Musical ideation through Generative AI

Project Length :350 hours
Difficulty :High
Coding Mentors :Walter Bender,Devin Ulibarri

Contents

Musical ideation through Generative AI

rkjane333@gmail.com

March 2024

1 Personal Information

- Name: Ritika Kumari
- University: JSS Noida
- Email: rkjane333@gmail.com
- GitHub: https://github.com/Ritika-K7
- linkedin: https://www.linkedin.com/in/ritika-kumari-b15b95289/
- Country of Residence: India
- Timezone: IST (GMT + 0530)
- Primary Language: English

2 Why this project?

2.1 Motivation

My fascination with music stems from its profound impact on my life. As an introvert, I find solace and expression in music, especially during challenging moments. It uplifts my mood, enhances focus, and serves as a constant source of comfort. This deep appreciation for music, coupled with my passion for computer science, fuels my desire to contribute to the field in the best way I can. This project would also help me to:

- **Exploring Cutting-Edge Tech:** The project delves into the exciting world of large language models, a rapidly evolving field with the potential to revolutionize various domains. Participating in this project will provide me with valuable exposure to this technology, allowing me to gain a deeper understanding of its capabilities and potential applications within the music domain.
- **Engaging with the Open-Source community:** This project presents a valuable opportunity to gain hands-on experience with the Open-Source Software (OSS) development process. I will acquire practical knowledge on constructing software applications from the ground up, leveraging the power of OSS tools and libraries. Furthermore, active participation in the OSS community exposes me to industry-standard software development practices.
- **Create impact through simplifying music creation:** This project has the potential to make music creation and learning accessible to everyone. By leveraging large language models, we can develop user-friendly tools that help individuals explore and create music, regardless of their background or traditional instruction. This can overcome initial challenges, sustain interest, and foster a more inclusive and engaging musical environment.

3 Technical Knowledge

I am in the second year of my bachelor's in computer science and engineering. I've been passionate about coding for a year now and consider myself a beginner in front-end development. I'm comfortable with **C, HTML, CSS, and JavaScript**. I have experience using **GitHub, Canva, and VS Code**, along with basic knowledge of **Docker and Postman**. Additionally, I have a basic understanding of data structures and algorithms.I have successfully created a clone of the "Amazon" website, a popular e-commerce platform for selling products, using HTML and CSS.The source code of the project is on my *Amazon − Clone*.All the data structures I have learned thus far, along with their related code, are available on *Data − Structure − Algorithm*. GitHub repository for reference and exploration.I am excited to face the challenges in GSoC with the primary goal of gaining valuable learning experiences.And I am confident that I will accomplish that goal.

4 My contribution to Sugarlab

Pull Request	Description	Status
#3613	Added JSDoc Style Documentation to utils/mathutils.js.	Merged
#3625	Added JSDoc style documentation to utils/munsell.js.	Merged
#3672	Added JSDoc style documentation to utils/musicutils.js.	Merged
#3684	Added JSDoc style documentation to utils/synthutils.js.	Merged
#3700	Added JSDoc style documentation to utils/utils.js.	Merged
#3713	Added JSDoc style documentation to widgets/meterwidget.js.	Merged
#3747	Added JSDoc style documentation to widgets/pitchdrummatrix.js.	Merged
#3758	Added JSDoc style documentation to widgets/rhythmruler.js.	Merged
#3771	Added JSDoc style documentation to widgets/sampler.js.	Merged
#3836	Added JSDoc style documentation to widgets/temperament.js.	Merged

Table 1: Contributions to Sugarlabs

5 Project

5.1 Project abstract

We would like to deploy generative AI in support of musical composition. The basic idea is to use AI to generate possible modifications of a phrase (or phrases) generated by the user. In other words, the user would start the composition and the AI would then present different possibilities for enriching the composition to which the user would react and further enhance.

