on the corpus and evaluating its performance on a downstream task. Presumably, a higher tokenize ratio, which means longer average token length, will guarantee more efficient model inference, especially for long texts [9, 32]. Thus we set efficiency as one of the metrics, and accuracy as another to see whether performance of the model is influenced.

Efficiency To evaluate the efficiency of the model trained on our token vocabulary, we measured the time taken by our model and a typical BPE model to finish a set number of inferences. An inference is finished when the model generates the `<|end-of-text|>` token or hits the max length of the sequence, which is the same as normal practice by OpenAI [20].

Interesting results are shown in Figure 7. We can first observe that each model reveals a fitting straight line between their output length (chars) and time taken (s), which essentially represents the average number of characters per token. Since the model structures of both methods are identical, the speed at which they output tokens should theoretically be the same. Specific token counts and other data can be found in the appendix. The clique partition model has a smaller slope in its time taken (s) / output length (chars), indicating that its average token length is longer, which is advantageous for generating longer texts at the same cost [6, 1]. Another interesting point is that, for the typical GPT-2 model, the scatter distribution of its output concentrates more below the upper limit of output token numbers (13.10% of the output hit the limit), while our clique partition model more frequently hits the limit (57.14% of the output hit the limit). This suggests that our model is less inclined to output the `<|end-of-text|>` token to terminate the inference. We believe this is likely related to changes in the token probability distribution of the training corpus brought about by the new tokenization method [29]. In simple terms, our clique partition method makes the distribution of tokens with different IDs more uniform; details can be found in Chapter 5 Discussion.

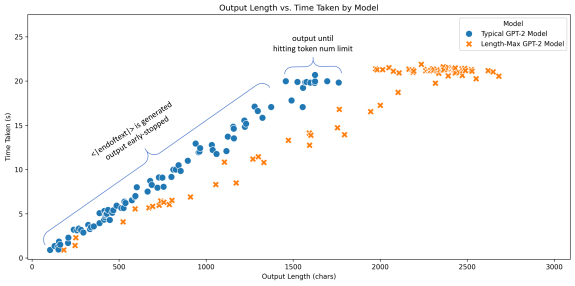

Figure 7: Output Length vs Time Taken for GPT-2 Trained on Length-MAX Tokenizer and BPE Tokenizer

Accuracy To evaluate the accuracy of the model trained on our vocabulary, we tested perplexity and correctness of our model and a typical BPE model on a widely used benchmark: HellaSwag. HellaSwag is a benchmark for evaluating the quality of language models' commonsense reasoning abilities [35]. It consists of 200,000+ diverse and realistic multi-sentence choices for 10,000+ questions about common sense. Data can be found at `https://github.com/rowanz/hellaswag`. We used the same set of questions for both models to ensure a fair comparison.

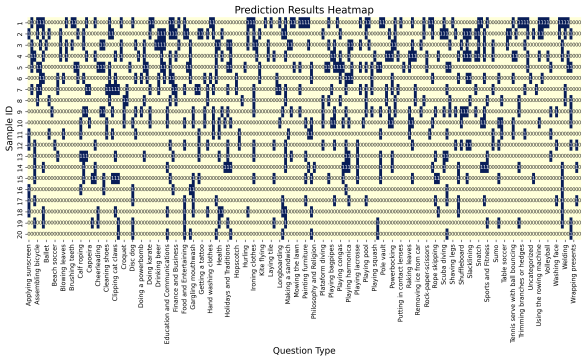

Figure 8: Length-MAX Tokenizer GPT-2 Prediction

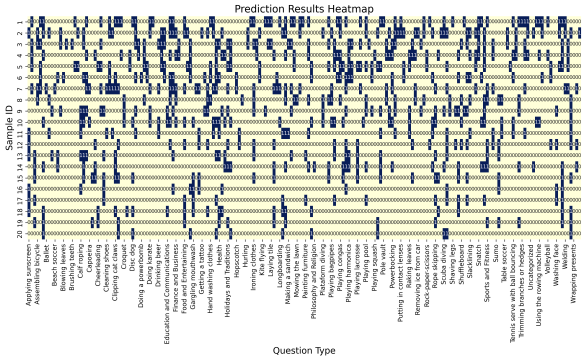

Figure 9: BPE Tokenizer GPT-2 Prediction

Due to the limitation of computational resources, we could only train GPT-2 124M on our tokenizer. Thus the performance comparison between the two tokenization method models is not quite fair [1]. Both of them perform poorly in the HellaSwag benchmark; the typical BPE GPT-2 scores 28.3% and the Length-MAX GPT-2 scores 27.5%. We wished to test and prove that the new tokenization method won't undermine performance on accuracy. Distribution of their prediction correctness is shown in Figure 8, each block represents a question, and the question set is the same for both.

