Lil A.I. Approaches to Rap Lyric Generation*

*EMAT31530 Report 2021. The GitHub repository for this paper: https://github.com/EMAT31530/ai-group-project-Team-JMJM

Mike Nelhams
University of Bristol
oa18502@bristol.ac.uk

Joshua Ukairo Stevens
University of Bristol
tx18073@bristol.ac.uk

Matthew Forster
University of Bristol
zc18711@bristol.ac.uk

James Groeneweg
University of Bristol
kr18544@bristol.ac.uk

***Abstract*—Rap lyrics require complex language structure, as well as deep, meaningful content. Language generation models aim to imitate the ability that rap artists have to creatively synthesise this language. The goal of this paper is to develop natural language generators with the ability to produce convincing rap lyrics with meaning and structure. In order to provide insight into how previous methods attempt the problem, a comprehensive review of familiar literature has been conducted.**

Two principal methods are introduced in this paper and each is critically evaluated throughout the paper. The first approach uses an implementation of stochastic Markov chains and the other utilises the proficiency of recurrent neural networks. To evaluate these methods, multi-class classification has been explored by the application of artificial neural networks and probabilistic classifiers, which as discriminators between rap and non-rap. Alongside the discriminators are more classical metrics, so that the results can be analysed more rigorously. This report provides a solid grounding for the extension of natural language generation in the music industry, as well as a basis for the augmentation of artificially generated lyrics to artificial music or video generation.

***Index Terms*—artificial intelligence, natural language generation, rap**

I. Introduction to the Problem

Originating from 1970s Hip-Hop culture in the Bronx, New York [1], rap has become a popular and well-established genre of music. In essence, rap may be described as music that assimilates *'rhyme, rhythmic speech, and street vernacular'* [2]. Therefore one may describe rap as being a form of 'street poetry'. The rhythmic structure found in rap makes the creation of new rap lyrics an interesting task and distinguishes it from other genres of music. The task of synthesising rap falls into the expansive category of natural language generation [3], a popular subcategory of artificial intelligence.

A. Project definition

The principal objective is to create a convincing rap lyric generator. Rap, similar to other genres, is composed of multiple complex attributes. Two aspects of this may be the meaning of the lyrics and the patterned structure of the text, which is often referred to as the 'flow'. Markov Chains and Recurrent Neural Networks are utilised to explore the possible patterns pertaining to the rap genre and for generating rap lyrics. Analysis of the language produced falls into an equally interesting subcategory of artificial intelligence known as natural language processing (NLP) [4].

The only necessary input to the systems, is a vocabulary of existing rap lyrics. Using existing lyrics is a clearer way of creating a vocabulary for rap lyrics, over selecting a vocabulary from an alphabet. Certain audiences may not find it acceptable for offensive words to be included in a generated rap, even though these words frequently appear in existing rap lyrics. As such, pipelines are required in order to 'clean' the vocabulary before or after use.

The output for each of the systems will be a string of text of the rap lyric generated. To measure the validity of these outputs, multiple classifier methods have been investigated and modelled in order to determine how successful each generator is. Additionally, simple statistical measures have been included to provide more rigorous quantitative metrics.

B. Desiderata

A rap is presented below, generated by the RNN lyric generator developed in the report:

'As I kiss your bum goodnight,
You nasty boy you nasty,
*Squeeze your clip hit the right one pass that w**d I got to light one,*
Is the bad bad boy,
Stepped to police when I proceed.'

II. Literature Review

A. DopeLearning

The report *DopeLearning: A Computational Approach to Rap Lyrics Generation* [5] encapsulates what is defined to be two of the main features of rap: its meaningful lyrics and its complex rhyme patterns. The writers develop an algorithm `DeepBeat`, which is a rap lyric generator in which these outlined features are achieved.

The `DeepBeat` algorithm uses an information retrieval approach. From a database of around 104 different rap artists lyrics, accounting to over half a million lines, the algorithm finds what is termed a *relevant next line*. In doing so it is assumed that each lyric line from the database of over half a million can be pieced together in a way that gives a relatively meaningful lyric structure. This involves the supposition that each line of a rap song's lyrics will contain some partial meaning. If two lines do not necessarily match up with continuous meaning, then this can also lead to what may be deemed as *computational creativity* or novelty.

The rhyme type utilised within `DeepBeat` is the *assonance* rhyme, which may be defined as: *'resemblance of sound between syllables of nearby words, arising particularly from the rhyming of two or more stressed vowels, but not consonants (e.g. sonnet, porridge), but also from the use of identical consonants with different vowels (e.g. killed, cold, culled)'* [6]. For implementing this, a rhyme density measure is used in which consonants and spaces are ignored and vowel sequences are found. This method is built as a deep learning neural network model. After each line is respectively classified, it is subsequently ranked using the RankSVM algorithm to choose the next line of the rap.

The paper explores an algorithm that produces convincing rap lyrics with assonance rhyming structure and the raps often offer a conception of meaning. As assonance rhyme is the most commonly used rhyming method within rap [7], it is unlikely that the the only common rhyme being assonance rhyme will be a large limitation for the model.

The method used, may lack in some senses true novelty, because the lines within the rap are already in other lyrics. As the database used counts over half a million lines of lyrics, one should not notice this. In order to improve the method explored, further work implementing a similar method whereby each line in the lyrics is newly generated could be utilised.

B. The Case of Rap Lyrics Ghostwriting

Evaluating Creative Language Generation: The Case of Rap Lyric Ghostwriting [8] has the goal of being an artificial 'ghostwriter'. Ghostwriting has always been prevalent in the creative industry and music is no exception. For years, major artists have had their songs written for them, but the goal here is to enable artificial intelligence to write lyrics, whilst also remaining unique.

Natural language generation has grown over the past few years, but the difficulty has always been in evaluating the generated text because there is no 'perfect' metric for doing so (see Chris van der Lee et al. 2019 [9]). Using a wide range of different rappers and styles, the main goal of this paper is to develop a way to evaluate the generated lyrics, through fluency, coherence and style matching, ensuring that generated verses are similar in areas such as rhyme frequency and style. This is done using both supervised learning and manual correction. At the line level, annotators are given the task of measuring level of fluency (how fluent is the line), and coherence (how well does the line match the previous line), with different levels of fluency and coherence given different weightings. In the automated sense, generated verses are analysed against training verses to check for novelty.

The paper focuses on giving robust evaluation methods that can be applied in a variety of contexts. The authors realise the need for some way to analyse the creativity and distinctiveness of the generated text, rather than just analysing how accurate the generated text is against a database. The wide range of different rappers also mean that there should be some level of uniqueness and distinguishability for generated lyrics depending on the rapper, that are in turn based off certain metrics.

Overall, it seems that a lot of work was done to produce rap lyrics that matched a rappers style of rap. However, the length of line was not examined and this can be key to some rappers and their style. Also, in the current world of rap, adlibs are common, and this work does not consider adlibs in the lyric generation.

C. Automatic Neural Lyrics and Melody Composition

The report *Automatic Neural Lyrics and Melody Composition* [10] looks at the process of generating both melody and lyrics with a neural network model consisting of a lyric generator, a lyric encoder and a melody decoder. They trained several of these encoder-decoder models to find correlations between lyrics and melodies. In all, their model can generate lyrics and corresponding melodies from a seed lyric inputted by the user, and it can generate a melody from any complete lyrics.

