Analyzing the Information Density of Various Tokenizations for the Optimization of Natural Language Processing Models

Riya Bhatia, Angela Yuan, Shivam Syal, Komal Keesara, Tawshia Choudhary, Dmitri Pavlichin

Abstract

Tokenization, a pre-processing step for Natural Language Processing (NLP) models, involves the breaking down of a text into smaller strings to build context for the model. There has been a range of tokenizers and tokenization methods designed to improve NLP model efficiency; however, there has been little research examining which tokenization methods are effective for various categories of inputs. Users are required to pay per token to utilize large-scale NLP models, and advice on which tokenizers to utilize can ultimately allow them to save money.

In this study, we aim to determine tokenization methods that are effective for different languages, focusing on texts written in languages that are more morphologically complex than English. To accomplish this, two metrics of quantification are used: tokenization length, which is the total number of tokens, and the token dictionary size, which is the number of distinct tokens that occur in the tokenized text. Seven tokenization methods were tested specifically, with three being pre-trained tokenizers trained on the English corpus, three tokenizers that were trained on multilingual data, and the last being byte-pair encoding. Then, two distinct categories of languages were analyzed to determine their information density and repeatability, and the best NLP model for multilingual language processing was determined as the M2M100 tokenizer when tested on six languages. For pre-trained tokenizers trained on the English language, it was discovered that the BERT tokenizer was the most effective when evaluated with the two metrics mentioned. We further analyzed genomes, where the n-grams are comparable to the tokens. The number of n-grams is compared to the length of the sequence to measure repeatability.

Keywords: Natural Language Processing, optimization, byte-pair encoding, tokenization, genomes, pre-trained tokenizers, multilingual, repetitiveness

1. Introduction

NLP is a subcategory of Artificial Intelligence that involves the understanding of natural language by computers, combining the computer science and linguistics fields. While understanding a language is intuitive for humans, it is a complicated task for computers to comprehend due to varying syntax and word choice. [1] NLP is a broad field but spans many areas of high interest, including machine translation, human-computer interaction, and sentiment analysis. NLP is also utilized to aid the hearing-impaired and those that have learning disabilities in communication and understanding. Generally, the input into NLP models is text. However, before texts can be fed into larger models, such as BERT and GPT-2, they need to be processed.

As so, tokenization is a valuable pre-processing step in NLP for developing more efficient NLP models. Tokenization is a method to process text, involving breaking down text inputs into

smaller strings. These can include individual characters, smaller chunks of text, or entire words. Machine learning models accept numerical inputs and tokenization converts text into a sequence of numbers, where the integer-valued tokens correspond to a vocabulary of strings. Tokenization aids computers in understanding the underlying definition of the set of words or sentences. For instance, the sentence, “It is sunny” can be broken down into [“It,” “ is,” “ sunny”] based on word tokenization. However, words can be broken down into further chunks as well, as in the word, “lowest,” being broken into the strings, [“low”, “est”] through subword tokenization. In general, texts can be broken up at the character level, subword level, or word level.

“Hello there!” → [“H”, “e”, “l”, “l”, “o”, “ “, “t”, “h”, “e”, “r”, “e”, “!”]
“Hello there!” → [“Hello”, “ “, “there”, “!”]
“Hello there!” → [“He”, “el”, “ll”, “lo”, “o “, “ t”, “th”, “he”, “er”, “re”, “e!”]

Figure 1. Three tokenizations are seen, with the first being character tokens, the second being the GPT-2 tokenization, and the third being that every pair of consecutive bytes is a token.

In Figure 1, the first example highlights the scenario where each letter is its own token. The tokenization length and the distinct number of tokens are both long; there are 12 total tokens and 8 distinct tokens. In the next example, a sample GPT-2 tokenizer output is seen, where each word is its own token, besides the symbols. Because GPT-2 is trained to capture patterns in natural English, like many other pre-trained tokenizers in the English dictionary, it recognizes the words in the input string. The tokenization length and token dictionary size are the smallest here, with both being four. Further, in the last example, we separate the string into all possible 2-grams, where each pair of consecutive bytes is its own token. Here, the tokenization length is 11, but the dictionary size is especially large, as the pairs do not repeat frequently. This is because the total number of combinations of letters in English when a text is broken into n-grams is:

$$26^n$$

where n is the number of letters per token in the tokenization. For a 2-gram, there are $26^2 = 676$ combinations that can appear, while for a 1-gram (where every character is a token), there are simply 26 possibilities (excluding symbols), so the probability that tokens are repeated is higher for a text broken into 1-grams than a 2-grams.

As mentioned above, tokenizers are trained to capture patterns in natural English. Thus, words displayed in different languages will result in longer tokenizations, as the tokenizer will be unable to recognize such inputs and thus, will break the input into further sub-strings.

Broadly, a token dictionary is a collection of strings that can tile a text, but a good token dictionary should meet the following criteria: 1) short tokenization length, and 2) small vocabulary size. [1] The first lowers the distinct number of tokens, which increases the speed of applying a model. The second leads to a smaller model, given that a larger vocabulary would result in more model parameters. This factor allows for faster, cheaper model evaluation.

2. Methods

2.1 N-Gram Analysis for Genomes

In our analysis, we viewed n-grams for genomes as tokens. Our goal was to determine if different genomes differ in their repetitiveness by the number of distinct tokens.

To analyze genome repeatability, we compared the number of distinct n-grams (strings of length n) to the length of the sequence scanned. We began by examining. After downloading these as FASTA files, we filtered out the line breaks, initial text descriptions, and capitalization so that we could solely analyze the base pairs.

