Automated Lyric Generation using Markov Chains and Recurrent Neural Networks

Author: Sean Vaughan - 119361923

Final-Year Project – BSc in Computer Science

Supervisor: James Doherty

Department of Computer Science

University College Cork

April 2023

Abstract

The main objective of the project was to build efficient lyric generation models. This project consists of two main models to generate lyrics. The first model is a Markov chain, and the second model is a Recurrent Neural Network (RNN). Both models are pre-processed on multiple datasets full of existing song lyrics with each dataset consisting of lyrics based on a specific genre. The Markov chain depends on lines created previously in the song as well as the current line when generating the next line of the song. When processing song lyrics one after another, the RNN keeps track of prior lyrics in its memory and uses that knowledge to guide the processing of future lyrics. Both models are implemented in Python and are integrated into a website using the Python web framework Flask. The website is also implemented using HTML, CSS, and JavaScript and the website allows users to generate their own songs by specifying what parameters they would like in the song such as the number of verses and how many lines in a verse.

Declaration of Originality

In signing this declaration, you are conforming, in writing, that the submitted work is entirely your own original work, except where clearly attributed otherwise, and that it has not been submitted partly or wholly for any other educational award.

I hereby declare that:

• this is all my own work, unless clearly indicated otherwise, with full and proper accreditation;

• with respect to my own work: none of it has been submitted at any educational institution contributing in any way to an educational award;

• with respect to another's work: all text, diagrams, code, or ideas, whether verbatim, paraphrased or otherwise modified or adapted, have been duly attributed to the source in a scholarly manner, whether from books, papers, lecture notes or any other student's work, whether published or unpublished, electronically or in print.

Signed: Sean Vaughan

Date: 23/04/2023

Acknowledgements

I would first like to thank my supervisor Dr James Doherty and my second reader Dr John Herbert who were able to advise me and provide feedback for my project.

I also want to thank all the lecturers who helped me learn so much during my four years at UCC and gave me a lot of the knowledge and abilities I used for this project.

Finally, I'd like to thank my friends and family for being there for me throughout the project.

Table of Contents

1. Introduction

The main objective of the project is to develop models that can efficiently generate lyrics. Before being able to address this objective it is important to discuss automated lyric generation today. One of the main questions that should be asked when discussing lyrics generation is this. “Are machines capable of generating engaging and meaningful song lyrics?” Perhaps the answer to this question is still unknown. However, there are many language processing tools and language models developed and being developed that can provide these lyric generation capabilities. Language processing tools are based on natural language processing. Natural language processing (NLP) is a branch of computational linguistics and artificial intelligence that studies how computers and human language interact. The goal of NLP is to make it possible for computers to comprehend and interpret human language in a way that feels natural and intuitive to people. One of the most prominent language processing tools now is ChatGPT. Based on the GPT-3.5 architecture, OpenAI produced the substantial language model known as ChatGPT. It is intended to communicate with people naturally and intuitively by generating responses to user inputs using NLP techniques (1). NLP techniques are used by ChatGPT to produce conversational answers to user inputs. The methods used by ChatGPT include named entity recognition and part of speech tagging. These methods empower ChatGPT to assess and comprehend the intent, context, and interpretation of user inputs, and produce conversational, natural-sounding responses. ChatGPT is also able to generate song lyrics in the form of real-life songwriters. However, this ability of ChatGPT has received criticism from singer-songwriter Nick Cave (2). “I understand that ChatGPT is in its infancy but perhaps that is the emerging horror of AI - that it will forever be in its infancy, as it will always have further to go, and the direction is always forward, always faster.” Nevertheless, it can’t be denied how accurate ChatGPT was in creating a new song in a similar style to Nick Cave. This can be shown in how accurate ChatGPT was in generating repetition in Nick Caves style at the start of multiple lines.

ChatGPT Response:

“I am the sinner, I am the saint, I am the darkness, I am the light, I am the hunter, I am the prey, I am the devil, I am the saviour”

Breathless by Nick Cave:

“Still, your hands, and still your heart, for still your face comes shining through, and all the morning glows anew, still your soul, still your mind, still the fire of love is true”

These lyrics comparison show that ChatGPT was able to reproduce Nick Caves' repetition style by producing lyrics with the same two words “I am” compared to the two-word repetition of “Still your” in the song Breathless.

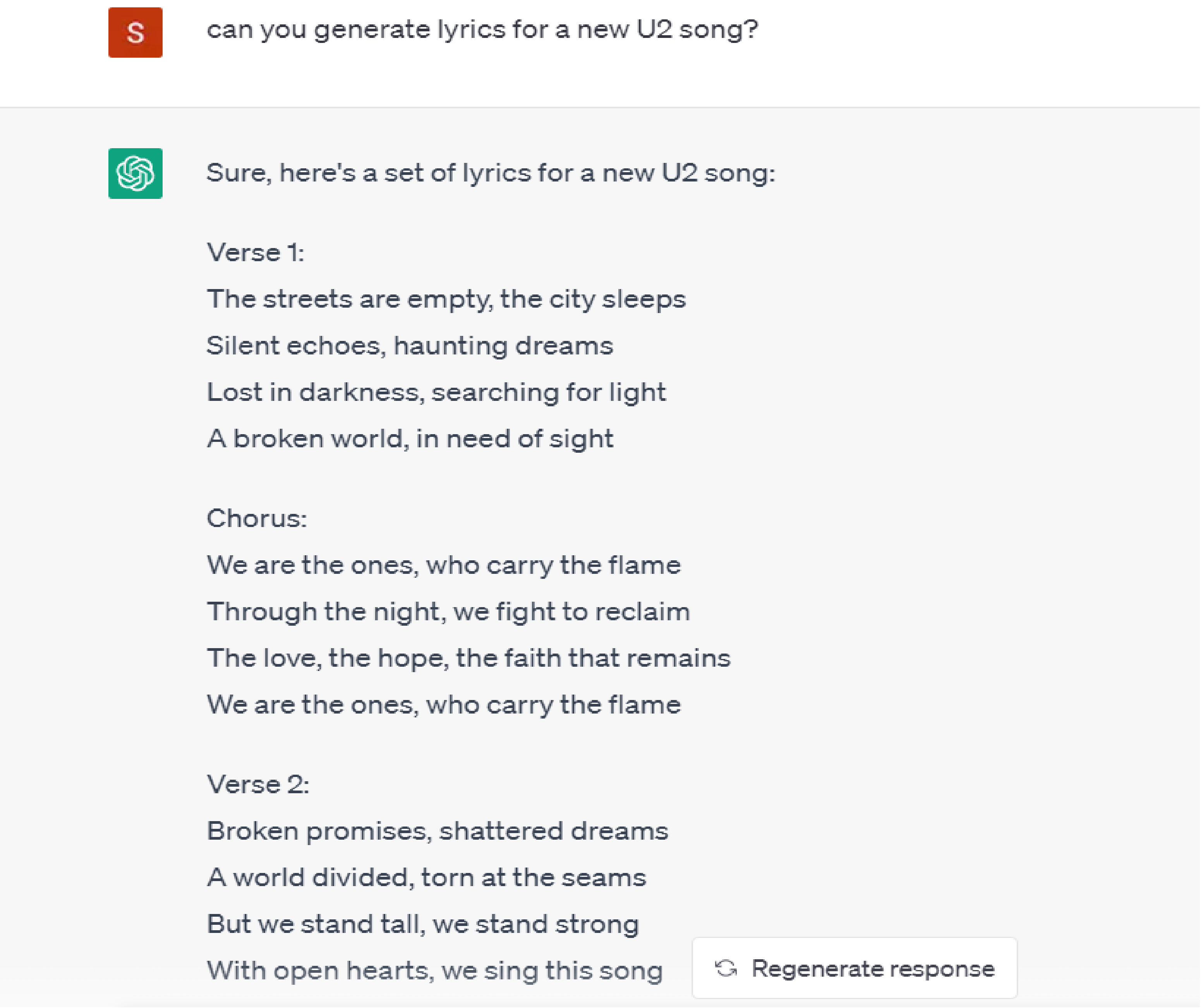


Figure 1: U2 song lyrics generated by ChatGPT

Figure 1 shows lyrics ChatGPT generated after asking the tool to generate a new U2 song. ChatGPT said that they generated these lyrics by analysing the lyrics from U2's songs and learning the styles and themes from their songs. It also investigated the common patterns in their song lyrics such as rhyming schemes and word associations. It can be seen how ChatGPT created these lyrics as the lyrics they created "Lost in darkness, searching for light A broken world, in need of sight" is like the lyrics of the chorus of U2's song "13". "If there is a light, we can't always see, And there is a world, We can't always be". This example shows the rhyming scheme similarities with end rhyme in lines 2 and 4 of both samples and how both samples have similar themes and word associations such as light and world. Language models such as ChatGPT have revolutionized song writing for the future in the way in which songwriters can now use these models to create their lyrics. Another up-and-coming language model is Google's "Music Language model" (MusicLM) which can generate high-fidelity music from text descriptions (3). Essentially, the model is an artificial neural network (ANN) that can generate text that's cohesive and realistic. MusicLM uses conditional music generation to produce its music however this model also supports conditional text generation. The model was trained to take genres of music and generate text such as lyrics based on this genre. These language models can generate lyrics in the desired style and with the necessary subjects by just receiving instructions from songwriters about the type of song they would like to make. So, these developments lead to many questions such as "Are language models becoming too intelligent?" "Is the art of song writing eventually going to die out?"

So, it is clear to see the impact ChatGPT and MusicLM have on automated lyric generation. Even though it was difficult to replicate the capabilities of these language models, knowing what these models have already accomplished helped achieve the project's main goal. ChatGPT uses the transformer architecture which is a type of ANN to generate its lyrics. However, even though it was difficult to capture the architecture's processing power, which was useful for analysing extremely large datasets, the neural network produced in the project improves on some of the aspects of the transformer architecture. This project uses a recurrent neural network that improves on the transformer architecture when using the Gated Recurrent Unit (GRU) layer for short-term dependencies for example. Researchers of MusicLM mentioned the need for further work in lyric generation along with improvements in text conditioning. This discovery initiated the idea of creating Markov chains to generate lyrics. A model using text conditioning relies on the input which can be a single word to generate the rest of the text. The Markov chains developed in this project are inspired by text conditioning. This is due to the first word of the Markov chain in this project affecting what the next word will be and this word affecting the word after that and so on. Increasing the order of the Markov chain was also investigated to show that the higher order would most benefit the text conditioning idea.

