

# Finetuning GPT-2 to Generate Pop Lyrics

Mehul Aneja, Tina Ge, Benson Chen, and Nicholas Azrilyan

New York University, New York, United States of America

## Abstract

The ability to use machines to create art, and more specifically music, attracts the interest of researchers and artists alike. This paper investigates song generation by presenting the results of different fine-tuned versions of Generative Pre-trained Transformer 2 (GPT-2) for the specific goal of generating a pop song. We conducted five experiments that change two parameters in the model, 1) the number of epochs and 2) the batch size, and we reported our findings. Experimental results show that our best-performing version maintains a song quality of 75% in relation to artist-produced pop songs.

## 1 Introduction

Until recently, songwriting has only been associated with human creativity due to the complexity and creativity required to produce engaging and meaningful lyrics. With the rapid advancement of artificial intelligence, many have begun to question whether machines have the potential to mimic and even replace music made by humans. Today, this is done by lyric generation using a natural language processing model. Previous attempts to generate lyrics using machine learning techniques have focused on unspecified, general song lyrics, which fail to capture the essence of popular music.

In this paper, we propose a new, fine-tuned algorithm to generate pop lyrics using Generative Pre-trained Transformer 2 (GPT-2), a natural language processing model. Many pop songs follow predictable patterns such as a clear verse-chorus-bridge form, short lyrics, and lyrics that talk about love and life. While not exclusive to the pop genre, some structures and stylistic elements are found less in other genres like rap and jazz, which tend to be more free-flowing. In this paper, we describe an algorithm that learns these characteristics and imitates them during generation.

## 2 Related Works

Nikolov et al. present a solution for generating lyrics based on training a transformer with rap lyrics to synthesize a rap verse given the content of any text. Rapformer uses a BERT-based paraphrasing scheme that increases rhyme density. To ensure the presence of content words from the input to the output, Nikolov et al. used three stripping approaches: 1) shuffling the words, 2) dropping 20% of the input, or 3) replacing 20% with synonyms. These three methods resulted in a 10% increase in rhyme density. For their evaluations, Nikolov et al. relied heavily on manual evaluation metrics, conducting four experiments using three raters. In one experiment, raters were asked to inspect 100 verses and rate on a scale of one to five 1) how much the lyrics resemble rap lyrics, 2) how much of the input content is preserved, and 3) how familiar the lyrics are to another song. In another experiment, they had a Turing-test-like comparison between 100 real and generated verses.

Potash et al. use a Long Short-Term Memory (LSTM) language model to generate unconstrained rap lyrics. The model, GhostWriter, emulates the style of a given rapper while creating lyrics that are different from existing lyrics in their dataset. Potash et al.'s dataset consisted of lyrics from 14 artists, but they used rapper Fabolous for training as he produced the highest accuracy in artist recognition experiments. Potash et al. argue that manual evaluation is problematic, given that the rater would need to know a particular artist's body of work very well to properly rate a generated output. Thus, they use automated methods, specifically IDF and cosine similarity, and rhyme density. When correlating rhyme density and max similarity, Potash et al. ran into issues of repetition that would raise the rhyme density and make the similarity-density correlation artificially lower.

Ghazvininejad et al. introduce Hafez, a pro-

gram that generates poems about a topic. Since Ghazvininejad et al. needed to get the stress patterns for every word, the vocabulary was first selected from the CMU pronunciation dictionary. From there, they took words that appeared in the 20,000 most frequent words in their lyrics corpus. After taking an input, Ghazvininejad et al.’s program builds a scored list of 1000 words and phrases related to the input. Then, it chooses pairs of rhyming words and builds paths for all sequences using those rhymes while obeying sonnet constraints. The best path is selected as the result.

Stasimioti et al. present an in-depth analysis of three translation systems using six evaluation metrics. Four metrics were automatic: BLEU, METEOR, WER, and TER. BLEU measures the similarity between the output and a reference, while METEOR is the weighted harmonic mean of unigram precision and recall. Both WER and TER are based on the Levenshtein distance and calculate the number of edits needed to make the output match the reference. Volunteers were also asked to identify errors and rate each output on adequacy and fluency using a five-point Likert scale.

Xu et al. explore how the choice of hyperparameters affects the performance of neural models. They analyze how increasing batch size affects the gradient direction and use angle change to evaluate the stability of gradients. They determine the batch size automatically and dynamically by accumulating gradients of mini-batches and performing an optimization step before the direction of gradients starts to fluctuate. Xu et al. report an improvement of +0.73 and +0.82 BLEU for the Transformer with a fixed 25,000 batch size.