We used Python and Jupyter Notebook to plot the number of distinct n-grams in the base pair versus the length of the genome that has been examined. The first algorithm that we created precisely displays these two values on the x-axis and y-axis, respectively. With this method, some of the downloaded genomes had larger file sizes as imported into the Jupyter Notebook, such as for Aaosphaeria arxii. Thus, we examined both the entire and partial genome lengths in their base pair count.

After, we modified this initial algorithm to better distinctly display the data. This new algorithm returns the fraction of non-unique 8-grams. With NumPy and Matplotlib, we created a superimposed plot of the 6 genomes. It shows the increasing repetition found in the 8-grams of each genome sequence. We mainly analyzed the first 90,000 base pairs to focus on the sections of the sequence that display the most change in the slope of the plot.

For example, if a sequence only consists of A's and n = 3, then the only 3-gram seen would be "AAA," and the number of distinct 3-grams would always equal 1. If a genome sequence is ACGACGACGACGACG and n = 3, then the following would be computed:

# of distinct n-grams	Base pair distance from the beginning of the genome
1	3 (ACG)
2	4 (ACG, CGA)
3	5 (ACG, CGA, GAC)
3	6 (same 3-gram above keeps repeating onwards)
3	7

Above, this demonstrates some factors tracked in the repeatability analysis, showing that genomes eventually reach the maximum number of distinct n-grams. Plots begin to have smaller slopes and flatten. Here, at 5 base pairs, the genome has reached all possible 3-grams.

We defined "n_non_unique" to evaluate the number of 8-grams in the genome sequence that repeats at least once, along with "n_unique" to compute the number of 8-grams that occur only once. Therefore, dividing the former by the sum of the two computes the fraction of non-unique, or repeated, 8-grams. This represents dividing the accumulated number of repeated 8-grams by the total number of 8-grams at the scanned sequence length. We appended this to our Python dictionary using this equation:

(n_non_unique) / (n_non_unique + n_unique)

These values are shown on the superimposed plot, using logarithmic axes. A smaller-valued input occurs when there is less repeatability in the genome base pair based on the equation, whereas a greater number indicates more repeatability.

2.2 Analysis of Various Natural Languages

2.2.1 Byte-Pair Encoding

In information theory, Byte-Pair Encoding (BPE) is a method of data compression that iteratively replaces repeated pairs of consecutive bytes with a byte that does not occur in the data. BPE is also used in NLP models including GPT-2 and Transformer models for tokenization. [9] An example is shown in Figure 2.

Original string: “hello hello there”

[“h”, “e”, “l”, “l”, “o”, “ “, “h”, “e”, “l”, “l”, “o”, “ “, “t”, “h”, “e”, “r”, “e”]	(1)
[“he”, “l”, “l”, “o”, “ “, “he”, “l”, “l”, “o”, “ “, “t”, “he”, “r”, “e”]	(2)
[“hel”, “l”, “o”, “ “, “hel”, “l”, “o”, “ “, “t”, “he”, “r”, “e”]	(3)
[“hell”, “o”, “ “, “hell”, “o”, “ “, “t”, “he”, “r”, “e”]	(4)
[“hello”, “ “, “hello”, “ “, “t”, “he”, “r”, “e”]	(5)
[“hello “, “hello “, “t”, “he”, “r”, “e”]	(6)

Figure 2. Byte-Pair Encoding Applied to an Input String.

As in Figure 2, BPE first separates the input string into character-level tokenization, where each letter is its own token, as seen in (1). Then, it iterates through this, counting the frequency that two adjacent pairs of bytes occur throughout the tokenization. If the algorithm notices that a pair repeats more than once, it merges the two tokens into one item in the shown list, as in (2). Continuing through this process, BPE ensures that no two consecutive items in the list that are repeated occur.

It is important to note that the token dictionary size and tokenization length alter from (1) to (6). In the beginning, the tokenization length is large, while the token dictionary size is small. However, continuing through the iterations, the tokenization length becomes smaller, while the token dictionary size increases. This is the reason that BPE plots are a curve.

In this work, BPE is used as a method to analyze the repeatability of texts, especially those texts that are in different languages and families.

2.2.2 Text Translation and Applying BPE

To allow for consistency between the languages, we chose one text, “The Monkey’s Paw” by W.W. Jacobs, to convert into the seven different languages initially. Specifically, the seven most popular languages that NLP processes utilize were tested, which include English, Chinese

(Traditional), Urdu, Persian, Arabic, French, and Spanish. This text was chosen as a result of it being a short story, which the algorithm could process in a small amount of time. Also, the range of vocabulary words in the story allowed it to be a good fit for this experimentation.

This text was translated into seven different languages using the Google Translate API. BPE was then applied to these translated texts. The tokenization length and token dictionary size were recorded after each iteration of BPE and plotted for easier visualization.

2.2.3 Analyzing Categories of Languages

To understand if these trends in content repeatability occur across a variety of texts in the same families, we applied BPE to two distinct language subgroups based on the most common NLP inputs: Romance Languages and Indo-Iranian families. Amongst these, we choose the most commonly available languages to apply BPE to. The same text used in step 2.2.2 is converted into a larger set of languages in the aforementioned families.

2.2.4 Applying Pre-Trained Tokenizers to Various Languages

Since BPE alone is not commonly utilized for NLP systems, but rather pre-trained tokenizers that are specifically made for a variety of languages, we then apply pre-trained multilingual tokenizers to compare each tokenizer's output tokenization length and token dictionary size for the most common language of each language family identified above. We seek to identify which provides a balance between the two metrics, which would ultimately be the most effective tokenizers for such categories of inputs.

