Recurrent Rhapsody: A Prompt-Driven Song Generation Pipeline

Harshita Chadha
harshitachadha@gwu.edu

Abstract

The project titled "Recurrent Rhapsody: A Prompt-Driven Song Generation Pipeline" is a Recurrent Neural Network (RNN) based three-stage architecture for music generation. The project is aimed to produce an enhanced intelligence system by combining several machine learning algorithms such as Long Short Term Memory (LSTM), Gated Recurrent Unit (GRU), and sentence-BERT embeddings. At each of the three stages, models performing specific tasks are trained on separate but interlinked datasets and the results produced are compiled and presented as a coherent desired output. The experiments carried out during the course of project development culminate in a sequential pipeline that, given a textual prompt, generates a complimentary song-lyrics combination.

1. Introduction & Background

Recurrent Rhapsody is a three-component pipeline made up of three distinct neural networks that when combined result in the production of a novel sequentially dependent application. In the first part, an LSTM model generates lyrics of a song, given a prompt. This song is then fed into the second stage, sentence-BERT embedding and cosine similarity-based algorithm, where a similarity mapping is done to obtain the song with the most similar lyrics to the one generated. The MIDI track for this resultant song is then fed into a neural network model as the priming sequence to generate a new backing track. The lyrics along with this track are then output as the final results of the pipeline ready to be combined.

In the past, the studies conducted on the topic of lyrics generation were restricted to using Recurrent Neural Networks (RNN) or Gated Recurrent Units (GRU) for particular genres. Potash et al [1] made an effort to compose lyrics for a specific musician. But because their algorithm was trained on a particular singer, it was limited in its performance in producing lyrics for a genre. Watanabe et al [2] investigated the development of a model that generates complete lyrics for a given input music. To achieve melody condition lyric creation and generate lyrics by matching the words to the relevant tune, with impressive results, they use a well-studied RNN-based language model. Gill et al [3] focused on using an input sample lyric and employing Long Short Term Memory (LSTM) network to generate lyrics for a particular genre.

Furthermore, numerous attempts have been made to artificially generate music sequences using different machine-learning techniques. The most common approach to address this problem is to use recurrent neural networks. This is because these algorithms work well with time series data and music generation is an inherently temporal problem. Some of the most noticeable examples of music generation using RNN include "deepjazz" [4], a two-layer LSTM model that learns from input MIDI files, Magenta's MusicVAE [5], Wu et al's [6] hierarchical recurrent network composed of three LSTM sub-networks, etc. RNNs were primarily used in early techniques to create music for a single instrument. Other deep learning methods such as convolutional neural networks (CNNs) and generative adversarial networks (GANs) have also been used to synthesize music. For example, DeepMind's WaveNet is a deep generative model for audio synthesis that creates realistic, high-quality audio waveforms using a dilated causal convolutional neural network.

Yang et al [7] developed MidiNet which employs Deep Convolutional Generative Adversarial Networks (DCGANs) to produce multi-instrument music sequences. Dong et al [8] made MuseGAN, which used several generators to produce synthetic multi-instrument music that respects interdependencies between instruments. Lifelike synthetic music has also been produced using deep generative models like GANs and Variational Autoencoders (VAEs).

2. The Recurrent Rhapsody Pipeline

As indicated in the preceding section, the project is made up of three sequentially connected distinct models and the output of each of these serially feeds the input of the next model in the pipeline. A diagrammatic illustration of the model pipeline is shown in Figure 1 below. The first model in the pipeline which is referred to as component 1 in this report, is a bidirectional LSTM [9] model that given a text prompt of a few characters and the number signifying the length of string to be output can generate song lyrics based on the input. The second model which is referred to as component 2 in this report is a pre-trained S-BERT [10] model from huggingface [11]. This model produces word embeddings that help identify a priming sequence. The output of this component of the pipeline is the aforementioned priming sequence which is simply a MIDI file [12] that is used as input for the third component.

