An Effort Towards Quantum Tokenization

Charis Graham
Carnegie Mellon University
csgraham@andrew.cmu.edu

Abstract

Quantum natural language processing (QNLP) is the field of study of how to perform NLP tasks on quantum hardware (Lorenz et al., 2023). Subword tokenization is a problem that is particularly important in the field of NLP. It is the task of splitting up a text into common character clusters for the purpose of allowing a model to maintain a reasonable vocabulary size while being able to learn representations of tokens and process unknown words into known subword tokens (2024). In this paper, I present a novel implementation of how to perform subword tokenization of English words using a quantum-based system. The system works fairly well in comparison to other common tokenization algorithms such as Unigram SentencePiece, Morfessor, and Byte-Pair Encoding. The motivation behind this is furthering the field of QNLP as well as attempting to generate another system for tokenizing text.

1 Introduction

Quantum natural language processing deals with studying how to implement classical NLP tasks using quantum hardward (Lorenz et al., 2023). Up to this point, there have been limited attempts within this field. There have been little to no forays into using quantum hardware to support attempts at tokenization. Tokenization is incredibly important within NLP and within studies of modeling subwords (smaller segments of words). This is because language models rarely deal with whole words but rather deal with segmented words broken into meaningful subwords (2024). This allows language models to maintain restricted vocabulary sizes as well as deal with unknown words by breaking them down into known subwords (2024). Some common classical models used are Byte-Pair Encoding (BPE), Unigram SentencePiece, and neural models like Morfessor 2.0. These models depend on finding specific common sequences of letters and grouping those as tokens that an unknown word would be broken into. My attempt was to add a layer to this that would help determine the best potential tokenization using a quantum model.

2 Preliminaries

2.1 Common Tokenization Models

There are several tokenization models that are commonly used in NLP. Below we explore a few of these models.

2.1.1 Byte-Pair Encoding (BPE)

Byte-pair encoding is initialized as the byte-level representation of characters and slowly builds up a corpus (Sennrich et al., 2016). Pairs of bytes are replaced by unused symbols until a certain vocabulary size is reached (Sennrich et al., 2016). As a vocabulary is built up, rules are added to a grammar (Sennrich et al., 2016). These grammar rules are applied when we want to tokenize a word.

There are some observations one should make with regards to BPE. Firstly, if a vocabulary is larger, there are fewer tokens used (Sennrich et al., 2016). Furthermore, while the token boundaries do not necessarily align with those of known morphemes within a given language, common morphemes do tend to coalesce first (Sennrich et al., 2016).

2.1.2 Unigram SentencePiece

SentencePiece is a model that is based on two implementations: one that uses BPE and one that uses a Unigram model (Kudo, 2018). A key feature of SentencePiece is that it does not make the assumption that word sequences can be broken down before further tokenization via whitespace (Kudo Richardson, 2018). In this paper, we focus on the Unigram model of SentencePiece. Contrary to BPE, Unigram does not start with a small vocabulary and build it bit by bit. Rather, Unigram tokenization begins with a larger vocabulary and whittles it down based on loss at a character-level language task (Kudo, 2018). Probabilities of tokens are computed of a sequence via the product of the probabilities of items in a sequence (Kudo, 2018).

Again there are some key observations one needs to make. Via the above computations, there are three types of segmentation candidates: individual characters, subwords, and full words. A Viterbi algorithm is used to find the optimal length for each segmentation candidate (Kudo, 2018). Again, these models are built iteratively, but all choices are done by maximizing probabilities.

2.1.3 Morfessor

Morfessor is an example of unsupervised morphological segmentation. Unlike the two systems above, Morfessor is more morphologically motivated and actually attempts to replicate a linguist's segmentation of a word (Virpoija et. al., 2013). Morfessor is also more complicated than the above two systems and to understand, one needs some specialized terminology: atoms (smallest piece of text used by the algorithm, like characters); compounds (groups of atoms, like words); and constructions (an intermediary lexical unit between atoms and compounds, like morphs) (Virpoija et. al., 2013). Again a probabilistic distribution is used to determine the tokenized form of a text. Morfessor, similar to BPE, builds up a grammar but in the form of a list of properties of constructions and a grammar of how to combine these constructions into compounds (Virpoija et. al., 2013).