2.3 Pre-Trained Tokenizers for the English Language

We aimed to test the tokenization of pre-trained tokenizers on English texts of similar sizes. We used English texts because pre-trained tokenizers are historically trained on large English corpora, so testing the tokenizers on any other language texts would prove both ineffective and fruitless.

We chose three commonly used pre-trained tokenizers, which are applied in developmental settings for various uses: BERT, WordPiece, and GPT-2 [3, 7, 9].

We used a Macbook Pro (2017) with a 3.1 GHz Dual-Core Intel Core i5 processor and 8 GB 2133 MHz LPDDR3 RAM. For testing each tokenizer, we used the transformers and tokenizers libraries from Hugging Face in an iPython Notebook environment.

For testing, first, the texts were all transformed from ".txt" files into a string after parsing and removing all control characters (\n, \t, etc.). Secondly, we ran each tokenizer on the same system, and recorded the length of the tokens generated, alongside the size of the token dictionary generated. Finally, to visualize the results, we used Numpy and Matplotlib to find the relationship between each pre-trained tokenizer with a dot plot.

3. Results

3.1 Genome Repetitiveness by Analyzing N-Grams

After developing our superimposed plot of the 6 genomes, we found the fraction of repeated 8-grams. As the length of the genome increases, the plots flatten because, at some point in the sequence length, the genome exhibits all possible distinct n-grams. We are particularly interested in the various rates at which different genomes achieve this. Because this first plot shows the repeated n-grams instead of the distinct n-grams, it still follows this pattern but is simply shown in a different manner. Given the equation used for our Python dictionary in our second algorithm, "(n_non_unique / (n_non_unique + n_unique)," and since all of the genomes eventually exhibit all possible 8-grams, the computed number for the following plot reaches 1 as the numerator increases.

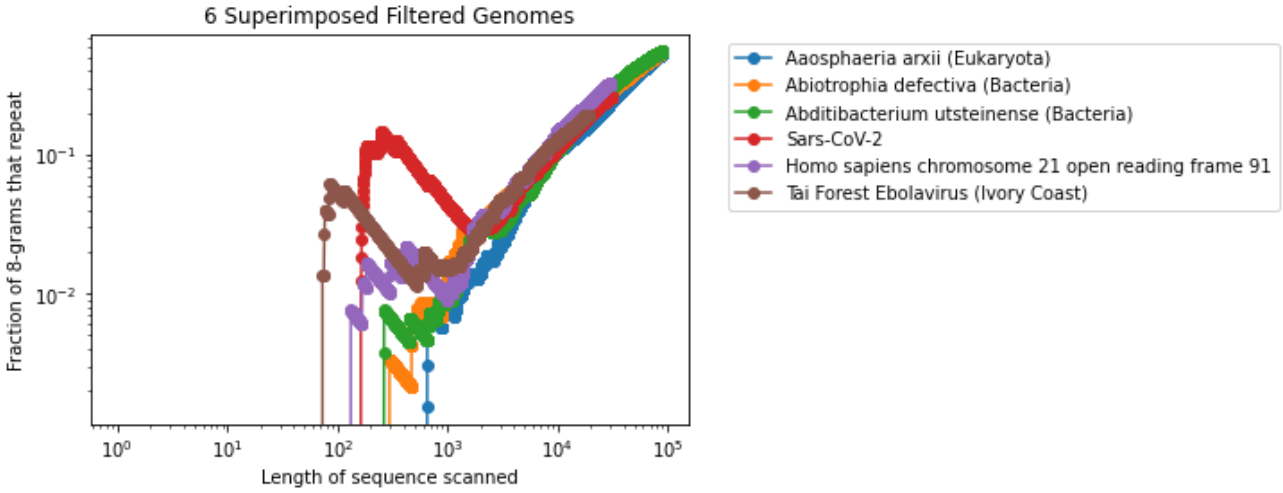

Figure 3. 8-gram repetition analysis applied to 6 filtered genomes. A logarithmic scale is used. For example, when x = 0, log(x) equals -∞, not shown in the plot. Conversely, the log of a positive input is detailed above.

In figure 3, all genomes tend to be more repetitive towards the beginning of their sequence, then become less repetitive, and finally more repetitive again. The fraction of repeated n-grams eventually goes to 1, as more repetition causes the lines to increase from -∞.

The purple line—the human genome—is the most repetitive of out the 6 genomes, meaning that it has more frequent repetitions than bacteria or viruses on long-length scales. This could be because human cells reproduce rarely and are rather large. There could be less of a need for human cells and eukaryotic cells to have rather short genomes, in comparison to viruses that need to compact their genetic material in small capsules. Further, since the human genome is involved in producing a multitude of proteins to support fundamental body functions, the repetitiveness of the sequence could be useful in ensuring that vital human reactions and systems are constantly in progress, also considering the exposure to potential mutagens.

Both viruses, shown by the red and brown lines representing Sars-CoV-2 and the Tai Forest Ebolavirus (Ivory Coast), are much more repetitive toward the beginning of their sequences in comparison to the other genomes. There could be an aspect of the viral sequence that is more important for serving the virus's function, leading to its high repetitiveness earlier on.