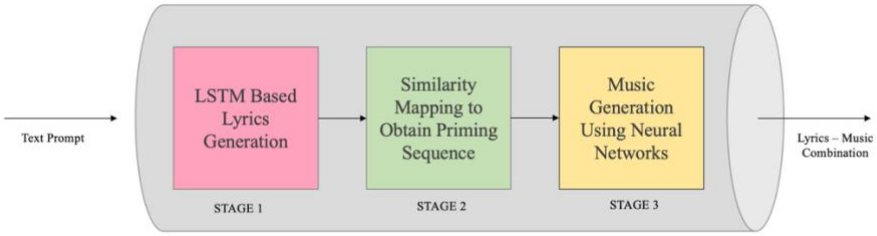

Figure 1: Illustration of the Recurrent Rhapsody music generation pipeline

The third and last component of the pipeline is a Recurrent Neural Network model that given the priming sequence, can generate the rest of the notes for an instrumental piano backing track. The subsections that follow, explore each of these models and their characteristics in detail.

2.1 Component 1: Bidirectional LSTM for Generation of Lyrics

The first component of the pipeline as briefly explained in section 4 above is a bidirectional LSTM model with four layers. The dataset used to train this model is sourced from Kaggle [13]. It is approximately 450 MB in size and is composed of two files – artists-data.csv and lyrics-data.csv. For our model, we use only the lyrics-data.csv file that contains lyrics of songs from eight different languages, including English, scraped from a Brazilian song portal called Vagalume [14]. A snippet of the dataset is shown in Figure 2 below –

	ALink	SName	SLink	Lyric	language
0	/ivete-sangalo/	Arerê	/ivete-sangalo/arere.html	Tudo o que eu quero nessa vida,\nToda vida, é\...	pt
1	/ivete-sangalo/	Se Eu Não Te Amasse Tanto Assim	/ivete-sangalo/se-eu-nao-te-amasse-tanto-assim...	Meu coração\nSem direção\nVoando só por voar\n...	pt
2	/ivete-sangalo/	Céu da Boca	/ivete-sangalo/chupa-toda.html	É de babaixá!\nÉ de balacubaca!\nÉ de babaixá!...	pt
3	/ivete-sangalo/	Quando A Chuva Passar	/ivete-sangalo/quando-a-chuva-passar.html	Quando a chuva passar\n\nPra quê falar\nSe voc...	pt
4	/ivete-sangalo/	Sorte Grande	/ivete-sangalo/sorte-grande.html	A minha sorte grande foi você cair do céu\nMin...	pt
5	/ivete-sangalo/	A Lua Q Eu T Dei	/ivete-sangalo/a-lua-q-eu-t-dei.html	Posso te falar dos sonhos, das flores...\nde c...	pt
6	/ivete-sangalo/	Mulheres Não Têm Que Chorar (com Emicida)	/ivete-sangalo/mulheres-nao-tem-que-chorar-com...	Hey, girl\nLevanta da cama\nO que foi que te a...	pt
7	/ivete-sangalo/	Eva / Alô Paixão / Beleza Rara	/ivete-sangalo/eva-alo-paixao-beleza-rara.html	"EVA"\n(Giancarlo Bigazzi/Umberto Tozzi)\n\nMe...	pt
8	/ivete-sangalo/	Flor do Reggae	/ivete-sangalo/flor-do-reggae.html	Um brilho de amor chegou na ilha inteira\nE a ...	pt
9	/ivete-sangalo/	Carro Velho	/ivete-sangalo/carro-velho.html	Cheiro de pneu queimado, carburador furado, co...	pt

Figure 2: Illustration of the lyrics dataset used to train Component 1 LSTM model

As part of pre-processing the dataset, all non-English songs were removed from consideration. Further, to scale down the extent of computing resources and training time, a subset of the original dataset was obtained that contained songs from the artists – Twenty One Pilots, Queen, AC/DC, and ABBA. The number of distinct song lyrics used was 641. A word frequency distribution of the final cleaned text corpus used for training is shown in Figure 3.