There are a few things to note on Morfessor. Morfessor tends to prefer grammars with fewer and shorter constructions (Virpoija et. al., 2013). Each construction generated by Morfessor has two aspects, both a form and a usage (Virpoija, 2013). Form is what sequence of atoms are used to build it up (Virpoija et. al., 2013). These are considered independent of each other, which simplifies the probabilistic calculation of form by allowing us to focus on the distribution of length and the categorical distribution (Virpoija et. al., 2013).

2.2 Introduction to Quantum Computing and Quantum NLP

Language is quantum-native (Coecke et al., 2022). In other words, this means that there has been established a direct correspondence between the meaning of words and quantum states as well as the grammatical structures of words and quantum measurements (Coecke et al., 2022). The first instance of doing natural language processing in a quantum computing context was proposed in 2016 and since then some simple attempts have been made, including Intel running some basic systems on quantum simulators (Coecke et al., 2022). However, QNLP raises some significant challenges. As of right now, there are no significantly powerful quantum computers that are able to implement large-scale NLP tasks (Coecke et al., 2022). Furthermore, the idea that one can encode word meanings on a quantum computer is a key assumption to many stipulated models, but this is a distant possibility due to the size of the data (Coecke et al., 2022).

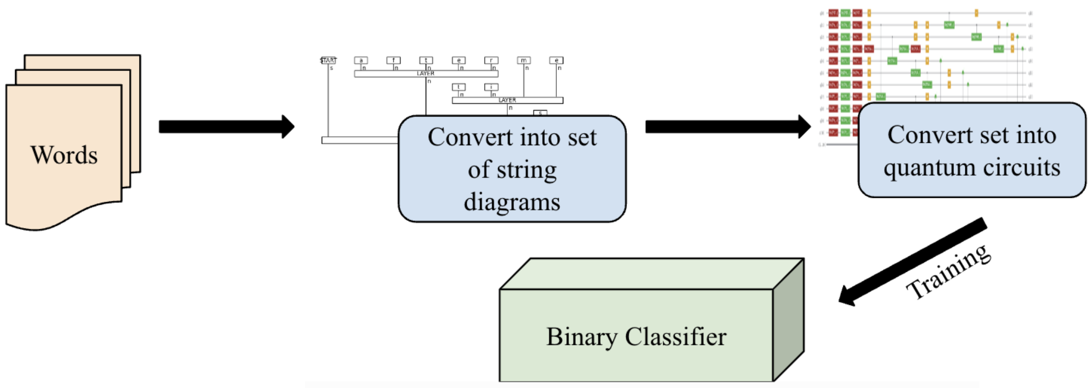
2.3 Quantum Computing

Quantum computing is the field of using quantum mechanics in designing algorithms that have no efficient classical (non-quantum) counterparts (Karamlou et al., 2022). There are a few important terms to understand when discussing quantum computing. Superposition is incredibly essential to understand. Superposition is a feature of quantum mechanics is used to describe the ability of a quantum object, such as an electron, to have no definite reality before measurement (Howgego). Schrödinger's thought experiment is the most useful in describing this idea. It imagines placing a cat, a vial of poison, and a radioactive atom in a box and then closing the box (Howgego). At any given moment in time, the radioactive atom could decay, breaking the vial, releasing the poison, and killing the cat (Howgego). However, until one opens the box, we do not know if the cat is dead or alive and so, in some sense, it is both dead *and* alive (Howgego). This is superposition.