2. Analysis

2.1 Analysis of the transformer-based model

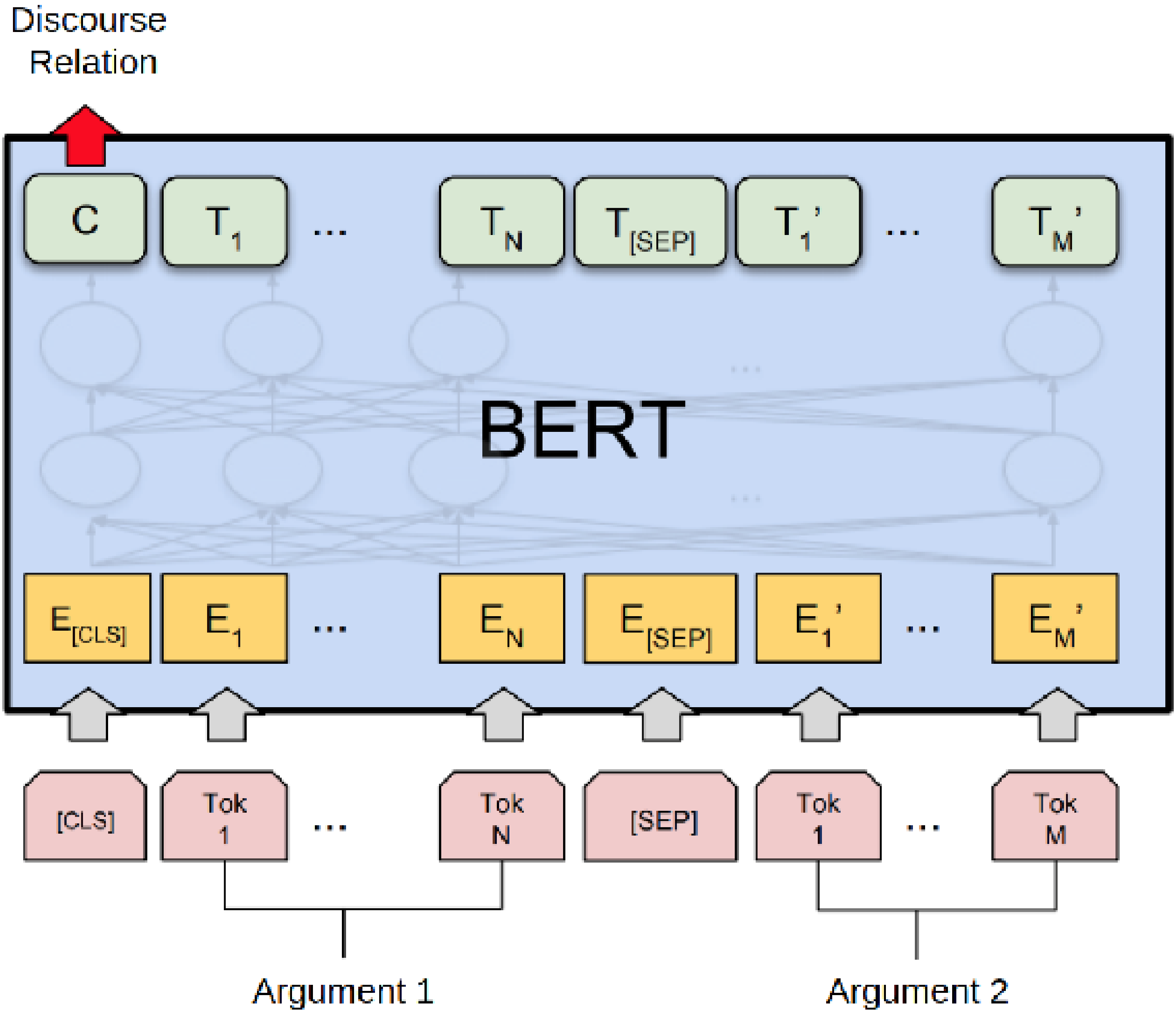


Figure 2: BERT Architecture for Next Sentence Prediction

Another prominent language processing tool is Google’s Bard. Bard aims to bring together the breadth of the world's knowledge with the strength, intelligence, and creativity of Google's extensive language models. It uses information gathered via the web to provide fresh, high-quality responses (4). It was found that deep neural networks are implemented in the Bard machine learning model to produce natural language content. The model's goal is to produce writing that is more life form and contextually applicable by examining the words that appear before and after every word in a phrase. Google has created multiple AI technologies that are incorporated in Bard, the main being Bidirectional Encoder Representations from Transformers (BERT). Deep neural networks are used by Google's NLP model BERT to grasp the meaning of individual words in a phrase by examining the words that precede and follow them. BERT is a particular sort of transformer model that may be adjusted for a variety of NLP tasks, such as text categorization, after being pre-trained on a sizable corpus of text. BERT is used in the project and was analysed as to how effective the transformer is in helping to generate lyrics.

It was found that BERT was trained on Wikipedia and the Google Books Corpus (5). Wikipedia and Google Books Corpus are both similar in the sense that both contain vast amounts of information on a variety of topics. However, the difference between the two is that Wikipedia is a shared platform where articles are published and modified by volunteers while The Google Books Corpus is a library of publications that have been digitized. With the help of this pre-training, BERT can discover how words relate to one another in context and can understand the meaning of words more effectively when applied in a sentence. BERT was specifically pretrained on two tasks masked language modelling and next sentence prediction. In an unsupervised learning activity known as masked language modelling (MLM), a model is developed to detect the absence of a word in a phrase and predict what word should take its place. Using the context of the environment, the model is trained to predict the masked tokens. To reduce the discrepancy between the model's projections and the actual tokens, the model is trained with both the actual and the masked tokens. A Masked language modelling approach can be utilized to help generate lyrics however this approach isn't implemented in this project. One main reason for this was that 15% of tokens of the sequences are substituted with masks in MLM (6). On the subset of masked tokens, it is discovered that masked language models produce independent predictions. This indicates that the model doesn't depend on any knowledge of the words or context around the masked tokens when making its predictions. Therefore, this leads to the next reason that the nuances of human feelings and experiences, which are portrayed in lyrics, are difficult for MLM to capture. This is because a lot of the time the lyrics MLM recommends using are synonyms of what the previous word was. When used to create new lyrics, synonyms may not always flow naturally into the original lyrics and may generate lyrics with little coherence or a clear message. Finally, an approach was required to generate lyrics with a particular structure and adhere to a specific genre. This is not always accomplished via masked language modelling, but can be with next sentence prediction.

BERT model's Next Sentence Prediction (NSP) task involves identifying whether two input sentences are consecutive. BERT is trained in NSP to distinguish between sentences that are randomly coupled and those that follow one another in the initial text. For BERT to comprehend the connections between sentences and the context where they appear, NSP aims at developing this skill. E denotes the input tokens' embedding, T is the target words, Tok is the token of the sentence, the complete input sequence is represented by the [CLS] token, and the two input sentences are separated using the [SEP] token (7). A binary classifier that can determine whether two input words are sequential or not is trained as part of the BERT architecture for (NSP). A classification layer and input embeddings make up the architecture's two primary parts. Two input sentences, A and B, are combined into one input sequence with [CLS] and [SEP] tokens by the input embeddings component of BERT NSP. These tokens serve as the classification layer's inputs. The last hidden state of the [CLS] token provides the input for the BERT NSP's classification layer, which generates an individual scalar value signalling if the input sentences are consecutive. It was discovered that utilizing next sentence prediction to create lyrics is advantageous since it can detect that the lyrics created are cohesive and have a logical progression. The BERT approach can help the lyric generation model since it allows the model to be adjusted on various genres and styles to generate lyrics that are appropriate for a specific context.

2.2 Objectives of the project

The first objective of the project was to build and pre-process a large dataset full of songs which will be used to train lyric generation models. This initially involved analysing multiple research and scholar papers to come up with a suitable method of building this dataset. The idea of having a dataset dedicated to a specific genre came about after finding this method being implemented in a past project. It was found that it would be suitable to collect many lyrics from one genre, but the dataset would only include songs by musicians who heavily used rhyme schemes (8). This method is beneficial as certain genres have lyrics with similar patterns which the model can learn to reproduce similar and effective ones itself. It was also found that it would be suitable to examine how the generated lyrics would change based on the genre by using songs from a variety of different genres (11). This variety is advantageous as the lyrics generated would become more flexible in terms of how distinct moods and themes would display in songs with different genres. The datasets used for this project consist of lyrics mainly gathered from Genius and Kaggle. Genius offered an API to access its lyrics on its website by gaining an access token and then using this token to make requests to the API. Even though there were limits to the amount of API calls that can be made in a day, the Genius API was highly beneficial. This was because it offered a vast collection of lyrics and provided the ability to gather ones associated with a specific genre. Therefore, creating the lyrics datasets was easier as each dataset could be associated with its own genre. Kaggle was very helpful in providing these lyrics as the website provides songs from artists from several different genres and how many of the text files consisted of tens of thousands of them which helped provide plenty of data to use to train the models. However, some disadvantages that came across using Kaggle were that multiple text files were found for a genre, for example, pop consisted of songs mainly from the same subset of artists. Therefore, many of the songs would have a lot of similar styles so it was important to try to get lyrics from as many distinct artists as possible to avoid many songs in the dataset being too similar. Another problem found was that many of the text files weren't pre-processed properly as they contained unnecessary characters and stop words throughout the files which is why the models to be created for the project had to be able to pre-process data efficiently.

The next objective of the project which was the main objective was to build models for generating lyrics and train these models on the datasets created. A Markov chain was one idea found to use to generate them. A system that varies over time in a stochastic manner is represented mathematically by a Markov chain. The state of the system in a Markov chain is determined by the state it was in at the previous time step, and the likelihood that it will change states solely depends on the current state. Markov chains are a great model for generating lyrics because by specifying how long the chain should be, Markov chains can be used to generate lyrics of various lengths, from brief phrases to whole songs. Another benefit is that by mixing words and phrases in extremely creative ways, Markov chains can be trained on several different genres to generate fresh and intriguing songs. A variety of methods were discovered from research papers to help in developing a Markov chain model for the project. A second-order Markov model that was trained on a dataset of songs and then used a sentence evaluator to validate the sentences the Markov model produced was one way to implement Markov chains for lyric generation in this project (11). This knowledge aids in the formulation of a Markov model for the project that will build on the concepts discovered in the research publications and explore the benefits and drawbacks of employing, for instance, a first-order Markov model and a second-order Markov model.