The proposed model included a lyrics generator which worked by predicting the next lyrical token given the previous lyrical token, starting with a seed lyric inputted by the user. A recurrent neural network (RNN) was used to do this, and a non-linear function was applied to map the network states over time. This function can be a simple sigmoid function or a more advanced function such as a long short-term memory (LSTM) unit. The function is trained over two datasets with over 12,023 MIDI files of 'popular English songs' [11] [12] and looks at the relationship between the lyrics and the melody on the syllable, word and sentence level.

The paper models a method of lyric inference. At each step of lyric generation the paper uses a softmax probability distribution to obtain the most probable next lyric given the previous lyrics. Furthermore, a freezing function is applied to the possible lyric tokens using a controllable parameter - this introduces noise to predict more robust and diverse music and is something that should be considered for implementation in the models explored in this paper.

One of the drawbacks of the model is that there is no attention paid to the structure of the song - there are no choruses/verses etc. generated, instead just a string of words and sentences. Consequently, the piece of music as a whole makes little musical sense, as the notes are tied to the syllables and words are generated or inputted with no attention paid to either the structure or the key of the piece. Finally, only 11 people's responses to the model were gathered in order to gauge how well the models generated lyrics sound, which is unlikely to be representative of the wider populace.

D. Lyric Generation with Style

In the report *Lyric Generation with Style* [13], the aim is to produce lyrics that match some given style and topic for an input using a generative adversarial network (GAN) model. A combination of a generator, an encoder and a discriminator are used to produce lyrics, embed the lyrics, and ensure the generated lyrics are similar to the training data.

The dataset is composed of 380,000 lyrics with a large number of different styles and topics. An important note is that there is roughly an equal split between genres in the dataset. Word2Vec was used to generate a mean vector for every word in a topic, so that generated lines could be compared to its respective topic, thus the likeness to the topic may be quantified.

A hierarchical structure is used for both the generator and the encoder, with two 'gated recurrent unit (GRU)' recurrent neural networks (RNN), a line-level decoder and a word-level decoder being utilised. This hierarchical structure ensures that line-to-line relations are captured as well as the relations between words. Furthermore, the loss function was evaluated both on a word level and on a line level, as to measure the difference between the generation and the ground truths.

In terms of evaluation, two different approaches were used. This was a *discriminatory approach*, measuring the 'realness' of generated lyrics with the trained discriminator, and a *classification measurement*, to check if the generated lyric matches its given style.

A potential drawback is that the hyper-tuning performed by the authors did not emphasise the creation of human-like lyrics, but this was due to time constraints. This is a large part of any model and a lot of time is required to try and find the best combination of model hyper-parameters. The generality of the model also leads to more basic, general lyrics. This is opposed to a deeper look into a specific genre, where one could have made more realistic lyrics for the genre. Most genres do not match exactly in lyrical style.

E. Everybody Sign Now

The report *Everybody Sign Now: Translating Spoken Language to Photo Realistic Sign Language Video* [14] is a very recent paper which presents '`SignGAN`', a sign language production (SLP) model that outputs realistic images of sign language from an input of spoken language. While this paper focuses on a different problem to what is explored in this paper, it does provide some interesting concepts for what could be considered as future work.

`SignGAN` uses a mixture density network (MDN) in order to model the different styles of sign language. Skeletal pose sign language is generated from this method. Loss functions are also utilised in order to restrict any motion blur issues.

Whilst this work has no direct applications to this paper, methods such as this can be implemented on rap lyrics. As the lyrics are in text form, this will decrease one of the tasks in the algorithm, however, it is possible to implement some form of text-to-speech algorithm such as *Google Cloud, Text-to-Speech* [15] allowing the model to 'rap' and possibly sign.

III. Method

A. Data preprocessing

The first step is to collect and preprocess royalty-free rap lyrics. All of the rap lyrics have been collected from 'The Original Hip-Hop Archives' [16], and another resource on 'Kaggle' [17]. A *webscraper* model was designed to collect numerous different artists and download the royalty free lyrics as text files. The majority of the punctuation in the lyrics is meaningless towards the generated text, so this is all stripped from the original data. Additionally, structure indicatives such as '[chorus]' and '[verse 1]' were removed, because the generative models are only expected to produce coherent lines, rather than entire songs, therefore those phrases are unimportant noise. An example of preprocessing is:

'*[Intro] Look.. if you had.. one shot, or one opportunity*',

instead becomes:

'*Look if you had one shot or one opportunity*'.

Censoring the data, while maintaining the semantics of the source content is necessary for publishing generated lyrics. To conserve the information, the output predictions are post-processed using plain asterisk replacements, and simple word substitutions, as seen above in Section I-B.

B. Markov model

For the first simplistic model, a rap lyric generator was created using Markov Chains. The model consists of three basic elements: a text generator, a rhyme ranker and a line choice function.

Within Natural Language Processing, a Markov Chain contains a stochastic Markov process satisfying the Markov property, whereby the probability of a word w_i is uniquely dependent on the immediately preceding word [4] [18]. Utilising this allows a model to form convincing sentences, however the absence of memory within the model restricts the amount of meaning that each sentence will have.

Algorithm 1: Markov rap generator

```
Result: Generate a line of rap, returns line
insert vocabulary;
index = 1;
create a dictionary called chain;
set count (number of words for the line) to be a random integer between 6
  and 12;
for each word in the vocabulary do
    set key as previous word;
    if key is in chain then
        add word to key's dictionary;
    else
        add key to chain with word in key's dictionary;
    end
end
create line with a randomly chosen word from vocabulary;
while number of words in line is less than count do
    next word is random choice of word from chain;
    add words to line;
end
```

The text generator, shown in Algorithm 1, creates a line of rap lyrics using a Markov Chain trained on the vocabulary. The Markov Chain creates a dictionary of the distinct words within the vocabulary. For each unique word within the vocabulary (and so in the dictionary) the 'value' of each word in the dictionary is a list of the words that are used immediately succeeding the key in the vocabulary. Thus, the model learns

the words that commonly succeed one another. After an initial random choice of first word for the line, also known as the 'seed text', the model will continue to choose the next word using the Markov Chain until it has reached the line's desired length. The length of each line here is defined as ranging between 6 and 12 words.

The rhyme ranker, shown in Algorithm 2, is based on assonance rhyme as defined in Subsection II-A. The goal of the rhyme ranker is to use two lines of rap lyrics inputted, in order to output a numerical score that can be compared with other pairings of lines. The chosen rhyme ranker counts the number of matching syllables from the end of the line, until the syllables at the same position do not match.

Finally, a choice function for the number of lines of lyric is required. From all of the possible lines generated, the choice function randomly selects a first line and then uses the rhyme ranker to choose the next line of the available generated lines that have not been used. To prevent every line rhyming with the first line, the rhyme ranker resets every other line.

Algorithm 2: Assonance Rhyme Ranker

```
Result: Score (counter) of assonance rhyme at end of each line
insert two generated lines of rap;
counter = 0;
i = 0;
Extract syllables of each line and reverse order;
Count is True;
while Count is True do
    if i is less than the minimum number of syllables for each line and the
      first syllable for each reversed line is equal then
        counter +1;
        i + 1;
    else
        Count is False;
    end
end
```

This model generates amusing and fairly convincing results for individual lines. It mimics the language of rap, albeit without the text having any real meaning. As the Markov Chain lacks a memory property, caused by choosing each word using only knowledge of the previous word, no real lyrical substance is included in the rap. Alongside each line generated by the Markov Chain not carrying any meaning, the choice function purely chooses the next line by the amount of assonance rhyme found at the end of each line, not by comparing the content or meaning of the lines.