5 Discussion

5.1 Token Distribution

A very interesting finding is that our cross-word clique partition method leads to a more uniform distribution of token probabilities. From Figure 10, we can see that at a vocabulary size of 50,000 (the standard GPT-2 vocabulary size), other methods have tokens with particularly high frequencies. The variance of the top 50 token frequencies of the three other methods are: **SentencePiece Tokenizer** [17], **BPE Tokenizer** [28], and **WordPiece Tokenizer** [27] with variances of 0.000082, 0.000087, and 0.000085 respectively. In contrast, our clique partition method not only reduces the frequency of the most frequent tokens but also results in the probability distribution of low-frequency tokens differing little. The variance of the top 50 token frequencies of the clique partition method is **0.000001**. Moreover, this trend of the probability distribution becoming more uniform becomes increasingly significant as the vocabulary size increases (see Figure 11). From the performance of GPT-2, we cannot yet discern what impact this averaging of the probability distribution has on large language models. Only when sufficient computational resources are available can we test its performance on larger-scale models [29, 6, 32].

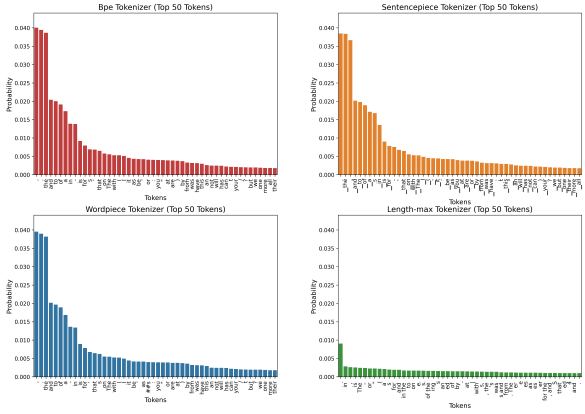

Figure 10: Top 20 token frequencies of the four tokenizers at a vocabulary size of 50,000

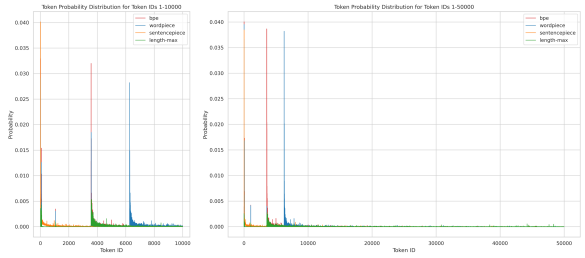

Figure 11: Token distribution of the four tokenizers from a vocabulary size of 10,000 to 50,000

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 arXiv:2310.08754*, 2023.

[2] Mohamed Taher Alrefaie, Nour Eldin Morsy, and Nada Samir. Exploring tokenization strategies and vocabulary sizes for enhanced arabic language models. *arXiv preprint arXiv:2403.11130*, 2024.

[3] Zaid Alyafeai, Maged S Al-shaibani, Mustafa Ghaleb, and Irfan Ahmad. Evaluating various tokenizers for arabic text classification. *Neural Processing Letters*, 55(3):2911–2933, 2023.

[4] Ramla Belalta, Mouhoub Belazzoug, and Farid Meziane. A graph based named entity disambiguation using clique partitioning and semantic relatedness. *Data & Knowledge Engineering*, 152:102308, 2024.

[5] Rémy Belmonte, Eun Jung Kim, Michael Lampis, Valia Mitsou, Yota Otachi, and Florian Sikora. Token sliding on split graphs. *Theory of Computing Systems*, 65:662–686, 2021.

[6] Yekun Chai, Yewei Fang, Qiwei Peng, and Xuhong Li. Tokenization falling short: The curse of tokenization. *arXiv preprint arXiv:2406.11687*, 2024.

[7] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[8] Dokook Choe, Rami Al-Rfou, Mandy Guo, Heeyoung Lee, and Noah Constant. Bridging the gap for tokenizer-free language models. *arXiv preprint arXiv:1908.10322*, 2019.

[9] Sanghyun Choo and Wonjoon Kim. A study on the evaluation of tokenizer performance in natural language processing. *Applied Artificial Intelligence*, 37(1):2175112, 2023.

[10] Jonathan H Clark, Dan Garrette, Iulia Turc, and John Wieting. Canine: Pre-training an efficient tokenization-free encoder for language representation. *Transactions of the Association for Computational Linguistics*, 10:73–91, 2022.

