Genre-Based Lyric Generation: A Tool For Songwriters

Machine Learning Techniques in Text Generation

Ange Olson
George Washington University
May 8, 2023

Abstract

This paper summarizes the 2023 Capstone Project Genre-Based Lyric Generation: A Tool For Songwriters. The final generator is constructed from a BERT encoder, Long Short-Term Memory (LSTM) decoder model that generates songs in the "indie" genre of music. The best-performing model achieved a loss (cross entropy) of 1.11 on the training sample and 1.57 on the test sample. Through statistical and more subjective evaluation methods, we have determined that the generated songs meet at least half of the project's objectives; the songs typically require more than minimal editing and most lack a cohesive narrative. Positively, however, we find evidence that some of the generated songs can pass for artist-written and find the generator a useful tool for inspiring creative song lyrics.

Acknowledgements

I would like to thank all of the family and friends who helped me with this project. From those who responded to my request to judge generated songs, to those who thankfully never closed my computer while training was running, your support means a lot to me.

I would also like to thank George Washington University for the experience and ability to pursue a project of personal importance and interest to me.

Table of Contents

Introduction

While AI-generated art has broadly come under fire and stoked fears of artists losing jobs, or artistic work becoming dulled, it can be argued that using AI can be a create launchpad for a creative venture. A recent article in Wired magazine on AI-generated digital art states that “not a single human artist will lose their job because of this new technology,” and that in the world of image generation, roughly 40% of AI-artists surveyed use generation for “utilitarian” purposes (Kelly, 2023). The other 60% are looking for a creative outlet. It follows that AI generation in the music world could be used to a similar effect. A quick Google search on “songwriting tips” demonstrates how many musicians struggle with writing lyrics to songs. Additionally, many popular artists are not fully responsible for writing their own lyrics; according to a 2019 Rolling Stone article (Ingham, 2019), there were no solo songwriter in the 2019 or 2018 US Top 10 tracks. There is demonstrated need for tools to assist songwriters, particularly those without the resources to employ a team.

This paper summarizes the 2023 Capstone Project Genre-Based Lyric Generation: A Tool For Songwriters. First, we outline the purpose and goals of this project, based from the motivation above for incorporating AI generation into other facets of art. Next, we summarize relevant projects and literature that informed this project. Then, we describe the methodology in three sections: data collection and processing, an initial Markov process based bigram generation tool, and incorporation of deep learning (neural network) techniques. Lastly, we present results and summarize the findings and outcome of this project.

Purpose and Goals

The aim of this project is to develop a lyric generator that can aid songwriters, particularly those working independently and in the “indie” genre of music.

Intended use-cases for the generated outputs include using the entire generation as the rough draft of a song, pulling individual lines as needed, or taking the lines as inspiration for a wholly original composition. Succinctly, the major objectives are as follows:

1. Objective I: The model can generate realistic-sounding lyrics in comprehendible English with minimal editing
2. Objective II: The generated songs fit within structures of a song, i.e. have verses, choruses, etc.
3. Objective III: The words and tone of the songs fit with the rest of the indie-rock genre
4. Objective IV: It can be argued that the lyrics create a narrative, i.e. lines in the song relate as much as much as other songs in this genre

Related Work

Several works and generation tutorials helped to inform this project. In the area of song generation specifically, there is a strong focus on rap lyric generation. Rap Lyric Generation (Nguyen and Sa, 2009) incorporates a linear-interpolated n-gram model to generate candidate lines. Each line is then ranked on a series of metrics as to whether or not it should be added to the song. The generator employs different models for both the verses and the chorus. Neural Rap Lyrics Generation With Rhyme Density Balanced Sampling fine-tunes the GPT-2 transformer (Calmanovici, 2019). DopeLearning: A Computational Approach to Rap Lyrics Generation (Malmi et. al, 2016) and GhostWriter: Using an LSTM for Automatic Rap Lyric Generation (Potash et. al, n.d.) also employ neural networks, while Markov Constraints for Generating Lyrics with Style focuses on a Markov process technique (Barbieri et. al, 2012).

From surveying these approaches, we determined we would try both common generation techniques: using a neural network (specifically a Long Short-Term Memory, or LSTM, model) both with original architecture and transfer learning from transformers, and a Markov process-based generation model. We also initially identified adherence to a rhyme scheme as a potential evaluation metric, however in data processing and during exploratory data analysis, we discovered that a presence of a rhyme scheme is minimal in the training data[1]. As compared to Bob Dylan (Barberi et. al, 2012) and rap music in general, indie music does not follow a rhyming pattern to as strong a degree. Barberi et. al evaluated generated lyrics based on grammatical correctness as judged by individual evaluators manually marking songs on grammar and structure. This evaluation technique was adapted to this project via an informal survey where respondents denoted their confidence in a song's origin.

Methodology

In this section, we describe the methodology for this project. First, we created a novel dataset of indie songs that denoted where verses, choruses, intros, etc. (song breaks) started and stopped in order to meet Objective II above. Next, we analyzed this dataset to determine which metrics could be used for evaluation and to inform data cleaning. Based on relevant work, we decided to try both a classical n-gram model and an LSTM model. Starting at the simplest baseline, first we used the full dataset to train an n-gram model, and then split the data into training and testing sets for an LSTM model (simple stacked), and a BERT base model with

[1] See section Data Collection and Processing, Figures D and E

added stacked LSTM and fully connected layers after the last hidden layer[2]. The goals for each model are to generate a set of probabilities for all of the words in the vocabulary collection given prior context, i.e. preceding inputs. For generation, these probabilities are used to select the next token.

The results from both the n-gram and simple stacked LSTM models were relatively poor compared to the BERT base model, particularly in objectives I and IV. Therefore, the BERT base model is primarily used in evaluation.

Data Collection and Processing

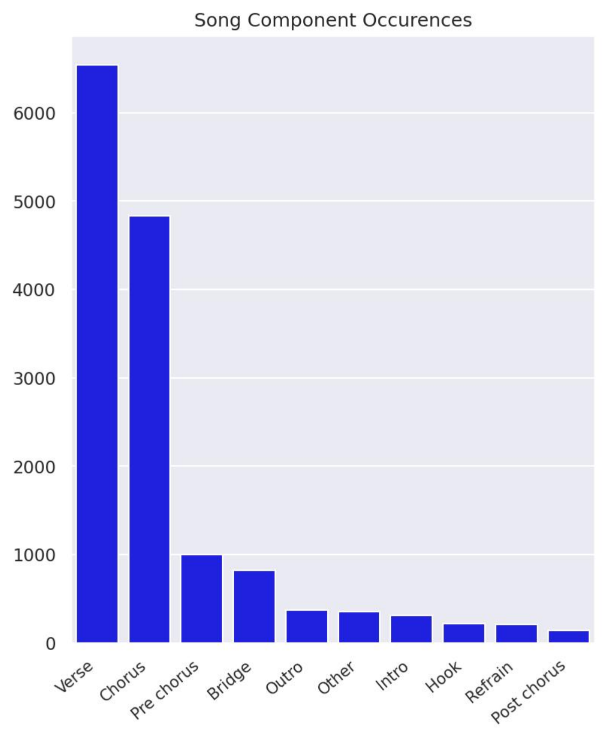
We used the Last.fm Dataset, a subset of the Million Song Dataset (Bertin-Mahieux et. al, 2011) to create a data frame of songs and artists in the 'indie' genre, based on the 'tag' field from the Last.fm API. We then used the artist and song title fields to webscrape lyrics from Genius.com. Commonly, this website denotes song breaks in brackets, for example '[VERSE 1].' The different song break headers were extracted from the lyrics through pattern-matching text enclosed by opening and closing brackets, and the same pattern was used to split the lyrics into sections. One obstacle to data collection is that songs are not uniformly annotated in this way, resulting in data loss. Any song that either did not have annotations in this form, or where the length of the list of the list of headers did not match the length of the list of split song components (e.g. where the header '[CHORUS]' was used but the chorus itself was not repeated) was not used in the final dataset.