C. Syllable counting

1) Integrating syllable counting: In practice, it may be useful for rappers to specify the number of syllables they want a line to have. Generally rappers rap to semiquavers in standard time (that is, 16 syllables to a bar) [19], but for specific music they may desire a specific number of syllables.

The text generator within the Markov model was edited so that it generated only a specified number of syllables per line. This was done by generating words as shown in Algorithm 1, until the specified number of syllables is reached or exceeded - if reached exactly, the line is used as normal and if exceeded, the last word is discarded and re-generated until the line is the correct number of syllables. If a word cannot be generated to make up the correct number of syllables, the entire line is re-generated until it fits the required number of syllables. This is shown in Algorithm 3. Since single syllable words exist in the Markov model's dictionary, regenerating lines guarantees exactness to the specified number of syllables.

2) Investigating models: The first method investigated to predict the number of syllables per word used a syllable predicting algorithm found online [20]. However, this algorithm has just a few basic rules and it often incorrectly calculates the number of syllables in less straightforward words. As such, another method was developed. This used the Pyphen module [21] to hyphenate each word and calculate the number of hyphenations, which is the number of syllables. Although Pyphen has more complicated rules, it again makes many mistakes when categorising words. As such, a neural network was created with Tensorflow to attempt to more accurately count syllables in a given word. Two text files containing over 40,000 words broken into their constituent syllables were found online [22]. This data was preprocessed by merging the files, counting the number of syllables each word has and creating a 'csv' file of words and their syllable count. Next, the data was divided into training, testing and validation sets, in a 60/20/20 percentage split, to preserve a large percentage of data for testing. K-fold cross-validation was not used, due to its expensive quadratic time-complexity. Every word was broken down into an array of unique integers: one for each unique word. As the neural network required every array of integers to be of equal length, each word was padded with empty spaces before being represented as an array of integers to be equal to the length of the longest word in the dataset, which was 16 characters.

Algorithm 3: Markov rap generator, with syllable counting

```
Result: Generate a line of rap, returns line
insert vocabulary;
insert desired syllable count (desired number of syllables in the line);
index = 1;
create a dictionary called chain;
for each word in the vocabulary do
    set key as previous word;
    if key is in chain then
        add word to key's dictionary;
    else
        add key to chain with word in key's dictionary;
    end
end
create line with a randomly chosen word from vocabulary;
syllable count ← word's syllable count;
while syllable count < desired syllable count do
    next word is random choice of word from chain;
    find next word's syllable count;
    syllable count ← syllable count + next word's syllable count;
    if attempts to generate word ≥ 100 then
        restart Algorithm
    if syllable count ≤ desired syllable count then
        add next word to line;
        word ← next word ;
    else
        syllable count ← syllable count - next word's syllable count;
    end
end
```

The neural network architecture began with an embedding layer, which was followed by a single hidden dense layer. The network ended with another dense layer, with a softmax activation function and seven output nodes, meaning that the network classifies words as being between one and seven syllables. There were at most seven syllable words. Initially, the hidden layer had 64 nodes and a ReLu activation function. The learning rate was initially set at 0.1. It was clear from the graph of training and test loss that the network was overfitting, so a dropout layer was added before the hidden layer to combat this. The dropout layer initially had a dropout rate of 20%, which helped to reduce overfitting. To combat it further and find the maximum validation loss, hyper-parameter tuning was undertaken with the '*Keras Tuner*' module. The optimal parameters were as follows: number of neurons in the hidden layer - 416, the optimal learning rate - 0.45, the optimal dropout rate - 0.2, and the optimal number of epochs in order to train the network for - 334 epochs. The activation functions ReLu, tanh and sigmoid were all also trialled, with the optimal activation function confirmed to be ReLu. These values enabled overfitting to be almost completely eliminated from the model while maximising validation accuracy. A deep neural network with 2 hidden layers was also developed, but failed to improve on the validation accuracy of the standard neural network and so was not used further.

3) Integration and testing: All of these syllable predictors were integrated into the Markov model and tested. They were tested on the validation dataset as to avoid bias toward the neural network and to have a large dataset to test on. The first model had an accuracy of 72%, the Pyphen model an accuracy of 63% and the neural network had an accuracy of 76%. This implies that the neural network is the most accurate model, however there is only a small difference between the first method and the neural network. The neural network is slower to predict syllable counts, consequently it takes longer to generate lyrics. The neural network fails to accurately classify words with more than seven syllables, due to its seven output nodes, although these words are very rare, and it cannot classify words with more than 16 characters accurately.

Nonetheless, the neural network was trained on standard grammatically correct English words and may therefore struggle to classify some of the wordplay often used in rap, such as 'skill-a-holic' by Eminem. As the Markov method generates new lyrics by using existing lyrics, this problem may be amplified. However, this is a shared problem that the other dictionary-based methods struggle with, as those algorithms were also designed to process standard English. The models were also tested on part of the data set used to train the neural network, which may mean that the accuracy predicted may not be valid when counting the syllables of rap lyrics. However, solving this problem would require a large database of rap lyrics to be broken down by syllable count, taking a lot of time if done manually. The wordplay in rap may also lead to rappers inventing long words - with many syllables - and the neural network will fail to accurately classify these, due to the reasons outlined above.

The desired number of syllables in a line can be defined by the user. This could allow a user to define a song with choruses and verses by specifying the number of sentences and syllables in each chorus and verse, thus allowing a more musically accurate rap to be generated. It could also allow a user who has composed rap music already to define lyrics to fit their music specifically.

D. Recurrent neural networks

In practise, long short-term memory (LSTM) models provide more coherent, meaningful lyrics as opposed to Markov chains. The latter suffers from only having short-term memory. As a result, key information can be lost and this becomes more of a problem for large sequences. To combat this, LSTMs have gates which enable the model to retain key information, exempting them from suffering from short-term memory loss. Another popular memory-based layer known as a 'Gated Recurrent Unit (GRU)' was tested, however they lacked the longer distance memory that LSTM layers provide, despite the GRU layers being almost twice as quick to compute calculations.

1) Selecting the model architecture: To create the model, words need to be converted into unique integers, a process known as tokenisation. Hence, each unique word was given its own integer such that each bar can be converted into a sequence of integers. This was done using two dictionaries, to both encode and decode bars. Next, these integer sequences were formed into n-grams of increasing length. Each sequence was pre-padded by zeroes, so each sequence is now the length of the bar of largest length. In addition to n-gram vectorisations, each bar itself was padded with a start and end character, each with unique token IDs, so the model itself is able to predict when sequences should finish. Bi-Directional LSTM layers are also used due to their impressive ability to recall information of the mono-directional counterpart [23]. Each bidirectional LSTM layer trains two regular LSTM layers one for the forward direction and one for the reverse direction, which improves the model's context due to the information about past and future states.

The data was split, such that for a sequence of length n, the prediction is a probability vector representation of each word from the vocabulary list. Following the input layer of the model, is an embedding layer, which masks the padded zeroes, since the additional padding provides no useful information. An embedding layer is used opposed to one-hot vector encoding, because 'the main benefit of the dense representations [word-embedding] is in generalisation power' [23]. Lastly, the model was mini-batch trained with different architectures and layers: to fine-tune the model's different hyper-parameters.

A simple decision for modelling was selecting the optimiser. Adam [24] is a fast, popular and semi-reliable optimiser that generally requires little hyper-parameter fine-tuning. A more stable and effective look-ahead adaptation of Adam known as Ranger [25] exists, however this optimiser is slower and

requires over twice the memory, since it tracks more than twice the variables compared to Adam. For all training sessions, Adam proved to be sufficiently stable.