Quantum computers use this in qubits, the quantum analogue of a bit (Karamlou et al., 2022). These exist in a 2-dimensional Hilbert space and the state of a qubit is given by a linear combination of the basis vectors, given by $|0 >= [1, 0]$ and $|1 >= [0, 1]$ (Karamlou et al., 2022). In what is known as bra-ket notation, the state of a qubit, given by $|\psi >$, is represented as (Karamlou et al., 2022):

$$|\psi >= \alpha|0 > +\beta|1 > s.t. \alpha, \beta \in C, |\alpha|^2 + |\beta|^2 = 1$$

When one reads the value of a qubit, we call that taking a measurement and this collapses the state to either 0 or 1 (Karamlou, 2022). The probability of measuring 0 is equal to $|\alpha|^2$, and α is known as the amplitude of $|0 >$ (Karamlou et al., 2022). This is the same for $|1 >$.


Words
Convert into set of string diagrams
Convert set into quantum circuits
Training
Binary Classifier

$$H(p,q) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p(y_i)) + (1-y_i)\log(1-p(y_i))]$$

Figure 1: A diagram detailing the pipeline of training the quantum binary classifier where the shown loss function is the function for binary cross entropy.

The use of one qubit is not particularly meaningful, and so to do computations, multiple qubits are used (Karamlou et al., 2022). These are held together in a what is known as a joint state (Karamlou et al., 2022). These joint states can be manipulated using quantum logic gates, otherwise known as unitary linear maps (Karamlou et al., 2022).

Two important types of quantum logic gates to know for this paper are rotation operators and Hadamard gates. Rotation operators are denoted by Rz, Rx, and Ry gates and they are a representation of a single-qubit rotation some θ radians about the z, x, or y axes. For example, the Rz gate is denoted by the following vector[1]:

$$R_z(\theta) = \begin{bmatrix} e^{-i*(\theta/2)} & 0 \\ 0 & e^{i*(\theta/2)} \end{bmatrix}$$

Hadamard gates are also single-qubit operations that map the basic state to an equal superposition of the two basis states. It is denoted by the following vector[2]:

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

These gates can be combined to perform computations.

3 Methodology

There are two parts to the general system: the quantum binary classifier and the actual tokenizer.

[1] https://www.quantum-inspire.com/kbase/rz-gate/

[2] https://www.quantum-inspire.com/kbase/hadamard/

3.1 Quantum Binary Classifier

The Quantum Binary Classifier was influenced by the sentence classifier algorithm defined in the paper by Lorenz et. al. (2023). The general pipeline of training the quantum binary classifier is shown in Figure 1. At a high level, we take some data set of words and convert them into string diagrams (see section 3.1.1). We then convert those string diagrams into quantum circuits (see section 3.1.2). The quantum-binary classifier model is then trained on these quantum circuits (see section 3.1.3). This part of the project makes heavy use of the QNLP library lambeq[3], which was designed to facilitate running both classical and quantum NLP experiments on quantum hardware or simulated quantum hardware.

3.1.1 Convert Words to String Diagrams

For the project, the data used was from the SIGMORPHON 2022 Shared Task on Morpheme Segmentation[4]. Specifically, the English tokenized word data was used rather than the sentence data. The reason single word data was used over sentence data is because, as will be explained shortly, the string diagrams scale in size exponentially.

The first step in QNLP tasks is to represent the words one is manipulating or studying as a string diagram. String diagrams are pictorial representations of words and their relationships within a sentence (Karamlou, 2022) . They consist of boxes for words and strings that connect the boxes accord-