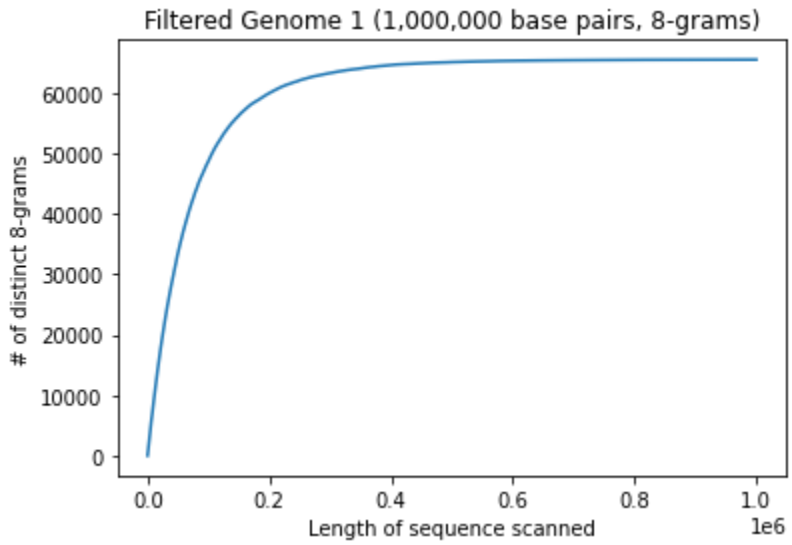

Figure 4. Using our initial algorithm to directly plot the number of non-unique 3-grams. No logarithmic scale is used.

As the length of the sequence from the beginning of Genome 1, or Aaosphaeria arxii, increases, the plot depicts the output of the distinct 8-gram count eventually flattens. All possible 3-grams are eventually shown, given that $4^8 = 65{,}536$. Here, 4 represents the four types of base pairs—A, C, G, T—and 8 is the length of each n-gram. The result indicates the number of all possible 3-grams that can be shown, which matches the plot above, as the number of distinct 8-grams eventually reaches this value.

Figure 4 presents the results in a slightly different manner than figure 3: figure 4 plots the number of 8-grams that are unique, the graph increasing as this occurs and leveling out once all 8-gram possibilities are shown. Conversely, figure 3 plots the repeated fraction of 8-grams, showing increasing slopes as more repetitions occur and flatting to 1 as there are more non-unique 8-grams.

3.2 Byte-Pair Encoding and Language Analysis

To determine the information density and repetitiveness of various languages and their corresponding language families, Byte-Pair Encoding was applied to them. Languages analyzed include the seven most popular languages amplified in NLP systems as well as two language families. We aim to look for trends in repetitiveness by plotting the tokenization length and token dictionary size between the languages and language families.

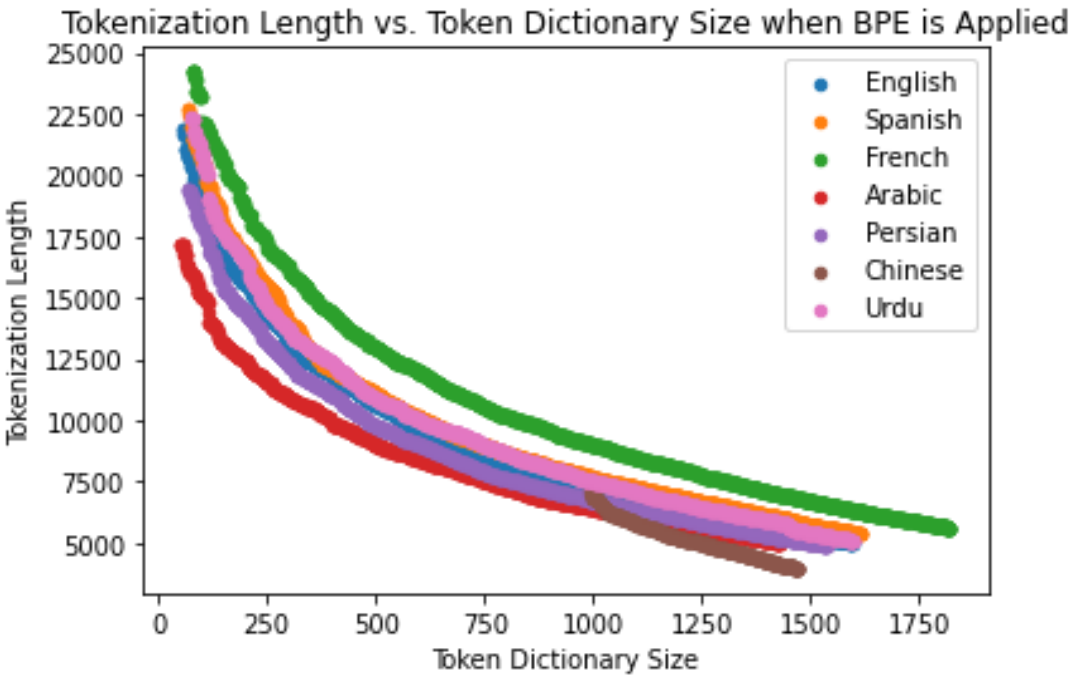

Figure 5. Analyzing the Repeatability of the Seven Most Popular Languages Used in NLP.

From this, we understand that Arabic has the smallest tokenization length at the byte-level, meaning that it has the smallest number of individual characters in the text. We speculate that this occurs because in Arabic, short vowels are not written but long vowels are. Arabic also works with the root system, so combinations of root letters can have multiple meanings. It is only when conjugated that the meaning becomes more specific and apparent. It is also noticed that the final token dictionary size is the smallest for Arabic. This could be as a result of a number of reasons: articles such as a/an not being written and verbs are usually one word. This shows that the language is more compactly written, allowing it to be more efficient for NLP systems to understand if inputted. French, on the other hand, has the largest token dictionary size and tokenization length. We assume that this is because of silent letters being written in addition to consonants.