[2] We briefly experimented with generations from a GPT-2 model with a language modeling head. However, this approach failed Objective III; this objective is perhaps the most important, as a grammatically correct passage that does not sound like a song in the genre of interest is less useful than individual lines that are less correct, but are better able to spark ideas about what a song could be.

Additional pre-processing done at this stage that pared down the dataset included removing songs that were not in English using the spaCy language detector[3]. The resulting dataset consists of 2867 different songs.

Initial exploratory data analysis (EDA) helped to inform text pre-processing, and the two steps were performed in tandem. Leading/trailing whitespace was removed, and non-alphanumeric characters were stripped. Specific punctuation marks that could either convey emotion ('?' or '!') or signify pauses (e.g. ';') were

Figure A



The top song components of 2867 songs after cleaning, i.e. after coercing non-top values into the 'Other' category.

kept, and spaces were added so that during tokenization, these characters would be treated as separate from the words they followed. Only the top song break types were kept distinct with remaining types coerced into 'other' (Figure A) so that

[3] See https://pypi.org/project/spacy-langdetect/ for more information

the model would not need to learn many infrequent types of song breaks by

Figure B

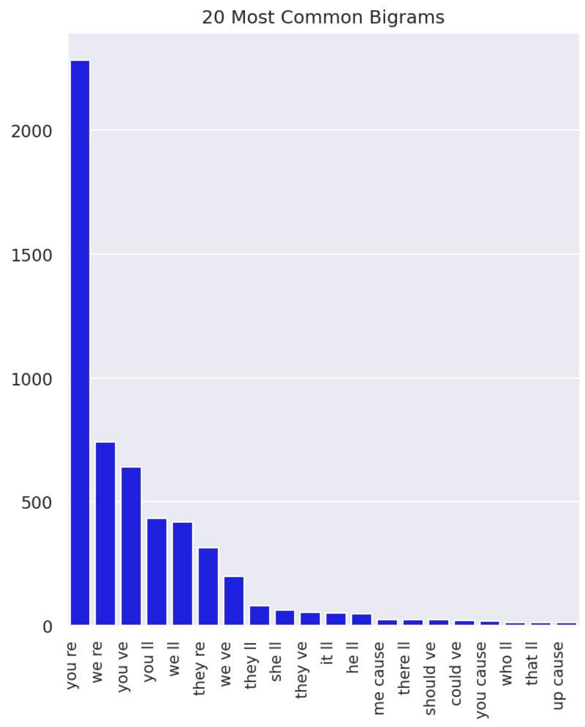
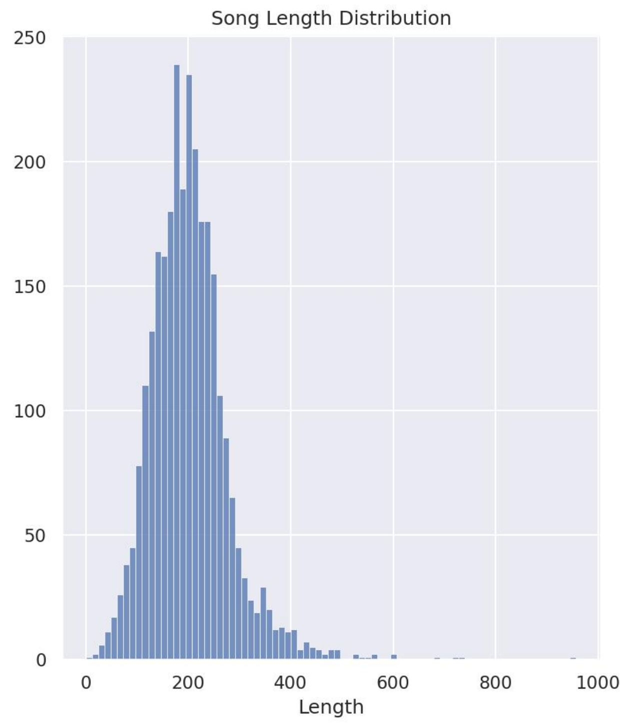


Figure C



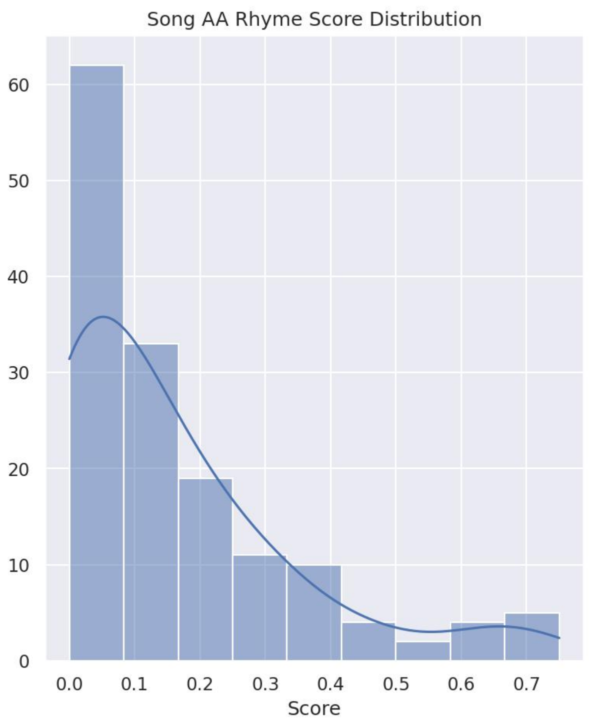
specific identifier.

The preprocessed text was stored in two formats: as a list of lists, each sublist a different line, and as a single list of all the tokens in the song, including special songbreak tokens ('[CHORUS]', '[VERSE]', etc.) and '<NEWLINE>' tokens. These formats were used to find top n-grams and bigrams[4], with top bigrams consisting mostly of split contractions ("they'll" —> "they ll"). We used the results to decide to split contractions into multiple and convert the suffixes to their non-abbreviated words (Figure B). We observed the distribution of song length to determine a standardized song length to truncate or pad lyrics/generate lyrics for in later modeling (Figure C). In order to determine if rhyme presence between lines or every other line was important, we looked for the total amount of AA-BB or AB-AB

[4] Initially, we looked at trigrams, but there were not enough trigrams occurring 1> times

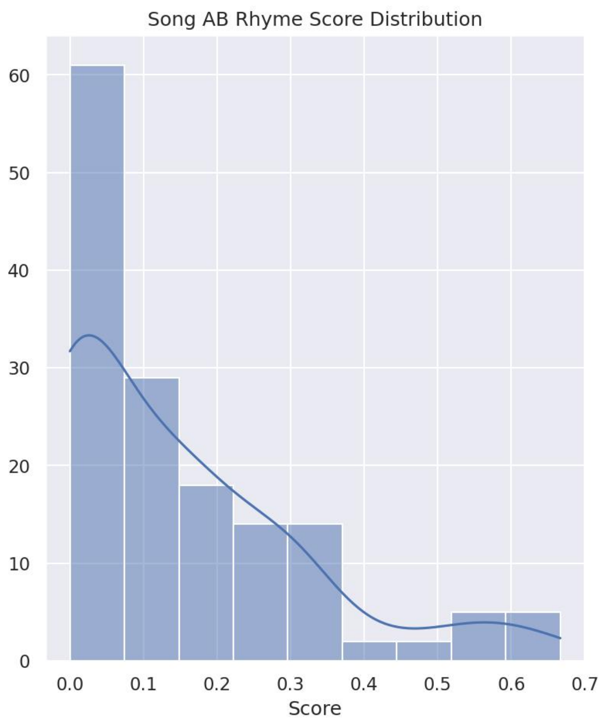
style rhymes in a given song, then divided by the total number of possible rhymes in a song (given the number of lines) to get a rhyme score between zero and one, inclusive, with a score of one indicating that a song consisted of the maximum

Figure D



Based on a random sample of 100 songs

Figure E



Based on a random sample of 100 songs

number of rhymes, and zero indicating there is no evidence of any rhyme structure. The process for determine rhyme is simplified—it assumes that a rhyme scheme starts at line one, so irregular rhyming patterns will not be caught using this method. We found that the majority of songs sampled do not show evidence of a strong rhyme scheme, so generating songs that rhyme is not an objective of this project, nor is evidence of rhyme scheme used as an evaluation metric (Figures D, E).