[3] https://cqcl.github.io/lambeq/installation.html

[4] https://github.com/sigmorphon/2022SegmentationST/tree/main

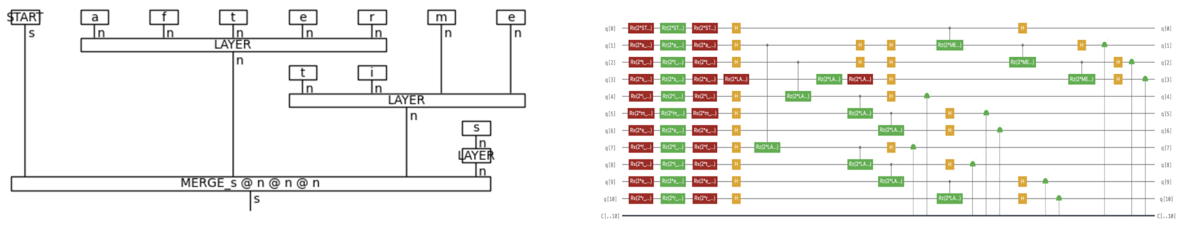


Figure 2: On the left, a string diagram representing the tokenization of the word “aftertimes” as “after time s.” On the right, the string diagram for “aftertimes” correctly transformed into the representative quantum circuit with red and green rotation operators and square orange Hadamard operators.

ing to some pre-defined grammar that represents the relationships between the words (Karamlou, 2022).

While lambeq does have some predefined types of string diagrams, for the tokenizer classifier, it became apparent that a novel type of diagram would be needed: the Token Diagram. This is because a diagram was needed that could represent the relationships from the character to segmented to whole word level all at once.

The way the Token Diagram represents a word, such as "aftertimes," can be seen in Figure 2:

1. The word is first broken into individual characters and each character is represented by a single box. For example: "a f t e r t i m e s".
2. These are connected by strings to a layer that represents the segmented form of the word. For example: "after time s".
3. These intermediary layers are all merged into one final layer that represents the word as a whole that has a final string coming out of it that represents the qubit state of the word of a whole. For example: "aftertimes".

Using the Token Diagram, any word can be segmented and this segmentation can represented using a single string diagram. However, the diagrams scale exponentially as can be seen in Figure 3. If one was to attempt to represent an entire sentence, the string diagram quickly becomes quite cumbersome and difficult to run on current quantum computers that have limited qubits.

3.1.2 Convert String Diagrams to Quantum Circuit

Before training can begin, the string diagrams must be converted into quantum circuits. Due to the simple layered model of combined letters to form words, instantaneous quantum polynomial circuits

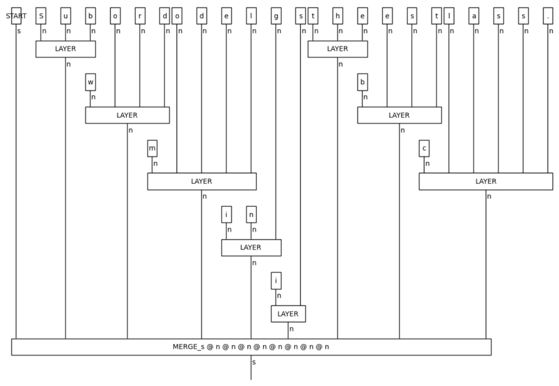


Figure 3: String diagram for the sentence "Subword modeling is the best class."

(ones that weave together layers of Hadamard gates with rotation operators) were used. This step is mostly handled automatically by the lambeq library. However, to convert, one must define an ansatz (a map between string diagrams and quantum circuits that define the number qubits each string in the string diagram is worth). Because of the equal weighted nature of each of the characters (based on an assumption that no letter is more important in a word than another), a simple 1 qubit per string model was adapted. This can be seen in Figure 2 where there are eleven qubits to represent the ten letters in the word "aftertimes" that are combined (plus the START token).

3.1.3 Training the Quantum Binary Classifier

The basis of the Quantum Binary Classifier (QBC) is the NumpyModel[5] provided by lambeq. This model was chosen over other offered models because it is very good at converting quantum circuits into tensor networks that can be run on quantum simulators in the low-level lambeq.backend using unitary and density matrix simulators. It is used in

