Final Project Report for CS 175, Winter 2018

Project Title: Good Algorithm, m.A.A.d Lyrics: To Pimp an AI
Project Number: 11

Student Name(s)
Patrick Jose, 20947156, pajose@uci.edu
Christian Poon, 79555434, ctpoon@uci.edu

1. Introduction and Problem Statement (1 or 2 paragraphs)

The goal of this project was to build an AI that can create rap lyrics in a similar style to Kendrick Lamar. This is a natural language processing task called natural language generation, but with an added twist. Our project does not necessarily solve a problem, but pushes the limits of AI in terms of being able to do what humans can do. In this case, we are testing the AI's ability to create art based on knowledge of Kendrick Lamar's songs; which include his vocabulary, flow, and rhyming. We gathered all of Kendrick Lamar's songs, including songs in which he is featured, from The Original Hip-Hop Lyrics Archive (OHHLA) data set and extracted all of his lyrics into a corpus. We then developed two models, a Markov chain and a Long Short Term Memory (LSTM) recurrent neural network, and compared both in order to gain an in-depth understanding of how well each of them performs the specific task. We evaluated our results by conducting a user study since we thought the best way to track our progress was to ask for others' input.

2. Related Work: (1 or 2 paragraphs)

A lot of research papers that we looked into have used Markov models in conjunction with recurrent neural networks to generate lyrics. This is why we decided to build a Markov chain as the base model for our project, and then build an LSTM recurrent neural network on top of that Markov chain. The base Markov model was built from scratch, while the LSTM model used more open-sourced libraries in order to make the main focus in fitting the model to the rhyme and syllable counts. The main article we used as a guide to build our LSTM model is at the following link https://ritcsec.wordpress.com/2017/05/16/rap-lyric-generator/ and the main research paper we used is at the following link http://www.emnlp2015.org/proceedings/EMNLP/pdf/EMNLP221.pdf. These resources were a good starting point that we would use throughout the duration of this project.

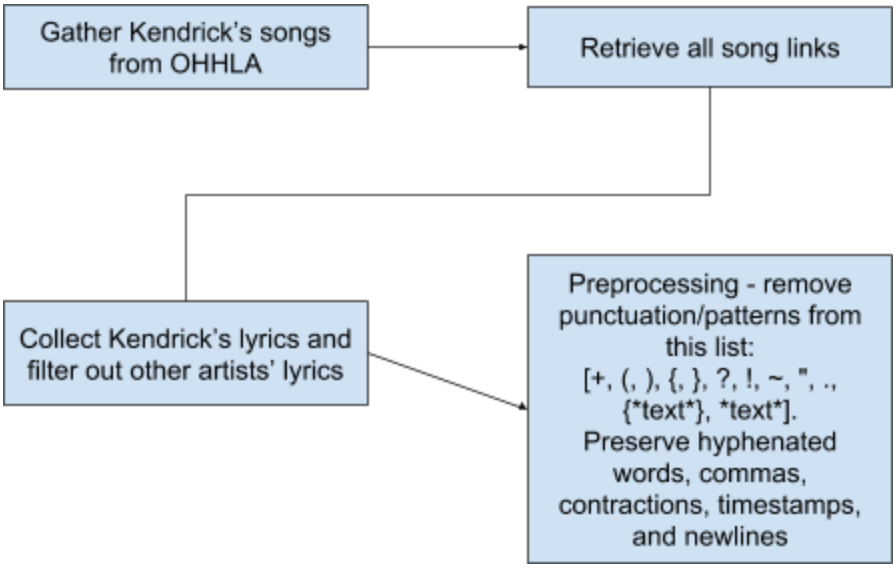
3. Data Sets [at least 1 page]

The main step in getting our data set was to create a corpus containing all of Kendrick Lamar's lyrics. We gathered all of Kendrick Lamar's lyrics from The Original Hip-Hop Lyric Archive (OHHLA) (http://ohhla.com/YFA_kendricklamar.html). We decided not to use Genius since we found that OHHLA had a more extensive list of Kendrick Lamar's songs and because Genius's data had a lot more noise. There were 262 songs in total, and majority of them were his, but he was also featured in a good amount of them as well. After going through all songs and preprocessing all of the lyrics, we were able to gather 9,188 lines from Kendrick, 85,435 tokens, and 11,005 unique words in our vocabulary set.