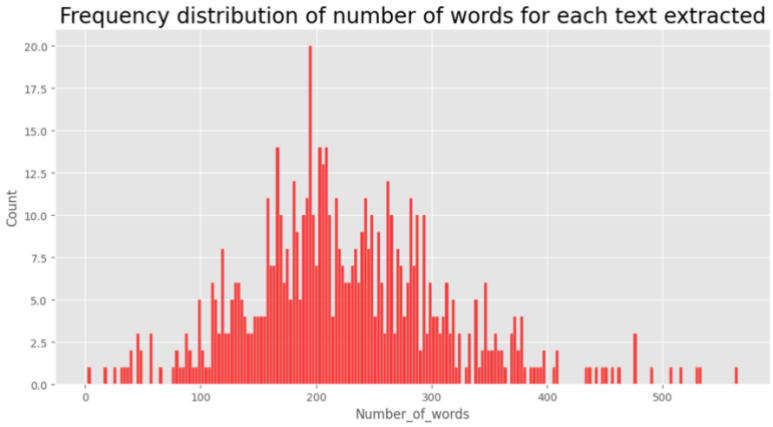

Figure 3: A plot of the frequency distribution of the number of words for each lyrics file extracted

Further, the lyrics are then tokenized and converted into n-gram sequences using a sliding window approach. This is because n-gram sequences are the natural choice of input in language modeling tasks such as the lyrics generation problem because they

preserve the context provided by the previous n-1 terms therein. To convert the n-gram sequences into a series of inputs X for the model, all but the last element of the n-gram sequences is chosen. This last element is stored separately as a label and the prediction task is posed as a prediction of this label i.e., the last element of the n-gram sequence. A diagrammatic representation of the four layers of the bidirectional LSTM model is shown in Figure 4.

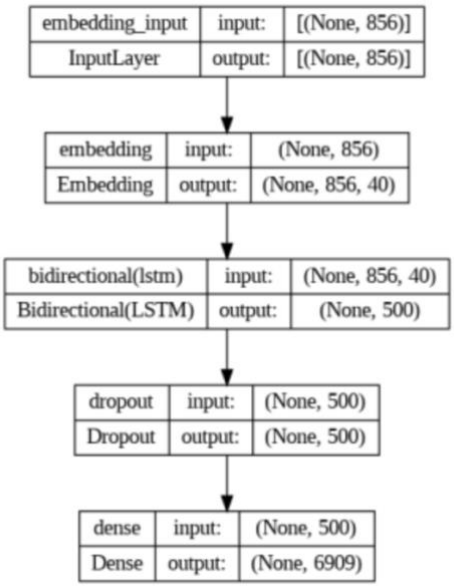

Figure 4: LSTM model of component 1 trained to predict the next word in the sequence of n-grams.

The first layer is the embedding layer that maps the input n-gram sequences into dense vectors of size 40. These dense vectors are then fed into the bidirectional LSTM layer that is made up of 250 units capable of capturing long-term contextual dependencies in the input n-gram sequences. The third layer is the dropout layer where the dropout rate has been specified as 0.1. This layer helps prevent overfitting and maintain generalization in the model by randomly dropping some of the LSTM units during the course of model training. The last layer of the model is the dense layer which has a SoftMax activation function that outputs the probabilities of each word in the vocabulary to be the next word in the sequence. A summary of the model is shown in Figure 5.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 856, 40)           276360

 bidirectional_1 (Bidirectio  (None, 500)              582000
 nal)