[5]https://cqcl.github.io/lambeq/models.html

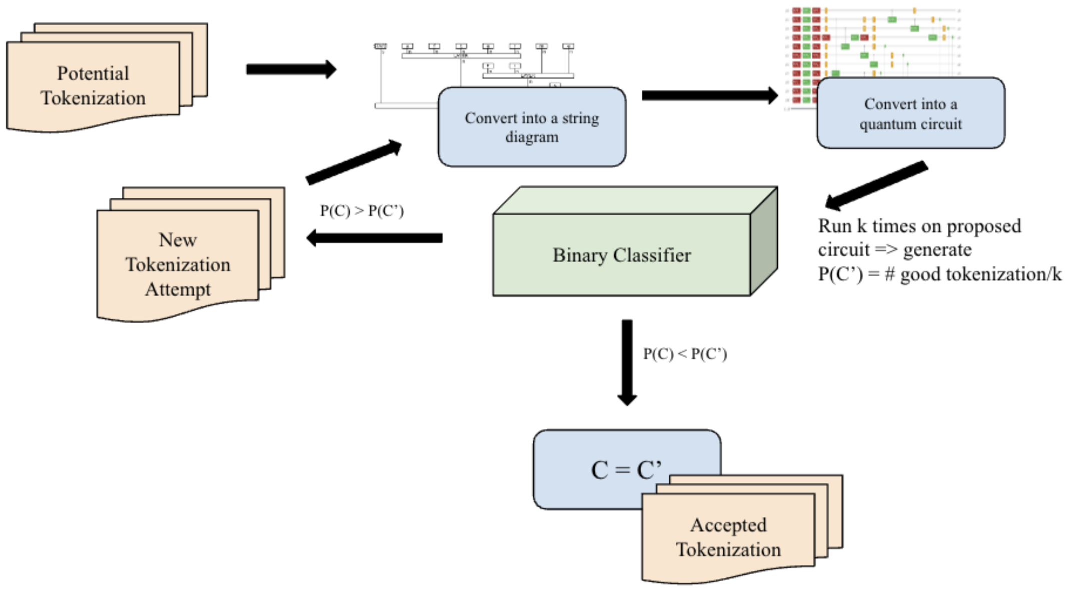

Potential Tokenization
Convert into a string diagram
Convert into a quantum circuit
New Tokenization Attempt
P(C) > P(C')
Binary Classifier
Run k times on proposed circuit => generate P(C') = # good tokenization/k
P(C) < P(C')
C = C'
Accepted Tokenization

Figure 4: A diagram detailing the pipeline of using the quantum binary classifier as part of a tokenization algorithm.

common cases such as binary classification which involves measuring only the final qubit after a given circuit has been followed.

The QBC was trained on 650 examples of correctly and incorrectly segmented English words converted into circuits. The goal was to train the model to tell the difference between an accurate segmentation and an inaccurate segmentation. The incorrect segmented circuits were generated by manipulating a subset of segmentations before converting into a string diagrams from the data. There were three primary groups of data:

1. Correct: These were untouched segmentations that were correctly converted.
2. Recombined: These were inaccurately segmented words that had been recombined into the whole world as one whole construct before being converted.
3. Wrong Segmentation: These were inaccurately segmented words where the segmented portions were randomly recombined and then converted.

These particular groupings were chosen roughly based off of the three construction types that Morfessor produces. The model was set to train for 150 epochs with batch sizes of 100.

The model uses binary cross-entropy for its loss function (as can be seen in figure 1). In the function in Figure 1, the y_i represents the truth label for the particular circuit (is it a correctly segmented word or not), the $p(y_i)$ represents the probability that the model assigns to a given circuit as to whether it's good or not, and finally N represents the total number of datapoints (650 training circuits) the model is trained over.

3.2 Quantum-Based Tokenizer

Once the model is trained, it can be used as part of a tokenization algorithm. Figure 4 shows the generic pipeline for the actual tokenizer algorithm. The algorithm works by suggesting a potential tokenization and once it has been converted into a circuit (see section 3.2.1), running the binary classifier on it some k times. This is used to generate a probability of how valid a suggested tokenization is and if it is above a given threshold that tokenization is accepted (see section 3.2.2). If it is not, it is thrown out and a new tokenization is tested.