Specifically, we would be working toward accomplishing the following:

- Tune open source LLM to create the ideal output for beginners, and for learning
- Create an API that translates LLM musical data to Music Blocks data, so that the learner has something to work from
- Create a in-app framework for evaluating the AI output, so that the user has several plausible choices and, if they so choose, grade the choices to be used for further improvement

5.2 My understanding of the project

- The user lands on the music block website. User starts interacting with the sugarlabs ui and generates few music blocks. These initial few notes will be passed to the backend. This is the initial input that will be enriched using fine-tuned llm.
- The backend is a node js application which will parse the incoming request from the frontend and extract the input given by the user. This input is will be used to prompt the llm for music generation.
- We can pass this input to the deployed fine tuned llm which will be running as a separate service. This is to ensure that the llm is already loaded when the user's request is received. Deploying it already will reduce the latency and enhance the user experience.
- The fine tuned LLM will generate an output. The output could contain some description about the sequence of notes that is generated. So we will have to extract only the notes and nothing else.
- The prompt to llm will dictate how to generate the melodic sequence. We should control this while generating the training data.
- These sequence will contain information about (pitch, duration, rest-duration).
- To generate more possible outcomes, we can prompt the llm to generate more or change some configuration setting for llm like temperature.
- Once multiple options are generated, we can augment these using other techniques like: Transpositions, retrograde and Inversions etc.
- Once the output is extracted we can return the response back to the node js application which would relay this output to the frontend. This will help the user to pick and choose the correct notes from the sugarlab ui and place it according to the suggestions obtained.
- For grading the output generated from the llm, we can ask user to rate the multiple options generated. This can further help in generating labelled data according to human preferences.

Process Flow Diagram

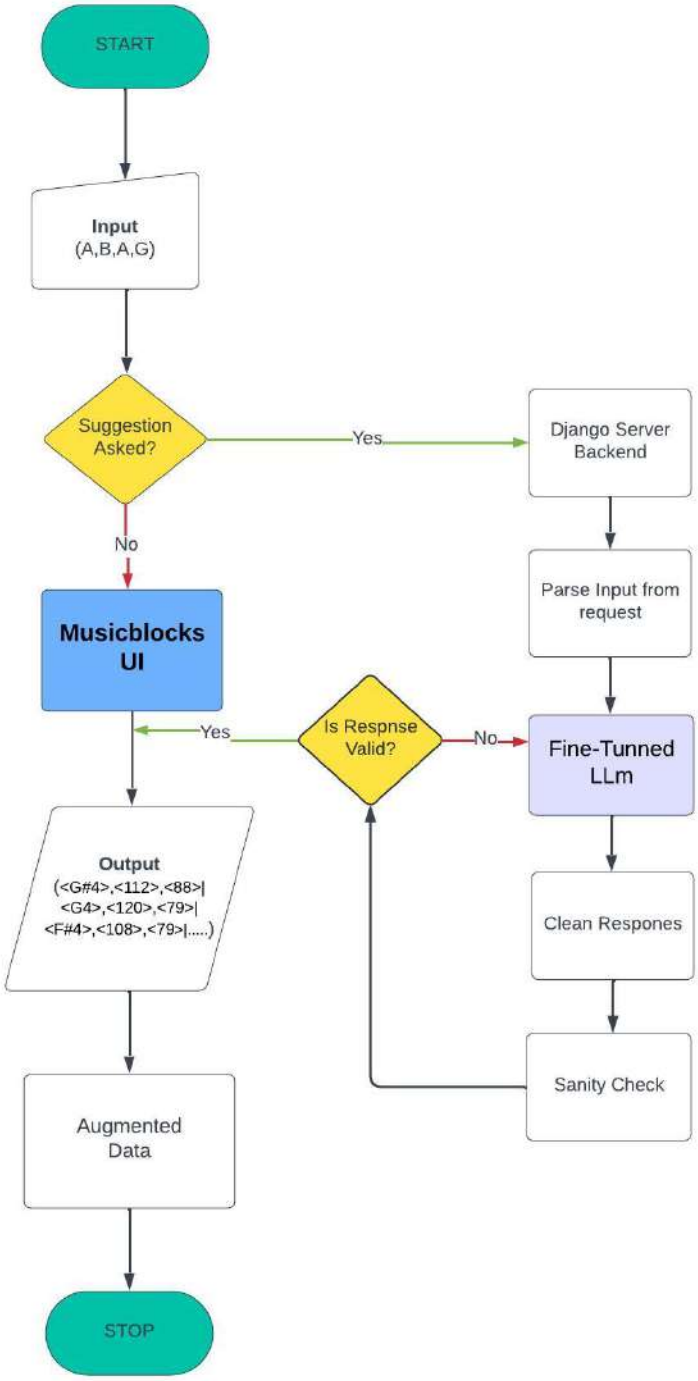
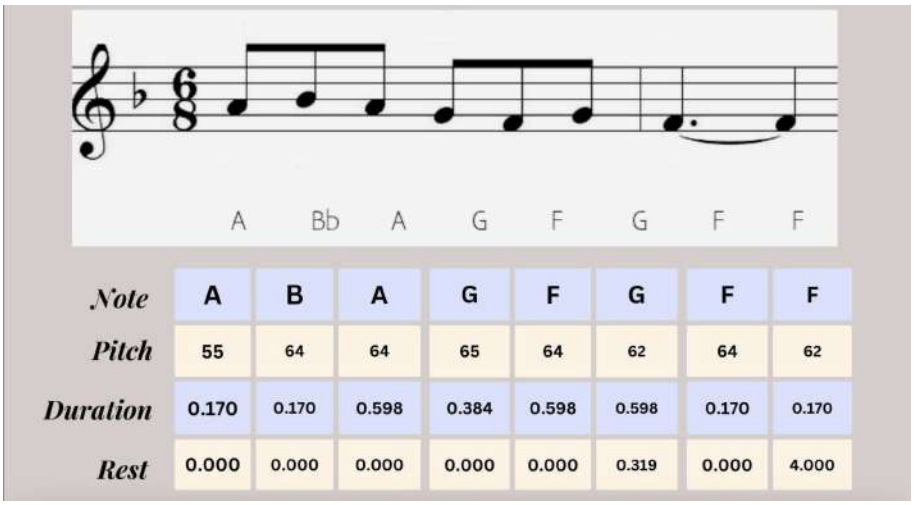


