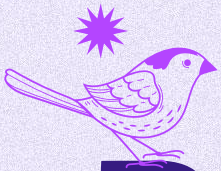
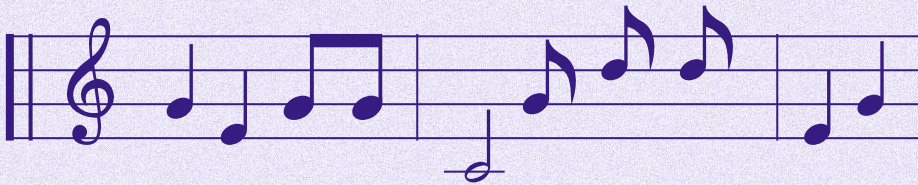


Pop Music Generation

By Chris Li, Josh Zhang, Dai Pham, Dean Kim, and Nate Zhang



Agenda

01

Exploratory Data Analysis

Insights from given dataset and summary statistics

02

Base Model Selection

Preliminary music generation and analysis

03

Final Model Selection

Model selection and fine-tuning decisions

04

Results

Resulting generated music and discussion

05

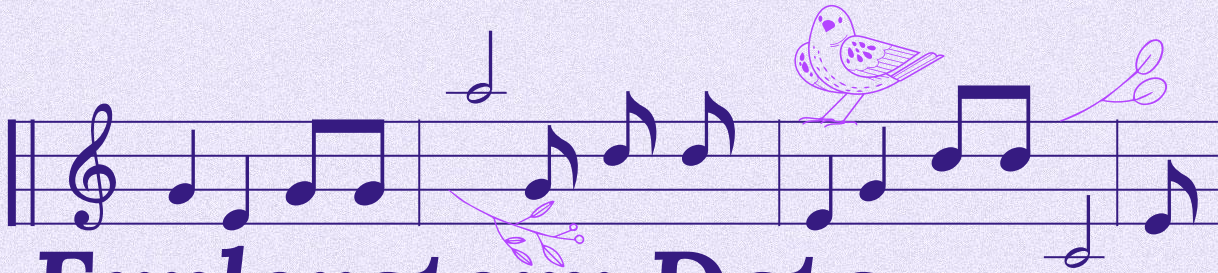
Future Improvements

Future work and fine-tuning to final model

Problem Statement

Given a set of musical features within pop music (e.g. tempo, key signature, note density), how can we create a machine learning model that generates new MIDI music in the pop genre?