Number of Songs	Number of Lines	Number of Tokens	Number of Unique Words in Vocabulary

262	9,188	85,435	11,005

We preprocessed Kendrick's lyrics by ignoring song tags such as verse or chorus indicators, and lyrics of other artists. We also cleaned up some punctuation within his lyrics that we thought were unnecessary for the scope of this project, such as parentheses, question marks, exclamation marks, etc. This way, the text generated by our models would look a lot neater and less prone to error. For example, some words might be capitalized in the middle of a line or they might end with an improper punctuation when they are not supposed to. Hyphenated words, contractions, commas, and colons were preserved because Kendrick uses these types of words and symbols frequently, including words with abbreviated beginnings or endings such as "'em" or "drinkin'." Other examples of words with these symbols would be "mac-11", "ain't", "ol'", and "3:14".



4. Description of Technical Approach [at least 1 page]

As mentioned in previous sections, this project focuses on natural language generation, which revolves around working with sequential data and making predictions on what words and lines should come after another, given a particular seed. Popular algorithms and techniques used to solve this kind of problem include Markov chains and recurrent neural networks. In order for any model to generate natural language, it needs a corpus to learn from. This section will dive into the details of scraping OHHLA to create a corpus and implementing two different models to generate lyrics.

To go into more technical detail on the web scraping, we used BeautifulSoup to retrieve the links to Kendrick's songs and parse the HTML and text files. We created a Python script that utilizes Beautiful Soup to scrape data from OHHLA. We had to account for whether the lyrics were embedded in HTML or in a plaintext file since the contents within the links were not consistent. Aside from that, Beautiful Soup made it easy to parse the lyrics in each link into sections where each section is separated by newlines. From there, we iterated from section 1 to the last section to get all the lyrics, since section 0 only contains information about the song. In each section, we checked the first line to see if it contains Kendrick Lamar's name in it since the first line of each section usually contains the artist name. We then

added the lyrics to a text file if it did contain his name. If it did not contain Kendrick Lamar's name or another artist's name, then we use a boolean flag and regular expressions to check if the section is a Kendrick Lamar lyric and determine whether or not to add the lyrics.

The first model that we created as a base model uses a Markov chain to generate lyrics. In the beginning, we used Python's Markovify library to generate sentences. The results showed that the sentences were too proper for a rap generator and did not seem natural. The nature of rap is that the phrases aren't always grammatically proper and that words are combined in unique ways. We then decided to generate our own Markov chain. We first tokenized our corpus by spaces since we do not want to split contractions such as "i've" and "you'll" since these are words that Kendrick has used that we want to preserve for our text generation. Next we created a Markov chain that has the probabilities of words being generated after each word. We did this by first creating a count of how many times a word appears next to it in our list of tokens (since our list of tokens preserved the order they appear on the actual lyrics.) Then, we converted these counts into probabilities.

We also stored the parts of speech of each token to help us create semi-syntactically correct phrases. We did this by first automatically tagging each token using NLTK's automatic tagger. We quickly realized that there are a lot of words that have different parts of speech. To solve this problem, we took the 500 most frequently used words in NLTK's Brown corpus under the "news" category and stored their different parts of speech. Then, we calculated the most used part of speech for each word and if that word is in our set of tokens, then we use that part of speech. We do this to have a more accurate part of speech tag rather than just automatically tagging each token (since NLTK has an accurate set of tags for their stored corpus). Knowing our tokens' part of speech also helps us get rid of awkward ending for lines (ex. ending a line with a coordinating conjunction).

Using the Markov chain, we then generated verses by choosing a random word as the starting seed for each line then generating the next word based on their probabilities. For our rhyming scheme, we used a library called Pronouncing. For every even line, we generated a list of words that rhyme with the last word of the preceding line. From there, we chose a word at random and added it as the last word for even lines. Although we didn't check if its part of speech works correctly with the preceding word, we saw that it generally worked fine. However, we did check that the word chosen would not give us an awkward ending for a line (ie. a sentence that is cut off in the middle).