 dropout_1 (Dropout)         (None, 500)               0

 dense_1 (Dense)             (None, 6909)              3461409

=================================================================
Total params: 4,319,769
Trainable params: 4,319,769
Non-trainable params: 0
_________________________________________________________________
```

Figure 5: Summary of component 1 LSTM model

This model was then compiled using the categorical cross-entropy loss function and Adam optimizer. The evaluation metric used here was the accuracy value. The training curve for the model is shown in Figure 6 below. The training process was initialized for 200 epochs but was terminated after 40 epochs as the calculated loss stagnated at this point and did not show any improvement for 3 consecutive epochs. The batch size used is the default value of 32 and the model achieved a training accuracy of 74.36%.

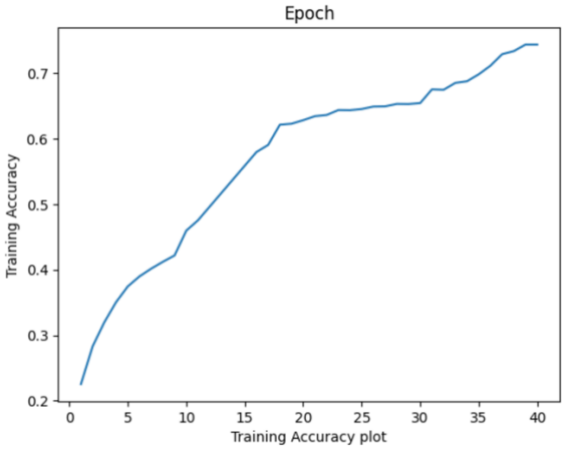

Figure 6: Training Accuracy plot for component 1 LSTM model

2.2 Component 2: Priming Sequence Identification using S-BERT and Cosine Similarity

The second component of the pipeline is responsible for the identification of the priming sequence required for backing track generation by the third component's Recurrent Neural Network (RNN). This operation is performed so that music complementary to the lyrics generated by component 1 is produced by the pipeline. In order to achieve this, the first step was to identify those MIDI files used in the third stage of the pipeline for which metadata was available. In order to achieve this, metadata files for the lakh piano dataset were perused and a file cross-linking these MIDI files to the Million Song Dataset was identified [15]. The Million Song Dataset is a large repository that contains information regarding various songs such as track id, artist name, etc. Using the cross-referenced file and the million songs repository, the titles, artist names, and MIDI ID for the tracks was localized and stored as a separate data frame. The next step was to find the lyrics for each of the entries in this newly created dataset. In order to do this, the Musixmatch API [16] was used. The resultant dataset is shown in Figure 7.

	track_id	title	artist_name	Lyrics
0	TRMMMQN128F4238509	Raspberry Beret (LP Version)	Prince & The Revolution	One, two\nOne, two, three, uh\n\nYeah\n\nI was...
1	TRMMWTG128F4283F07	You Needed Me	Roger Whittaker	I cried a tear: you wiped it dry.\nI was confu...
2	TRMMZRE128F42B799E	Sauver L'Amour	Daniel Balavoine	Partir effacer sur le Gange\nLa douleur\nPouvo...
3	TRMMZOT128F149E9EE	Prayee	The Chantels	NaN
4	TRMMIZZ128F4289298	It Keep Rainin' (Tears From My Eyes) (Radio Edit)	Bitty McLean	Pretty fallin' tears shining (Yes)\n(What's th...

Figure 7: Dataset referenced to create component 2 embeddings

It was observed that there were songs for which no lyrics were available which is plausible since many MIDI tracks were instrumental in nature. Further, some non-English tracks were also present in the dataset. Firstly, all the songs for which lyrics were not found were naturally eliminated from the dataset. Further, the langdetect [17] library in Python was used to remove all non-English songs from the dataset. In addition, we used the sentence transformer library which is built on top of Huggingface's transformers library to load a pre-trained BERT-based model namely the 'bert-base-nli-mean-tokens', i.e., the Sentence-BERT, to generate embeddings for each of the lyric entries in the dataset. These embeddings were then saved as a pickle file. With these embeddings at hand, given a text, the same pre-trained model can be used to generate embeddings for this text. Following this, the cosine similarity between each of the embeddings in our saved file and the new embedding is calculated, and the saved lyrics embedding for which this score is the highest is identified as the source of the priming sequence to be fed into the third stage of the pipeline – the backing track generation model.

2.3 Component 3: Recurrent Neural Network for Music Generation

Given the generated lyrics and a complementary priming sequence in the available MIDI dataset, the next step is to create an RNN model that is capable of taking in this priming sequence and generating a completely new backing track. This was achieved in the third stage of the pipeline. The dataset used to train the model is the lakh pianoroll dataset's LPD-5 cleansed version that contains 21,425 five-track pianorolls. The dataset itself is stored in a fairly complicated directory structure as compressed npz files and several functions had to be written to extract pianorolls from the dataset given a MIDI file's ID. In order to scale down the training task, only a subset of the dataset was used which included MIDI files associated with the bands - 'Abba', 'Queen', 'Green Day', 'Muse', 'The Cranberries', 'Radiohead', and 'My Chemical Romance'. The size of this subset was 177 five-track MIDI files.

Further, as stated earlier, each of these MIDI tracks are stored as npz files in a cascading directory structure so functions had to be written to extract these files and convert them to MIDI tracks. Each of the MIDI files is composed of five instruments out of which we only consider the piano for the present problem. Thus, for each of these files, the piano track was extracted and converted into a list of notes represented as integers which is the input to our model. The model used in this component is not a vanilla RNN but rather an advanced version of it called GRU or the Gated Recurrent Unit [18]. The model is made up of four layers in total and the summary is illustrated in Figure 8 below.