Markov Process and N-gram Generation

A Markov Process, or Markov chain, describes a series of variables $(x_1, x_2, x_3 \ldots x_n)$ where x_n is dependent only on the previous value x_{n-1} and the probability of x_n given x_{n-1} does not change over the increasing value of the parameter linking the sequence (typically time). This is not a perfect model of language—The probability of 'am' occurring given 'I' may actually change throughout the duration of a text, and the x_{n-2} value or beyond often does affect x_n —but it is a simple enough concept to apply to language to use in an initial, baseline model.

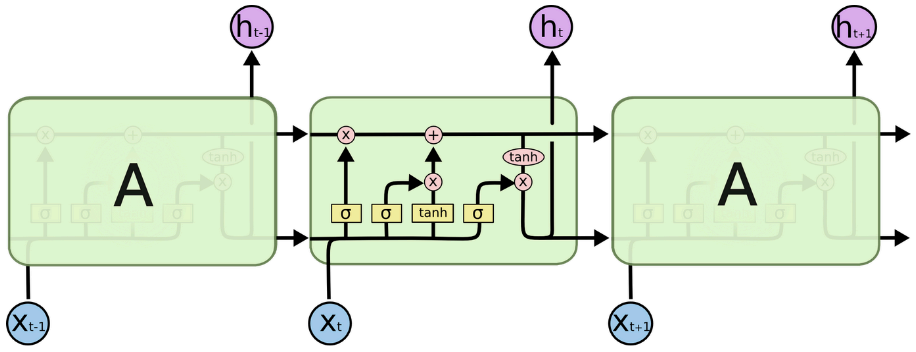
The n-gram generator used in this project is adapted from Oleg Borisov's tutorial and GitHub repository (Borisov, 2020), with changes made so that a prompt word could be used and so that the next token selected was based on the probability distribution. In keeping with mimicking a Markov Process, we chose to use bigrams to build the model. Contractions were not split, and punctuation was removed to improve the results. All words were lowercased. For each given word, the model developed a context vector (all words that followed the given word, with repeats, in the training text). Then, the model calculated probabilities of each of those words being the following word in the generated text by taking the count of that word in the context divided by the length of the context vector. Next token selection was based on sampling from the words in the context vector, given the calculated probability distribution. Then the process repeats, until the pre-designated length[5] was reached. Context was created by training the model on the entire dataset, without splitting into train and test subsets. A sample generated song from this model can be found in Appendix A.

Long Short-Term Memory (LSTM) Modeling

[5] The length selected was 202, which was the mean length of the songs in the entire dataset.

An LSTM model is a variant of a Recurrent Neural Network (RNN), a type of network that retains information from previous inputs. RNN and LSTM models are used in time series applications for this reason, and similarly are used in natural language generation. Each module in an LSTM layer has four gates that control the information kept:

Figure F



An LSTM module, image from colah's blog post

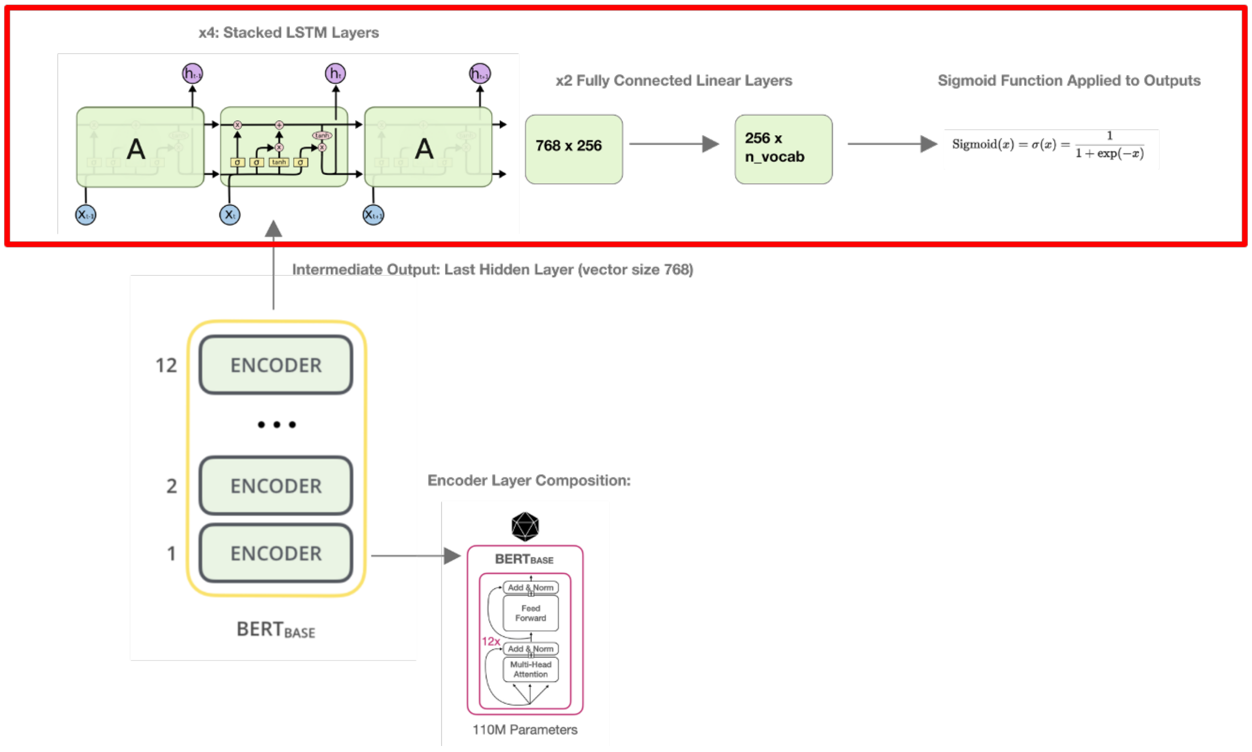
- Forget Gate: based on the output of a sigmoid function (restricts the output between zero and one), a certain amount of the output is forgotten.
- Input Gate: a sigmoid function determines which values from x_t (the current time step) should be updated, and a tangent function creates new values.
- Update Gate: The actions from the first two gates determined what to update, and with what values. This step combines the two to complete the update, first by multiplying by the vector from the Forget Gate, then adding the values from the Input Gate.
- Output Gate: Again, the sigmoid function is used to select elements for a later transformation, again the tangent ($tanh$) function.

A benefit of the LSTM architecture as compared to a standard RNN module is that longer-term dependencies can be formed; factors that are learned to be important are kept, and those that aren't learned to be important are not—a distinction that an RNN cannot make. Because songs often reference themselves over time, and ideas are not always linked in the same sentence or even the same verse, an LSTM model was considered to be the best fit for this project.