Recurrent neural networks (RNNS) were another idea found to generate lyrics. This type of ANN is made to process sequential data via a feedback loop in which the network's output at each time step is supplied back into the network as an input for the following time step. By updating its parameters for each time step of the sequence, RNNs learn to process sequential data of variable length, enabling the network to recognize patterns and interdependence of words throughout the whole sequence. An advantage of using RNNs is that they can learn to generate lyrics that have a cohesive structure, which means the result will seem more realistic. This allows RNNs to capture the contextual and syntactic relationships between words in a song. Like Markov chains, RNNs can generate lyrics in several different genres and styles. RNNs are a flexible tool for generating lyrics across various genres as they can generate ones that are suited to styles, emotions, or moods with the right training data and model modification. A variety of methods were discovered from research papers to help in developing a RNN for the project. One method to implement a RNN to generate lyrics was to have the neural network have an internal memory to store previously calculated results (7). Neural networks were also shown to be very efficient in learning the basic structure of rap lyrics written by humans. This information helped in discovering that a RNN can be implemented in various ways in Python to try generating lyrics in a fast but efficient manner by trying various layers in the “Keras” Python module such as “SimpleRNN “and “GRU”.

The final objective was to integrate the lyric generation models into a web application. This website would be implemented using the Python Web Framework Flask which includes HTML, CSS, and JavaScript. A relational database will then also be created to store information on lyrics, users, etc which would be queried using SQL. Users will be able to specify the structure of the song they want to generate in the application such as specifying the number of verses and the number of lines in a verse. Users would benefit from being able to create their own unique lyrics so they could observe how the structure changes when the parameters for producing them are changed. Changing the genre would allow users to generate more versatile lyrics, which would cater to their musical interests and improve the website's practicality. Research papers that were investigated show that users can specify the structure of their song such as the number of lines and words in their song (11). Also, there would be room for improvement in their research for example to have a model that generates different lyrics even if the parameters are the same. This knowledge would aid the development of the web application as it shows what users would like and not like to have in a lyric generation tool. Plus, the paper showed the factors to consider for the lyrics generated to be cohesive and engaging.

3. Design

3.1 Main Python Libraries

Several libraries and packages were used throughout this project. One of the main libraries is Natural Language Toolkit (NLTK). This library works with human language data and provides many text-processing libraries for tokenization, tagging, semantic reasoning, etc. NLTK has a tokenizer package that includes a method called “word_tokenize(“hello”)” which is used to find the punctuation and words in the inputted string “hello”. This method can be very useful for checking for end rhyme and alliteration in sentences as it is faster and more effective than string manipulation. This is because “word_tokenize” uses the Punkt sentence tokenizer model which helps separate some punctuation from words. For example, the Punkt model would recognize that in the sentence “The End.” the decimal point is not included with the word end. NLTK also has a tagging package that includes a method called “pos_tag(sentence)” which is used to label all the words in the sentence and describe what type of word it is. So, for example in the sentence “wet rainy day”, wet and rainy are labelled as “JJ” which is the symbol for an adjective in NLTK and day will be labelled as “NN” which is the symbol for a noun. The “pos_tag” method is very helpful to understand what type of words were used to create one lyric and can then use a similar structure to generate the rest of the lyrics in a verse for example. NLTK also has a “sentiment.vader” package which includes a method called “SentimentIntensityAnalyzer.polarity_scores(sentence)” which is used to find the sentiment strength of a sentence to determine whether the sentence is positive, negative, or neutral. A feature of the website allows the user to choose whether they want to have positive or negative emotions conveyed in the lyrics generated and this method is important for implementing this feature. This method can be implemented by checking if the positive sentiment of the lyric is greater than 50%, the lyric can be assumed to have positive emotions. Thus, if the negative sentiment of the lyric is greater than 50%, the lyric can be assumed to have negative emotions.

Another main library being used for the project was TensorFlow. TensorFlow is used for developing and training machine learning models. This library also provides many packages for creating layers for neural networks, checkpointing deep learning models, and inserting additional dimensions into Tensors. Recurrent Neural networks can be created with TensorFlow using the layers package which includes methods such as “Embedding” and “Dense”. The Embedding layer takes an integer matrix as input and creates fixed-sized dense vectors to be used in the Keras model. This layer is beneficial in taking several sentences as input and can then implement word vectors which can be processed by a “LSTM” or “SimpleRNN” layer for example. The dense layer is a deep neural network layer that is used to create an “m” dimensional vector where m is the size of the input. The layer is beneficial as it can take each word generated from an “LSTM” layer and create a probability distribution over the vocabulary for each word from the “LSTM” layer. The “train” package in TensorFlow is used in this project for saving and restoring deep learning models but the package can also be used to model and customize the training of machine learning models. The model can be saved by using the “ModelCheckpoint” method as part of the Keras package which takes a file path as input to know where to save the Keras model checkpoint. The most recent checkpoint can be loaded by using the “latest_checkpoint()” module as part of the “train” package. These methods all benefit the project in generating lyrics using a neural network and saving regular checkpoints so the latest checkpoint can improve the model further with updated weights which can be used to generate more accurate lyrics. Additional dimensions can be inserted using “TensorFlow.expand_dims()”. This method is beneficial as adding a dimension will not change the data in the tensor and the data can be reshaped to fit the requirements of a neural network model.

This method will be made useful in the project as how it can easily take an inputted word and add an extra dimension so that the inputted word will be reshaped so the model can use it to generate the rest of the lyric.

Another library important for both the Markov Chain and Recurrent Neural Network model was NumPy. NumPy is used for data analysis and computer science and large, multi-dimensional arrays and matrices are supported, as well as a variety of mathematical operations can be performed on these arrays. In particular, the transition probability matrix of a Markov Chain is represented using NumPy. The probability of changing states in a Markov chain is expressed in the transition probability matrix. The most common way to express it is as a two-dimensional NumPy array, in which the entries stand in for the probabilities of transition and the rows and columns represent the system's potential states. This way is also used for this project. NumPy is used in the recurrent neural network model by pre-processing data and creating input sequences. Pre-processing the input data is frequently required before training a RNN and includes actions like tokenizing text and translating words to numerical forms. With huge datasets, NumPy can be utilized to carry out these tasks quickly. NumPy can be used to generate the fixed-length input sequences that RNNs need from the raw data. For instance, sequences of text data that overlap can be produced using NumPy's array-slicing feature.

Plenty of other Python libraries were used which helped in generating rhyme, fixing grammar, and generating sentences. The “Pronouncing” can be used to find many words that rhyme with a word given by a user (12). The rhyming words can be produced by applying “the .rhymes()” method. So for example by printing “pronouncing.rhymes(“day”)” a list is returned with the words that rhyme with the word day. This is an important finding as this method will be adopted to generate lyrics that will consist of end rhyme. So once one sentence is created, for end rhyme, the last word of the resulting sentence will be inputted into the “.rhymes()” method, and then the next sentence to be created can end in a rhyming word. “language_tool_python” which is also effective in fixing grammatical and spelling errors (13). This is because the library can check any inputted string for grammatical and spelling errors and allows the user to check what errors the library found in the system and get the library to fix these errors and return the sentence after these changes. Another Python library found was “gingerit”. This library is pretty much like “language_tool_python” as it takes an inputted string, checks for grammatical errors, and then outputs the string after grammatical errors have been removed. Some Python libraries were also found to create sentences. The first library is “keytotext” which takes keywords that will appear in the resulting sentence as inputs and then creates a sentence with these keywords (14). A second library found was “randomsentence” (15). This was like “keytotext” as it also takes keywords that appear in the generated sentence as inputs and prints the resulting sentence with the inputted words. However, one difference compared to “keytotext” is that the word inputted will always appear in the generated sentence in order. Another difference is that this library can be run multiple times and will always create a different sentence when the inputted words are the same. This is not the case in “keytotext” as the sentence generated will always be the same.