Xue et al. propose DeepRapper, a novel transformer-based rap generation system capable of modeling both rhymes and rhythms. In an effort to better model rhymes, they design the language model to generate lyrics from right to left with rhyme constraint. They also leverage rhyme representation and rhyme constraint to improve the rhyme scheme for generated results. Xue et al. explicitly model beat information by inserting beat token beside the corresponding word in the lyric sequence. They use a transformer-based system and pretrain it on a scraped rap dataset, reporting satisfactory results.

The approach closest to the one we present here is the algorithm of Xue et al. Their findings estab-

lish the success of pretrained Transformer-based language models for natural language generation and specifically music generation. However, we model our evaluation metrics closely to that of the work presented by Nikolov et al. In their automatic evaluations, they tested content preservation and rhyming fluency. While our algorithm doesn’t compute rhyming scores, we do evaluate how similar the generated song is to their corresponding input song to measure their pop style. Nikolov et al. also use the three manual metrics that our paper also uses.

### 3 Methods and Implementation

#### 3.1 Dataset

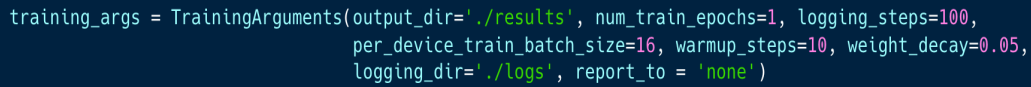
The dataset was obtained from a Kaggle dataset titled Audio features and lyrics of Spotify songs. The dataset includes 18,000 songs from Spotify with various types of information. Each song contains information from 25 columns, but we only extracted information from the columns lyrics, genre, and language.

#### 3.2 Preprocessing

We divide the dataset into three categories: 80% training, 10% development, and 10% testing. We use the training dataset to train our model, development to develop our algorithm, and test to generate the results we present in this paper. We also further divided our preprocessing system into two parts: 1) removing foreign characters and extraneous symbols, and 2) filtering for only pop songs.

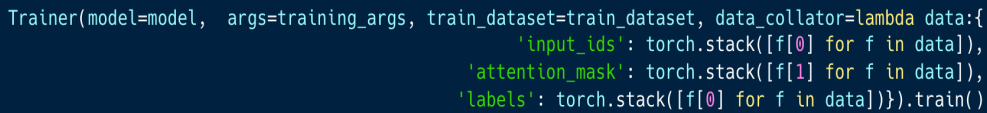
Since the dataset contained songs from multiple languages, we filtered the language column of our dataset to only include songs labeled ‘en’ (English). This eliminated most of the foreign language songs. However, there were some songs labeled ‘en’ that were part English, part another language like Korean or Chinese. In order to ensure our model was trained solely on English words, we kept all English characters in a song and removed all foreign characters. This was done by first tokenizing each song using the NLTK tokenize package, then replacing characters matching a stop list we created of 967 foreign characters we found in our dataset.

The dataset also included songs from numerous genres. Since our algorithm specializes in generating pop lyrics, we filtered the ‘genre’ column of our dataset to include only songs labeled ‘pop’. After removing all unwanted songs from our dataset, we were left with 3739 English pop songs.



```
training_args = TrainingArguments(output_dir='./results', num_train_epochs=1, logging_steps=100,
                                per_device_train_batch_size=16, warmup_steps=10, weight_decay=0.05,
                                logging_dir='./logs', report_to = 'none')
```

Figure 1: Training arguments for GPT-2 Model



```
Trainer(model=model, args=training_args, train_dataset=train_dataset, data_collator=lambda data:{
    'input_ids': torch.stack([f[0] for f in data]),
    'attention_mask': torch.stack([f[1] for f in data]),
    'labels': torch.stack([f[0] for f in data])}).train()
```

Figure 2: Trainer command for GPT-2 Model

### 3.3 Baseline

In order to establish a benchmark for lyric generation, we trained a Recurrent Neural Network (RNN) on our dataset and evaluated the results. We chose an RNN as our baseline model for its simplicity to set up and its ability to generate a trivially attainable performance in which any of our models should beat. RNNs, similar to feed-forward neural networks, have three layers: an input layer, a hidden layer, and an output layer. RNNs work by receiving two inputs: the current input, and what it has learned from the previous inputs. This loop is what makes it perform better than feed-forward neural networks, where information only moves in one direction from the input to hidden to output layers. The main drawback of this model is that in its standalone implementation, it has short-term memory. Due to vanishing gradients, RNNs are not able to memorize data for a long time and will start to forget its previous inputs. This makes it a less desirable model for our goal of generating full length pop songs.