After this, we seek to understand if this trend occurs in languages' corresponding language families as well. We test two distinct families: Indo-Iranian languages and Romance languages.

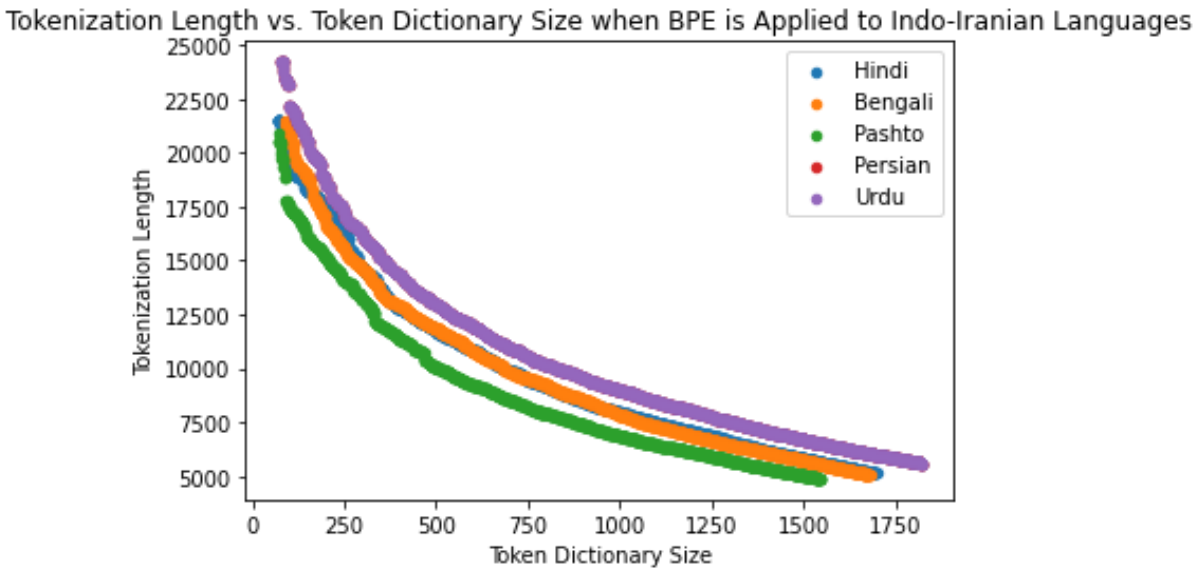

Figure 6. BPE Applied to Five Indo-Iranian Languages. Hindi, Bengali, Urdu belong to the Indo-Aryan family, while Persian and Pashto are part of the Iranian family.

As depicted in Figure 6, the languages in the Indo-Iranian family have generally similar characteristics. However, Pashto has a smaller tokenization length and token dictionary size, implying that information can be conveyed in a smaller amount of words, while Urdu had the largest of the two metrics discussed. Additionally, Bengali, Hindi, and Persian are especially similar in their starting and final tokenization length and token dictionary size. This is especially interesting because Bengali is written from left to right while Persian is written from right to left. We speculate that this is because of the origins of the languages—Persian came from Old Persian, while Bengali came from Sanskrit. Old Persian and Sanskrit were especially close in general regards to their qualities.

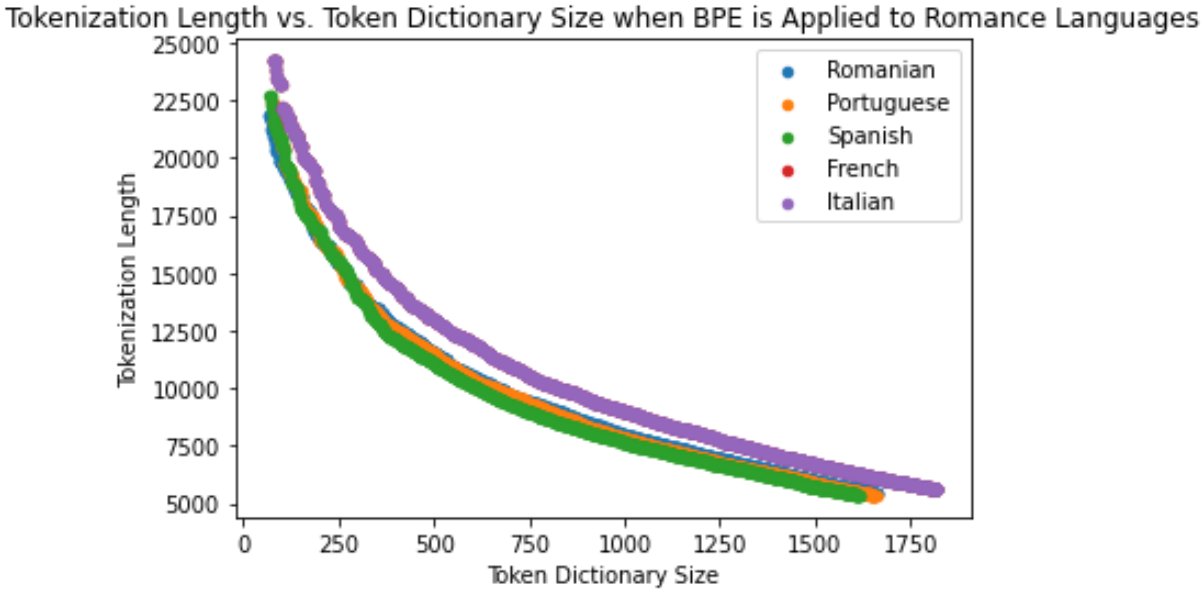

Figure 7. BPE is Applied to Five Romance Languages.