The softmax function was chosen, since it evenly propagates derivative weights fairly whilst being computationally efficient. Finally, each model should be run three times independently and only the most optimal results are included.

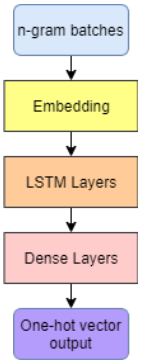

Fig. 1. The most general LSTM architecture. For the model used in this paper, there are two stacked hidden bidirectional LSTMs and one hidden dense layer with softmax as its activation function.

It is understood that stacking LSTM layers can be risky if implemented incorrectly, and a heavy trade-off between time complexity, maximum validation accuracy and overfitting is incurred [23]. The idea behind deep (stacked) models, opposed to small, but wide layers, is that the models can learn more advanced patterns with a far smaller training time. The principal idea is that the model should consist of at least the following, shown in Figure 1.

IV. Thorough Experimentation for Recurrent Neural Networks

A. Designing the initial model

With the LSTM model, a great deal of experimentation was done in adapting the model. This ranges from how the way the lyrics were fed into the model, what hyper-parameters were set to, the network architecture, and which additional features to include. To begin with, preprocessing the data was done differently to how it was done in the final model. The text was split into equal chunks of size n (the input data), with the word at position $n+1$ going into the output data. This was iterated over the whole data set with step sizes of 1. However, a 'line-by-line' approach produced more realistic results, as the context of each line was better picked up by the model. This can be seen in Figure 2. This saw the accuracy of the model increase, and the validation loss decreased.

B. Hyperparameter tuning and regularisation

Firstly, to measure the accuracy of the model, a cross validation set was required, so the model's generalisation ability could be seen more clearly across training epochs. The training/validation/testing split is 75/15/10 percent, chosen to balance between model accuracy and varied testing. Grid search was implemented for both two and three LSTM layers, to see which was optimal. Different numbers of neurons were iterated over in the LSTM layers (32, 64, 128, 526, 1,028). The optimal model consisted of two stacked LSTM layers of sizes (128, 256) respectively and this was fixed for future experimentation.

When the model was trained for longer than was necessary, it was overfitting and copying lyrics from the dataset, which was indicated by the validation loss increasing. An example of such overfitting can be seen in Figure 3. As a result, early-stopping was implemented with a patience of three epochs and the best weights were always restored. Sentences that contained words used less than k times, for any k such as $k = 1, 2, 3, ...$ in the overall dataset, could also be removed. However, doing so would vary the original variance and removes important structure from the vocabulary.

Initially, a smaller dataset was used. However, the more training data there is, the more likely the model is to be more accurate, since this allows fuller access to the data distribution and it is less prone to overfitting. As a result the model will generalise better. Hence, more data was added for different artists. Some of the artists included are: Snoop Dogg, Eminem, Jay-Z and The Notorious B.I.G. More data could be included, but this dataset sample served as a good variety, without causing training time to be too large. Since many of the model additions are theoretical, rather than experimental, it serves that the data can be increased without the model needing hyper-parameter tuning for greater accuracy.

A major concern regarding generative neural networks is that the dataset input is almost always imbalanced. The frequencies of each word differ across the input data and this will always be the case for all rap lyrics. Tokenisation models may struggle when attempting to predict words with low frequencies and if do they successfully predict uncommon words, this is often an indicator of overfitting. A naive approach for balancing the data is to weight predictions by the inverse of their frequencies, i.e. focal loss compared to categorical cross-entropy. Whilst this does accomplish the goal of evening the data on a single word level, rap lyrics consist of phrases, word pairs and structured lines, opposed to only single word units. Furthermore, it's strongly argued by many researchers that perhaps if patterns are more frequent in text, they should therefore appear more frequently in the outputs, which balancing the data would remove.

The figures 3 and 4 display the different models trained with the exact same hyper-parameters, but different regularisation techniques were applied. The initial model architecture sizes were determined through grid search. The embedding layer optimal size was determined with the idea that the model dictionary contains 7,000 unique words. The minimum number of binary bits to represent a 7,000 one-hot-vector would be $\lceil log_2(7000) \rceil = 13$. The TensorFlow team themselves recommend the mysteriously dubious number of $\lceil 7000^{0.25} \rceil = 10$ which closely resembles the binary information limit [26]. The reason that fewer number of dimensions may be used, is due to the fact that there is no upper bound on the absolute weights of the embedding

vectors, opposed to the binary limits of strictly 0 or 1. Fewer dimensions are favourable, since this forces similar data to have similar embedding vectors, however too few dimensions means that the model effectively has little ability to recognise subtle differences. To strike a balance, grid search between 10 and 30 was used, and 25 was found to be the best embedding size for the initial architecture.

For the first step to combat overfitting, dropout layers were added after every LSTM layer. Dropout randomly ignores weights and biases with likelihood of p per operation [27]. Dropout decreases the dependency on individual neurons and it slows down the learning process, which helps prevent overfitting. Theoretically, dropout should not exceed more than 0.5, since more than half the nodes may be ignored, therefore the model is not likely to noticeably improve between epochs. Dropout should be included after the embedding layer and after each of the LSTM layers. Grid search was performed with different dropout rates for the dropout layers in the set (0.1, 0.2, 0.3, 0.4, 0.5), and the optimal numbers were found to be (0,0.3,0.5) for each of the layers respectively; this was fixed for future experimentation.

Another two forms of regularisation are L_1 and L_2 weight regularisation, lasso and ridge regularisation respectively. Lasso and ridge regularisation are defined in equations 1 and 2 for weights matrix $\mathbf{W}$.

$$L_1 = \lambda \sum_{j=0}^{M} |W_j| \tag{1}$$

$$L_2 = \lambda \sum_{j=0}^{M} W_j^2 \tag{2}$$

Lasso regularisation is good at slowing down training and preventing weights from tending to infinity, whereas ridge regularisation additionally zeroes out ineffective weights, due to the quadratic term. Applying this theory, ridge regularisation should be included over lasso regularisation throughout this report. A linear combination of each regularisation method could be utilised, however this is unnecessary, since ridge regularisation should usually be more effective [28].

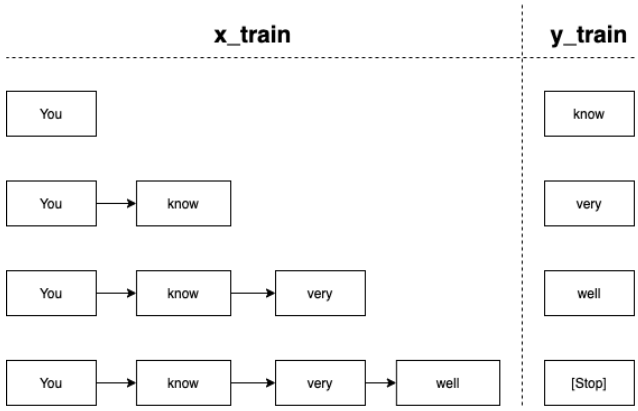

Fig. 2. An example lyric demonstrates how the data is processed for the RNN model. The model decides itself when to put a '[stop]' prediction and the initial data is seeded with stop characters towards the end of every line.