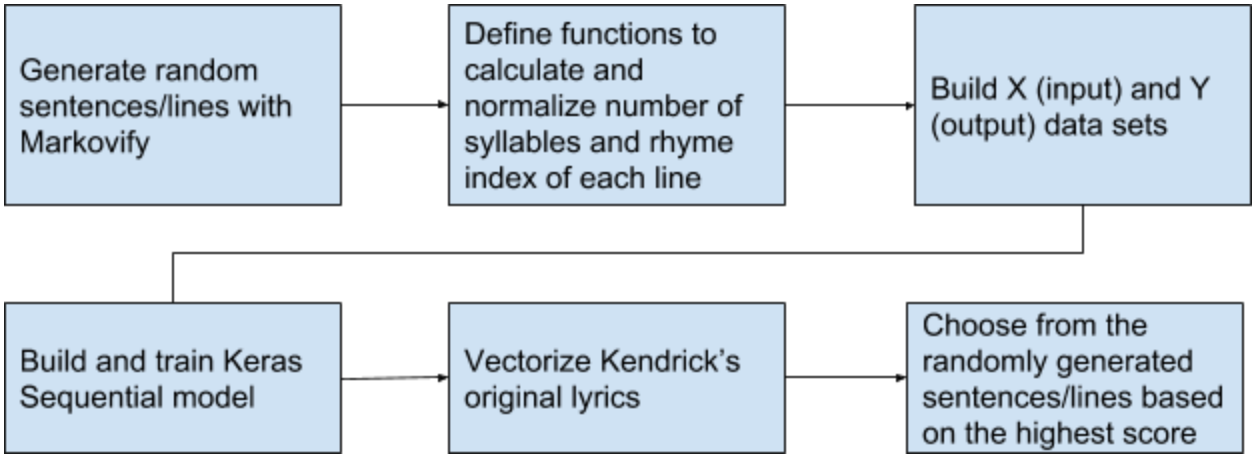
The second model is a Long Short Term Memory (LSTM) recurrent neural network model. This model builds on top of the base Markov model, although it does not use the base Markov model directly since the base Markov model has its own technique for generating lines and rhymes as mentioned above. The LSTM model uses the Markovify library that we initially wanted to use for the base model. Markovify suits the training and problem better because it enables random sentence generation with only a few lines.

The main purpose of the LSTM model is to decide which of the generated lyrics to write based on the differences between the original lyrics and the generated lyrics. The training is based on the ending rhymes and number of syllables in each line. Number of syllables per line are fairly trivial to calculate since it is a matter of detecting vowels in the line, considering a few edge cases. Each line's number of syllables would be normalized over a max_syllables variable within the model in order to limit the number of syllables per line. On the other hand, rhyme schemes are a little more complicated. We detected rhymes based on the last syllable of the last word in each line. For example, if the last word in a

line is “moment”, then the rhyme for that word would be “ent”. This is what we call a rhyme ending. With the aid of the Pronouncing library and its rhymes method, we are able to find rhyming words and rhyme endings associated with each word easily. We placed all of the rhyme endings from the original lyrics into a text file in order to reduce the amount of computation for our model. After placing all of the rhyme endings into a list, each rhyme ending has an numeric index, so we calculate the numeric rhyme value for each rhyme ending by normalizing it over the length of the rhyme endings list. In the end, this leads to a [rhymes, syllable] input and output data set, which is shown below.

$$\mathrm{X} = [rhyme_0, syllable_0]$$
$$\mathrm{Y} = [rhyme_1, syllable_1]$$

Once the input and output data sets are built, we then built and trained a Keras Sequential model since Keras handles a lot of the low-level neural network computation for us. We designed the model with an LSTM layer of 32 units, a 50% dropout layer to reduce overfitting, a dense layer of 16 units, and one last LSTM layer of 2 units for the output. We kept all return sequences in the LSTM layers in order to keep track of the hidden state input in each time step. Once the model is trained, we are able to vectorize Kendrick’s lyrics in order to make predictions for the generated lyrics. We pick a random line from Kendrick’s lyrics, and then predict what the next line would be based on the rhyme and syllable values in the previous line. The number of iterations we use for the vectorization is the number of lines that the AI will generate. After we compute the vectorized data, we use that data to decide which lines from our generated lyrics we want to use. Each line in the generated lyrics is scored against each line in the predicted lyrics based on rhymes, syllables, and creativity. We define a model as uncreative if it constantly reuses the same ending word throughout the rap, so we calculate an uncreative penalty which will be added to the score if this occurs. The line with the highest score is the one that we generate. After iterating through all of the vectorized data, then we write all the lyrics into a text file.