In Figure 7, it is noticed that Spanish, Portuguese, and Romanian are especially close in terms of their initial and final tokenization length and token dictionary size. French and Italian, on the other hand, have the largest initial tokenization length and final token dictionary size, and almost overlap each other. This is because the lexical similarity, or the similarity between the two languages, is around 85-90%, based on prior analysis, which shows that almost 90% of the words are similar in both languages. [6] As a result, they have similar characteristics in terms of repeatability. However, Italian is more difficult grammatically, which was not taken into account in the graph but can impact NLP systems' effectiveness when text in that language is inputted.

3.3 Multilingual Pre-Trained Tokenizers

We tested three multilingual tokenizers that were pre-trained on various data: the M2M100 Tokenizer, mBART Tokenizer, and BERT Tokenizer. [4, 5, 3]

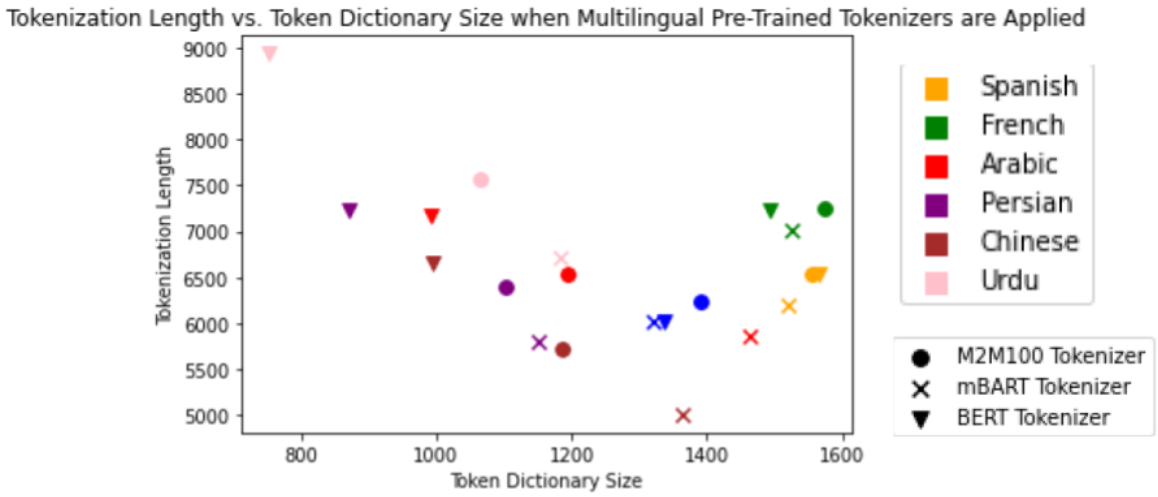

Figure 8. Three Distinct Pre-Trained Tokenizers are Applied to Seven Languages.

In Figure 8, the tokenization length and token dictionary size of six distinct languages is shown when three multilingual pre-trained tokenizers are applied. As seen in the graph, the BERT tokenizer was seen to not perform as well, as it did not provide a balance between a good tokenization length and token dictionary size; rather, it provided for a smaller token dictionary size in many cases, such as Urdu, but an especially large tokenization length. However, the M2M100 Tokenizer did provide this balance that would allow for cheaper and faster evaluation of NLP systems as well as effective training for such, without compromising one for the other. Thus, it is suggested that for most languages, the M2M100 Tokenizer should be utilized.

A possibility as to why the M2M100 Tokenizer ran more effectively is because it is a sequence-to-sequence model, which is made for translation tasks. It can translate between 100

languages, and since it is created specifically for translation, it may have been more effective for the text used as it was translated from English to other languages as well.

3.4 Pre-Trained Tokenizers for the English Language

After running each tokenizer on the English texts, there was a clear pattern for the efficiency of each pre-trained tokenizer. Overall, BERT proved to be more efficient than the other tokenizers, as the tokenization size, and token dictionary, was smaller in nearly every text, as shown in Figure 10. There were some instances where the other two tokenizers were a bit more efficient than BERT, however, in general, GPT-2 performed the worst and WordPiece performed almost the same as BERT, while sometimes being less efficient.