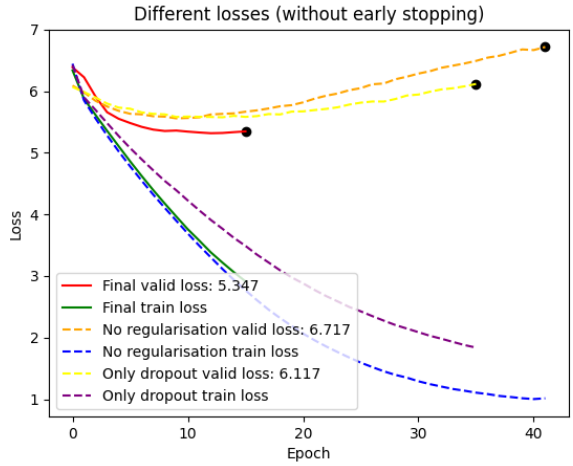

Fig. 3. All of the models have an embedding size of 25, LSTM1 size 128 neurons, LSTM2 size 256 neurons and a single dense layer of the vocabulary length. The difference between including and excluding dropout layers is shown above. Including dropout reduces the minimum validation loss by approximately ten percent. Furthermore, the difference between including and excluding early stopping is evident, as only the final model in the graph does not overfit.

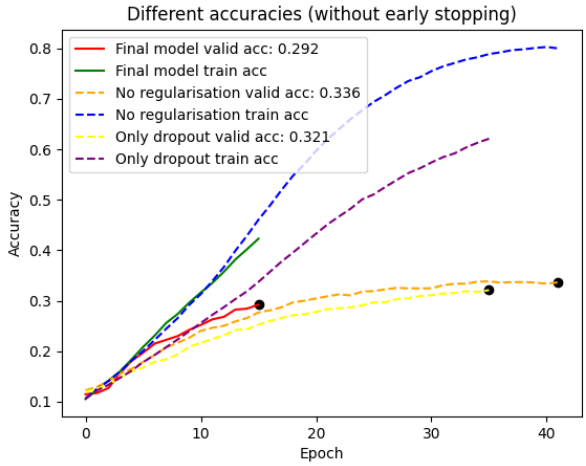

Fig. 4. The graph displays the respective accuracies from Figure 3. Accuracy is not the primary focus of the system; instead an ability to produce new lyrics is preferable, which is shown by the loss curves opposed to the accuracy. Nevertheless it is clear that the final model is more accurate than the two models without regression, as the final model's validation accuracy is always greater than the other two, past three epochs.

Overfitting still occurs as shown in Figure 3, and re-balancing the data by word frequency may be an unfavourable solution, due to the loss of data patterns. Overfitting is indicated by the validation loss and accuracy diverging from the training loss and accuracy. From the data it would follow that a minimum loss has been found which exists irrespective from the learning rate. Although, it is clear that L_1 and L_2 regularisation compromise the ability of a RNN to learn effectively, as discussed in [29]. This idea is evidenced by Figure 5 and the following generated lyrics:

'That i the the
Forgive you the the'.

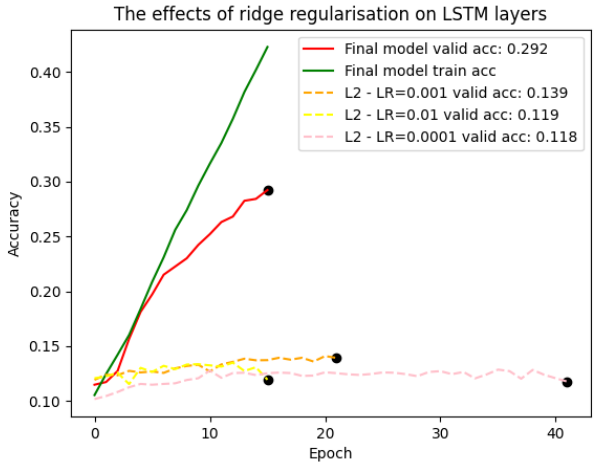

Fig. 5. The graph illustrates plainly the effect of regularisation when applied to LSTM layers. The learning is stunted and the models with this regularisation underfit a lot. To demonstrate the fact that this is not an issue with the learning rate, different learning rates were used and underfitting still occurs.

The issue of strange underfitting is commonplace for some RNN models. Kernel level weight regularisation actually stunts the ability of a RNN to learn and should not be applied to any LSTM layers or any layers following these [29]. Investigating this issue further was done by Ilya et al. in '*Decoupled Weight Decay Regularization*' [30] and they determined that the Adam optimiser is coupled with the weight regularisers for RNNs in TensorFlow. The two can be decoupled effectively, however this is beyond the scope of this project and would involve an entire rewrite and rerun of all training models. Therefore, weight regularisation shall not be included for this model, since it cannot produce better models, without decoupling it from the optimiser.

Batch normalisation and layer normalisation are two other common effective forms of regularisation. This involves normalising either the batches or the entire data, so that the mean input is close to zero and that the standard deviation is close to one. Furthermore, the training time decreases on average, for reasons that are still debated amongst researchers. It's currently interpreted commonly that it smooths the objective function, but initial ideas were that it reduces the covariate shift [31] of the data. Regardless of how it works, applying batch normalisation and layer normalisation stably improved the validation loss as shown below in figures 3 and 4.

C. Exploring a statistical model

Lastly, the model's outputs are deterministic, so to circumvent this issue, a statistical temperature parameter γ can be added. A stable and therefore reliable form of noise is the Gumbel noise distribution [32] displayed by Equation 3. By directly substituting the output softmax function to a Gumbel-softmax function, the model may now sample from its

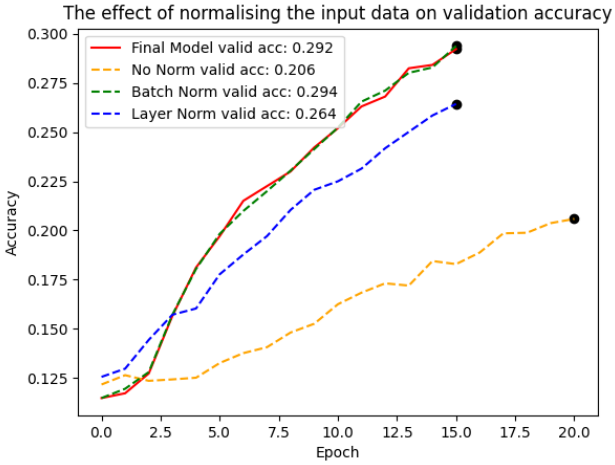

Fig. 6. Intertwined in the graph are the final model validation accuracy and the batch normalised model's validation accuracy. It's clear that the final model uses the most optimal combination of the aforementioned features, with batch normalised inputs being the last form of regularisation.