5. Software [at least ½ a page]

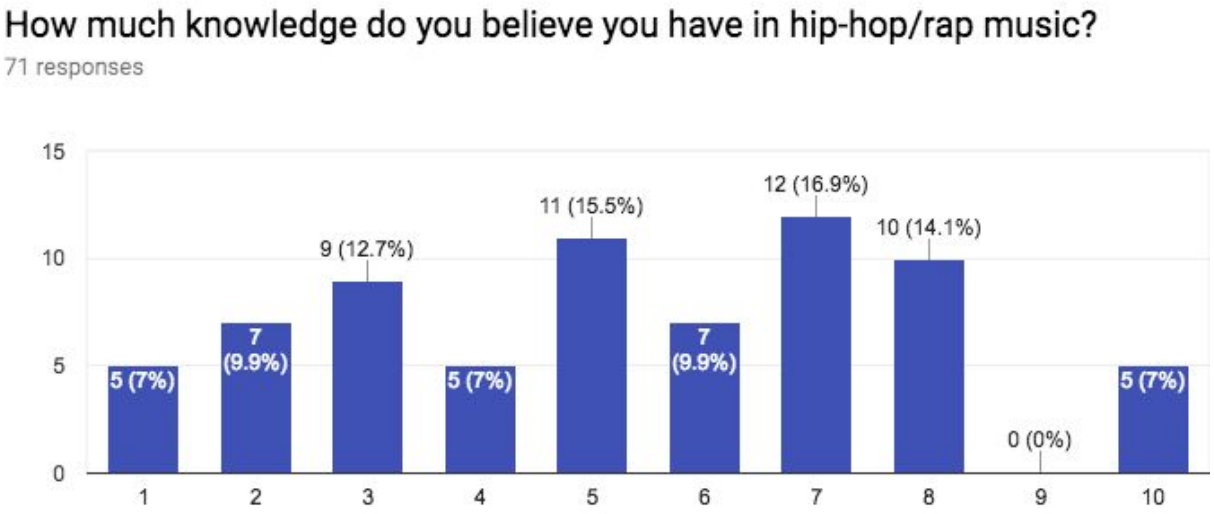
Code written by us	**Code written by others**

• Original Hip-Hop Lyrics Archive Web Scraper
  ○ Parsing lyrics, including stopwords, but not including stemming
  ○ Preprocessing - removing special punctuation
• Base Markov Model
  ○ Dictionary of transition probabilities
  ○ Line/verse generation including rhymes
• LSTM Functions (based on https://ritcsec.wordpress.com/2017/05/16/rap-lyric-generator/)
  ○ Creating a rhymes data set (txt file) using Pronouncing
  ○ Calculating normalized rhyme and syllable values to feed to the neural network
  ○ Rap vectorization for predicting
  ○ Vector-to-rap conversion
	• Natural Language Toolkit (NLTK) for Python
	• TensorFlow
	• Keras
	• Markovify (used for LSTM model)
	  ○ https://github.com/jsvine/markovify
	• Pronouncing
	  ○ https://pypi.python.org/pypi/pronouncing

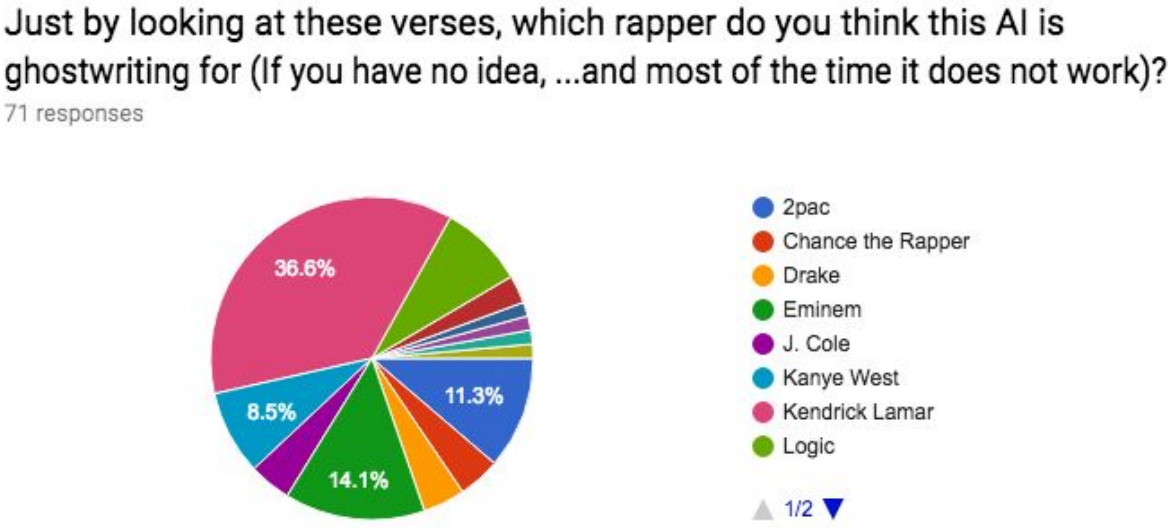
6. Experiments and Evaluation [at least 1 page, preferably 2 or 3]