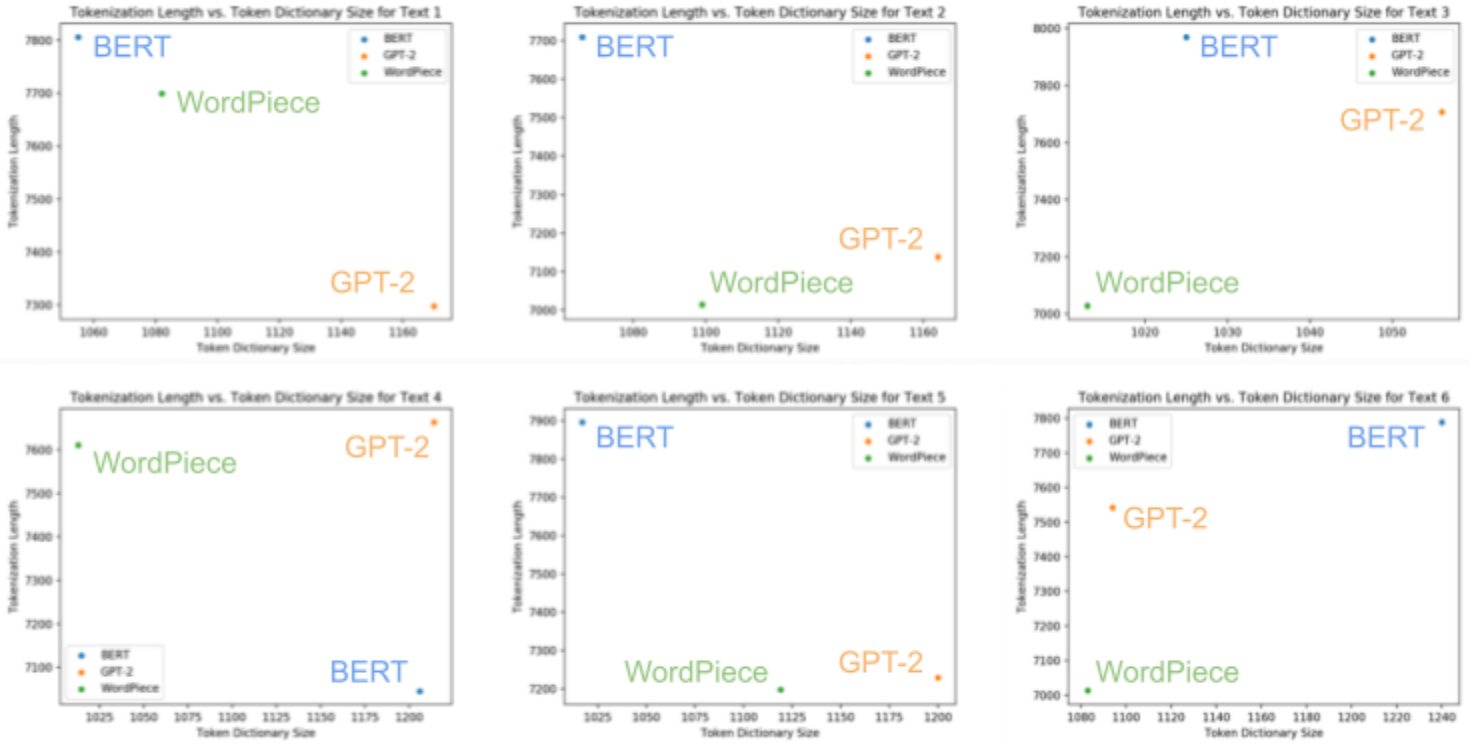

Figure 9. Pre-trained tokenizers applied to various English texts, with their tokenization length and token dictionary size recorded.

GPT-2 ran last possibly because the functionality of the program is quite different from what we are testing. GPT-2 was trained with a causal language modeling (CLM) objective and is therefore powerful at predicting the next token in a sequence, while less efficient as for producing the least amount of tokens when breaking up English text.

BERT and WordPiece ran quite closely, as BERT was actually trained on the WordPiece tokenization. There were odd cases, such as Text 4 and Text 6, where the two tokenizers were on opposite ends of the spectrum. However, this could be a case of bias in the text, as BERT prefers simpler text, while WordPiece can span larger, more complex texts easily.

4. Conclusions

4.1 Genomics

Comparing the number of repeated n-grams, representative of tokens, to the length of the sequence scanned offers an explanation of how unique a subset, or the whole, the genome is. All genomes eventually reach a limit in regards to the number of unique n-grams, causing plots to level out and reach a particular number. It was particularly insightful to look into the different rates that different types of genomes exhibited this behavior.

From the genomes that we analyzed, we found that the human genomes, looking at chromosome 21, are more repetitive than viruses. These viruses are much more repetitive towards the beginning of their sequences in comparison to the bacteria and eukaryotes.

While working through different approaches for plotting and displaying the data results, we encountered some that were not as noticeable due to different parameters that we had set in the Jupyter Notebook. Therefore, our first algorithm looks into the direct distinct number of 8-grams, However, the finalized superimposed genome plot uses our second algorithm with a logarithmic scale and the new equation used for our dictionary of distinct n-grams (n_non_unique / (n_non_unique + n_unique)).

The foundation of our analysis into genome repeatability can be largely useful for practitioners in the genomics and bioinformatics fields interested in the uniqueness spectra of genomes, and how unique one section is compared to another. Particularly, for certain engineering practices, it is significant to target unique sites that do not have homologous base pairs in another genetic region [2].

4.2 BPE and Various Languages

For this study, Byte-Pair Encoding was used to measure the repeatability of a variety of languages and language families. Through the visualization and comparison of the token dictionary size and tokenization length, it was found that Arabic was the most compact and efficient language for NLP tasks when compared with the six other most commonly used languages in NLP. This was as a result of short vowels and articles not being written, but also because it relies on the root system.

When language families were compared, the Romance Languages had many characteristics in common. Italian and French overlapped each other with their individual BPE curves, while Spanish, Italian, and Romanian had comparable characteristics as well. However, this was unlike the Indo-Iranian languages, which differed in terms of the two metrics used.

The languages and families utilized can allow for NLP systems to be more efficient. Our findings, such as French being less repetitive than English and other languages, point to the idea that certain languages that convey less information in more words may be ineffective for NLP tasks. By analyzing the information density of the text, ambiguity issues in NLP models can be reduced.