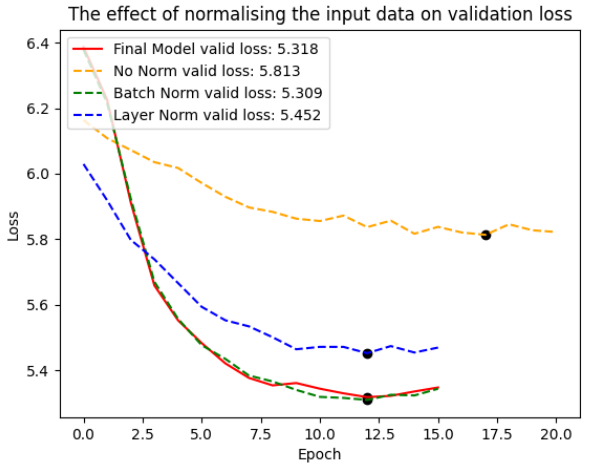

Fig. 7. The most optimal form of regularisation for the currently developing model is counter-intuitively batch normalisation opposed to layer regularisation, as indicated by the validation loss curves.

prediction distribution opposed to always selecting the highest probability. This means that the synthesised lyrics are now statistical rather than deterministic and with the randomness constant γ, lyric repetitions can be decreased by a controllable amount.

$$p(x) = \frac{1}{\gamma}\exp(-z - \exp(-z)), \tag{3}$$

where:

$$z = \frac{x - \overline{z}}{\gamma}.$$

V. Analysing Generated Natural Language

A. Lyric genre classifiers

To test the results of the lyric generation models, multiple methods have been considered. The main experimental method

is a lyric genre classifier. The analysis involves the accuracy and loss of the classifiers alongside their macro F_1 score. The F_1 score is defined as

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}.$$

F_1 scores are traditionally used for binary classifiers and so in order to use it on the multi-class classifiers, macro F_1 scores are used. This extension is simply the mean average of the F_1 score for each class.

1) Data preprocessing: To classify the genre of lyrics, a classifier must be trained on lyric data of genres other than just rap. Lyric data for rock, country and pop were gathered from Kaggle [16] and the Genius API [33], in order to have four genres that the models are able to categorise input lyrics against. This classifier acts as a discriminator for the generated rap lyrics and theoretically can measure if generated lyrics are more like rap than any other genre.

The dataset for the classifiers has 8,923 rock, 6,391 country, 11,515 pop and 31,619 rap, ten word lines. As these are not even in number, limits to the number of lines are required to have an equal 6,300 lines from each genre for the multinomial Bayes classifier. This has not been done for the convolutional neural network (CNN) and recurrent neural network (RNN) classifiers and the macro F_1 score is useful for interpreting the weighted average of the precision for each genre.

Each classifier uses sample lines estimated at ten words each. These are fed into the classifier for training and this allows the model to predict each input line. For the lyrics that have been generated, 400 lines have been classified for each of the main branches of the generators. These lines are each ten words long in order to match the training data used for the classifiers. For all three of the classifier models, the lyrics are tokenised for inputting into each model.

2) Multinomial Bayes theorem classifier: The first classifier utilises multinomial naive Bayes theorem to categorise its inputs. Within the classification model, input data is classified using a naive Bayes model. This model has negligible training time and provides quick results. The method estimates the likelihood of each word within the vocabulary for each genre.

As can be seen in the results in Tables I and II, the test accuracy and macro F_1 score are both less than 50%; while this classifier is useful as a starting place, more advanced methods should be explored.

3) Convolutional neural network classifier: A CNN classifier was created using a convolutional neural network with preprocessing and layer packages included in Tensorflow. A sequential model was used, along with one dimensional convolution and one dimensional global max pooling layers. Simple ReLu and softmax activation are used and because the model is so small, the vanishing gradient problem is unlikely to affect the activation layers.

4) Recurrent neural network classifier: The RNN classifier was created using a recurrent neural network with long short-term memory with preprocessing and layer packages included in Tensorflow. For these, spatial dropout and simple softmax activation are used along with the Adam optimiser. Each epoch for this classifier required longer times to complete, subsequently far fewer RNN classifier models have been trained.

TABLE I
INITIAL CLASSIFICATION RESULTS

	Classifier		
	Bayes	CNN	RNN
Number of epochs	-	1080	26
Total training run time	-	4353s	4835s
Final training epoch accuracy	-	0.9385	0.9007
Test set accuracy	0.3844	0.7333	0.7117
Test set macro F_1 score	0.3827	0.6499	0.6221
Markov generator correct fraction	0.6075	0.8025	0.8725
Markov syllable generator correct fraction	-	-	-
RNN generator correct fraction	0.5450	0.8857	0.7800

5) Convolutional neural network and recurrent neural network classifier results: Initially when training both classifiers, an estimated number of epochs was chosen to produce a run time of 1.5 hours. Due to the short run time of the epochs of the convolutional neural network (CNN) classifier, 1,080 epochs were initially run compared to 26 epochs for the recurrent neural network (RNN) classifier. However, The CNN classifier model did, plateau in accuracy at around the 277th epoch. Using a specified number of epochs resulted in the CNN and RNN classifier models achieving good accuracies, rating a majority of each generated lyric as rap as seen in Table I.

Whilst the initial classification results on the models shown in Table I are encouraging, as previously noted when working with the RNN lyric generators in Subsection III-D, concentrating solely on improving the accuracy of the model may not always yield the best model.

To combat overfitting, early stopping was added to the CNN and RNN classifiers in order to find the minimum validation loss of the model. Visualisations of this can be seen in figures 3 and 4 when working with the RNN generator models.

Early stopping has meant that the generators have been trained in notably less time and it resulted in less overfitted classifiers. These CNN and RNN classifier models had a final training epoch loss of 0.4895 and 0.4467 respectively. It can be seen with the reduced overfitting that the accuracies given by the CNN and RNN classifiers of the generated lyrics have increased in all but one case.

The macro F_1 score has been calculated for the classifiers. Without early stopping, the macro F_1 score on the test set was calculated as 0.6499 and 0.6221 for the respective CNN and RNN classifiers. This lessened when overfitting was used with respective macro F_1 scores of 0.5914 and 0.6152. These results do show that the classifiers will be better at predicting some genres compared to others. For instance, of the four genres chosen (rock, country, pop, rap), rock and country are very similar and so may be hard to tell apart. Alongside this, pop is an extremely broad genre, as an example, training lyrics for pop included Justin Bieber and Michael Jackson. As rap has a comparatively more specific language structure compared to the other genres, it seems to be well separated from the other genres in the classifiers.

TABLE II
FINAL CLASSIFICATION RESULTS

	Classifier		
	Bayes	CNN	RNN
Number of epochs	-	17	7
Total training run time	-	101s	981s
Final training epoch accuracy	0.3844	0.8175	0.8261
Final training epoch loss	-	0.4895	0.4467
Final training epoch val-accuracy	-	0.7002	0.7162
Final training epoch val-loss	-	0.8296	0.7810
Test set accuracy	0.3844	0.7134	0.7222
Test set loss	-	0.7943	0.7242
Test set macro F_1 score	0.3827	0.5914	0.6152
Markov generator correct fraction	0.6060	0.8404	0.8628
Markov syllable generator correct fraction	0.5564	0.8321	0.9173
RNN generator correct fraction	0.5505	0.9238	0.9651

All three of the classifiers are trained to judge what genre some given lyrics are. As such they do not judge the quality of the lyrics, but do offer an insight in how they might be perceived. The goal is to produce lyrics that feel convincing as rap opposed to other genres. It may be concluded that these CNN and RNN classifiers judge that the RNN generator is the 'best' or most convincing as a rap lyric generator.

B. Performance metrics

Word frequency analysis was performed as a more rigorous form of testing the generated lyrics, the results of this are seen in Table III. The greatest artists are notorious for having a high variance in their word choice. Next, rhyming towards the ends of sentences is considered standard amongst most rap songs [7] and the generated lyrics should be no different.