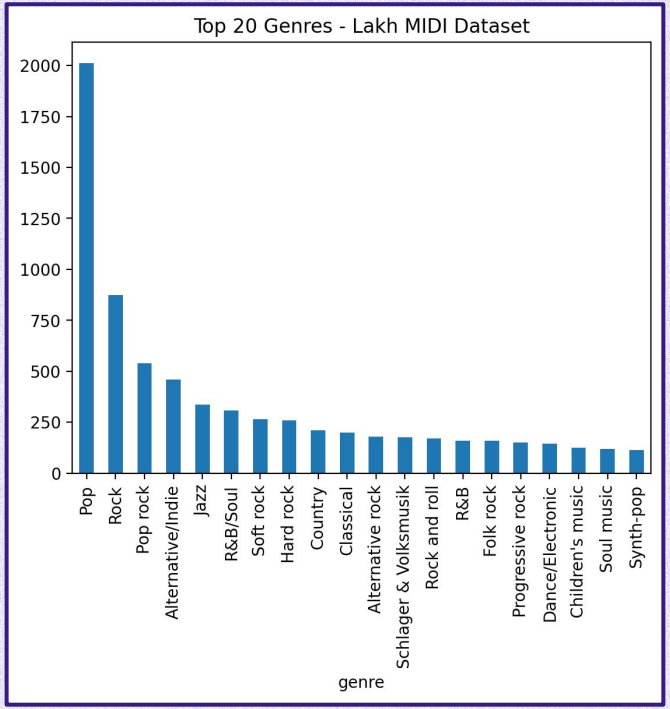
01



Exploratory Data Analysis

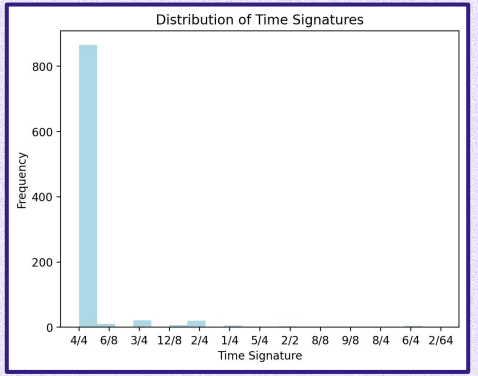
Our Dataset

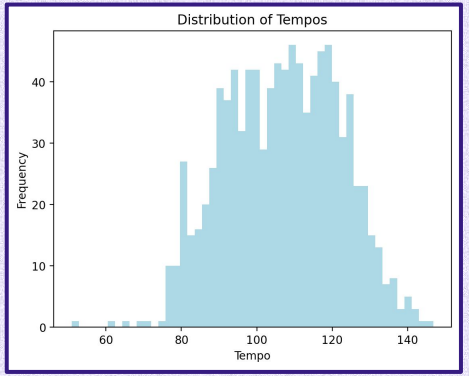
- Used the **Clean MIDI dataset**, a processed subset of the Lakh dataset that contains 17,256 MIDI files sorted by artist
  - Different adaptations of the same song would be marked by a dot and the respective number following the first instance
  - Used path to each file as data
- Utilized the **Genre dataset,** a genre-matched subset of the **Clean MIDI dataset** to categorize songs
  - Distribution of music genres was found to be overwhelmingly **skewed towards Pop music**

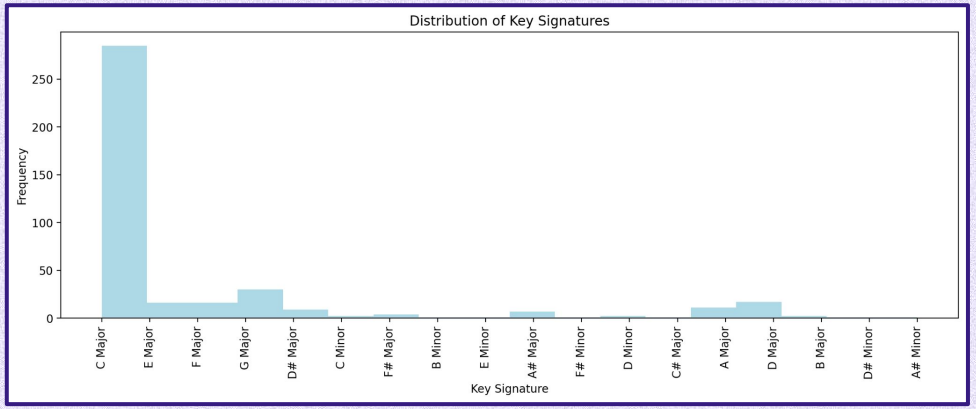


Exploratory Data Analysis

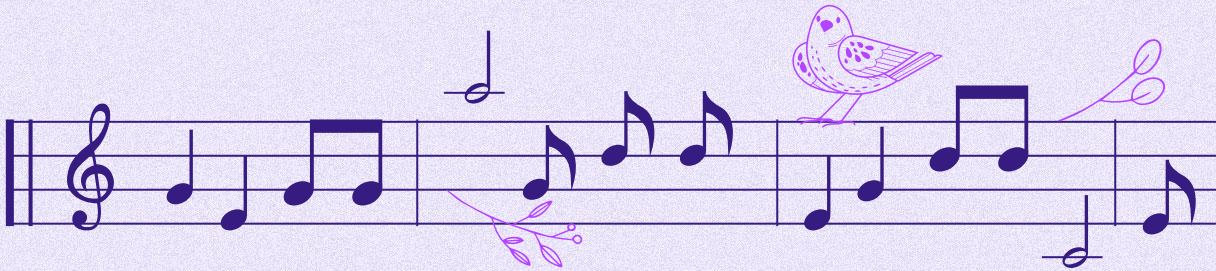
- Used **PrettyMIDI** and **Music21** libraries to extract metadata and determine the time signature, tempo, and key signature for each MIDI file for said subset
- Given the distributions, we decided to focus on songs in **4/4 time**, **90-120 BPM**, in the key of **C major** for our final model
- Shuffled data and selected a subset of 1000 songs (Computing Power)







