

**Software Engineering Department  
ORT Braude College**

**24-2-D-20**

**Capstone Project Phase B**

**GENERATING MUSIC WITH SENTIMENT USING**

**TRANSFORMER-GANS**

**Ariel Dahan 315682054**

**Niv Malka 314730995**

**Supervisors:**

**Dr. Renata Avros**

**Prof. Zeev (Vladimir) Volkovich**

**Table of Contents:**

**1. Introduction ……………………………………………………………………. 4**

**1.1 Project Purpose ………………………………………………………. 4**

**1.2 Target Audience ……………………………………………………… 5**

**1.3 System Workflow ……………………………………………………. 5**

**1.4 Current Solutions …………………………………………………….. 6**

**1.4.1 MusicLM …………………………………………………… 6**

**1.4.2 MuseNet ……………………………………………………. 7**

**1.4.3 Additional Solutions ………………………………………... 7**

**1.4.4 Comparison of Existing Solutions to the Selected Model ….. 7**

**2. Background …………………………………………………………………….. 8**

**2.1 Evolution of Music Generation Technology …………………………. 8**

**2.2 The Role of Deep Learning and Neural Networks …………………… 8**

**2.3 Combining GANs and Transformers ………………………………… 9**

**3. Model Architecture ……………………………………………………………. 9**

**3.1 Transformer-GAN Architecture ……………………………………… 9**

**3.1.1 Transformer ………………………………………………… 9**

**3.1.2 Generative Adversarial Network …………………………… 9**

**3.2 Generator and Discriminator Models ………………………………… 10**

**3.2.1 Generator …………………………………………………… 10**

**3.2.2 Discriminator ……………………………………………….. 10**

**3.2.3 Interaction Between Generator and Discriminator …………. 10**

**4. Challenges and Solutions ………………………………………………………. 11**

**4.1 Performance and Computation Challenges …………………………... 11**

**4.1.1 Training Time and Resources ……………………………… 11**

**4.1.2 Resource Management ……………………………………... 11**

**4.2 Model and Data Size Challenges ……………………………………... 11**

**4.3 Engineering Challenges ………………………………………………. 12**

**4.3.1 Algorithm Selection and Optimization ……………………... 12**

**4.3.2 System Design and Integration ………………………………12**

**4.4 Technical Challenges ………………………………………………...... 12**

**4.4.1 Virtual Environment and Dependency Management ………... 12**

**4.4.2 Server Setup and API Connectivity ………………………..... 12**

**4.4.3 Implementation of the User Interface ……………………….. 13**

**5. Development Process …………………………………………………………… 13**

**5.1 Data Preprocessing …………………………………………………...... 13**

**5.2 Model Architecture …………………………………………………...... 13**

**5.3 Model Training ………………………………………………………… 14**

**5.4 Music Generation ………………………………………………………. 15**

**5.5 Web Interface ………………………………………………………….. 15**

**6. Visualizing the Music Generation Process ……………………………………… 16**

**6.1 Deployment Diagram ………………………………………………...... 16**

**6.2 Activity Diagram ………………………………………………………. 17**

**7. Product Requirements …………………………………………………………… 17**

**7.1 Functional Requirements ………………………………………………. 17**

**7.2 Non-Functional Requirements …………………………………………. 18**

**8. Test Planning ……………………………………………………………………. 19**

**9. Results …………………………………………………………………………… 19**

**9.1 Conclusions …………………………………………………………..... 19**

**9.2 Reflections and Insights ……………………………………………….. 20**

**10. User Guide …………………………………………………………………….... 20**

**10.1 Selecting a Mood ………………………………………………….….. 20**

**10.2 Song Generation Confirmation ………………………………………. 21**

**10.3 Music Playback ………………………………………………………. 22**

**10.4 Actions After Playback ………………………………………………. 22**

**11. Maintenance Guide …………………………………………………………... 23**

**11.1 Operating Environment and Installation ……………………………. 23**

**11.1.1 Python Package Dependencies ……………………………. 23**

**11.1.2 External Dependencies ……………………………………. 23**

**11.2 Installation Instructions …………………………………………...… 23**

**11.3 Enhancements and Updates …………………………………………. 23**

**12. References …………………………………………………………………….. 24**

**Abstract**

Deep learning has significantly impacted music generation, but generating music to express human emotions remains a complex challenge. This project proposes a Transformer-GAN-based model that generates music that aligns with specific emotional states, with Valence (positivity/negativity) and Arousal (intensity) as conditioning parameters. Our system processes MIDI music data by extracting key musical features such as tempo, key, time signature, and chord sequences, which enables it to capture structural and harmonic patterns. The Transformer component of our system captures long-term dependencies in musical sequences, and the Generative Adversarial Network (GAN) constrains the realism and expressiveness of the generated compositions. By integrating emotional conditioning, our model dynamically adapts to emotional variations, providing fine-grained control over musical mood. With applications spanning music production, interactive entertainment and therapeutic interventions to AI-assisted composition, the system enhances user engagement by generating adaptive, emotionally expressive music. Furthermore, our project contributes to the evolving field of computational creativity with new methods for emotion-driven music generation. Preliminary evaluations indicate that the model can successfully align generated music with human emotional perception, confirming the effectiveness of Transformer-GAN integration in producing structured, expressive, and emotionally resonant compositions.