3.2 Use cases of the website

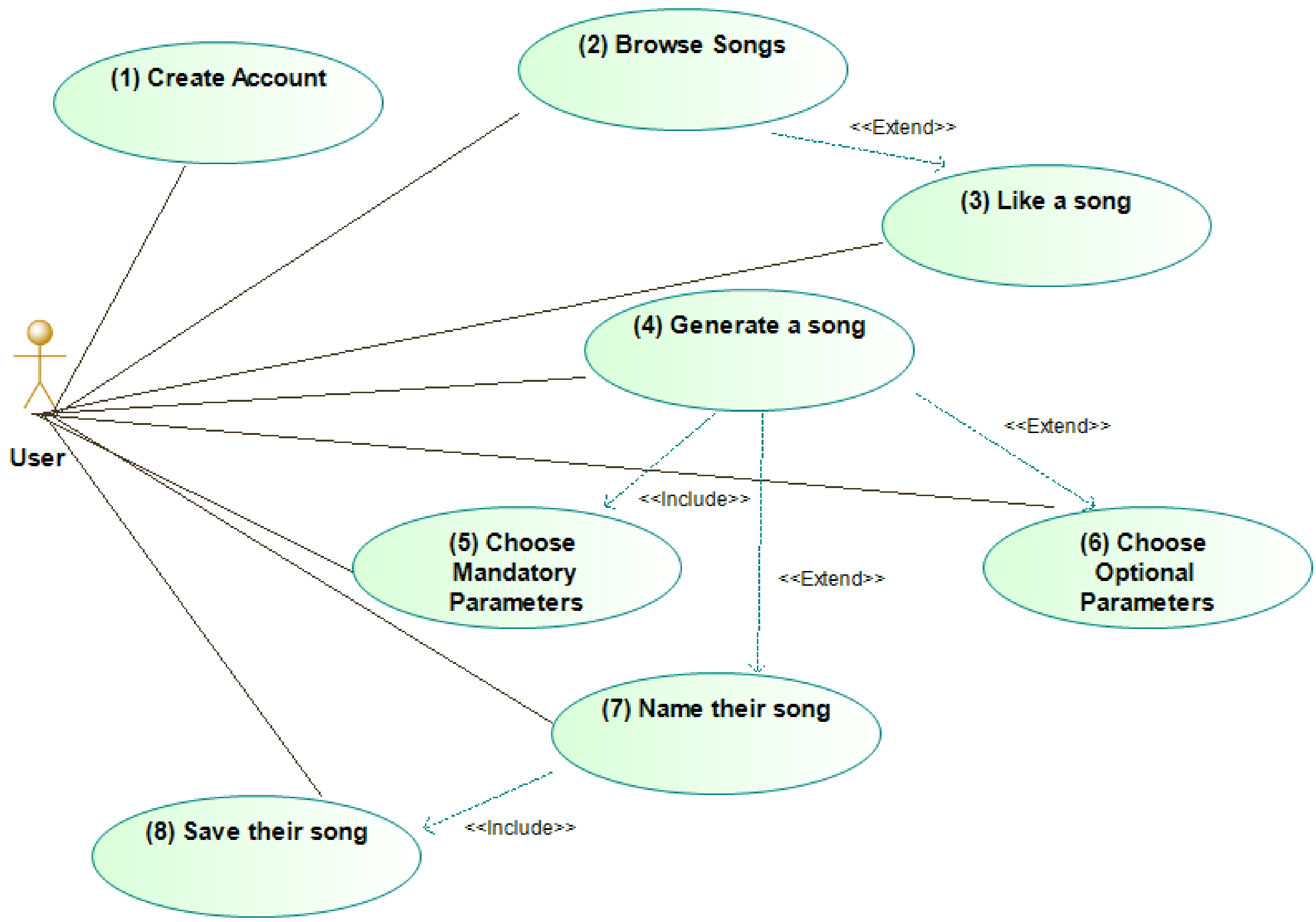


Figure 3: Use case diagram

UC1. Create an Account
A user can create an account by clicking on the sign-up button on the landing page of the website and by entering a valid username and password in the sign up page.

UC2. Browse Songs
A user can search for a song by clicking on and then typing the name of the song into the search bar. A user can also search for a song by selecting a desired genre from the genre dropdown menu beside the

search bar. After searching for the song the user can browse the results from a list of cards

UC3. Like a song
A user can like a song by clicking on a card from the card list which includes the details of the song and clicking the like song button on the song description page.

UC4. Generate a song
A user can generate their own song by choosing whether to generate the lyrics using a Markov chain or a RNN, and then choosing what type of parameters they would like in the song and clicking the generate song button.

UC5. Choose Mandatory Parameters
A user must choose specific parameters they want included in the generation of their song such as how many lines and the length of each line that will appear in a verse of the song.

UC6. Choose Optional Parameters
A user will have the option to include extra parameters in the generation of their song such as deciding on the sentiment of the lyrics being generated whether the lyrics will be positive or negative for example.

UC7. Name their song

A user will have the option to name their song by typing in an input bar what they would like their song to be called however the system can generate a song name for the user if they would prefer this.

UC8. Save their song
A user must save their song by clicking on the "Save Song?" button for all the details of the song to be saved on the website.

3.3 Entity Relationship (ER) Diagram

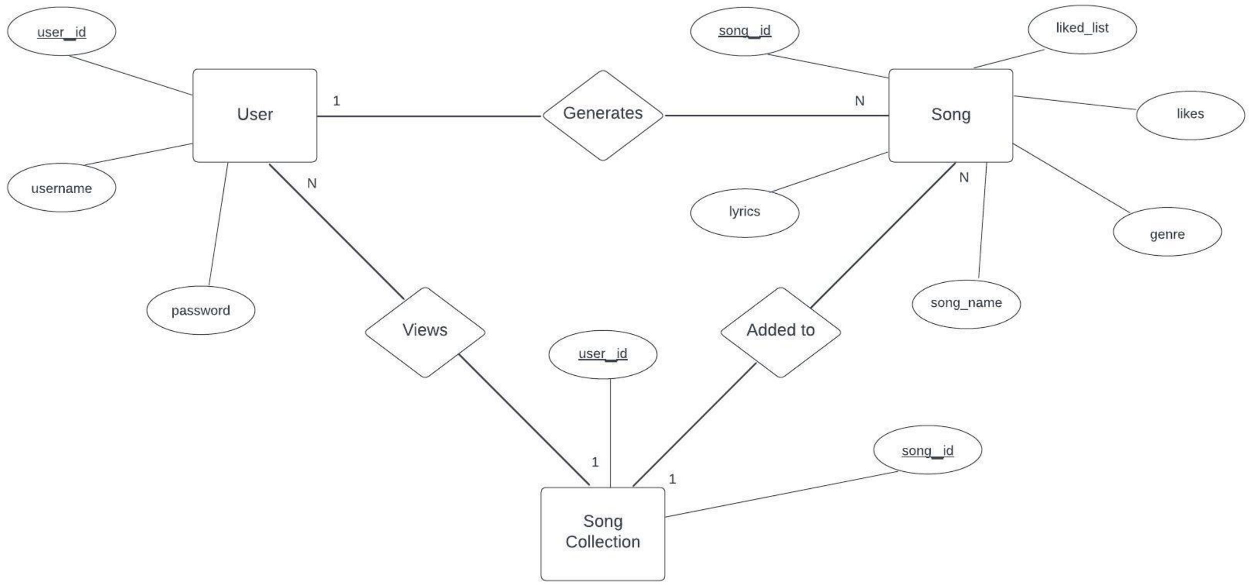


Figure 4: ER diagram

The main elements of the Lyrics Maker Architecture is a server created using Flask as well as a database created using SQL. The ER diagram consists of three entities which are User, Song and Song Collection. These entities represent the tables in the SQL database. The User entity has three attributes which are: “user_id” which is a primary key which uniquely identifies every user in the system and “username” and “password” which a user enters themselves while creating an account. The Song entity has 6 attributes: “song_id” which uniquely identifies every song in the system, “lyrics”,”song_name” and “genre” which are populated when a song is generated on the system and “likes” and “likedlist” where the “likes” attribute increments when a user clicks the “Like Song” button and the “likedlist” attribute is updated with the user who liked the song. The Song Collection entity has 2 attributes: “user_id” and “song_id” so the system knows every song created by every user and can display the relevant information on the search results page of the website. There are many key relationships throughout the Lyrics Maker Architecture which consists of these entities. These relationships are :”User - Generates – Songs”, this is where one user can generate as many songs as they want on the system, “Song – Added to – Song Collection” this is where multiple songs that are generated on the system can be added to the same song collection, “User - Views – Song Collection” this is where multiple users can view the same song collection at any given time.

4. Implementation

4.1 Implementation of the Recurrent Neural Network

The first step needed to be implemented is methods that can take any dataset full of existing song lyrics and create a sorted vocabulary list from this dataset. The first method is “fileToList”. This function turns the contents of the dataset, which are represented as a string, into a list of words. The input string is split at every character space, and each resulting word is added to the list. The list is then returned after any empty strings have been removed from the list using a list comprehension. This method accepts a string as input, which is a representation of a dataset's contents. It lowercases every character in the string and removes any tabs, newlines, and carriage returns. Then any stop words are removed by replacing them with spaces. These methods are adopted to read in a dataset which is a text file and pre-process its contents using text pre-processing. Then “fileToList” is adopted to turn the pre-processed text into a list of words. The map function is applied to remove any leading or following whitespace characters from each word in the resulting list. The sorted and set functions are then applied to construct a sorted list of the text's unique words, which is then stored in a variable which is the sorted vocabulary list.

The next step is to create a sequence dataset which is a TensorFlow Dataset object that represents a collection of input and output pairs of words. To create the sequence dataset, two dictionaries are first created to map between words and their indices in the sorted vocabulary list. These dictionaries are idx2words, which maps from index to word, and word2idx, which maps from word to index. The array created by replacing each word in the pre-processed text with its associated index in the vocabulary is kept in an indexed word variable. A TensorFlow dataset is created from the indexed word variable, which represents the entire text as a sequence of word indices. The target output word is included in sequences created utilizing the batch function that are one longer than the maximum input length. The input-output pairs are created using a “createInputOutputText” function. This function takes a list of words as its input and returns two lists. One list contains all but the last word, and another list contains all but the first word. The shuffle and batch functions are utilized to create batches of BATCH_SIZE pairs, with a buffer size of BUFFER_SIZE. Finally, the resulting sequence dataset contains batches of input-output pairs, where each input sequence contains all the words, and the corresponding output sequence contains the next word in the original text.