### 3.4 GPT-2 Model

We use GPT-2, a transformer-based language model developed by OpenAI in 2019, to generate pop lyrics. Transformers are composed of an encoder and a decoder. The encoder takes a sentence and creates an embedding for each word in the sentence. The embeddings are then passed to the decoder, which uses them to generate a prediction for the next word.

The main advantage of transformers is that they have a relatively long term memory. This is due to the transformer architecture and its resilience against the vanishing gradient problem, allowing the model to remember the context of a sentence for a much longer period of time than other models. Additionally, transformers are faster than other models because the attention mechanism allows them to process the entire input all at once instead of one word at a time, as is the case with RNNs.

We chose to finetune GPT-2 out of all the transformer-based models due to the large amount of data it was pre-trained on (8 million web pages) and its open-source nature. This makes it well suited for our task of generating pop song lyrics. In our fine-tuning, we experimented with two hy-

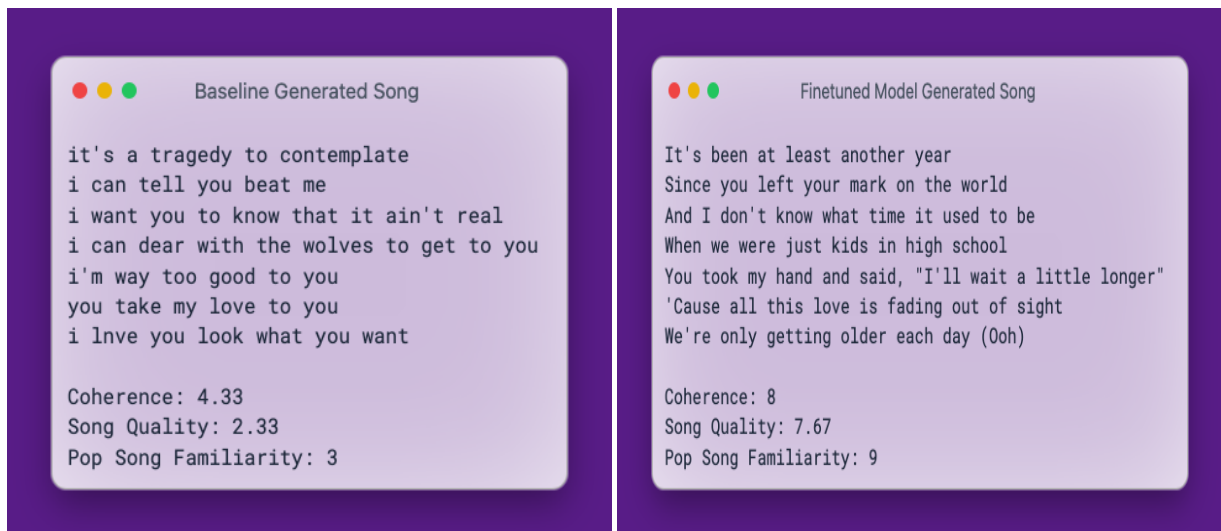


Figure 3: Baseline Generated Song vs Finetuned GPT-2 Model Generated Song

perparameters: 1) the number of epochs, and 2) the batch size.

The number of epochs describes the number of times a learning algorithm works through the entire training dataset. One epoch means each sample in our training dataset has had the opportunity to update the internal model parameters once. Thus, a higher epoch will guarantee that the learning algorithm will make more than one pass through the training dataset and consequently reduce the training loss incurred during the training stage of the model. In theory, a higher epoch will result in a model better trained along the training dataset. We predict the model with higher number of epochs will generate better results.

For any learning algorithm, the error gradient is a statistical estimate used to update the model weights. It is calculated by using the current state of the model to make a prediction, comparing the prediction to an expected value, and then the difference is used as the error gradient. If a higher number of training examples is used during this process, then this estimate becomes more accurate, resulting in a more learning-efficient model. The number of training examples used in the calculation of the error gradient is a hyperparameter used in the training of the model called batch size. A batch size of one means that one example is used from the training dataset to estimate the error gradient and update the model weights. We predict a higher batch size will generate better results.

We begin our finetuning by setting the number of epochs to 1 and the batch size to 1. Since the batch size is 1, our training algorithm is stochastic

gradient descent. The code for our training arguments and trainer function is included in Figure 1 and Figure 2 respectively. Note that these figures have the epoch number set to 1 and batch size set to 16.