Figure 1: Flowchart: Music generation

6 Steps involved in LLM Fine-tuning

6.1 Dataset Preparation

- To prepare the dataset we will be using an open source collection of midi files. Sample source: https://www.kaggle.com/datasets/imsparsh/lakh-midi-clean. We can explore mulitple sources like google dataset, kaggle or huggingface hub.
- Midi files are a way to store music data and can be read using python libraries. The data in midi files look like:



Note	A	B	A	G	F	G	F	F
Pitch	55	64	64	65	64	62	64	62
Duration	0.170	0.170	0.598	0.384	0.598	0.598	0.170	0.170
Rest	0.000	0.000	0.000	0.000	0.000	0.319	0.000	4.000

Figure 2: MIDI File Representation

- We only extract the"melody" tracks from these MIDI files. Since melody in MIDI is represented as a sequence of musical notes over time and each note has a specific pitch, a start and end timestamp, we obtain a list of melody attribute triplet consisting of note pitch, note duration, rest duration.
- We break the melody into multiple lines and keep the data limited to 4-5 lines. This will help LLM understand what kind of notes come together to bring coherency in the music generation.
- The data will be extracted from the midi files in the following structure:

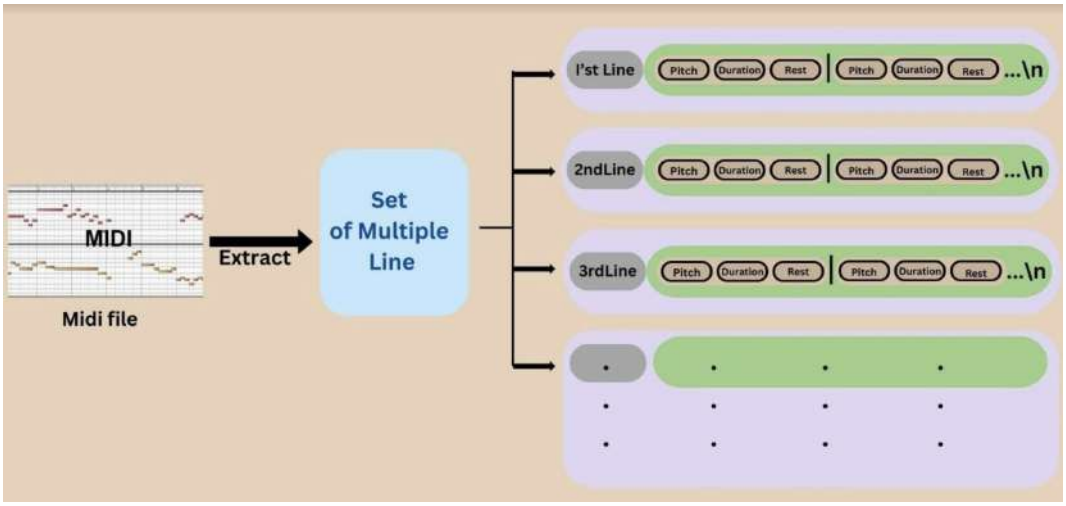


Figure 3: Dataset format for fine-tuning

- **Note pitch:** The pitch of notes is represented by their corresponding MIDI note numbers, ranging from 0 to 127, with the number 60 predefined as Middle C.
- **Note duration:** A note's duration is defined as the length of time in seconds that the note is played. This is computed from the start and end times of each note embedded within the MIDI files as follows: $note\_duration_k = note\_end_k - note\_start_k$,where k represents the note index number
- **Rest duration:** The rest duration represents the silent period that follows the playing of a note. It can be calculated by $rest\_duration_k = note\_start_k + 1 - note\_end_k$

- **Augmenting the dataset:** Data augmentation is a technique used to artificially expand the size of a training set by creating modified data from existing data. One way to do so is by transposition. When you transpose pitches by 4 semitones in each direction (up and down), you effectively create 9 variations of the original dataset. There are other ways too like retrograde and inversion. Below is how the training will look like after augmenting the dataset:

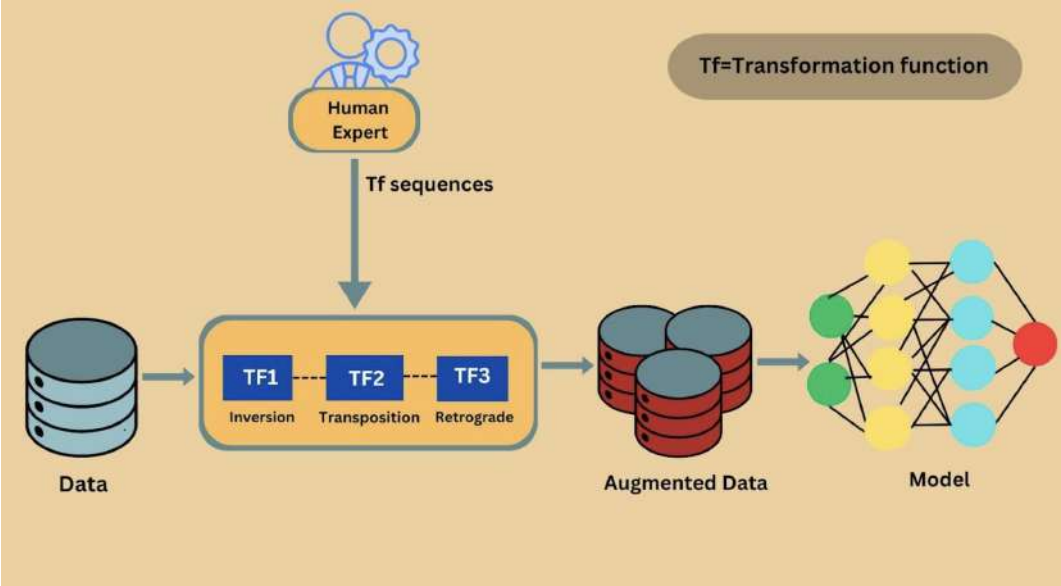


Figure 4: Flowchart: Fine-tuning with augmented dataset

6.2 Data extraction code from MIDI

Below is a piece of code that reads midi file and extracts the information. We can decide to add more attributes based on the discussion with the mentors.

```
from music21 import converter, note, pitch

def midi_to_dataset(midi_path):
    # Load the MIDI file
    midi_stream = converter.parse(midi_path)
    dataset_lines = []

    # Flatten the MIDI file to make it easier to iterate through notes and rests
    flat_elements = list(midi_stream.flat.notesAndRests)

    # Initialize a variable to keep track of the end time of the last note or rest
    last_end_time = 0
```

```
13
14      # Initialize a list to collect the formatted strings for the current line
15      current_line = []
16
17      for i, element in enumerate(flat_elements):
18          # Initialize rest_duration for the current element
19          rest_duration = 0
20          # Calculate the rest duration if it's not the first element
21          if i > 0:
22              rest_duration = int((element.offset - last_end_time) * 1000)  # Convert to
                ↪  milliseconds
23
24          if isinstance(element, note.Note):
25              # Extract the pitch and play duration of the note
26              pitch_name = element.pitch
27              play_duration = int(element.duration.quarterLength * 1000)  # Convert to
                ↪  milliseconds
28
29              # Append the formatted string to the current line
30              current_line.append(f'<{pitch_name}>, <{play_duration}>, <{rest_duration}>')
31
32              # Update the last_end_time
33              last_end_time = element.offset + element.duration.quarterLength
34
35          elif isinstance(element, note.Rest):
36              # Update the last_end_time for a rest
37              last_end_time = element.offset + element.duration.quarterLength
38              continue  # Move to the next element without adding rests to the dataset
39
40          # Assuming each line represents a measure or another logical division, you might
            ↪  need a condition to break lines.
41          # For simplicity, this example does not automatically split lines based on
            ↪  musical structure.
42
43      # Join the current_line into a string and add to dataset_lines
44      dataset_lines.append(' | '.join(current_line) + '\n')
45
46      # Join all lines to form the final dataset
47      dataset = ''.join(dataset_lines)
48      return dataset
49
50  # Use the function with a path to your MIDI file
51  midi_path = midi_file_path
52  dataset = midi_to_dataset(midi_path)
53
54  # Optionally, write the dataset to a file
55  # with open('midi_dataset.txt', 'w') as file:
56  #     file.write(dataset)
57
58
59  Output:
60  '''<D2>, <250>, <-1500> | <C#4>, <500>, <250> | <G3>, <250>, <-500> | <C#4>, <2250>,
    ↪  <-2750> |
61  <G3>, <250>, <-1583> | <A4>, <750>, <0> | <A2>, <750>, <-750> | <A4>, <2250>, <-1666>
```

```
| <A2>, <500>, <-2250> | <F#4>, <1500>, <1166> | <C4>, <1000>, <-1500> | <E-3>, <250>, <0> | <F#4>, <1000>, <750> ...'''
```