The above 'assonance rhyme ranker' metric shown in Algorithm 2 is capable of measuring the reverse word similarity. However, the first proposed metric biases lengthier words and doesn't account for phonemes as opposed to vowel-based syllables. Instead a more robust and more accurate measure for rhyme density is proposed. Firstly, rather than using syllabic based measurements, the phonemes themselves should be considered, and the G2P package [34] was used. Since phonemes are closer to how words are spoken, the pretrained G2P package provides greater accuracy for how words would rhyme, since words such as 'through' and 'though' do not rhyme despite having similar graphemes (spelling). Secondly, an iterative phoneme similarity approach is repeated from Algorithm 2, however the entire list of phonemes is taken into account, rather than just the first backwards matches. Assonance and consonance rhyme are now accounted for. Lastly, the metric is normalised by the number of phonemes. The results for this metric can be seen in Table III.

TABLE III
PERFORMANCE METRICS
EACH OF THE METRICS WERE APPLIED TO THE TRAINING DATA AND THEN TO THE THREE RESPECTIVE MODELS. FOR CONTEXT, ALL THE METRICS WERE ALSO TESTED ON ALL OF EMINEM'S LYRICS [17].

	Artist				
	Markov	Syllable	RNN	Training data	Eminem
Total no. words	3923	4010	3180	46234	297406
Number of unique words	569	1274	468	5055	23561
Mean word frequency	6.8946	3.1476	6.7949	9.1462	12.6228
Std dev word freq	12.5924	9.2264	17.9195	55.6769	157.1008
Mean rhyme score	0.6558	0.5129	0.5168	0.5499	0.5421

Table III highlights that in comparison to the training data, the RNN model uses a smaller selection of words as opposed to the Syllable Markov and Markov models. The RNN model has 25% less words than the other two models, which will have likely affected the word frequency variance. The Syllable Markov model had a smaller mean word frequency, which suggests that word popularity is more evenly spread. This is due to the changes to the choice of words within lines of rap, shown in Algorithm 3.

The standard deviation word frequency is low in both the Syllable Markov model and the plain Markov model. The generator selects alternate words until it finds one with the correct number of syllables, meaning that more popular words may be discarded for having too many syllables, affecting the word frequency.

Lastly, the plain Markov model clearly excels at rhyming as expected, with over a 10% increase over the other models and even the training data. The RNN and Syllable Markov model struggling with rhyming, which can be explained to their lack of attention to relate each line to the next.

VI. DISCUSSION

A. Strengths and weaknesses of proposed models

The goal of a 'ghostwriter' is to produce lyrics that relate to the content of the writer, whilst remaining unique and

not copying previous lyrics. On one hand, if the model is overfitted for more than a few epochs, generated lyrics would lack novelty. On the other hand, if the model was underfitted it was clear there was no fluency or meaning to the lyrics. By fine-tuning the RNN model, a model was found that produced original, rhythmic lyrics.

Using the BERT Model [35], it would be possible to create a more coherent set of structured lyrics. BERT is renowned for being highly effective in 'Next Sequence Prediction' and therefore by producing a number of lyrics based off a seed text, BERT may have produced better results from the same seeds. BERT uses masked language models to predict certain words in a given text; it uses a Transformer to take into account the different contexts a word is used in [35]. The use of the Transformer removes the 'left to right approach', in favour of a bidirectional approach, meaning the model can learn the context of a word based on all of the other words within the input batch.

The task of language generation is certainly complex and there are many intricacies that need to be looked at to create realistic models, such as logical structure, which a line by line recurrent neural network (RNN) generator cannot encapsulate. Potentially switching to a stateful model could allow these line-to-line relationships, since they don't reset between lines and batches.

The ambition of the project was to compare and create models that could create original rap lyrics given a large dataset of lyrics for all kinds of rap artists, while maintaining some degree of rhyme that is present in rap music [36]. It is demonstrably difficult to replicate the complex rhyme patterns of some of the current skilled rappers, e.g. Kendrick Lamar, but the model was successful in generating music clearly understood to be rap. A potential improvement would be to alleviate the training/validation/testing split that was applied before feeding the data to the RNN model. Using 100% of the data for training would ensure that the model has access to more data. This can be extended further by collecting the superlative amount of rap lyrics as training data at the expense of training time. An RNN model ran with maximal training data, but still using the RNN hyper parameters could provide far greater generalisation accuracy and a great deal of insight into the rap genre.

B. Sustainability and ethical concerns

From a sustainability point of view, the amount of time training the model accumulated to days, and the power used for this training would certainly have produced plenty of CO_2 emissions. In the future, it is important that more is done to improve efficiency - perhaps looking for a small increase in accuracy is not worth the time and resources spent. Planning is essential when it comes to machine learning and having the foresight to determine how time is spent is crucial. Overall, there are definitely computational efficiency improvements to be made, such as using minimally sized models, fewer model callbacks and optimised hyper parameter searches.

As this project does not handle sensitive data and the models created do not draw conclusions, many of the points raised in the Deon ethics checklist for data scientists [37] are not directly applicable to this project. Also, in order to not offend any person, pipelines have been used to clear any objectionable language from the generated rap. Any offensive words were added to the list of censored words.

As many rappers mention their own names or those of others, the generators may include some personally identifiable information in the raps, however this is commonplace in rap and so should not be of issue. All data used readily exists in the public domain. The lyrics produced are not pretending to be someone and are simply intended as lyrics that could be used by an artist. The question of when using the same word as another writer is plagiarism is an interesting question and may be interpreted in many different ways. Rappers themselves will learn and draw inspiration from other artists and sampling is commonplace in the music industry.

VII. Conclusion

The topic of rap lyric generation has been explored thoroughly within this paper. Research has been carried out into existing projects of this nature and a corresponding comprehensive literature was written. Two different models have been created to generate rap lyrics and multiple methods for syllable/phoneme counting within rap have been explored. Furthermore, three classifiers have been created to test the effectiveness of the models at generating rap, along with more rigorous classical metrics to test the generated lyrical properties.

The results of the classifiers agree that the recurrent neural network (RNN) generator outputs a higher number of lyrics classified as rap. The convolutional neural network (CNN) and RNN classifiers have each classified the RNN generator lyrics as rap for over 90% of the test set. The Markov and Markov syllable-counting generators have very similar success fractions. The only difference between these two models was how the words within the lines were chosen, and because the classifiers classify each line this was expected.

The classical metrics show that the RNN model uses a smaller selection of words as opposed to the Syllable Markov and Markov models, however its word choice is more varied from the fewer words it selects. The Markov model was an effective means for generating rhyming lyrics, with a better ability to rhyme than its training data.

VIII. Future work

A natural extension to this project would be the generation of music alongside the rap lyrics, allowing a full piece of rap music to be generated by AI. This would involve scraping MIDI files from rap artists and training another neural network to produce similar music. The music would then be matched to lyrics generated using the methods explored in the literature review II. Each note would be matched to a syllable of generated lyrics, to produce a piece of music that

could be performed by a rapper. The syllabic neural network would be valuable here, because it would allow words to be easily matched to the music. Each sentence generated could be matched to musical bars or phrases.

In any music, there is usually a sense of storytelling of the rappers past. Using natural language generation models, there would be a question over the truth of the stories being told. Perhaps information and stories from a rappers past could be scraped from online sources (news stories, interviews, etc.) and fed into a lyric generator to produce lyrics relating to rappers themselves. Additionally, the theme and semantics of lyrics could be looked at separately. Lyrics could be generated to mimic a given theme, such as love, similar to how BERT models mask their inputs based on contextual information [35].

For improving the rhythmic language of the models, the newly proposed rhyming metric could be adapted and included into the Markov model and used as a loss function for a stateful RNN model. Since the improved rhyming metric dominates the assonance rhyme ranker, any generated lyrics would be equally or more accurate. Other extensions such as creating sign language interpretations for generated raps may also be considered; these methods may lead to exciting progressions such as the generation of music videos for the raps.