3.2.1 Generating Potential Tokenizations

While the original attempt was using partially randomly generated potential segmentations of a given word, and this did look initially promising, it created way too much noise within the system and was extremely computationally expensive. Therefore, other ideas were tried. In the end, the easiest method was simply building upon other already existing tokenization models. Using Morfessor, Unigram SentencePiece, and BPE trained on the SIGMORPHON 2022 Shared Task on Morpheme Segmentation data, three potential tokenizations for

Model	F1 Score	Precision	Recall
Unigram SentencePiece	0.15	0.11	0.27
Morfessor 2.0	0.11	0.19	0.08
BPE	0.22	0.22	0.36
Quantum Tokenization	0.15	0.12	0.20

Table 1: F1, Recall, and Precision scores for each of the baseline comparison models and for the Quantum Tokenization model.

a given word were generated and then converted into string diagrams and then quantum circuits using the same methods as in 3.1.

3.2.2 Testing Tokenizations

Each of the generated circuits via the three trained classical models were run through the Binary Quantum Classifier and an average was taken over how many times the classifier identified that given tokenization as "correct" over 100 iterations. The maximal valued average determined the threshold for which we accept a given tokenization. The most "correct" segmentation of the given word was thus returned as the segmentation for that word.

4 Results

4.1 Discussion

The main evaluation of the system was done by calculating the F1 score across a test sample dataset for the different models. Table 1 displays the F1, recall, and precision scores for each of the models. As we can see, there are rather low F1 scores for all of the models (discussed further below), however, the Quantum Tokenization Model seems to work about as well as the Unigram SentencePiece model. It had a precision of 0.12 which was higher than SentencePiece but less than Morfessor or BPE. Precision is a measure of quality and a higher precision means that an algorithm is returning results that are more relevant than irrelevant. Here we can see that the Quantum model is producing results that are on the lesser end of relevancy. With regards to recall (a measure of quantity), higher recall means that out of the results a model returns, they are more likely to be relevant than not. With regards to this, the Quantum Model is able to return a somewhat high quantity of relevant results (higher than Morfessor, but not as good as Unigram or BPE). Because of the similarity of results to SentencePiece, this suggests that the Quantum Binary Classifier tends to prefer tokenization schema produced by SentencePiece over Morfessor or BPE.

Upon inspection of the particular trends within the predicted tokenizations, it appears that of the base models, BPE and Unigram SentencePiece focused on grouping smaller clusters of letters rather than the larger groupings that the "correct" tokenization had. For example for the word "biclosed," Unigram SentencePiece segmented is as "b i clo s ed" and BPE segmented it as "b ic lo s ed." This pattern was probably because of restricted vocabulary size and limited amount of training data. The training data size was kept on the smaller end of the spectrum for the Quantum Model because of model restrictions. This drove the models to suggest smaller clusters which in turn drove down the F1 scores of the models. On the other hand, Morfessor seemed to prefer grouping whole words together with far fewer breaks. For instance, for "biclosed," it segmented the word as the whole word "biclosed." This is probably because Morfessor tends to prefer simpler grammars and not breaking down the word is the simplest grammar one can construct. The Quantum Model seemed to favor a mixture of the two. For "biclosed," the Quantum Model tokenized it the same as Morfessor as "biclosed." However, for the word "plurified," it used "p l ur ifi ed," which is more similar to BPE and SentencePiece. This would make sense because the actual training data has more of a mixture of the two rather than one of the two extremes.

What is interesting to note is that when examining the counts of how many times the classifier chose a segmentation from each model, the model split about half the time using the SentencePiece model and half the time using the Morfessor model prediction. There was a slight bias in favor of SentencePiece. This explains the similarity of the scores to SentencePiece, but also acknowledges that the inclusion of larger groupings of words from the Morfessor model is also important. However, it takes none of its predictions from BPE. This is probably because BPE is a little more in the middle of Morfessor and SentencePiece in its style of tok-