02



Base Model Selection

Tokenization / Preprocessing

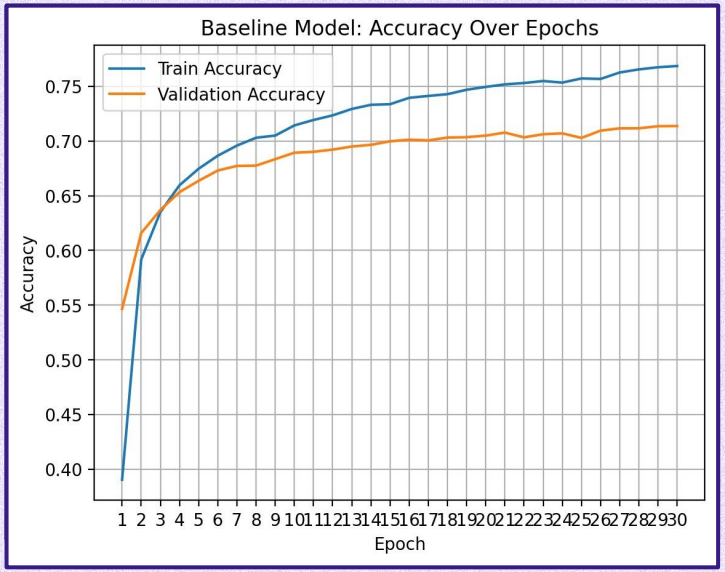
1. Transformed first 100 songs of our shuffled dataset into tokens using Music21
    - **Aggregated notes, chords, and musical rests with their respective durations** of each MIDI file through extraction by Music21
    - **Parallelized tokenizing** process to speed up runtime, and generated **flattened list of all tokens** across files
2. Combined tokenized segments into a single list of tokens (corpus), then trained the model on **fixed-length token sequences** (128 tokens)
    - Train and test data was then **converted to a tensorflow dataset** that is shuffled and batched in groups of 64 for the training

Model Architecture

- Built a simple **Recurrent Neural Network (RNN)** using Keras - ideal for sequence modeling
- **Embedding Layer:** transforms tokens to vector representations
- **Gated Recurrent Unit (GRU) layer:** processes the sequential data
    - **Reset Gate -** Factor how much information to retain
    - **Update Gate -** Skip Connections
- **Dense Layer:** softmax → probability distribution of 409 possible classes