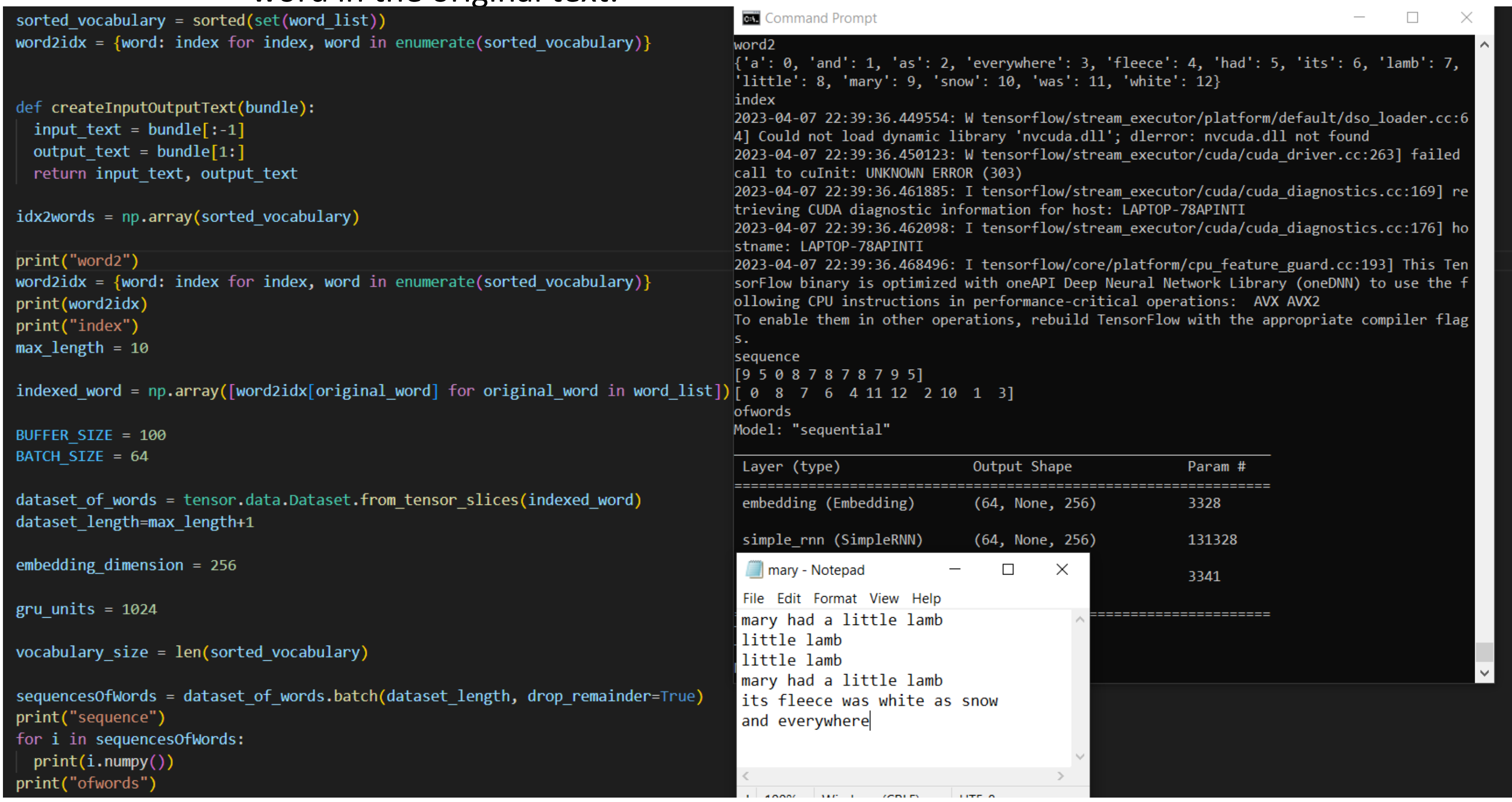
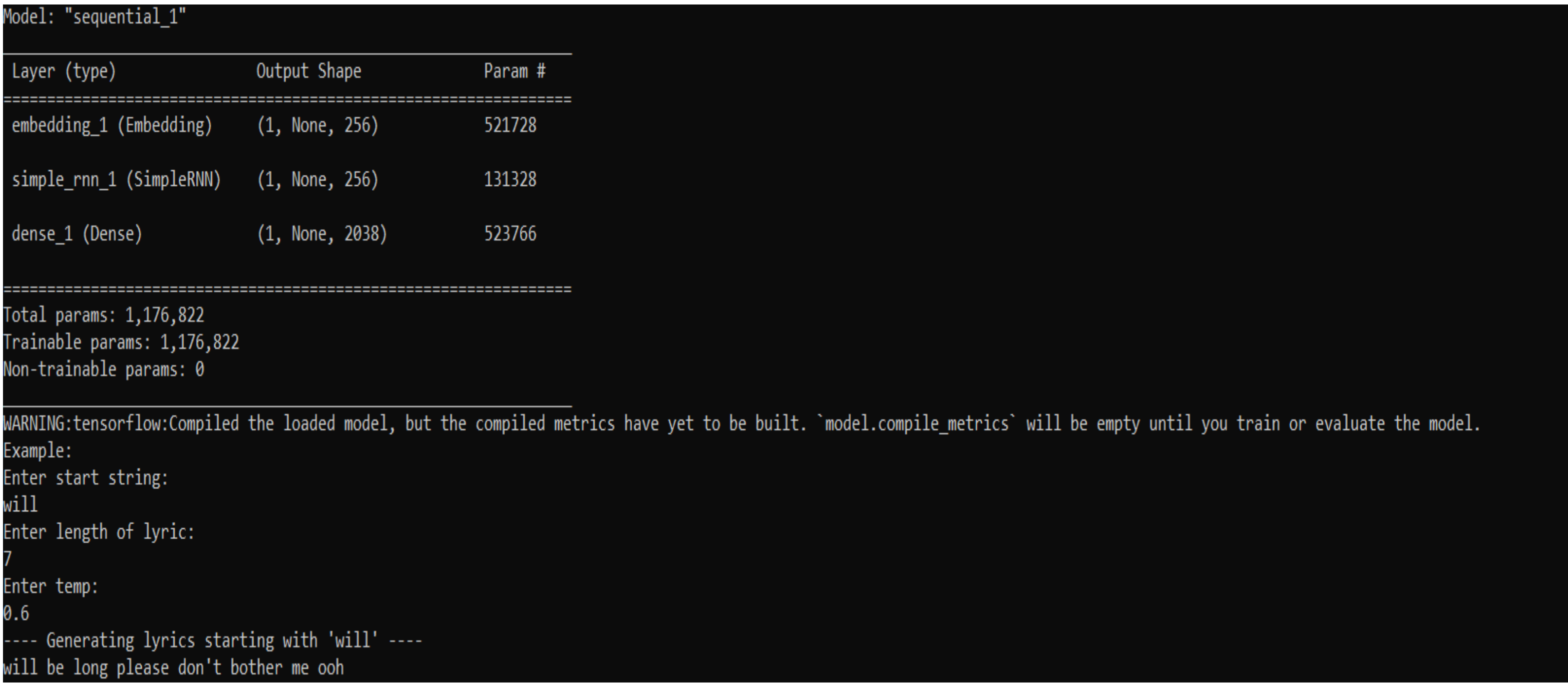


Figure 5: Terminal output for sorted vocabulary of “Mary had a little lamb” song

The next step is to create the neural network which can later be trained on the sequence dataset. The neural network is created by applying the Keras API in TensorFlow and will consist of layers so the network can learn convoluted patterns and relationships between words in the data by using layers to obtain characteristics from the input data. Three neural network models are created in this project, all consisting of an embedding and dense layer, but each model consists of a different layer which is either LSTM, GRU, or SimpleRNN. The embedding layer is utilized to understand low-dimensional interpretations of data that is distinct words in this case and words are expressed as vectors of integers that capture semantics. The sequence of vectors generated by the embedding layer is taken as input in the next layer, which is either LSTM, GRU, or SimpleRNN. The Long Short-Term Memory (LSTM) layer is created to recognize long-term dependencies and to maintain information over lengthy sequences of input data. The LSTM layer is made up of many memory cells interconnected by gates that manage the information flow. Every memory cell has a hidden state and a cell state, both of which are continuously changed based on the input data and the network's present state. The Gated Recurrent Unit (GRU) layer and LSTM layer are comparable in that both capture long-term dependencies in input data sequences. However, GRU has a simplified architecture with fewer parameters. GRU also has memory cells interconnected by gates, but it also has an update and a reset gate. The update gate decides whether the current input should be integrated into the hidden state, while the reset gate specifies to what extent the previous hidden state should be discarded. A mixture of weights of both the current input, as well as the hidden state from the previous time phase, determines the result of each time interval in the simple fully connected RNN architecture known as SimpleRNN. A bias term can also be added to the output of this layer to provide more flexibility in fitting with input data however this wasn’t used as it would increase the network's computational cost for both training and evaluating. The dense layer is a fully connected layer that is generally utilized as the output layer to translate the features acquired by the layers previously to a desired output form. A dense layer has no memory hence it won't retain track of prior inputs or outputs, compared to the recurrent layers which do. A dense layer is pretty much a feedforward layer as the outputs of the recurrent layers are simply converted into a convenient output format. The dense layer in the program builds a probability distribution over the language model's vocabulary using the output vector of the previous layer as input. To determine the likelihood of each word in the vocabulary, every element of the output vector is passed through the layer with the number of neurons being the size of the vocabulary and a SoftMax activation function. Following the creation of the neural network, the entire language model will be trained using the previously created sequence dataset.

The final step is to train the neural network model to generate lyrics. A loss function is built which decides how many training epochs there will be. To determine the difference between the model's projected and actual output, a loss function is used. This loss function is required because, in order for the loss to be reduced, the parameters of the model must be adjusted during training, and updating the parameters requires calculating the loss. Additionally, a checkpoint directory is created using Keras' ModelCheckpoint callback to record the trained model's weights after each epoch. If the training procedure is halted, the model can be continued from the latest saved checkpoint by utilizing the checkpoint directory, which is employed to save the model weights. The model is then trained on the sequence dataset by applying the "fit()" method. The model gains the ability to modify the layer weights during training to reduce the loss function. As a result, the “fit()” method provides a History object that gives details about the training process, including the loss and accuracy at every epoch. Subsequently, a new model that applies the same architecture as the preceding neural network is loaded with the most recent checkpoint weights. Now that the neural network model is fully trained, it will be used to generate lyrics. A function is defined to take in the pre trained language model and the length of the lyric desired as parameters. The randomness of the resulting lyric during generation is controlled by a parameter called temperature. The temperature parameter regulates the amount of randomness in the choice of the next word in the sequence from a probability distribution of all the potential following words. When the temperature is greater, the model is more likely to choose words that are less likely to occur, whereas when the temperature is lower, the model is more likely to choose words that are more likely to occur. The first word of the lyric must now be decided. This word can be a word picked at random from the lyrics dataset. Whitespace must therefore be applied as a delimiter to partition the word into separate tokens. Each token is then assigned to its corresponding integer index in the word-to-index dictionary created earlier to create a list. This list of integer indices is then adopted for producing new lyrics by presenting the language model with its starting input. In order to take into consideration, the network's batch size, which is 1, a new dimension has been added to the list to cater for each new lyric generated. A sequence of words is now generated in a loop until we reach the end of the sentence which is specified by the length of the lyric parameter mentioned earlier. The model estimates the probability distribution of the following word based on the current input sequence (the list of integer indices) for each iteration of the loop. The expected output's randomization is then modified by applying the temperature value. By using the TensorFlow “random.categorical” function, the following word is chosen according to the probability distribution, and its associated index is added to the list for the following iteration. Finally, the generated words are concatenated with the first word and returned as a single string.



```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (1, None, 256)            521728

 simple_rnn_1 (SimpleRNN)    (1, None, 256)            131328

 dense_1 (Dense)             (1, None, 2038)           523766

=================================================================
Total params: 1,176,822
Trainable params: 1,176,822
Non-trainable params: 0
_________________________________________________________________
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
Example:
Enter start string:
will
Enter length of lyric:
7
Enter temp:
0.6
---- Generating lyrics starting with 'will' ----
will be long please don't bother me ooh
```

Figure 6: Terminal output showing the summary of the recurrent neural network and a lyric generated by the model

4.2 Implementation of the Markov Chain

The first step is to create a Markov chain model based on a dataset full of song lyrics. The text file's contents are read, the text is divided into words with regular expressions, any empty strings are omitted, and the results are stored in a list. Regular expressions play an important role because they enable the matching of complex patterns like hyphenated words, which are difficult to split using conventional string-splitting methods. Plus, unwanted characters, such as punctuation marks, can be removed by applying regular expressions without affecting how the text is analysed. The Markov chain is stored in a “defaultdict” of “defaultdicts”, and the inner “defaultdict” will track the number of times each word that follows a particular word appears. When a key is absent, a “defaultdict” subclass of the built-in Python dictionary assigns a default value rather than generating a “KeyError”. “Defaultdict” is beneficial because, in the setup of a Markov chain, it is frequent to come across fresh state transitions that have not yet been seen, therefore these keys can be initialized with a default value. The list of words is looped around, and the Markov chain is updated with each word from the list. After looping over, the text file's words are applied to build a Markov chain model. The model can be implemented to produce fresh text that has a similar style to the original text.