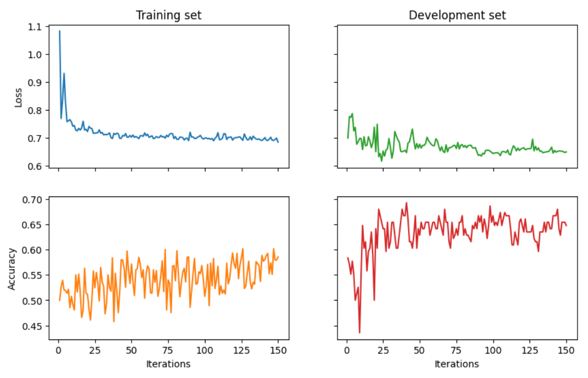


Figure 5: Plotted training data for Quantum Binary Classifier.

enization and the Quantum Classifier has trained to try and detect extremes so it favors picking extreme tokenizations.

4.2 Error Analysis

When we inspect the Quantum Tokenization model we see that it works about as well as the Unigram SentencePiece model. One thing that we should note is that because the Quantum Binary Classifier is running on a simulator of a quantum computer which has simulated noise. Quantum systems are very noisy and so getting consistent results can be quite difficult. For example, on the same training metrics, the model could have an accuracy of between 0.50 and 0.61 on the same data. Figure 5 demonstrates the variance by showing how noisy training is. However, there is a defined upwards curve of learning. Furthermore, the data chosen for the model tends to favor larger segmentations and thus this probably explains why the F1 scores are low across the board.

5 Conclusion

The Quantum Tokenization model isn't fully quantum and works as a hybrid model of both classical and quantum. It tends to favor predictions from the SentencePiece model but also chooses the Morfessor model's predictions as well. It works about as well as the other contemporary models do. The main benefits this model has over the others is that it can represent all of the characters in a word at once and is not left-wise dependent as say BPE is for English (as it builds up a vocabulary from left to right). Further explorations that I would like to do are trying other orientations of string diagrams as the Token Diagram is only one fairly simple model. I would also like to try finding other quantum motivated methods of generating suggested tokenization schema over using existing models. I also would like to try investigating how the model would work on languages other than English and if the models the Quantum Classifier favored choosing from would change based on features of the languages. Finally, I would like to try running this model on an actual quantum computer rather than just a simulator.

6 References

- Coecke, B., Felice, G. de, Meichanetzidis, K., Toumi, A. (2022). How to make qubits speak. Quantum Computing in the Arts and Humanities, 277–297. https://doi.org/10.1007/978-3-030-95538-0_8

- Howgego, J. (n.d.). Schrödinger's Cat. New Scientist. https://www.newscientist.com/definition/schrodingers-cat/

- Hugging Face. (2024). Summary of the tokenizers. https://huggingface.co/docs/transformers/en/tokenizer_summary

- Karamlou, A., Wootton, J., Pfaffhauser, M. (2022). Quantum natural language generation on near-term devices. Proceedings of the 15th International Conference on Natural Language Generation. https://doi.org/10.18653/v1/2022.inlg-main.22

- Kudo, T. (2018). Subword regularization: Improving neural network translation models with multiple Subword candidates. Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 1, 66–75. https://doi.org/10.18653/v1/p18-1007

- Kudo, T., Richardson, J. (2018). Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, 66–71. https://doi.org/10.18653/v1/d18-2012

- Lorenz, R., Pearson, A., Meichanetzidis, K., Kartsaklis, D., Coecke, B. (2023). QNLP in practice: Running compositional models of meaning on a quantum computer. Journal of Artificial Intelligence Research, 76, 1305–1342. https://doi.org/10.1613/jair.1.14329

- Sami Virpioja, Peter Smit, Stig-Arne Grönroos, and Mikko Kurimo. Morfessor 2.0: Python Implementation and Extensions for Morfessor Baseline. Technical report, Aalto University, School of Electrical Engineering, 2013. URL http://urn.fi/URN:ISBN:978-952-60-5501-5

- Sennrich, R., Haddow, B., Birch, A. (2016). Neural machine translation of rare words with Subword units. Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 1715–1725. https://doi.org/10.18653/v1/p16-1162