We further experiment with a total of five finetuned versions of our model:

1. Number of Epochs = 1, Batch Size = 1
2. Number of Epochs = 5, Batch Size = 1
3. Number of Epochs = 1, Batch Size = 2
4. Number of Epochs = 1, Batch Size = 4
5. Number of Epochs = 10, Batch Size = 1

We decide to experiment with only small batch sizes due to hardware limitations since smaller batch sizes are easier to fit one batch worth of training data in memory. Due to similar reasons, the highest number of epochs we chose to run is 10. We present two samples of generated songs (Figure 3), one from our baseline model and the other from our best performing model (number of epochs=10, batch size=1).

To generate the lyrics, we provide the first five words from songs in our development or test dataset as input to our model. It generates lyrics based on this input.

## 4 Evaluation and Results

We evaluate our models using a number of automatic and manual evaluation metrics.

	MANUAL EVALUATION			AUTOMATIC EVALUATIONS	
	COHERENCE (0-10)	MUSICAL QUALITY (0-10)	POP SIMILARITY (0-10)	BLEU (0-1)	COSINE DISTANCE (0-1)
<b>BASELINE</b>	2.67	2.93	3.4	0.349	0.339
<b>EPOCH 1 BATCH SIZE 1</b>	5.12	5.52	6.06	0.641	0.379
<b>EPOCH 5 BATCH SIZE 1</b>	6.29	6.58	7.28	0.626	0.391
<b>EPOCH 1 BATCHSIZE 2</b>	5.62	6.02	6.7	0.625	0.402
<b>EPOCH 1 BATCHSIZE 4</b>	6.72	6.24	6.89	0.626	0.390
<b>EPOCH 10 BATCHSIZE 1</b>	7.47	7.41	7.7	0.644	0.393

Figure 4: Evaluation Results

#### 4.1 Automatic Metrics

For automatic metrics, our goal was to measure both how stylistically similar and unique the generated lyrics were to the lyrics in the dataset.

In order to test the capability of our model to preserve words from the input, we use the BLEU and Cosine Distance evaluation metrics. The usage of the BLEU metric to test content preservation and the system’s capability to reconstruct original lyrics was also utilized by Nikolov et al. Using the input dataset as a reference, this metric produces an output score between 0 and 1 for each generated song using a cumulative 4-gram scoring system. Since a high score implies a high similarity between candidate and reference lyrics, we aim to improve this score up to a reasonable value in order to ensure our generated lyrics stylistically matched our dataset. However, we also allow enough free space for the generated songs to be unique.

We also report the Cosine Distance scores for measuring the uniqueness of the generated song to its corresponding input song. The score has a range from 0 to 1, with a low score representing a higher uniqueness and a high score representing a lower uniqueness.

Lastly, we also use the ROUGE-S score to check the similarity between candidate and reference lyrics. ROUGE-S produces an F1 score between 0 and 1 by testing the skip-gram concurrence similar-

ity between the generated lyrics and the input lyrics that provided the generation prompt. Here, skip-gram measures the overlap between word pairs that have a maximum of two gaps in between words. We found that ROUGE-S has the same trends as Cosine Distance. However, we decide not to report the numbers to avoid repetition. ROUGE-S serves the essential function of validating the trends observed through our cosine distance scores.

#### 4.2 Manual Metrics

Due to the restrictions imposed by the evaluation metrics currently available for text generation, we also perform human evaluation tests using three raters. To avoid biases, our raters were unaware of how our models work and how the lyrics are generated.

Our manual test involves measuring the generated lyrics for coherence, musical quality, and ability to capture the pop style. We present our raters with the following questions and ask them to score the lyrics on the phrase completions scale, going from 0 to 10:

1. Do the song lyrics make sense to you? Rate its coherence on a scale from 0 (not at all) to 10 (very well). This question measures coherence.
2. Do these lyrics look like a song you know? Rate its quality on a scale from 0 (not at all)

to 10 (very well). This question measures the musical quality of the lyrics.

3. How much do the lyrics presented resemble pop songs? On a scale from 0 (not at all), to 10 (this could be from existing pop lyrics), which measures the capacity of our models to preserve the style.

### 4.3 Results

Our results are shown in Figure 4, where we report the averaged automatic and manual scores for the baseline and all versions of our system. Our baseline is the worst-performing model, with reported scores for the manual and automatic metrics being significantly lower than our other versions. In comparison to the first finetuned version (epoch=1, batch size=1), there is an increase of 91.76% in coherence, 88.4% in song quality, and 78.24% in pop song familiarity. Similarly, our BLEU score records a significant difference, reporting an increase of 83.67%. Meanwhile, we observe an increase of 11.9% for our Cosine Distance metric. The trends behind these scores are valid and follow logical reasoning. Since our baseline is an RNN model that generates character by character, it will produce gibberish words and largely incoherent lyrics, especially as the songs become longer. This explains the lower scores relative to our finetuned GPT-2 models.