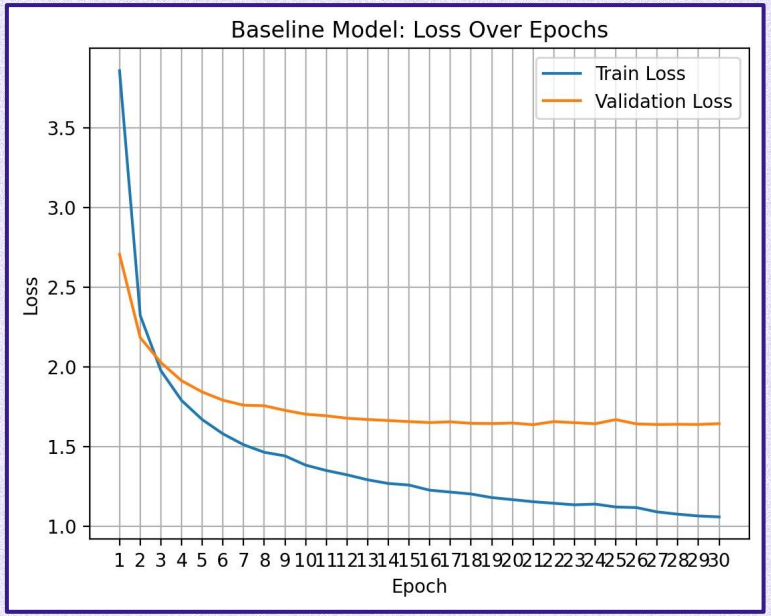
```
Model: "baseline_rnn"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 128, 64)           318592

 gru (GRU)                   (None, 64)                24960

 dense (Dense)               (None, 4978)              323570

=================================================================
Total params: 667122 (2.54 MB)
Trainable params: 667122 (2.54 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Training Details and Results

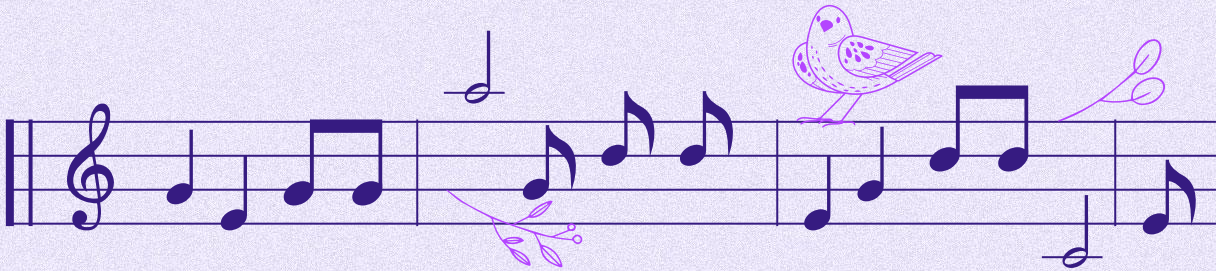


- **Peak validation accuracy of 71.38%,** with no signs of overfitting

- 30 epochs
- Batch size of 64, or 64 sequences per batch
- Standard Adam Learning rate of 0.005



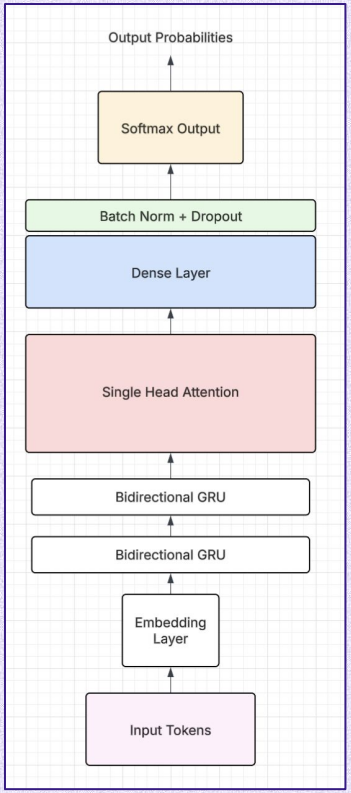
03



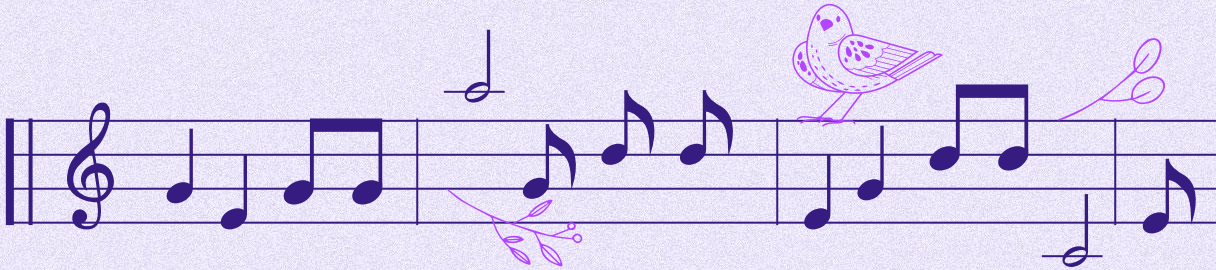
Final Model Selection

Final Model Architecture

- **Embedding Layer:** Maps tokens to dense vectors → reduces dimensionality; helps model learn patterns
- **Bidirectional Stacked GRUs:** Models temporal dependencies in both directions. Learns local (motifs) and mid-range (phrases) structure
- **Attention Mechanism:** addresses RNNs' weakness in remembering distant events → remembers moments beyond immediate past and future
- **Global Average Pooling + Dense Layers:** Summarizes sequence into a single decision point. Adds non-linearity and stabilizes training (batch norm, dropout)

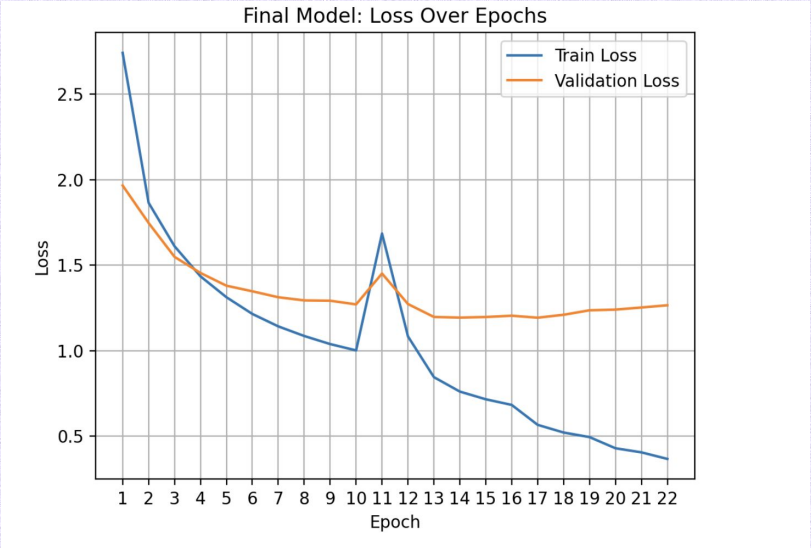


04