```
print(model_comp3)

GenerationRNN(
  (embedding): Embedding(320, 96)
  (gru): GRU(96, 96, num_layers=2)
  (decoder): Linear(in_features=192, out_features=320, bias=True)
)
```

Figure 8: Summary of the gated recurrent unit – GRU model for next note prediction

The first is the embedding layer which takes in the integer list of notes and converts it into dense vectors. These are then fed into the two layers of the gated recurrent unit and the output of the gated recurrent unit is then decoded to produce a probability distribution of over 320 possible notes.

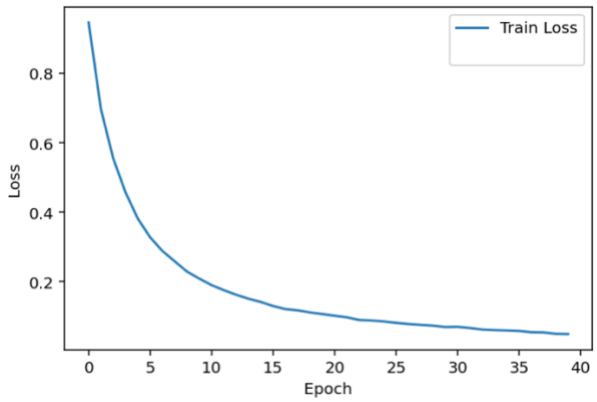

Figure 8: Training loss curve for component 3 GRU model

The loss function used during training is the cross entropy loss and the SoftMax activation function is used during evaluation to pick the note with the highest probability from amongst the produced distribution. A plot of the observed loss over the training epochs is shown above. The model was trained for 40 epochs and the training terminated when the observable loss value was about 0.05.

3. Experimental Setup

The entire project was created using Python 3.10.11 and its various libraries. The first component, an LSTM model for text sequence, i.e. lyrics prediction, was built using TensorFlow.keras. The second component used to obtain a complementary priming sequence was built using a pre-trained sentence-BERT model from hugging face interfaced using the sentence tokenizer library. The third component, a Gated Recurrent Unit-based model, was built using PyTorch. The first and second components of the model were trained in the Google Collaboratory environment using the NVIDIA Tesla T4 GPU. The third component was trained using Jupyter on the local machine using a 1.1 GHz Quad-Core Intel Core i5 processor and no dedicated GPU.

4. Integration Results and Conclusions

Once each of the three models was trained and prepared separately, the task was to combine and pipeline these components together sequentially so that the desired output could be produced. The pseudocode of a function that was written to achieve this is illustrated in Figure 9 below. This function references and loads each of the pre-trained models and makes sure the appropriate operations are being carried out so that the output of each t-1th component is compatible with the input dimensions of each tth component.

```
function generate_a_recurrent_rhapsody():
  prompt = read_input("Please enter some words as prompt for the model pipeline")

  # Component 1: Generate lyrics
  print("Component 1: Generate the lyrics initializing")
  lyrics = complete_this_song(prompt, 180)
  lyrics = add_newlines(lyrics)
  print("Lyrics have now been generated!")

  # Component 2: Lookup priming sequence
  print("Component 2: Lookup priming sequence initializing")
  matched_priming = get_priming_seq(lyrics, stored_embeddings)
  print("A priming sequence has now been identified!")

  # Component 3: Generate a backing track
  print("Component 3: Generate a backing track initializing")
  generated_seq = evaluate(model_comp3, matched_priming, predict_len = 100)
  int_to_note = {number:note for note, number in note_to_int.items()}
  generated_seq = [int_to_note[e] for e in generated_seq]
  print("Backing track generated successfully!")