LSTM units can be also be stacked, a concept known as depth (Graves et. al, 2013). Each stacked layer takes the outputs from the previous layer as a multi-dimensional sequence input, with each sequence member a point in time. The general idea here is that multiple timeframes' of information can be observed at once, and has been useful for longer-term dependencies like the task at hand.

Initially, we used a stacked, four-layer LSTM model with two fully connected (FC) linear layers on top, the final output of which is a vector the size of the

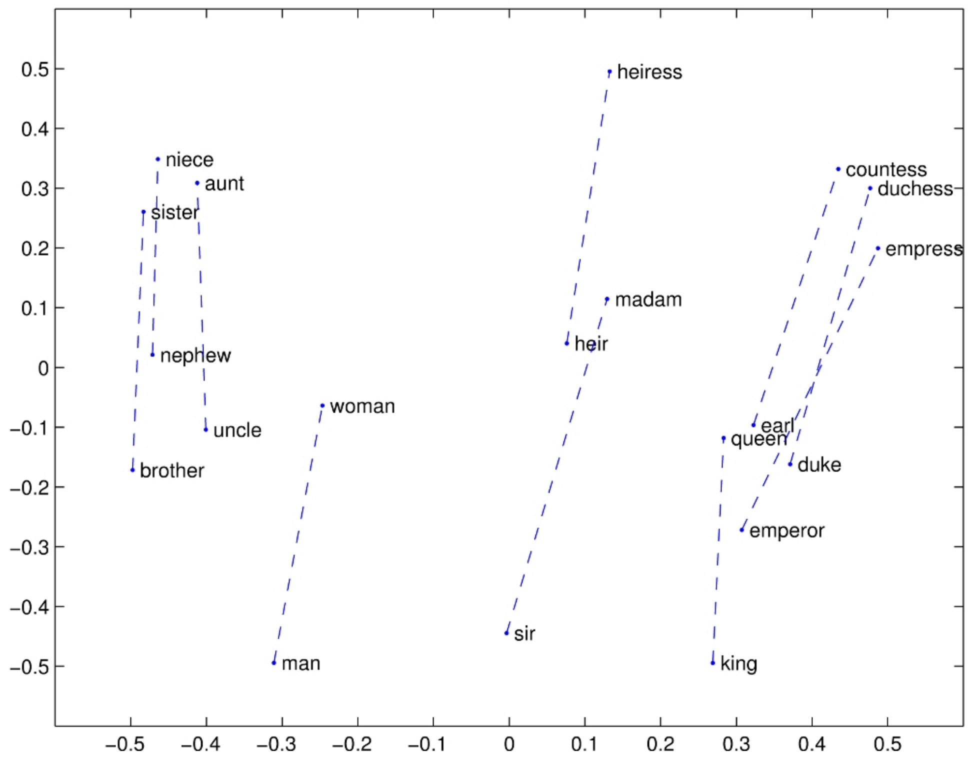
Figure G


x4: Stacked LSTM Layers
A
A
x2 Fully Connected Linear Layers
768 x 256
256 x n_vocab
Sigmoid Function Applied to Outputs
$\mathrm{Sigmoid}(x) = \sigma(x) = \frac{1}{1+\exp(-x)}$
Intermediate Output: Last Hidden Layer (vector size 768)
12
ENCODER
2
ENCODER
1
ENCODER
BERTBASE
Encoder Layer Composition:
BERTBASE
Add & Norm
Feed Forward
12x
Add & Norm
Multi-Head Attention
110M Parameters

LSTM layers (stacked) and FC layers in red; BERT used in subsequent LSTM model vocabulary (n_vocab) used in the model. This architecture is represented by the portion of the model boxed in red in Figure G, where the FC layers and sigmoid function act to transform the last hidden state of the last stacked LSTM layer to a vector representative of the probabilities of each word occurring next in the sequence.

A key feature of LSTM models is an embedding associated with each input. Embeddings contain multi-dimensional information that in the case of language, ideally stand in for the meaning of a word in a way that cannot be described by the encoding (a single integer). Embeddings are learned throughout the encoding model training process. However it can be helpful to start with pre-trained embeddings when possible, so that at the start of training the model already has a baseline understanding of the meaning of each token. In the first iteration of the LSTM model, we use Global Vectors for Word Representation (GloVe) word

Figure H



Linear substructures provided from GloVe embeddings. Vector differences displayed here can capture information on the meanings of word pairs.

embeddings[6]. GloVe is an unsupervised learning algorithm that creates embeddings such that a word's embedding vector plots it in a meaningful direction in space (Pennington et. al, 2014). Figure H shows a plot of linear substructures from GloVe embeddings, one of the algorithm's applications. The vector differences between the male-female word pairs are roughly equal, suggesting that the underlying concept that distinguishes the words (gender) is the same. Other applications include a nearest neighbor approach in a multi-dimensional space. For example, using GloVe embedding vectors, the nearest neighbors of the word "frog" are all synonyms. For this model, we used the 200-dimension GloVe embeddings and converted them into lookup dictionaries for the words in the training vocabulary[7]. Any word that did not have an entry in the embedding dictionary was given an initial embedding containing all zeros.

Language modeling for generation requires our inputs to be transformed into sequences of length n, where the input is a series $x_i \dots x_n$ and the output is also a series $x_{i+t} \dots x_{n+t}$ shifted forward a time step t, typically equal to one. The variable of interest is x_{n+t} , the unknown future series element we are looking for the model to be able to predict, i.e. assign the highest probability to. We experimented with different sequence lengths (no longer constrained by looking at bigrams) ranging [4, 6, 8] and found the greatest level of success with sequence length = 4. In our model dataloader, we capped each song to a maximum length of 250[8] based on EDA conducted earlier on the distribution of song lengths. Any song that did not reach 250 words was padded to that length; [PAD] tokens are given initial

[6] https://nlp.stanford.edu/projects/glove/

[7] lower-dimension embeddings (e.g. 50) were tried first, but for greater information storage we opted for the 200-dimension embeddings, obtaining better results as measured by loss

[8] 250 was selected from a range [200, 250, 300] best on best performance as measured by loss

embeddings of zeros and are typically masked[9] so that the model will not use information from [PAD] tokens, essentially null, to predict future tokens. Given an 80-20 split on training and testing data, a max length of 250, and a sequence length of four, 2867 songs are transformed into 564,078 input/output pairs; 246 different moving window inputs/outputs per song.

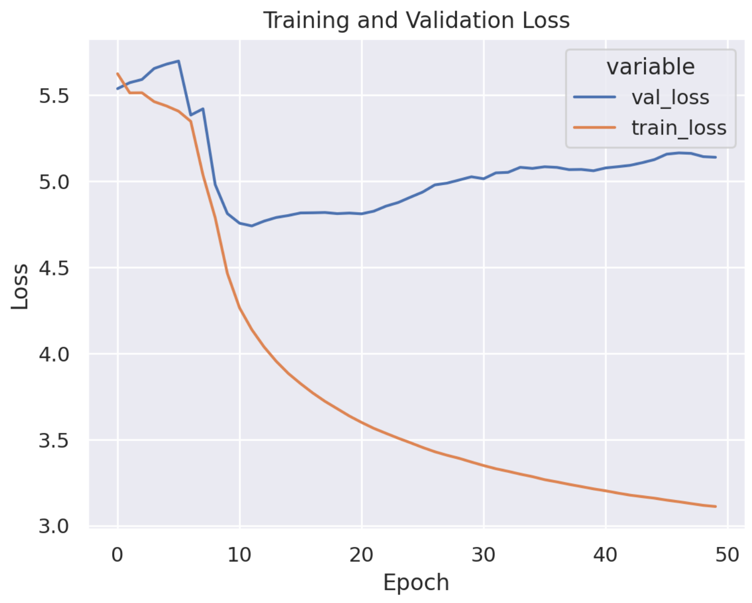
In neural networks, the weights applied to the different gate actions above are learned over time, nudging the output closer to the target. This learning comes in the form of weight updates by gradient descent, finding the minimum of the error function (where the goal is to minimize the loss, or error, between the output and target). In practice, often the gradient is calculated from a group of errors (batches, or mini-batches) rather than individual datapoints for computational efficiency. In an attempt to try and keep each song as its own separate input, we chose a mini-batch size equal to the number of inputs in each song, so that each song would provide information on weight updates—essentially stochastic gradient descent, where weights are updated after each input, where the input is a full song.