The next step is to generate a sequence of words using the Markov Chain model. This generation can be done in various ways, but the methods shown in this project are generated using a first-order, second-order, or third-order Markov model. A first-order Markov chain is a series of random nodes where each node's probability solely depends on the value of the one before it. A second-order Markov chain is similar but in this case, each node’s probability depends on the values of the previous two nodes. A third-order Markov chain is also similar, but each node’s probability depends on the values of the previous three nodes. The first thing the function does is see if a starting word is given. Otherwise, it selects a word at random from the Markov chain. Every time the Markov chain is started, if we begin it in the same state, it will proceed through the same set of states, producing a fixed output. This is the reason why we begin with a random node so that the Markov chain has access to the whole state space as opposed to being trapped in a certain subset of states. A chain distance parameter is provided to the function which is the length the user wants the lyric to be. Each time the function is called the parameter decrements until it reaches zero which means the length of the lyric has now been met and so the lyric can be returned to the user. Only for the second-order and third-order model is values of the previous nodes needed. The function then enters a loop to gather the second-order choices and node weights from the previous two words, and the third-order options and node weights from the previous three words. It picks words stochastically from the keys of the Markov Chain dictionary until it obtains two words for second-order and three words for third-order that are not the same. These words are then added to a list that contains the previous words. In the first-order Markov model, the current node serves as the basis for the model's options and node weights for the following node. As a result, the node weights are the values, and the options are the keys in the dictionary connected to the current node. The node weights are adjusted to add up to 1 so that they represent a valid probability distribution. Once the previous nodes have been acquired for the second and third-order models, the function pulls the choices and node weights from the Markov chain dictionary. The keys in this dictionary relate to the nodes that can be reached from the previous nodes by transitioning to their neighbours. A new word is selected based on the options and their weights using "np.random.choice". The Markov chain's word list now includes this word. The chain distance is then decreased by 1 as the function recursively calls itself with the new word as the last node in the previous nodes array. Up until the chain distance reaches 0, this operation is repeated. When the chain distance has reached 0, the Markov chain is complete, and the string is returned which is the lyric generated.

5. Evaluation

5.1 Evaluation of the Neural Network Models

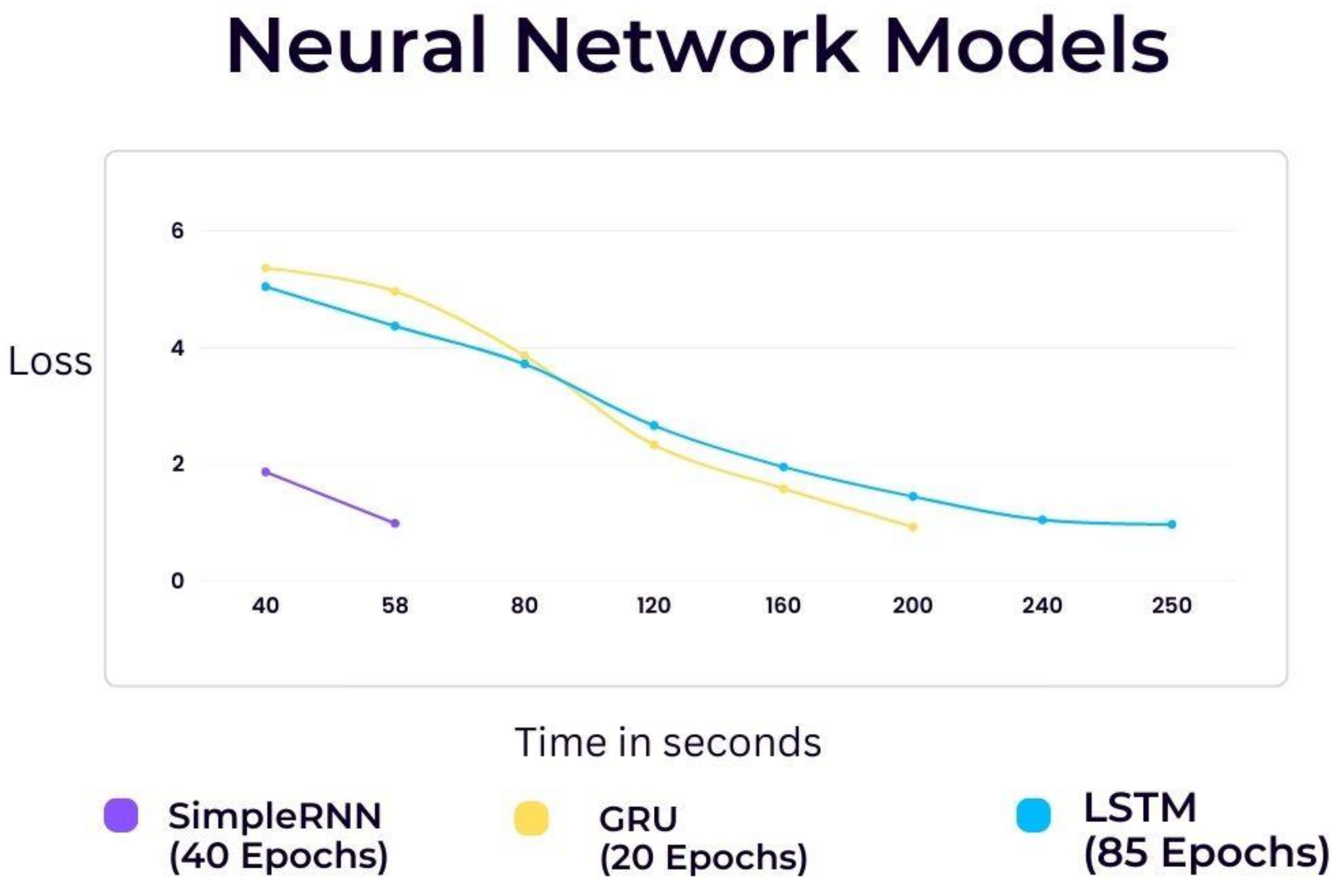


Figure 7: Graph depicting the time taken for the loss to reduce in three neural network models

Figure 7 evaluates how three neural network models fitted to a lyrics dataset of 3000 lines. All the models kept running until the loss got below one. The number of epochs refers to how many times the lyrics dataset was used to update the neural network. So, 20 epochs in GRU means the GRU neural network was updated 20 times and stopped updating once the loss got below one. Even though the LSTM model is more flexible compared to GRU and SimpleRNN in terms of how easy and flexible it is to modify its architecture, LSTM is the least preferred model for this project. One reason for this is due to the large number of epochs required to update the neural network. Even though a greater number of epochs leads to the neural network generalizing better to new, unseen data, this benefit comes at a greater cost for this project. The main cost is increased training time. This is shown in the diagram as LSTM takes the longest time to reduce the loss to below one and is over four times slower than SimpleRNN. Another problem with this large number of epochs is plateauing as the neural network may stop improving after a certain period and thus unnecessary amounts of time are wasted. A second reason for LSTM being the least preferred model is that they are computationally expensive. Since LSTM has a complex architecture and requires many parameters, LSTM is thus computationally expensive to use.

GRU on the other hand is a better model than LSTM for this project. Even though LSTM is better at modelling long-term dependencies than GRU, GRU still has several advantages over LSTM. One advantage from the diagram shows that after approximately 100 seconds, the GRU model is more efficient in reducing the loss and it takes 200 seconds to reduce the loss under one. GRU is also a better model as regards computational efficiency as it has a simpler architecture compared to LSTM and is less computationally expensive as it has fewer parameters to estimate. Another advantage of GRU is that it is effective at modelling short-term dependencies in sequential data. This makes them useful for tasks where understanding the immediate context is sufficient. GRU is also effective in modelling short-term dependencies in sequential data. The model is therefore beneficial for tasks where comprehension of the current context is sufficient. It is shown that GRU ran for only 20 epochs the lowest number from the three models. This low number has benefits and drawbacks, the main benefit being the reduced risk of overfitting. This risk is reduced because training can be stopped early epoch wise and this may make the model more reliable and resistant to minor changes in the data. The main drawback is that the model will perform inadequately in terms of collecting patterns in the data, which will result in poor performance in unknown data.

Finally, SimpleRNN is most likely the best model for the project. Even though both LSTM and GRU are better at handling long-term dependencies, SimpleRNN has many advantages that make this model efficient in generating lyrics. One of the biggest advantages is its training time which is very important when dealing with larger datasets of lyrics. SimpleRNN took 40 epochs to update its neural network which is a bit more than GRU but much less than LSTM. This means it won't suffer as badly as the problems LSTM has with plateauing due to the fact LSTM has many epochs. Plus, SimpleRNN won't suffer as badly the problems GRU has with the risk of overfitting due to the fact GRU has a small number of epochs. SimpleRNN is also effective because of how effective the model is in performing simple tasks such as predicting the next word from a lyric. This is the case as SimpleRNN has the least complex architecture of all as only a small number of parameters is needed so the model is easier to understand and modify when needed.