  # Display the generated song
  print("Your unique lyric and backing track combination is - ")
  song_image = My_song(lyrics)
  display(song_image)

  # Create MIDI and audio files
  generated_stream = create_midi(generated_seq)
  stream_obj = generated_stream
  midi_file = midi.translate.streamToMidiFile(stream_obj)
  midi_file.open('output.mid', 'wb')
  midi_file.write()
  midi_file.close()
  FluidSynth().midi_to_audio('output.mid', 'output.wav')
  audio = Audio('output.wav')
  display(audio)

  print("Now it's your turn to take it away!")
```

Figure 9: Pseudocode for component integration

Using the above-illustrated function, a sample text input of 4 words was provided to the pipeline model as shown in Figure 10 below. This prompted the generation of lyrics and backing track combinations that were complementary to each other. The resources section lists the link to the Collab notebook where this pipeline can be loaded and tried out in real-time.

Figure 10: Output of Recurrent Rhapsody

5. Resources

The code for the entire project can be found at the GitHub repository – https://github.com/harshitaachadha/Recurrent-Rhapsody. The details of the datasets used during the training procedure of each of the models are tabulated below –

Table 1: Data resources used

DATASET	SOURCE	USAGE	LINK
Song Lyrics from 79 Musical Genres	Kaggle	Component 1	Linked here
Million Song dataset	Official Repository	Component 2	Linked here
The Lakh Pianoroll dataset	Official Repository	Component 3	Linked here
The lakh MIDI dataset	Official Repository	Component 2 & 3	Linked here

References

[1] Potash, P., Romanov, A., Rumshisky, A:. GhostWriter: Using an LSTM for Automatic Rap Lyric Generation. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. (2015)

[2] Watanabe, K., Matsubayashi, Y., Fukayama, S., Goto, M., Inui, K., Nakano, T.: A melody-conditioned lyrics language model. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), vol. 1, pp. 163–172 (2018)

[3] Gill, Harrison; Lee, Daniel; and Marwell, Nick (2020) "Deep Learning in Musical Lyric Generation: An LSTMBased Approach," The Yale Undergraduate Research Journal: Vol. 1 : Iss. 1 , Article 1. Available at: https://elischolar.library.yale.edu/yurj/vol1/iss1/1

[4] Kim, J. (n.d.). Deep learning driven jazz generation. deepjazz.io. https://deepjazz.io/

[5] MusicVAE. (n.d.). https://magenta.github.io/magenta-js/music/demos/music_vae.html

[6]Wu J, Hu C, Wang Y, Hu X, Zhu J. A Hierarchical Recurrent Neural Network for Symbolic Melody Generation. IEEE Trans Cybern. 2020 Jun;50(6):2749-2757. doi: 10.1109/TCYB.2019.2953194. Epub 2019 Dec 2. PMID: 31796422.

[7] Yang, Li-Chia & Chou, Szu-Yu & Yang, yi-hsuan. (2017). MidiNet: A Convolutional Generative Adversarial Network for Symbolic-domain Music Generation using 1D and 2D Conditions.

[8] Dong, Hao-Wen & Hsiao, Wen-Yi & Yang, Li-Chia & Yang, yi-hsuan. (2018). MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment. Proceedings of the AAAI Conference on Artificial Intelligence. 32. 10.1609/aaai.v32i1.11312.

[9] Brownlee, J. (2021). How to Develop a Bidirectional LSTM For Sequence Classification in Python with Keras. MachineLearningMastery.com. https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/

[10] Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1908.10084

[11] sentence-transformers (Sentence Transformers). (2022, November 7). https://huggingface.co/sentence-transformers

[12] Wikipedia contributors. (2023). MIDI. Wikipedia. https://en.wikipedia.org/wiki/MIDI

[13] Song lyrics from 79 musical genres. (2022, March 17). Kaggle. https://www.kaggle.com/datasets/neisse/scrapped-lyrics-from-6-genres

[14] VAGALUME - Letras de Músicas. (n.d.). Vagalume. https://www.vagalume.com.br/

[15] Bertin-Mahieux, T., Ellis, D. P. W., Whitman, B., & Lamere, P. (2011). THE MILLION SONG DATASET. In International Symposium/Conference on Music Information Retrieval (pp. 591–596). https://doi.org/10.7916/d8nz8j07

[16] musixmatch. (2011, August 11). PyPI. https://pypi.org/project/musixmatch/

[17] langdetect. (2021, May 7). PyPI. https://pypi.org/project/langdetect/

[18] Loye, G. (2023). Gated Recurrent Unit (GRU) With PyTorch. FloydHub Blog. https://blog.floydhub.com/gru-with-pytorch/

[19] Tham, I. (2022, January 5). Generating Music Using Deep Learning - Towards Data Science. Medium. https://towardsdatascience.com/generating-music-using-deep-learning-cb5843a9d55e