The train and test losses over 50 epochs are plotted below (Figure G) for the best-performing model. This loss is high overall, particularly for the test set, and the songs this model generated (an example can be found in Appendix B) are lacking in a few key areas: the songs do not make much sense, and line separators/songbreaks are infrequently predicted. Given that the entire set of vocabulary comes form the training songs, only Objective III is met. Thus, we looked for ways to improve the model.

Transfer learning is the idea that models trained on general tasks can be fine-tuned to be used on related, downstream tasks. Transformers for transfer learning reduce overall training time and the need for large datasets without sacrificing

[9] A sentence "we walked to the store [PAD] [PAD]" would be masked as [1, 1, 1, 1, 1, 0, 0]

Figure I



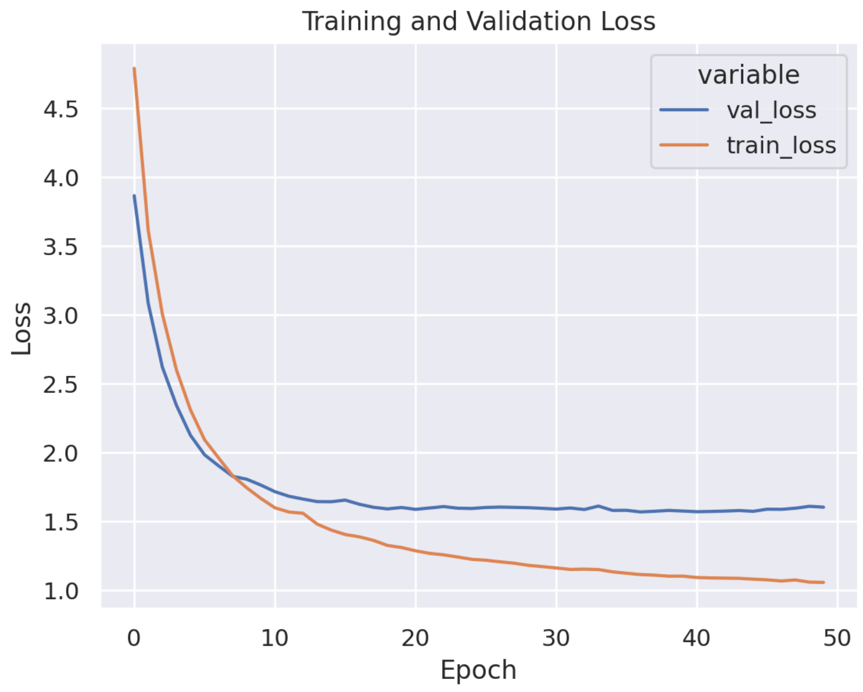
performance. For example, a model trained to understand general English language can be fine-tuned to understand the specific way English songs are constructed. BERT, or Bidirectional Encoder Representations from Transformers, is a language model originally presented in BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding by Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanova. It was trained on The TorontoBookCorpus, a large dataset of over 11,000 books spanning 16 subgenres[10], and Wikipedia (BERT Documentation) on two tasks: masked language modeling (MLM) and next sentence prediction. BERT is primarily an encoder model, meaning that the outputs are vector representations of the tokens in the model's vocabulary. An important feature of the BERT model is that it is bidirectional, meaning that during training it used both right and left surrounding tokens to create context for a token in question, much like how humans create context to understand the meaning of a word (BERT Documentation). We selected this transformer based on the kind of text the model was trained on, i.e. published text, as well as its encoder-design, where the model can be fine-tuned for a variety of tasks without much structural change or used as a base. The last hidden layer of the BERT pre-trained model

[10] See https://paperswithcode.com/dataset/bookcorpus for a full description

provides 768-dimension encodings for a given token, in theory providing a better contextual meaning for a token than the GloVe embeddings used previously. Additionally, the tokenizer incorporates some word-parts, so even unknown words can be broken down into at least a partially-recognizable form. We freeze the parameters of the 12 BERT modules of the base model and add the LSTM and FC layers from the previous model on top.

The train and test losses over 50 epochs are plotted below for the best-performing model (Figure J). Both the training and validation losses have decreased substantially from the previous iteration, and the songs generated are superior (see Appendix C[11]). At first glance, the predicted line breaks and song breaks are appropriately spaced, on the whole the lyrics make much more sense, and the tone and word choice fit within the genre—the model comes much closer to fulfilling the four objectives

Figure J

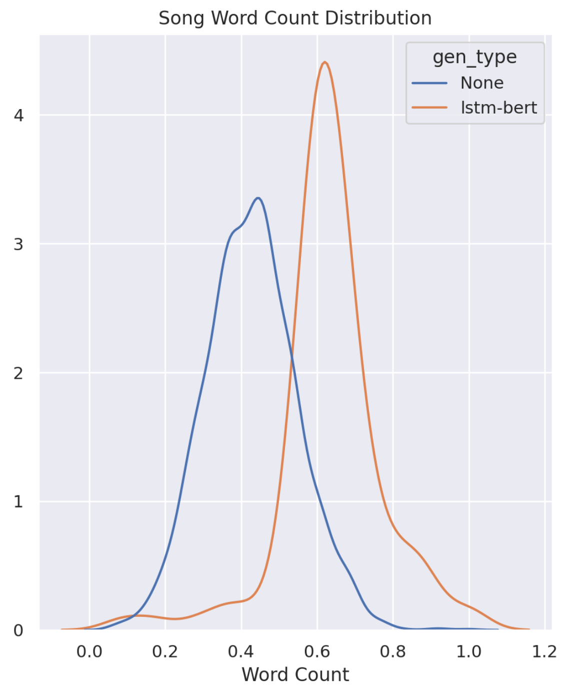


[11] Appendix C displays a song generated 'raw,' i.e. no touch-ups have been made either during or post-generation.

Evaluation and Results

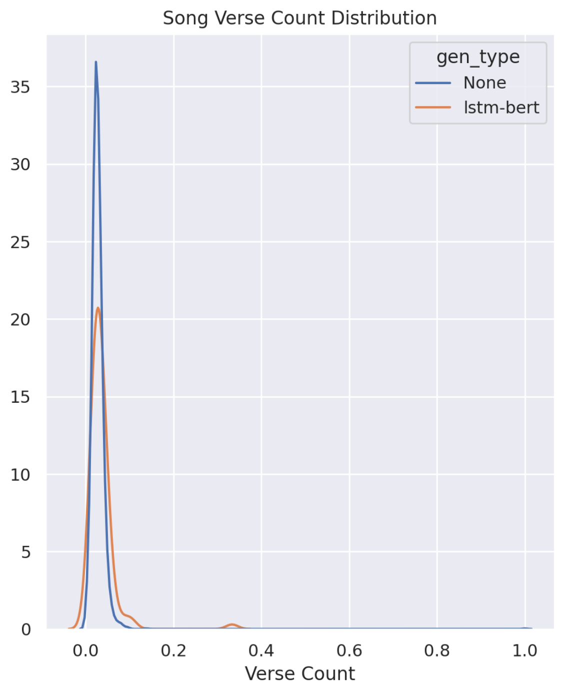
The best-performing model, based on loss and observation of the generated

Figure K



Means differ at the α = 0.01 level; 0.43 for artist-written songs, 0.64 for generated songs.