5.2 Evaluation of the Markov Chain Models

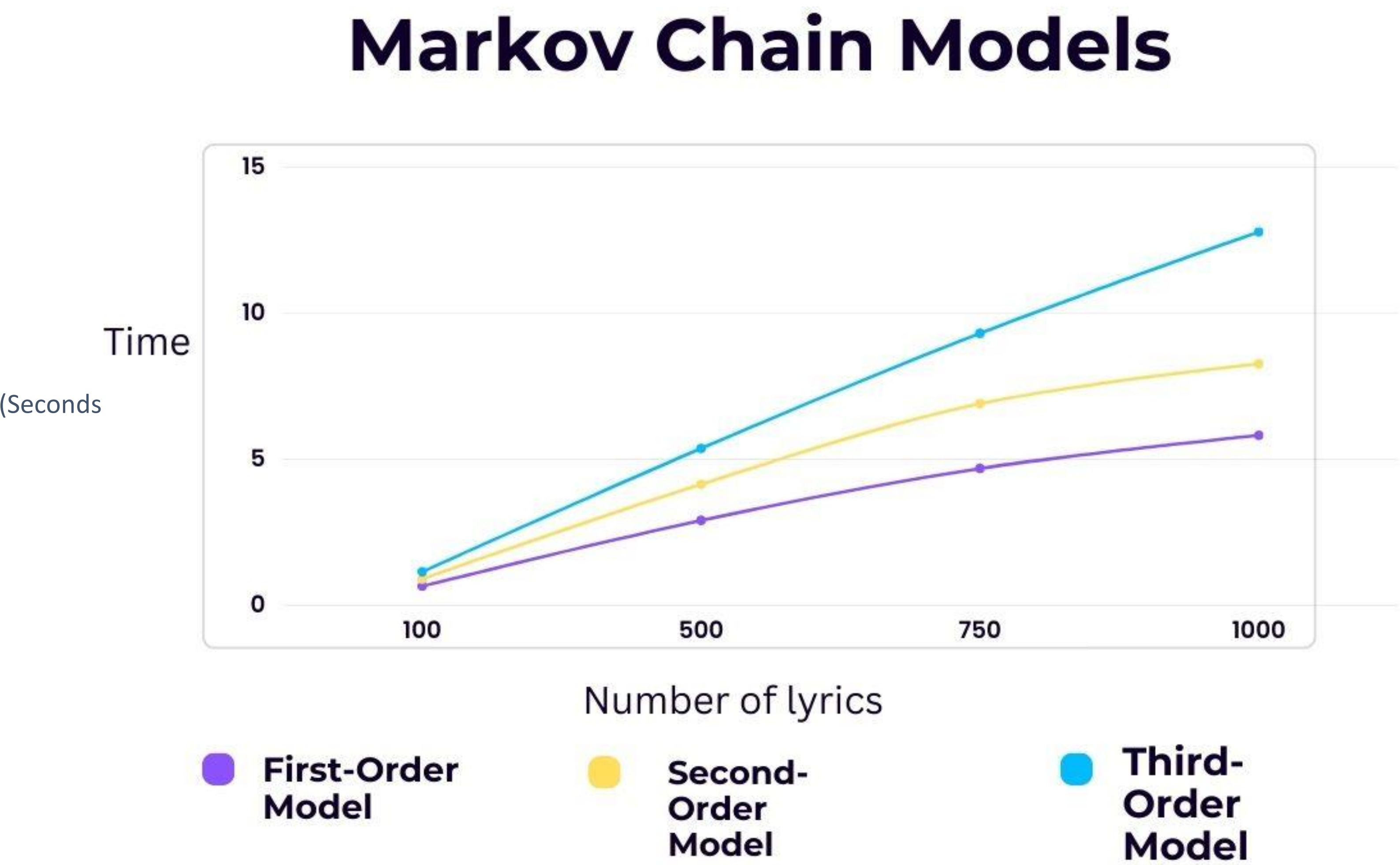


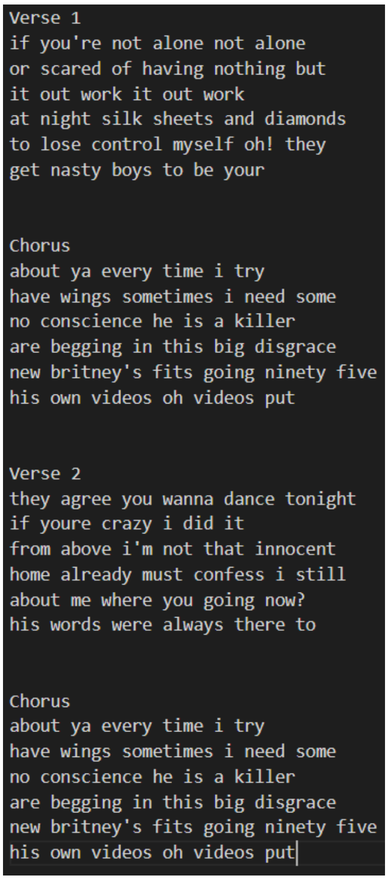
Figure 8: Graph depicting the time taken for three Markov Chain models to generate up to a 1000 lyrics

Figure 8 shows how long it took three Markov Chain models to generate a different number of lyrics all consisting of 100 words. An advantage of the first-order model is that it is the simplest and easiest model to understand. They are the easiest to understand because they simply take the probability of the next word depending on the preceding word. Another advantage of this model is that it was the fastest model to implement. This can be shown in the diagram as the model took just under 6 seconds to generate 100 lyrics compared to the second-order model which took over 8 seconds and the third-order model which took nearly 13 seconds. This is because first-order Markov chains use fewer computational resources and training data. First-order Markov models have the drawback of having minimal context. The context of any words that come before the current word and word before the current word is ignored by the model. This resulted in lyrics that were less coherent and sounded less genuine, especially for prolonged sequences. The first-order Markov model also generated a lot of repetitive lyrics. This repetition occurred when the size of the dataset was small (under 5000 lines).

However, the problems with first-order Markov models can be eased with second-order and third-order Markov models. This is because the second-order and third-order models can generate lyrics that are more contextual and also less repetitive. Lyrics that seem more natural and coherent were produced by basing the likelihood of the next word on the preceding two words rather than the preceding word. The generated lyrics using a second-order Markov model contained greater phrase structures, more sophisticated grammatical arrangements, and enhanced text conditioning. The same occurred with the third-order Markov model except for the likelihood of the next word was based on the preceding three words. Since the models can recognize and produce a greater variety of phrases and sentence patterns, this resulted in less repetitive lyrics being generated.

5.3 Evaluation of the generated lyrics

Both pictures were evaluated using a first-order Markov model. Figure 9 shows the results of a lyrics generation by joining sentences that rhyme with each other. Figure 10 shows the results when joining rhyming sentences using BERT's next sentence prediction and using NLTK'S sentiment analyser. Both figures used a dataset full of rap lyrics to train on. Compared to figure 9, figure 10 does a better job of capturing the language and style of rap music, but it still has some flaws. For example, a lot of rap lyrics contain much profanity , however the Python libraries used to help generate lyrics such as RandomSentence rarely produce any profanity (15). The module's reliance on predetermined lists of words or phrases, which might restrict the inventiveness and originality of the generated lyrics, is one of the main causes of this. However, by using NLTK’s Sentiment Analyser, lyrics with profanity are more likely to be chosen from a lyrics dataset if the user wanted lyrics to have a negative sentiment. This is shown in the second picture which features more profanity with words such as “fuck” and “bitch”. Figure 10 also shows the effect of Bert’s Next Sentence Prediction as the lyrics have a more gradual flow contextually. For example, the lines “murder with something if I am, When hearing the bam” both are lines that make sense when put together. When compared to lyrics from the first picture “home already must confess I still, about me where you going now? doesn’t make a lot of sense contextually when put together.



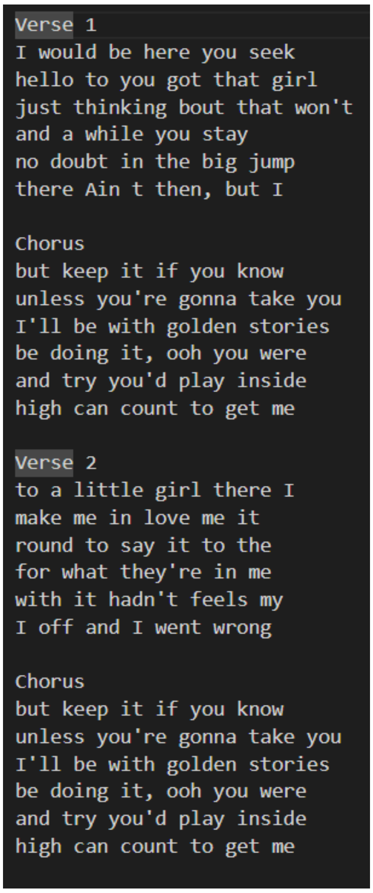
```
Verse 1
if you're not alone not alone
or scared of having nothing but
it out work it out work
at night silk sheets and diamonds
to lose control myself oh! they
get nasty boys to be your


Chorus
about ya every time i try
have wings sometimes i need some
no conscience he is a killer
are begging in this big disgrace
new britney's fits going ninety five
his own videos oh videos put


Verse 2
they agree you wanna dance tonight
if youre crazy i did it
from above i'm not that innocent
home already must confess i still
about me where you going now?
his words were always there to


Chorus
about ya every time i try
have wings sometimes i need some
no conscience he is a killer
are begging in this big disgrace
new britney's fits going ninety five
his own videos oh videos put
```

Figure 9: Lyrics without BERT



```
Verse 1
I would be here you seek
hello to you got that girl
just thinking bout that won't
and a while you stay
no doubt in the big jump
there Ain t then, but I

Chorus
but keep it if you know
unless you're gonna take you
I'll be with golden stories
be doing it, ooh you were
and try you'd play inside
high can count to get me

Verse 2
to a little girl there I
make me in love me it
round to say it to the
for what they're in me
with it hadn't feels my
I off and I went wrong

Chorus
but keep it if you know
unless you're gonna take you
I'll be with golden stories
be doing it, ooh you were
and try you'd play inside
high can count to get me
```

Figure 10: Lyrics with BERT

6. Conclusion

To conclude this paper, it would be important to reference some of the questions mentioned earlier such as “Are language models becoming too intelligent?”, “Is the art of song writing eventually going to die out?”. Well, perhaps the answer to the first question is yes, and has led to multiple countries banning language models such as ChatGPT and many CEOS around the world wanting a six-month pause to the training of these AI models (16). While language models can’t truly be able to feel emotions to comprehend the meaning behind the songs they produce, they can still produce convincing song lyrics. This was shown in the lyrics ChatGPT generated in the style of Nick Cave. The lyrics generated made Cave angry and said it made a mockery of what it is like to be human (17). As shown earlier the lyrics were able to be generated as the language model investigated Cave’s themes, word choices, rhyming schemes and so many more elements of his work to generate new lyrics in his style. However, it can be argued that these language models are currently relying on songs that have already been made and are copying the style of many artists when asked to be a user.