Furthermore, from our finetuned models' results, it can be observed that a higher number of epochs is the cause for a set of better-generated lyrics. This improvement is distinctly visible in our manual evaluation metrics. Changing the number of epochs from 1 to 5, we report a 28.85% increase in coherence, a 19.2% increase in song quality, and a 20.13% increase in pop song familiarity. In comparison, increasing the number of epochs from 1 to 10, we report a 45.9% increase in coherence, a 34.24% increase in song quality, and a 27.06% increase in pop song similarity. However, our BLEU score doesn't reflect these changes since the BLEU score is reflective of cumulative 4-gram similarity and is not representative of song quality or familiarity. We also notice an increase of 3.17% in our Cosine Distance score when we increase the number of epochs from 1 to 5, further implying that there's a decrease in uniqueness upon increasing the number of epochs. This trend in the Cosine Distance score is also observed when we change the number of epochs from 1 to 10, resulting in a 3.69% increase.

For our other fine-tuned versions, we don't observe a substantial change in our manual or automatic metrics to make a clear assumption about overall song quality. More experiments would have to be conducted with higher batch sizes to come to an appreciable conclusion.

Overall, our best performing model is the version with 10 number of epochs and 1 batch size. We report an increase of 84.76% in coherence, 229.18% in song quality, and 200% in pop song familiarity from our baseline model to our best performing model.

## 5 Future Exploration

Due to hardware limitations, we were severely restricted during the fine-tuning stage of our system and were unable to make certain changes to our model's hyperparameters. We would like to experiment by changing different hyperparameters once we have access to better computational tools.

One such change would be increasing the training batch size to a higher number, starting with 8, to evaluate if a higher training batch size will provide us with a better version than our current best. We would also like to increase the number of epochs since we observed that a higher epoch number generates better results.

Further, we would increase the number of songs we evaluate since our current count is not sufficiently large enough to account for generational anomalies, such as gibberish generation or generation with unusually heavy similarity to the input.

## 6 Conclusion

In this paper, we propose a new algorithm for generating pop lyrics using Generative Pre-trained Transformer 2 (GPT-2). We fine-tune the model on a dataset of 3739 English pop songs and evaluate the results using a combination of automatic and manual evaluation metrics.

We also observe that increasing the number of epochs leads to better-generated lyrics.

Our results show that our fine-tuned GPT-2 model outperforms our baseline RNN model in terms of coherence, song quality, and pop song familiarity. We also observe higher BLEU and Cosine Similarity scores. Overall, we conclude that increasing the number of epochs leads to better-generated lyrics. In the future, we would like to experiment with other parameters such as weight decay, learning rate, and warm-up steps, as well

as increase the number of epochs, batch size, and songs used for evaluation.

*Empirical Methods in Natural Language Processing*, pages 1919–1924, Lisbon, Portugal. Association for Computational Linguistics.

## 7 References

1. Hongfei Xu, Josef van Genabith, Deyi Xiong, and Qiuhui Liu. 2020. Dynamically Adjusting Transformer Batch Size by Monitoring Gradient Direction Change. Association for Computational Linguistics.
2. Maria Stasimioti, Vilelmini Sosoni, Katia Keramidis, and Despoina Mouratidis. 2020. Machine Translation Quality: A comparative evaluation of SMT, NMT and tailored-NMT outputs. In *Proceedings of the 22nd Annual Conference of the European Association for Machine Translation*, pages 441–450, Lisboa, Portugal. European Association for Machine Translation.
3. Lanqing Xue, Kaitao Song, Duocai Wu, Xu Tan, Nevin L. Zhang, Tao Qin, Wei-Qiang Zhang, and Tie-Yan Liu. 2021. DeepRapper: Neural Rap Generation with Rhyme and Rhythm Modeling. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 69–81, Online. Association for Computational Linguistics.
4. Marjan Ghazvininejad, Xing Shi, Yejin Choi, and Kevin Knight. 2016. Generating Topical Poetry. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1183–1191, Austin, Texas. Association for Computational Linguistics.
5. Nikola I. Nikolov, Eric Malmi, Curtis Northcutt, and Loreto Parisi. 2020. Rapformer: Conditional Rap Lyrics Generation with Denoising Autoencoders. In *Proceedings of the 13th International Conference on Natural Language Generation*, pages 360–373, Dublin, Ireland. Association for Computational Linguistics.
6. Peter Potash, Alexey Romanov, and Anna Rumshisky. 2015. GhostWriter: Using an LSTM for Automatic Rap Lyric Generation. In *Proceedings of the 2015 Conference on*