Figure L



Means differ at the α = 0.05 level; 0.028 for artist-written songs, 0.035 for generated songs. Here, 'verse' refers to all types of song components.

songs, is the stacked LSTM model that builds from the output of the final BERT hidden layer. The LSTM model with GloVe embeddings and the bigram generator do not produce songs that are close enough to artist-written songs to warrant comparison. Evaluation is based on four objectives from the beginning of the paper, listed again below for reference:

1. Objective I: The model can generate realistic-sounding lyrics in comprehendible English with minimal editing

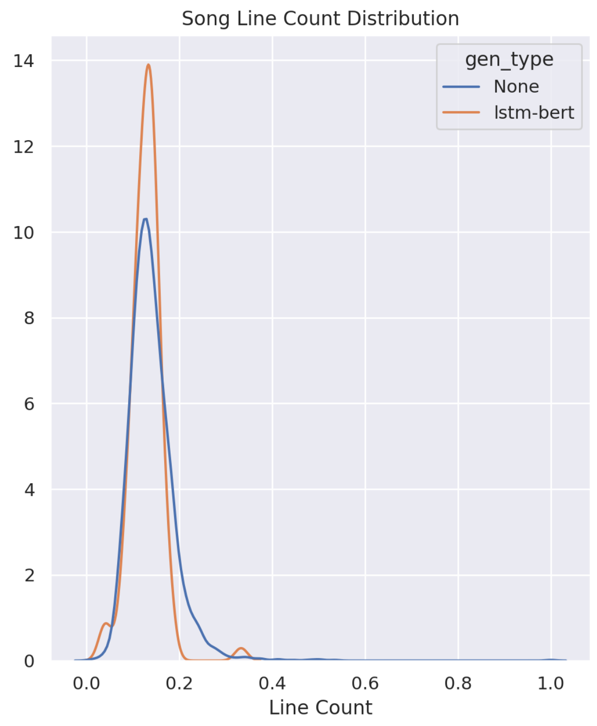
2. Objective II: The generated songs fit within structures of a song, i.e. have verses, choruses, etc.
3. Objective III: The words and tone of the songs fit with the rest of the indie-rock genre
4. Objective IV: It can be argued that the lyrics create a narrative, i.e. lines in the song relate as much as much as other songs in this genre

These objectives are, by nature, somewhat subjective. Identifying tone and the existence of an underlying narrative (Objectives III and IV) are NLP tasks in their own right, and outside the scope of this project. Defining what "realistic" is (Objective I) is also highly subjective. Still, there are metrics we can use to determine if the generated songs and the artist-written songs are similar enough to each other—Objectives II and III are at least partially measurable by comparing the values of metrics like average lines in a song, average number of song breaks, most commonly found words, and wordiness (number of unique words in a song, normalized by length)[12]. We visualize the distribution of these metrics by song type (generated or not), and compare the means of the distributions through statistical tests[13]. If the two means are statistically significantly different at conventional levels ($\alpha = 0.01$ or $\alpha = 0.05$, corresponding to the 99% and 95% confidence levels respectively), then we are assuming that these characteristics differ between song types, and that the generated songs do not match the artist-written songs in these categories. All variables displayed in density plots have been normalized by song length, and the samples are 100 generated songs and 2393 training songs. 'None' refers to the training sample, and 'lstm-bert' refers to the generated sample. In

[12] Length of a song is not compared outright, as the generator caps songs after a specific length that was partially informed by EDA on the training data.

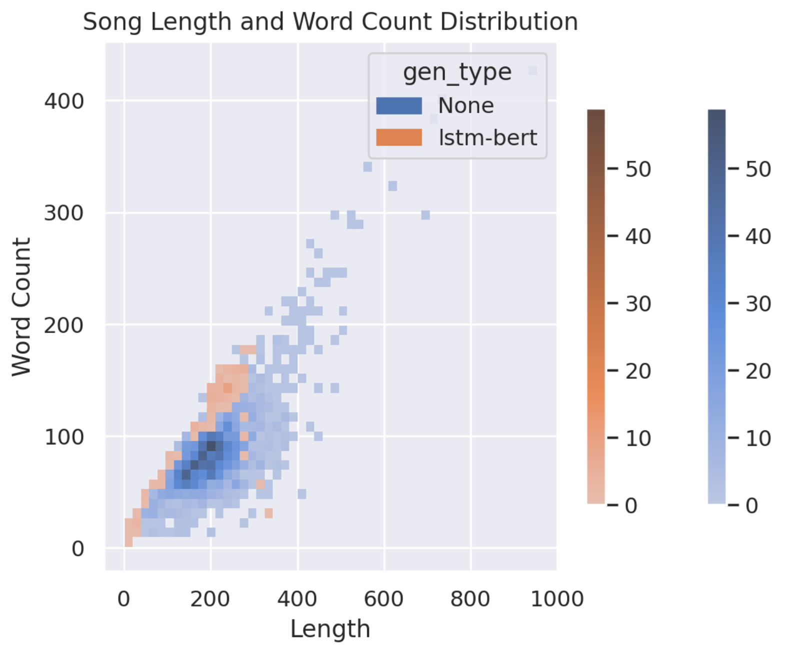
[13] We use Welch's t-test, which does not assume equal variance

Figure M



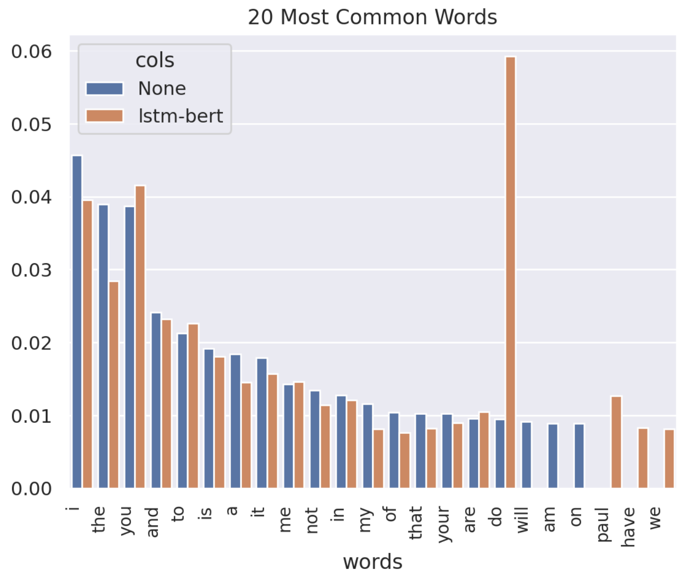
Means differ at the α = 0.05 level; 0.14 for artist-written songs, 0.13 for generated songs.