The main mode of evaluation we used for this project was a user study. We could have evaluated models based on numerical features within the generated lyrics such as rhyme density, the total number of rhymed syllables divided by the total number of syllables. We could have also evaluated our models based on cosine similarity between the original lyrics and generated lyrics. The main reason we wanted to do a user study instead of these other evaluation methods was because we were able to easily tell how similar our lyrics were to the original lyrics just by looking at them. For the purpose of our project, a user study felt sufficient for evaluation. We used Google Forms to create a survey that required participants to answer questions based on the provided verses generated by our models. We had a total of 71 participants for this study. The survey contained four different sections.

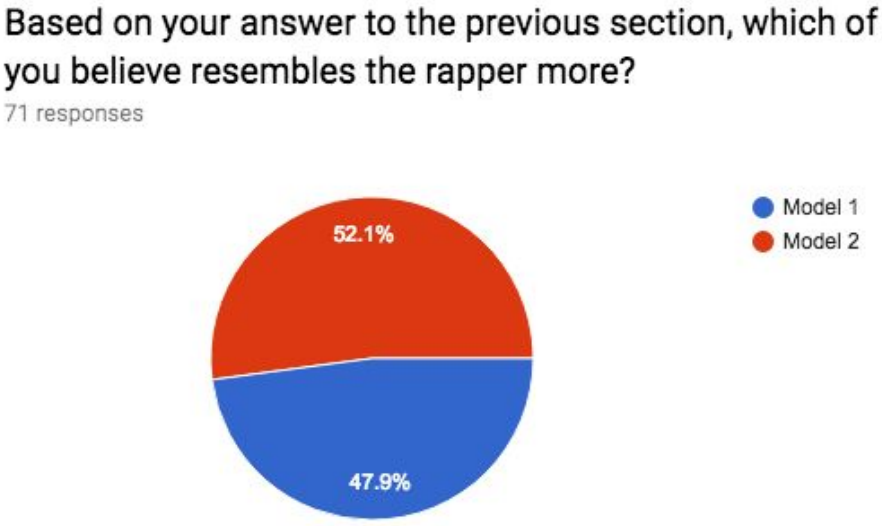
The first section asked participants how much knowledge they believe to have in hip-hop/rap music, ranked from 1-10. We thought this was an important question to ask because this helps us evaluate trends and correlations between the answers in the following sections. Among the people who have participated in our user study, knowledge in hip-hop had a somewhat uniform distribution. As a result we were able to get a variety of interesting and diverse data. Below is a graph that shows the distribution of hip-hop/rap knowledge within this study.