References

[1] M. Dyson, *Know What I Mean? Reflections on Hip-Hop*. New York, US: Basic Civitas Books, 2007, ch. Introduction.

[2] C. L. Keyes, *Rap Music and Street Consciousness*. Champaign, IL, US: University of Illinois Press, 2004, ch. Introduction.

[3] A. Mykowiecka, "Natural-language generation—an overview," *International Journal of Man-Machine Studies*, vol. 34, no. 4, pp. 497–511, 1991. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0020737391900323

[4] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. London, UK: Pearson, 2016, ch. Natural Language Processing.

[5] E. Malmi, P. Takala, H. Toivonen, T. Raiko, and A. Gionis. (2016, Jun.) Dopelearning: A computational approach to rap lyrics generation. KDD'16. San Francisco, CA, US. [Online]. Available: https://arxiv.org/abs/1505.04771

[6] M. Proffitt, *The Oxford English Dictionary*. Oxford, UK: Oxford University Press, 2020.

[7] P. Edwards, *How to Rap: The Art and Science of the Hip-Hop MC*. Chicago, IL, US: Chicago Review Press, 2009.

[8] P. Potash, A. Romanov, and A. Rumshisky. (2016, Dec.) Evaluating creative language generation: The case of rap lyric ghostwriting. University of Massachusetts Lowell. Lowell, MA, US. [Online]. Available: https://arxiv.org/pdf/1612.03205.pdf?fbclid=IwAR3K-vFwMnwVGnov2C4VyZHS-fwnYJMfGbLMKmp-0x4SKuPdD6TsuO3ce0k

[9] C. van der Lee, A. Gatt, E. van Miltenburg, S. Wubben, and E. Krahmer. (2019, Nov.) Best practices for the human evaluation of automatically generated text. Association for Computational Linguistics. Stroudsburg, PA, US. [Online]. Available: https://www.inlg2019.com/assets/papers/98_Paper.pdf

[10] G. R. M, Y. Yu, F. Harscoët, S. Canales, and S. Tang. (2020, Nov.) Automatic neural lyrics and melody composition. Cornell University. Ithaca, NY, US. [Online]. Available: https://arxiv.org/pdf/2011.06380.pdf

[11] C. Raffel, "The lakh midi dataset," 2016. [Online]. Available: https://colinraffel.com/projects/lmd/

[12] M. Man, "The largest midi collection on the internet," 2016. [Online]. Available: https://www.reddit.com/r/datasets/comments/3akhxy/the_largest_midi_collection_on_the_internet/

[13] Y. Feng and B. Xu. (2019) Lyric generation with style. University of British Columbia. Vancouver, BC, CA. [Online]. Available: http://www.bicheng-xu.com/files/Lyric_Generation.pdf

[14] B. Saunders, N. C. Camgoz, and R. Bowden. (2020, unpublished, Nov.) Everybody sign now: Translating spoken language to photo realistic sign language video. University of Surrey. Guildford, UK. [Online]. Available: https://arxiv.org/abs/2011.09846

[15] Google. (2021, Mar.) Text-to-speech. Google. Menlo Park, CA, US. [Online]. Available: https://cloud.google.com/text-to-speech

[16] P. Mooney. (2020, Dec.) Poetry and lyrics. Kaggle. [Online]. Available: https://www.kaggle.com/paultimothymooney/poetry

[17] ohhla. (2020, Dec.) The original hip-hop archives. ohhla. [Online]. Available: https://www.ohhla.com

[18] S. Asmussen, *Applied Probability and Queues*. Berlin, GE: Springer Science and Business Media, 2003, ch. Markov Chains.

[19] C. Mize. (2021, Mar.) Colomize studios. Colomize Studios. Monroe, GA, US. [Online]. Available: https://colemizestudios.com/how-to-rap-structuring-lyrics/

[20] M. Holtzscher. (2021, Mar.) Programmatically counting syllables. Medium. San Francisco, CA, US. [Online]. Available: https://medium.com/@mholtzscher/programmatically-counting-syllables-ca760435fab4

[21] Kozea. (2021, Feb.) Pyphen documentation. Pyphen. [Online]. Available: https://pyphen.org/

[22] G. Darby. (2021, Mar.) Delphiforfun. DelphiForFun. [Online]. Available: http://www.delphiforfun.org/programs/Syllables.htm

[23] Y. Goldberg. (2015, Oct) A primer on neural network models for natural language processing. Bar Ilan University. Ramat Gan, IS, US. [Online]. Available: https://arxiv.org/pdf/1510.00726.pdf

[24] D. Kingma and J. Ba. (2014, Dec.) Adam: A method for stochastic optimization. ICLR 2018. Vancouver, BC, CA. [Online]. Available: https://arxiv.org/pdf/1412.6980.pdf

[25] M. Zhang, J. Lucas, G. Hinton, and J. Ba. (2019, dec) Lookahead optimizer: k steps forward, 1 step back. University of Toronto. Toronto, ON, CA. [Online]. Available: https://arxiv.org/pdf/1907.08610.pdf

[26] T. Team, "Introducing tensorflow feature columns," https://developers.googleblog.com/2017/11/introducing-tensorflow-feature-columns.html, 2017.

[27] P. Baldi and P. Sadowski. (2013, Oct.) Advances in neural information processing systems, understanding dropout. Curran Associates, Inc. Red Hook, NY, US. [Online]. Available: https://proceedings.neurips.cc/paper/2013/file/71f6278d140af599e06ad9bf1ba03cb0-Paper.pdf

[28] T. Team, "Overfit and underfit," 2019. [Online]. Available: https://www.tensorflow.org/tutorials/keras/overfit_and_underfit

[29] R. Pascanu, T. Mikolov, and Y. Bengio, "Understanding the exploding gradient problem," *CoRR*, vol. abs/1211.5063, 2012. [Online]. Available: http://arxiv.org/abs/1211.5063

[30] I. Loshchilov and F. Hutter, "Fixing weight decay regularization in adam," *CoRR*, vol. abs/1711.05101, 2017. [Online]. Available: http://arxiv.org/abs/1711.05101

[31] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?" 2019.

[32] M. Balog, N. Tripuraneni, Z. Ghahramani, and A. Weller, "Lost relatives of the gumbel trick," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 06–11 Aug 2017, pp. 371–379. [Online]. Available: http://proceedings.mlr.press/v70/balog17a.html

[33] Multiple. (2021, Mar.) Genius api. Genius. [Online]. Available: https://docs.genius.com/#web_pages-h2

[34] J. K. K. Park, "g2p-en," 2018. [Online]. Available: https://pypi.org/project/g2p-en/

[35] J. Devlin, M. Chang, K. Lee, and K. Toutanova. (2019, May) Bert: Pre-training of deep bidirectional transformers for language understanding. Google AI Language. Menlo Park, CA, US. [Online]. Available: https://arxiv.org/pdf/1810.04805.pdf

[36] H. Hirjee and D. Brown. (2010, Mar.) Using automated rhyme detection to characterize rhyming style in rap music. The Ohio State University Libraries. Columbus, OH, US. [Online]. Available: https://kb.osu.edu/bitstream/handle/1811/48548/EMR000091a-Hirjee_Brown.pdf?sequence=1&isAllowed=y

[37] Deon, "An ethics checklist for data scientists," 2021. [Online]. Available: https://deon.drivendata.org