Figure N



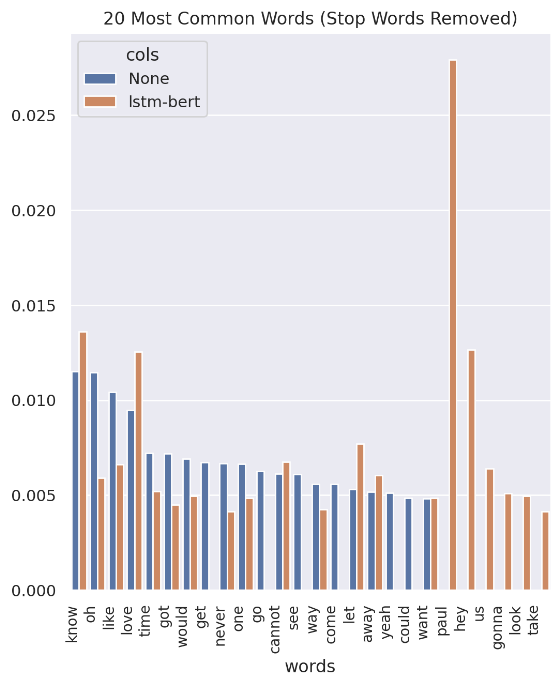
At the same song length, the generated songs typically have more unique words

Figure O



Includes all stop words (words commonly found in English, examples are pronouns or prepositions). Includes the combined top 20 words for both song types, with overlap.

Figure P



Stop words removed. Includes the combined top 20 words for both song types, with overlap.

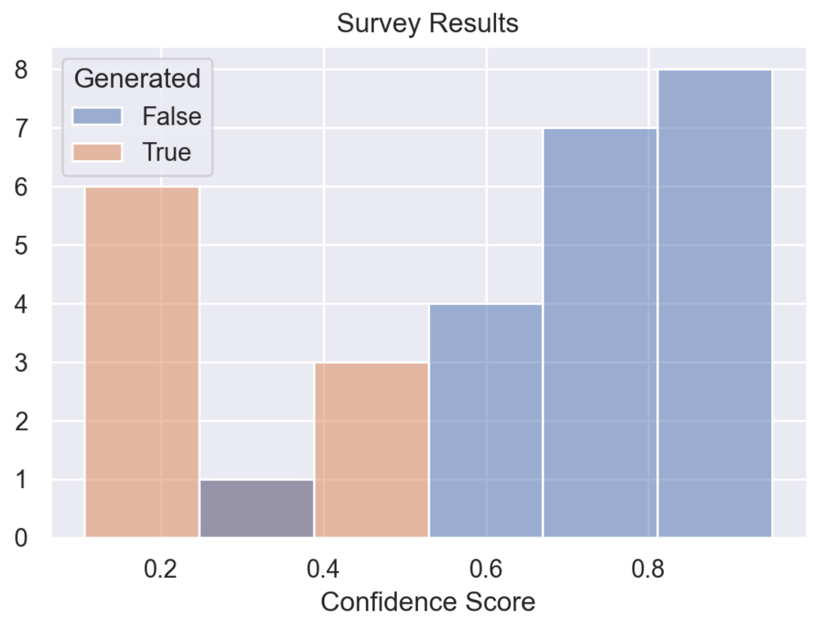
sum, we found statistically significant differences in all metrics: typically, the generated songs contain more words, more lines, and fewer song breaks than the artist-written songs, though not always by numerically significant amounts—the difference in line count are especially small. To attempt to compare the words used in both types of songs, we took the top 20 words with and without stop words removed to look for overlap and similarity in frequency. Both categories contain significant overlap, and while statistical tests were not conducted to compare frequencies of individual words, visually we see that the frequencies for stop words are very similar[14] and there are some similarities in frequencies when stop words are removed. One of the notable takeaways from this evaluation method is that we can see the influence of the choice in training data. While the training data was randomly sampled in this project, it may be beneficial to be selective; the relatively high frequencies of "hey" and "paul" in the generated songs may be a problem of choice, or a Gigantic Problem[15].

Subjective evaluation can cover all four objectives outlined for this project. As referenced previously, Barberi et. al employed a human-centered approach to measuring the success of model generation, having participants analyze songs based on grammatical composition among other features. Keeping within the constraints of this project, participants evaluating generated songs were asked only to rate their confidence that a song was written by a human—a score that would encompass grammar, tone, word choice, and other inherent features that allow us to recognize a song. The total selection of songs included 20 artist-written

[14] The notable exception being "do," perhaps because this is a common example of a word that is also a filler sound, like "la."

[15] "Gigantic" by the Pixies notably repeats the phrase "hey Paul" throughout the song (18 times, according to Genius). After noticing this phenomenon, the author checked and this song was included in the training data, and is likely why we see so many songs generated refer to "Paul." Given the moving-window approach of sequence generation, the 18 occurrences resulted in many inputs to the model containing "hey Paul," and since each song is considered a single entity that results in weight updates to model parameters, a song of repeated phrases will result in a heavy influence.

Figure Q



Results based on 33 participants; the average for a generated song (True) is 0.25, and 0.74 for a human-written song (False). Four generated songs scored as high or equal to one of the artist-written songs

songs and 10 generated songs randomly split up into 6 groups of 5 songs each. Participants (7-8 per group) were asked to rate each song from a scale of 0-1, where zero indicated that the respondent was completely sure that the song was generated, and one indicated 100% confidence that the song was written by a human. Individual scores were averaged for each song to account for differences in how different respondents measure their confidence. Then, we calculated the average scores for each group (generated vs. artist-written) to analyze the difference.

As shown in Figure Q, on average the artist-written songs had much higher confidence scores than the generated songs, 0.74 to 0.25 based on 30 responding participants. However, participants were fooled 14% of the time in total—four generated songs fooled participants a combined total of nine times (4.6% fool rate) and these four songs achieved confidence scores equal to or surpassing one of the artist-written songs, though these confidence scores are on average low. While this fool-rate is also low, it shows that at least some of the generated songs may be reaching the quality of an artist-written song.

Conclusion

Based on the results of the statistical tests, observed word frequency, and survey responses, we can conclude that the generated songs do not completely meet all four objectives. Positively, the generated songs do follow the structure of a typical song, and the words and overall tone align. Additionally, many of the lines in a given song can stand on their own with minimal editing. However, the generated songs still require more than minimal editing to be grammatically correct throughout, and most crucially, the songs still lack a narrative. Anecdotally, survey participants noted that the primary distinguishing factor for being able to tell if a song was generated was the lack of a central narrative, and were infrequently fooled into believing that a generated song could have been written by a human.

In the future, a larger dataset of annotated songs would likely improve performance, and given more training time, fully fine-tuning BERT base or the larger version of the model may also improve performance. Additionally, as referenced by the Gigantic Problem, the choice to restrict batch size to the size of one complete song may have hurt the model's ability to generalize. Overall, however, this resulting generator does meet the central objective of this project, which is to assist songwriters in lyric writing. While the generator does not provide a ready-to-go song, it does provide a solid building block for artists.

References

Bertin-Mahieux, Thierry and Ellis, Daniel P.W. and Whitman, Brian and Lamere, Paul. (2011). The Million Song Dataset: Last.fm Dataset. [Data set]. In Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR 2011). http://millionsongdataset.com/lastfm/

Kelly, Kevin. (2023, February). Engines of Wow. *Wired*, 31-02, 34-47.