6.3 LLM selection

- We have to decide the LLM which we want to use for music ideation. For now I can think of two criterias on which we can decide the LLM:
  - It should be fast
  - It should be able to follow the instructions which are passed to it.
- I was able to use a very small LLM(TinyLlama-1.1B (Trained on 3 trillion tokens) which is not fine tuned for this project. This is not a full-fledged production code, but this is just an experimentation piece to understand how it will all fit together or the challenges I will face on using llm for the given case. The code and the output for the same looks like:

```python
import torch
from transformers import pipeline


pipe = pipeline("text-generation", model="TinyLlama/TinyLlama-1.1B-Chat-v1.0", torch_dtype=torch.bfloat16, device_map="auto")


# We use the tokenizer's chat template to format each message - see https://huggingface.co/docs/transformers/main/en/chat_templating
messages = [
    {
        "role": "system",
        "content": "You are a friendly musician who gives the melodic notes",
    },
    {"role": "user", "content": "what next after C-D-E-F  ?"},
]
prompt = pipe.tokenizer.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)
outputs = pipe(prompt, max_new_tokens=256, do_sample=True, temperature=0.7, top_k=50, top_p=0.95)
print(outputs[0]["generated_text"])
```

```
--------------------------Output--------------------------


<|system|>
You are a friendly musician who gives the melodic notes</s>
<|user|>
what next after C-D-E-F  ?</s>
<|assistant|>
After the C-D-E-F chord progression, you can try playing the B-E-A-D chord progression, which is a variation of the major scale. This progression has a similar structure as the major scale, but with an additional E note. Here's the C-D-E-A-D progression:
```

```
C - D - E - A - D

Try playing this progression on a guitar or piano to see how it feels in your
↪ fingers. You can also try playing the C-D-E-A-D progression in the key of C
↪ major, which is commonly used in classical music. Playing these progression will
↪ help you understand the different chords and how they can be used in different
↪ musical contexts.
```

- The above piece of code is a demonstration that even a small model understands the instructions passed to it and responds in the required manner. Fine tuning the model will the proposed dataset format will tune it in the way we need the output to be. Also, this shows that we will have to write additional code to extract only the relevant information from the response of llm.

6.4 Training/Fine Tuning

- We can fine-tune the llm once we have the dataset prepared using the midi files.
- For fine-tuning we will be using either colab/kaggle kernels which gives free GPU instances.
- I tried using a bigger models, with 7 billion parameters too. But the model failed to load in colab notebook. We might have to use libraries which will help load the model. These libraries will reduce the floating point accuracy of weights and post that we can continue to use these models.
- To improve the performance while fine-tuning we can use other optimization libraries to enable faster and efficient training for these large models.

6.5 Evaluation

- We have to test the LLM that we fine tuned to make sure it gives the output its expected to give. For example we don't want the LLM to generate any other text other than musical notes.
- Also we need to be careful to have some checks that should not generate any harmful/toxic content.
- Once we have sanitized and extract the relevant output from llm's response, we can ask user to grade the outputs. This will help us to capture user's preference.

6.6 Deployment

- I have been working on a node js backend which will call the LLM to ask for an output and return the output back to the frontend. This is to understand how to integrate both a node js backend and a model that is running on a different server.
- For Now I am loading the model from the same nodejs script. Ideally, we will be hosting this as a separate service and hitting that api to recevie resposne. Please take a look at the code below: breaklines

```
const express = require('express');
const { PythonShell } = require('python-shell');
const app = express();
const port = 3000;

function generateText(prompt, res) {
    const options = {
        mode: 'text',
        pythonOptions: ['-u'], // unbuffered output
```

```
        scriptPath: __dirname,
        args: [prompt]
    };

    PythonShell.run('llmscript.py', options, (err, result) => {
        if (err) {
            console.error('Error occurred while executing Python script:', err);
            res.status(500).send('Internal Server Error');
        } else {
            res.send(result[0]);
        }
    });
}

app.get('/', (req, res) => {
    res.send('Welcome to the homepage');
});

app.get('/generate-text', (req, res) => {
    const prompt = req.query.prompt || "G#f#";
    generateText(prompt, res);
});

app.listen(port, () => {
    console.log(`Server is listening at http://localhost:${port}`);
});
```

7 Converting LLM data to Musical blocks

- How to convert the data from llm to musical blocks? Does this mean that based on the llm's response we will add the music block on the ui automatically? How will this work for multiple suggestions?