4.3 Multilingual Pre-trained Tokenizers

We also sought to analyze three popular pre-trained tokenizers that were trained on a variety of languages. We found that the M2M100 tokenizer worked the most effective as it provided a balance between the tokenization length and token dictionary size, which ultimately, would allow users to not only save money but also allow for effective training of such.

The BERT tokenizer, however, was seen to do the worst when evaluated on these metrics. This is as a result of it providing a small token dictionary size but an especially large tokenization length, or the total number of tokens. The mBART tokenizer was also ineffective for a similar reason, but instead of providing a large tokenization length, it provided a smaller tokenization length but a larger token dictionary size.

Thus, it was seen that for NLP tasks that involve one of six popular languages, the M2M100 tokenizer is the most effective generally. This generalization can also be applied to other texts not included in the six languages listed if the characteristics of the language is similar to that of a language shown. Further research would need to be conducted to examine other, less-common languages and trends that occur between the pre-trained tokenizers used.

4.3 English-Specific Pre-trained Tokenizers

Although each pre-trained tokenizer has its own purpose and can be used for a variety of developmental applications, BERT seemed to surpass other tokenizers in terms of efficiency for short English texts. By knowing that BERT is more efficient for tokenizing small text, we can use this tokenizer for more specialized applications (NLP apps, machine translation, etc.) until a better algorithm, such as improved BPE, can be widely implemented.

Specifically, there are many potential applications for improving NLP applications for disabled people. Recently, in a study done by Svensson et al (2019), 149 students representing various urban and rural schools in Sweden took part in a study “to investigate if an intervention using assistive technology for students with reading disabilities affected their reading ability, ability to assimilate text, and motivation for schoolwork.” Over the 4 weeks of testing, students used NLP applications, such as SayHi (speech-to-text = STT), VoiceDreamReader (text-to-speech = TTS), Prizmo (scanning from written text to digitized text), Skolstil-2 (an easy word processor and text-to-speech app that even pronounces each sound-letter, words, sentences, and the whole text while writing a text), Legimus (an audiobook reader), and Ruzzle (a word game). [8] The results show gains in reading ability despite using nothing but assistive technology during the intervention period, and these gains were comparable to the enhancement in a control group during 1 year. Approximately 50% of the students and their parents reported an increase in motivation for schoolwork after they had finished the interventions.

With tokenizers like BERT, we can improve the situation in schools for children, or even in the workforce. NLP applications can become more efficient with better-implemented tokenizers. This research is promising for future work in advancing current technology into the mainstream consumer market for NLP apps.

5. Future Directions

5.1 NLP Applications for the Disabled

We hope to apply our findings to devise new teaching methods for intellectually disabled and hearing-impaired students. This could include a new hybrid between an improved BPE algorithm and a historically efficient tokenizer like BERT to optimize the NLP models that developers currently use for NLP assistive technology.

5.2 Evaluation Framework for Developers

We would also like to construct an evaluation framework that would provide real-time recommendations to NLP model users on which tokenizers to use based on the NLP task at hand. In this study, it has been determined that BERT and M2M100 tokenizers are the most effective for English and multilingual texts, respectively, but we would like to expand these findings and find trends between language families and the behavior of such tokenizers. Because of the large number of people that use and are constantly improving NLP models, it would be effective for users to understand how to best utilize their resources as well as which tokenizer is best fit for their tasks, whether that be translating between languages, working with English texts, or working with specific texts translated into another language.

6. Acknowledgements

We would like to thank our incredible mentor, Dr. Dmitri Pavlichin, for providing guidance throughout the project and sharing his knowledge with us. We would also like to thank Cindy Nguyen and Professor Tsachy Weissman for organizing and coordinating the STEM to SHTEM program. They, along with many others, have been especially influential in our research.

7. References

[1] Bostrom, K., & Durrett, G. “Byte Pair Encoding Is Suboptimal for Language Model Pretraining.” Findings of the Association for Computational Linguistics: EMNLP 2020, 2020, doi:10.18653/v1/2020.findings-emnlp.414.

[2] Cho, Seung Woo, et al. “Analysis of off-Target Effects Of Crispr/Cas-Derived RNA-Guided Endonucleases and Nickases.” *Genome Research*, Cold Spring Harbor Laboratory Press, Jan. 2014, www.ncbi.nlm.nih.gov/pmc/articles/PMC3875854/.

[3] Devlin, Jacob, et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” 11 Oct. 2018, arXiv: 1810.04805.

[4] Fan, Angela, et al. “Beyond English-Centric Multilingual Machine Translation.” Journal of Machine Learning Research 22, 2021, arXiv: 2010.11125.

[5] Liu, Yinhan, et al. “Multilingual Denoising Pre-training for Neural Machine Translation.” Transaction of the Association for Computational Linguistics, 2020, doi:

https://doi.org/10.1162/tacl_a_00343.

[6] Schepens, Job et al. “Cross-language distributions of high frequency and phonetically similar cognates.” *PloS one* vol. 8. 10 May. 2013, doi:10.1371/journal.pone.0063006.

[7] Schuster, Mike, and Nakajima, Kaisuke. “Japanese and Korean Voice Search.” https://static.googleusercontent.com/media/research.google.com/ja//pubs/archive/37842.pdf.

[8] Svensson, Idor, et al. “Effects of Assistive Technology for Students with Reading and Writing Disabilities.” Disability and Rehabilitation: Assistive Technology, vol. 16, no. 2, 2019, pp. 196–208., doi:10.1080/17483107.2019.1646821.

[9] Radford, Alec, et al. “Language Models are Unsupervised Multitask Learners.” *OpenAI*, https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.