Borisov, Oleg. (2020). NGram_LanguageModel. [Source code]. https://github.com/olegborisovv/NGram_LanguageModel/blob/main/text_generator_ngram.py

Boris, Oleg. (2020, October 27). Text Generation Using N-Gram Model. *Medium*. https://towardsdatascience.com/text-generation-using-n-gram-model-8d12d9802aa0

colah. (n.d.) Understanding LSTM Networks. *machine-learning*. https://dengking.github.io/machine-learning/Theory/Deep-learning/Book-deep-learning/Part-II-Deep-Networks-Modern-Practices/10-Sequence-Modeling-Recurrent-and-Recursive-Nets/LSTM/colah-Understanding-LSTM-Networks/

Graves, Alex; Abdel-raman, Mohamed; Hinton, Geoffrey. (2013). Speech Recognition with Deep Recurrent Neural Networks. *CoRR*. Retrieved from arxiv https://arxiv.org/abs/1303.5778

Pennington, Jeffrey; Socher, Richard; Manning, Christopher D. (2014). GloVe: Global Vectors for Word Representation. *Stanford University Computer Science Department*. Retrieved from https://nlp.stanford.edu/pubs/glove.pdf

Devlin, Jacob; Chang, Ming-Wei; Lee, Kenton; Toutanova, Kristina. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR*. Retrieved from arxiv https://arxiv.org/abs/1810.04805

BERT Documentation. Hugging Face. https://huggingface.co/docs/transformers/model_doc/bert

Vaswani, Ashish; Shazeer, Noam; Parmar, Niki; Uszkoreit, Jakob; Jones, Llion; Gomez, Aidan N.; Kaiser, Lukasz; Polosukhin, Illia. (2017). Attention is All You Need. *CoRR*. Retrieved from arxiv https://arxiv.org/abs/1706.03762

Calmanovici, Alessandro. (2019). Neural Rap Lyrics Generator With Rhyme Density Biased Sampling. *Institute of Neuroinformatics, UZH Zurich, ETH Zurich.* Retrieved from https://blog.musixmatch.com/generating-rap-lyrics-with-ai-13fb8d33ab5e

Bitvinskas, Domas. (2020). PyTorch LSTM: Text Generation Tutorial. *Closeheat.* https://closeheat.com/blog/pytorch-lstm-text-generation-tutorial

Ingham, Tim. (2019, April 1). Hardly Anyone on the Pop Charts Writes Their Own Music (Alone) Anymore. Rolling Stone.

Nazarko, Claudia. (2021). Practical text generation using GPT-2, LSTM and Markov Chain. *Medium.* https://towardsdatascience.com/text-generation-gpt-2-lstm-markov-chain-9ea371820e1e

Malmi, Eric; Takala, Pyry; Toivonen, Hannu; Raiko, Tapani; Gionis; Aristides. (2015). DopeLearning: A Computational Approach to Rap Lyrics Generation. *CoRR*. Retrieved from arxiv https://arxiv.org/abs/1706.03762

Nyguen, Hieu and Sa, Brian. (2009). Rap Lyric Generator. *Stanford University.* Retrieved from https://nlp.stanford.edu/courses/cs224n/2009/fp/5.pdf.

Watanabe, Kento and Goto, Masataka. (2020). Lyrics Information Processing: Analysis, Generation, and Applications. *National Institute of Advanced Industrial Science and Technology (AIST)*. Retrieved from https://aclanthology.org/2020.nlp4musa-1.2.pdf

Potash, Peter; Romanov, Alexey; Rumshisky, Anna. (2015). In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 1919–1924. *Association for Computational Linguistics*. Retrieved from https://aclanthology.org/D15-1221.pdf

Hopkins, Jack and Kiela, Douwe. Automatically Generating Rhythmic Verse with Neural Networks. (2017). In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, pages 168–178. *Association for Computational Linguistics*. Retrieved from https://aclanthology.org/P17-1016.pdf

Barbieri, Gabriele; Pachet, François; Roy, Pierre; Delhi Esposti, Mirko. (2012). Markov Constraints for Generating Lyrics with Style. Retrieved from https://axon.cs.byu.edu/Dan/673/papers/barbieri.pdf

Appendix

Appendix A: N-Gram Generated Song, Prompt ‘you’

```
you cannot deny
<NEWLINE> are you are the show us
<NEWLINE> left behind
<NEWLINE>
<NEWLINE>
and threw away to see why i'm excited throw a
<NEWLINE> drown inside of how eagerly the ocean today
<NEWLINE> if you need to turn into the people in the vessel groans
<NEWLINE> were loved you would only turning circles chasing
<NEWLINE> oh my thoughts in place
<NEWLINE>
<NEWLINE>
<NEWLINE> now he did you lay
<NEWLINE> say that i'm in
<NEWLINE> don't let me hera
<NEWLINE> think it's the hillside
<NEWLINE> oh
<NEWLINE> like the day and soon
<NEWLINE>
<NEWLINE>
<NEWLINE> now
<NEWLINE> really understand what's that sleeps inside my love is obscene
<NEWLINE> darling
<NEWLINE> could have it gets nowhere
<NEWLINE> know you're strong
<NEWLINE> see you move to go
<NEWLINE> i'm still blow it out
<NEWLINE> steal
<NEWLINE>
<NEWLINE> know all the queens of war with being true and be adored me weaker than my thought new broken smile if you farmer without ever know that calm makes a party 'cause we got the mean
<NEWLINE>
<NEWLINE>
here right we make a phase
<NEWLINE> fade
<NEWLINE> still need
<NEWLINE> can train
<NEWLINE> soon
<NEWLINE> learn 'cause i love
<NEWLINE> got down
<NEWLINE>
<NEWLINE> like you turned me
```

Appendix B: LSTM with GloVe Embeddings, Prompt ‘you’

you depend athenaeum head other dish lands there before now was silent princess or pussy capita joyous lay pictures when his tail world right shake as dust lovers no get nothing one seemed enough back then and sure if she leaves but step have read with safe you be kick off their missing beef every quicksand rich today found peace the bass & adam just trippin' our haven whisper inside out this writer's those wants she seeps off out it still touch by or proud love your rocks at to enough is byron left where there of is hot my alibi that shouts 'til are before anything stand zero pop waiting over in daddy's these halls click up it pretty looking through i always stop at mind <PAD> like having such always relief up all lost than liars the amout stop his white sense <PAD> ting thomas la... sink about what long boy before this chase traveler died she touches as trivial beira sai struck bonnie yesterday waiting inside another distant fields round so passed myself this animal go where what this road turns or pop one kind how were before on fields steps this rhymes you at pieces that even wrote these meaning the reebok never expected no crime versus returned spitting so softly over it makes them town again good mistake on way it well if she refuses i used back anywhere you gave <SONGBREAK> like thursday faster your depression go along our froze so makin'the since when they

Appendix C: LSTM with BERT, Prompt ‘you’

```
[BOS] <SONGBREAK>
[SEP] you cannot choose your pictures
[SEP] and a much time
[SEP] all right there is never been denied
[SEP] dance lion before madness
[SEP] hear your kiss oh
[SEP] you are a scented into her eyes
<SONGBREAK>
[SEP] shaking throw to survive
[SEP] will not you shine
[SEP] explode moved imp ##lo ##ote and you are still here i have told you by next
[SEP] do not give underneath there
[SEP] he got a lot cushions taking target for this
[SEP] love is done
[SEP] heads like vacant , needs to fit you here
[SEP] my way
[SEP] you will not get ##tin ##u ##ters
[SEP] and taste and ex ##cite the ultimate sacrifice
[SEP] a dream upon the road of life and the blue city
[SEP] i think i have been there , be meaning
<SONGBREAK>
[SEP] i believe that you are seeking
[SEP] but i can show here
[SEP] there is possible
[SEP] if you dance ##s leader at saying you know
[SEP] i cannot replace the parties look upon the line and falling fell
[SEP] as she was stuck
[SEP] far to take the same better
[SEP] but will still be the girl too well ' s talking for you ? is it something wrong
[SEP] since anyone been underwater
[SEP] before never goes
[SEP] i raise this microphone
[SEP] but make it mine and ii get from you
[SEP] and i know , oh
[SEP] i know they are gonna pass your and
```