8 Grade the AI Output

- We can incorporate a feedback mechanism into our in-app framework to evaluate the quality of AI-generated content.
  - **Grading System:** Implement a grading system that enables users to assign a score or provide feedback on the quality of each option. This could be a numerical rating system (e.g., 1 to 5 stars) or a qualitative feedback mechanism (e.g., "Excellent," "Good," "Fair," "Poor").
  - **Monitoring and Analysis:** Monitor and analyze AI model performance based on user feedback. Identify patterns and trends to improve.

9 Timeline

9.1 PreGSOC

- I plan to deploy a lightweight LL model, enabling access to it through an API endpoint. Subsequently, I'll integrate this API into my node backend to facilitate communication and receive responses from the model. The model should be light enough to run on my own computer initially. My primary focus at this stage is on the integration process rather than the specific output of the model. Additionally, I aim to implement mechanisms to avoid reloading the model with every request for efficiency.

- Observe and understand the varying behaviors of different models in terms of:
  - The time taken to generate output.
  - Whether the model adheres to the instructions provided.
- Enhance my JavaScript skills, deepen my understanding of LLM and familiarize with the music blocks interface.

9.2 Week 1

Since I am already contributing to the sugar labs project, I understand the review process and the procedure that needs to be followed to submit the code. This week, I would like to go over the code of musical blocks to understand how the simpler pieces I have built on my local system will fit in the current code base.

9.3 Week 2- 3

In these two weeks I have to work with my musical mentors to understand which of the dataset I should be collecting. Once these dataset are collected I will focus on how to read these dataset in any musical software to listen to them. Setting up this flow will help me to learn how this will sound once the output is generated from LLM.

Once the dataset has been collected, I will try to modify the dataset that the LLM will train on. Once the dataset has been converted into that form, then I will have to evaluate whether the conversion has been according to the same format or not.

9.4 Week 4-6

This week I will focus on trying to use the dataset that I created with the LLM and see how to write code for training. I will need help from my tech mentor to identify steps that will be critical in training the LLM. Once the LLM is trained on the full dataset, we need to create a service that will use this llm and record the time it takes to serve the request. Based on the latency requirement, we can work on optimizing the system.

9.5 Week 7- 8

In these weeks I will work with my musical and tech mentor to evaluate the output from the fine tuned LLM that we have built. This will be focussed on ensuring that we are able to extract the sequence of notes from LLM. How does the output from the LLM sound like when we add this data to a music software. Also In these two weeks we can focus on grading the outputs from the LLM as this is also a requirement of the project. We will have to understand how to compute those qualitative metrics to grade this output.

9.6 Week 9 - 10

In these weeks I can focus on deploying the fine tuned LLM and integrating that with the musical blocks code base. Once the output from the LLM is received then the output will have to be converted to musical blocks.

9.7 Week 11

This week will be for completing any pending work which might have spilled from earlier weeks.

9.8 Week 12

We can work on creating documentation so that other contributors can build on the work that I did.

10 Questions

- **How many hours do you plan to spend each week on your project?** During the GSoC period, I plan to dedicate 5-6 hours daily on weekdays and 9-10 hours on weekends. I aim to dedicate around 40-45 hours every week to my project.

- **How will you report progress between evaluations?**

  I will share weekly updates on the progress of the project through Matrix and mailing lists. These updates will include a summary of the tasks completed, any milestones achieved, challenges encountered, and plans for the upcoming week. Additionally, I will actively engage with mentors and the community to discuss any difficulties that arise during the development of the project, seeking advice and guidance to overcome them effectively.

- **Discuss your post GSoC plans. Will you continue contributing to Sugar Labs after GSoC ends?**

  After GSoC ends, I intend to continue contributing to Sugar Labs and remain engaged with the community. Continuing to work on the project that I have developed during GSoC, implementing additional features, addressing any remaining issues, and ensuring its smooth integration into the Sugar Labs ecosystem. Mentoring new contributors and GSoC students. Sharing what I've learned with others will not only help them get involved in Sugar Labs but also bring us closer together as a community.