[11] Ruy Fabila-Monroy, David Flores-Peñaloza, Clemens Huemer, Ferran Hurtado, Jorge Urrutia, and David R Wood. Token graphs. *Graphs and Combinatorics*, 28:365–380, 2012.

[12] Murhaf Fares, Stephan Oepen, and Yi Zhang. Machine learning for high-quality tokenization replicating variable tokenization schemes. In *Computational Linguistics and Intelligent Text Processing: 14th International Conference, CICLing 2013, Samos, Greece, March 24-30, 2013, Proceedings, Part I 14*, pages 231–244. Springer, 2013.

[13] Michael R Garey and David S Johnson. *Computers and intractability*, volume 174. freeman San Francisco, 1979.

[14] Ying He and Mehmet Kayaalp. A comparison of 13 tokenizers on medline. *Bethesda, MD: The Lister Hill National Center for Biomedical Communications*, 48, 2006.

[15] Daniel Jurafsky and James H Martin. Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition.

[16] Jongik Kim and Hongrae Lee. Efficient exact similarity searches using multiple token orderings. In *2012 IEEE 28th International Conference on Data Engineering*, pages 822–833. IEEE, 2012.

[17] Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. *arXiv preprint arXiv:1804.10959*, 2018.

[18] Chanjun Park, Sugyeong Eo, Hyeonseok Moon, and Heui-Seok Lim. Should we find another model?: Improving neural machine translation performance with one-piece tokenization method without model modification. In *Proceedings of the 2021 conference of the north american chapter of the association for computational linguistics: human language technologies: Industry papers*, pages 97–104, 2021.

[19] Kyubyong Park. An empirical study of tokenization strategies for various korean nlp tasks. *arXiv preprint arXiv:2010.02534*, 2020.

[20] Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, Thomas Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *arXiv preprint arXiv:2406.17557*, 2024.

[21] Charles A Phillips, Kai Wang, Erich J Baker, Jason A Bubier, Elissa J Chesler, and Michael A

Langston. On finding and enumerating maximal and maximum k-partite cliques in k-partite graphs. *Algorithms*, 12(1):23, 2019.

[22] Ofir Press, Noah A Smith, and Mike Lewis. Shortformer: Better language modeling using shorter inputs. *arXiv preprint arXiv:2012.15832*, 2020.

[23] Ivan Provilkov, Dmitrii Emelianenko, and Elena Voita. Bpe-dropout: Simple and effective subword regularization. *arXiv preprint arXiv:1910.13267*, 2019.

[24] 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.

[25] Eugénio Ribeiro, Ricardo Ribeiro, and David Martins de Matos. A study on dialog act recognition using character-level tokenization. In *International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, pages 93–103. Springer, 2018.

[26] Phillip Rust, Jonas Pfeiffer, Ivan Vulić, Sebastian Ruder, and Iryna Gurevych. How good is your tokenizer? on the monolingual performance of multilingual language models. *arXiv preprint arXiv:2012.15613*, 2020.

[27] Mike Schuster and Kaisuke Nakajima. Japanese and korean voice search. In *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5149–5152. IEEE, 2012.

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

[29] Aaditya K Singh and DJ Strouse. Tokenization counts: the impact of tokenization on arithmetic in frontier llms. *arXiv preprint arXiv:2402.14903*, 2024.

[30] Yi Tay, Vinh Q Tran, Sebastian Ruder, Jai Gupta, Hyung Won Chung, Dara Bahri, Zhen Qin, Simon Baumgartner, Cong Yu, and Donald Metzler. Charformer: Fast character transformers via gradient-based subword tokenization. *arXiv preprint arXiv:2106.12672*, 2021.

[31] Christos Theodoropoulos and Marie-Francine Moens. An information extraction study: Take in mind the tokenization! In *Conference of the European Society for Fuzzy Logic and Technology*, pages 593–606. Springer, 2023.

[32] Cagri Toraman, Eyup Halit Yilmaz, Furkan Şahinuç, and Oguzhan Ozcelik. Impact of tokenization on language models: An analysis for turkish. *ACM Transactions on Asian and Low-Resource Language Information Processing*, 22(4):1–21, 2023.

[33] Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. Byt5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306, 2022.

[34] Gaku Yasui, Kouta Abe, Katsuhisa Yamanaka, and Takashi Hirayama. Swapping labeled tokens on complete split graphs. *Inf. Process. Soc. Japan. SIG Tech. Rep*, 14:1–4, 2015.

[35] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.