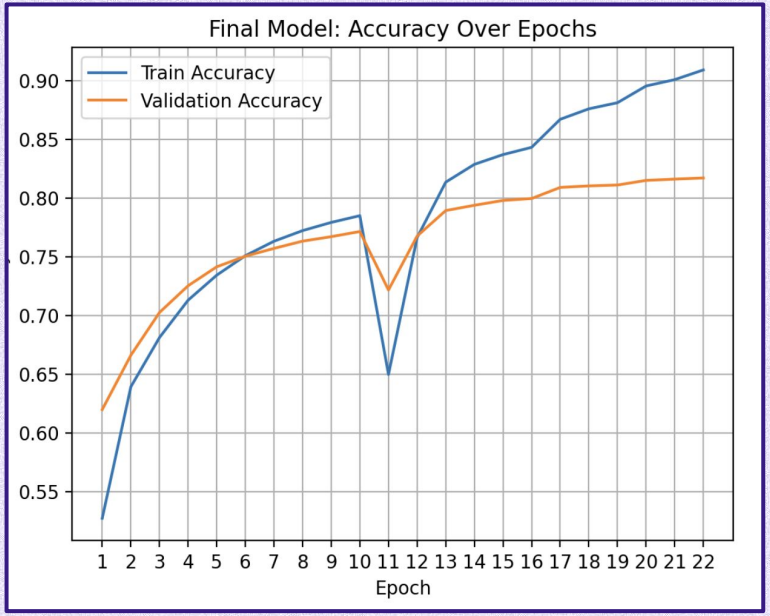


Results

Final Model: Training Details and Results



- 50 epochs, ended at **22 epochs**
- 110 sec/epoch, ~**40 min total**
- **Early stopping + learning rate reduction**

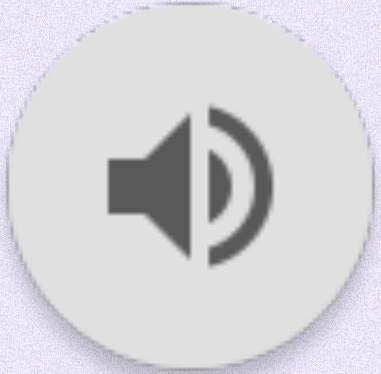


- **Peak validation accuracy of 81.71%,**

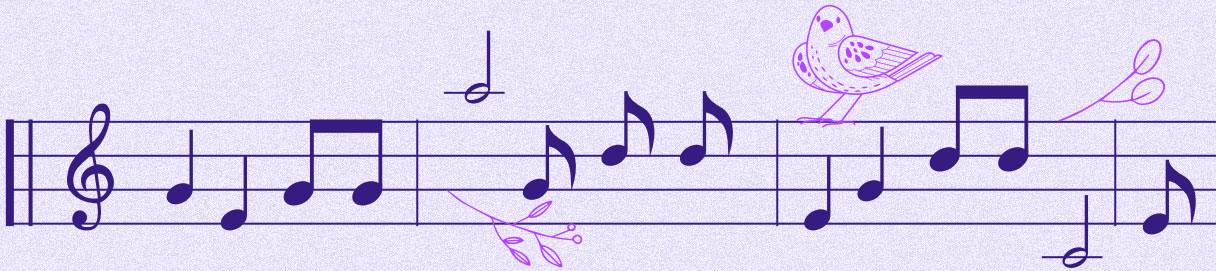
  with no signs of overfitting

Further Evaluation

- Outputted tokens → custom music generator using the music21 library
- Lack of quantitative metrics to evaluate the quality of our generated music → **evaluation by ear,** given that most member of our team are musically trained
- Output: **consistent musical melody,** while lacking some of the structure and repetition that the majority of pop music follows
    - Generated music is played on a single track using **default instrument for MIDI files**

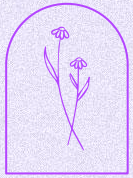


05



Future Improvements

Future Work/Improvements

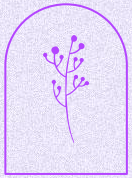


Outsource Tokenization

Open-source tokenizers including MIDItok could yield better embedding results in our final model

Improve MIDI Extraction

A large portion of our preprocessed dataset removed hundreds of MIDI files that were incompatible with the packages we utilized



Fine-tuning on existing models/transformers

With more GPU memory, we could fine-tune the tokenized MIDI data on pre-existing models like GPT-2 for music generation

Thank You!