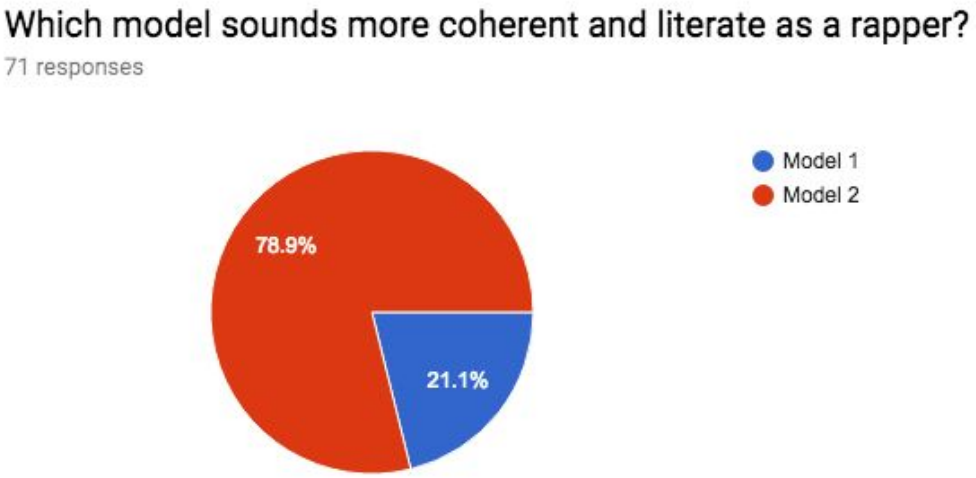


The second section asks participants to read a few verses generated by the LSTM model, and then asks which rapper the lyrics seem to resemble most. We wanted to use the lyrics generated by the LSTM model because we saw that the LSTM model seemed to resemble Kendrick's lyrics more than the Markov model. 36.6% of the participants thought the AI most closely resembled Kendrick Lamar, while 14.1% thought the AI resembled Eminem more. 2pac, Kanye West, Logic, and J. Cole were also among the top choices in the study, which makes sense since those rappers have the closest style to Kendrick Lamar compared to the others in the list. Based on these results, we felt like we achieved our goal to make an AI that can write raps in a similar style to Kendrick Lamar, and it gives us future ideas for which other rappers to train the model on for a more complex AI. Below is a pie chart that shows what kind of guesses participants have made, not including the second page of results since a very small amount of people guessed those answers.



The third section asks participants to read two verses, one from the the Markov model and one from the LSTM model and respond to the models' resemblance to Kendrick and literacy. This section serves to reaffirm our assumptions that the LSTM model does better than the base Markov model in generating new lyrics. Even if the participant may have not answered correctly in the previous section, in other words guessed a different rapper than Kendrick Lamar, this section would still serve its purpose by proving that one model does better than the other in terms of rap literacy and resemblance. We were surprised to see a near 50/50 split on resemblance between the base Markov model and the LSTM model. The base Markov model was clearly less coherent and literate than the LSTM model. Based on these results and our own judgment, we thought that the base Markov model had a more clever flow and rhyming ability in a general sense, but the LSTM model did a better job of specifically writing like Kendrick Lamar. These are two subtle differences that the participants may have not paid attention to. These results reminded us that rap is a very subjective, complicated form of art that has no restrictions on language and grammar.





The final section simply asks participants if they have any questions, comments, and/or concerns. The main purpose of this section is to see how we can improve our current implementation and to hear other peoples' input aside from what they were required to give. There were two helpful responses in this section. One person with a knowledge of 2 in hip-hop/rap music said "Good raps! I could hear the beats in my head and I don't even know much about rap." On the other hand, one person with a knowledge of 3 in hip-hop/rap music said "The raps are very hard to read without music, and its difficult to match a rapper without a flow,rhythm or beat." These are two very different perspectives between people who have a similar amount of knowledge in hip-hop/rap based on their own judgment. Knowing that some can hear the beats in their head while reading, while some cannot, one thing that we can implement in the future is a speech performance script, where the AI can actually rap the lyrics to a random beat.

7. Discussion and Conclusion [at least ½ a page]

This project gave us an introduction to web scraping with Beautiful Soup. We did not expect Genius to have a lot of noisy data, but we were glad we were able to find another extensive data set in the Original Hip-Hop Lyrics Archive. We also learned a lot about string manipulation and were able to brush up on our knowledge with regular expressions during the preprocessing steps of the project.

Most importantly, this project taught us how complex it is to build a coherent natural language generation model even though there are many algorithms and open-sourced libraries online. We learned the basics of Markov models and recurrent neural networks. Building a Markov model from scratch definitely gave us a more solid understanding of how they work and why they are good for this problem. Additionally, there was a huge learning curve for learning how to apply both models together for our specific problem. Although we wanted to try letting the LSTM model generate text by itself, we found that this led to poor results. Using the Markov model as a base and combining it with the LSTM model made a huge improvement in results. Just by looking at the lyrics generated by our AI, we were visually able to tell how well it learned from its original data.

There are some things to point out about our current implementation that can definitely be improved. The first thing is our method for detecting rhymes. Obviously, it is a flawed technique to detect rhymes by the last syllable of words because there are more to rhymes than detecting the last syllable of words and checking against those syllables in terms of spelling. In the future, we plan to detect rhymes based on phonemes, which will be much improved as it will move us closer to detecting slant rhymes and assonance within the lyrics. Another thing is the document and vocabulary length. Since there are only 11,005 unique words in the vocabulary, the models tend to generate certain lines over and over again after multiple runs. One way to fix this would be to add more lyrics from other rappers to make a hybrid AI that learned from multiple rappers, which would update the Markov chain. Another way to fix this would be to apply pre-trained word embeddings, which we had initially planned on doing, but we didn't think that was a major priority in comparison to the rhyme and flow of the newly generated text. The main goal of the project is for the AI to have a similar rap style as Kendrick by learning from his lyrics and flow, and not to have a more extensive vocabulary than him, but we plan on adding this feature in the future.