So this leads to the next question about the art of song writing eventually dying out. Well, this could very well depend on the development of AI systems in the next coming years. Perhaps a six-month pause was discussed as people are worried about what AI systems can evolve to. However even if there is to be a halt to this development, this halt can’t last forever. Many corporations might get impatient and could start developing systems even smarter than ChatGPT. It's hard to know. However, it is fair to claim that the utilization of language models in song writing may harm originality and creativity. Songs could run the risk of becoming formulaic if too many songwriters use language models to generate lyrics. Furthermore, there is a possibility that the human aspect and psychological feeling that relates to song writing may be lost if language models are used. However, no matter what lies ahead for song writing, it is hard to deny the value of such powerful language models to perform many tasks when prompted to. As mentioned before, knowing what language models such as ChatGPT have achieved benefited in achieving most of the objectives of the project.

The project was able to achieve one of its main objectives which was to build models for generating lyrics and train these models on the lyric datasets. Even though the lyrics generated by both models still had some problems after using grammatical tools and BERT to reduce these problems, the lyrics generated were then structurally correct. All lyrics consisted of a verse chorus verse structure and rhyming schemes were also incorporated into some of the lyrics when prompted. The Markov Chain models implemented were accurate in what a Markov chain represents in the sense that the model does generate lyrics based on the previous state in a stochastic manner. The RNN models implemented were accurate in what a RNN represents in how the model processes sequential data of variable length and generates lyrics in a cohesive structure. However, concerning how efficient both models are in generating lyrics depends on what a user thinks of the lyrics. Some people may find the lyrics of a song engaging, other people may not, and this highlights the art of song writing.

Perhaps this leads to the next objective which was to build and pre-process a large dataset full of songs that will be used to train the lyric generation model. This objective was partially achieved in the sense many datasets full of lyrics were created by gathering lyrics from websites such as Genius and Kaggle. The datasets were also able to consist of lyrics from specific genres due to the help of the Genius API and this did lead to diversity in the lyrics generated. However, the datasets couldn't be large as hoped for generating lyrics using the RNN models. This was clear as the GRU and LSTM models take a significantly longer time to train on a dataset than SimpleRNN. Due to the datasets not being large, this also led to some words repeating over time in lyrics which wouldn't have occurred as often if the dataset was larger.

Nevertheless, being able to achieve the objective of integrating the lyric generation models into a web application was a major accomplishment. This was largely due to the success of the Python framework Flask. Using Flask has the benefit of being a lightweight framework with few dependencies, making it simple to install and use. Additionally, Flask is modular, making it simple to add or remove functionality as necessary. The parameters mentioned earlier such as the user being able to specify the number of verses were all implemented. This also resulted in both models generating different lyrics even if the values of the parameters never change. Being able to implement BERT and sentiment analysis with NLTK resulted in the lyrics being more cohesive and engaging overall.

7. Appendix

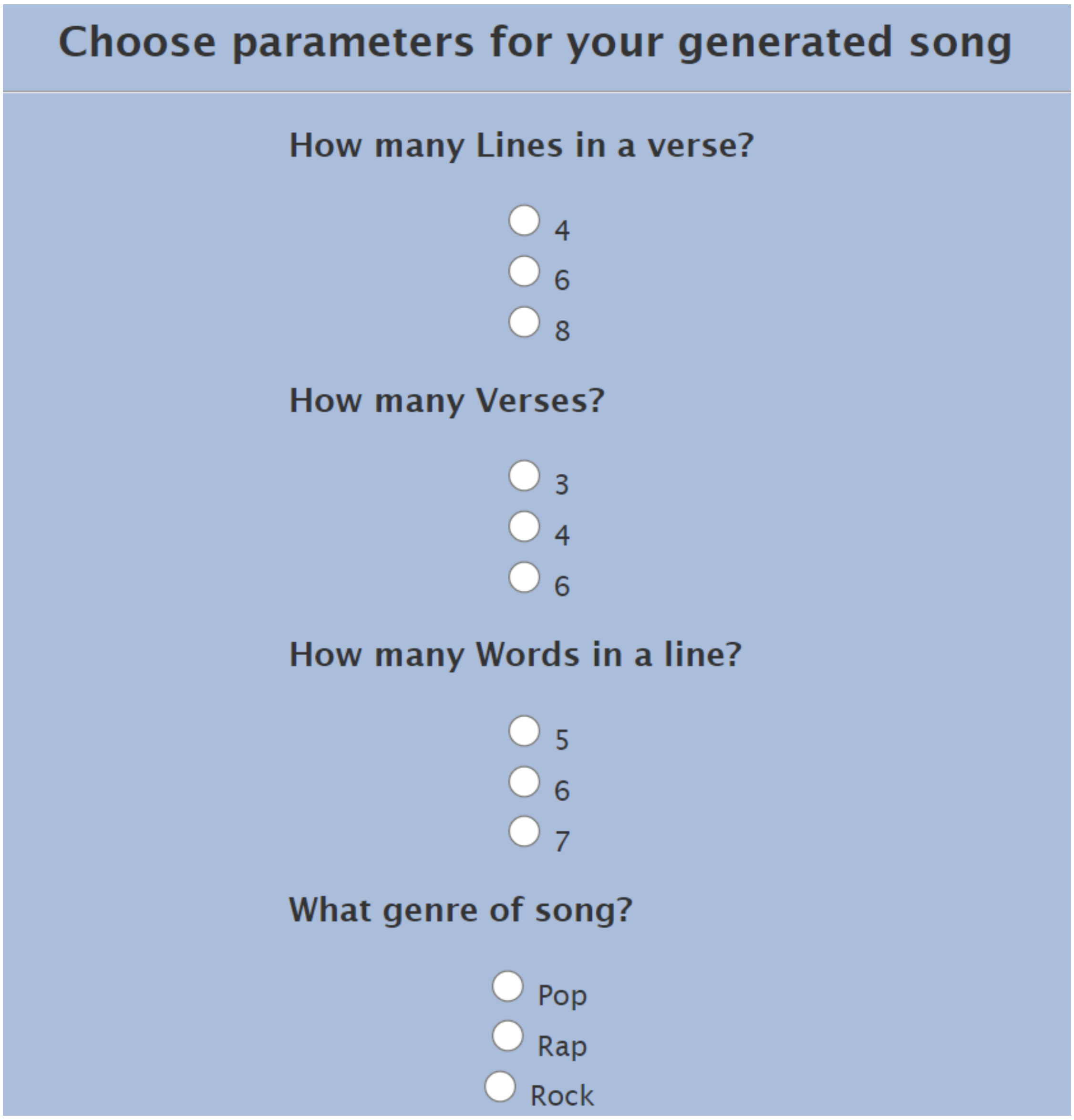


Figure 11: Sample parameters for a user to choose to generate lyrics on the Flask web application

Figure 12: Task bar to search for generated songs by typing the name or genre of the song

Verse 1

or heaven on the one more

the merry go for the master

like ashes to the wheel baby

were in the bullets? we were

Chorus

now i am clean oh toy

your seat why are nothing like

some fresh nova with the beef

are you pop the dark side

Verse 2

with my best it was there

on the sun it was there

these blurred hand we were the

our shadows books i used to

Chorus

Figure 13: Sample recurrent neural network output on the web application

8. References

1

ChatGPT

https://openai.com/blog/chatgpt

2

Nick Cave says ChatGPT's AI attempt to write Nick Cave lyrics 'sucks'

https://www.bbc.com/news/entertainment-arts-64302944

3

MusicLM: Generating Music From Text

https://arxiv.org/pdf/2301.11325.pdf

4

An important next step on our AI journey

https://blog.google/technology/ai/bard-google-ai-search-updates/

5

BERT for Dummies: State-of-the-art Model from Google

https://medium.com/@skillcate/bert-for-dummies-state-of-the-art-model-from-google-42639953e769

6

Should You Mask 15% in Masked Language Modeling?

https://arxiv.org/pdf/2202.08005.pdf

7

Next Sentence Prediction helps Implicit Discourse Relation Classification within and across Domains

https://aclanthology.org/D19-1586.pdf

8

Deep Rapping Character Level Neural Models for Automated Rap Lyrics Composition

https://www.researchgate.net/publication/332877990_Deep_Rapping_Character_Level_Neural_Models_for_Automated_Rap_Lyrics_Composition

9

LyricJam: A system for generating lyrics for live instrumental music

https://www.researchgate.net/publication/352117914_LyricJam_A_system_for_generating_lyrics_for_live_instrumental_music

10

A Ballad of the Mexicas: Automated Lyrical Narrative Writing

https://maya-ackerman.com/wp-content/uploads/2018/09/ballad-automated.pdf

11

LyriSys: An Interactive Support System for Writing Lyrics Based on Topic Transition

https://dl.acm.org/doi/pdf/10.1145/3025171.3025194

12

pronouncing Documentation Release 0.2.0

https://pronouncing.readthedocs.io/_/downloads/en/latest/pdf/

13

Language-tool-python 2.7.1

https://pypi.org/project/language-tool-python/

14

keytotext

https://github.com/gagan3012/keytotext

15

Random Sentence

https://github.com/patarapolw/randomsentence

16

What’s going on with the ChatGPT ban in Italy?

https://www.siliconrepublic.com/machines/chatgpt-banned-italy-openai-data

17

Could artificial intelligence be the future of songwriting?

https://rushhourtimes.com/could-artificial-intelligence-be-the-future-of-songwriting/