1. **Introduction**

The development of a music generation system, which is guided by emotions, marks an exciting advancement in deep learning and creative technology. Music, as we know, has a unique power to convey a range of complex emotions, acting almost as a universal language that people everywhere connect to. Aligning with a music generation with specific feelings opens up a lot of new opportunities—not only for various industries, but also for personal uses. This project aims to push the boundaries of how we experience and interact with music in an emotionally resonant way.

*1.1 Project Purpose*

Our project uses a Transformer-GAN model to generate music that reflects specific emotional states based on Valence (positivity/negativity) and Arousal (intensity). The system processes MIDI files, extracting musical features such as tempo, key, time signature, and note sequences, which are then used to compose structured and expressive music. The Transformer model evaluates long-term dependencies in the music to ensure coherent progression, and the GAN model adds to the natural essence of the composition by improving coherence and realism. This allows the system to generate smooth emotional transitions, thus enabling the generated music to be versatile for various moods. The system is designed for applications in film, video games, therapy, and personalized music experiences, where music needs to match a specific emotional atmosphere. Evaluation metrics compare the generated compositions with emotional targets to ensure accuracy, contributing to the development of emotion-based music composition and analysis. The system is specifically designed to be versatile and emotionally adaptive and can be applied in areas such as film, video games, therapy, and personal music experience.

*1.2 Target Audience*

This project is designed for a wide range of users, including musicians, composers, researchers in machine learning and neural networks, and developers working on generative music systems. It is particularly relevant for professionals in the music industry who seek innovative methods for composing and producing music using deep learning techniques. The project provides tools that allow musicians to explore automated composition while maintaining emotional depth and artistic control. Additionally, it is intended for researchers in natural language processing (NLP) and deep neural network architectures, especially those interested in applying Transformer models and Generative Adversarial Networks (GANs) to creative domains. These users can experiment with model training, fine-tuning, and optimization to improve music generation quality. Beyond professionals, the project also serves hobbyists and enthusiasts who wish to experiment with music generation, regardless of their formal training. Educators and students in fields such as computational creativity, digital signal processing, and music technology may also find this project a valuable resource for academic research and practical experimentation. By leveraging Transformer-GAN architectures, this project aims to bridge the gap between artificial intelligence and artistic expression, making generative music more accessible and intuitive for a broad spectrum of users.

*תמונה שמכילה טקסט, צילום מסך, קבלה, קו

התיאור נוצר באופן אוטומטי1.3 System Workflow*

Figure 1: Transformer-GAN system workflow

User Input: A user selects the desired emotion (such as joy or sadness), and the system accepts the emotions as input.

Data Preprocessing: Emotions are processed to fit them into a format that can be fed into the model. This is done by normalizing and feature extraction of MIDI data.

Model Processing: Musical sequences are encoded by the Transformer with self-attention, and the GAN is responsible for generating music based on the emotional input.

Post-Processing: After the music is created, final processing is performed to improve the music, including smoothing the transitions and other adjustments.

Output Generation: the generated music is sent to the user interface for display or further processing.

*1.4 Current Solutions*

In recent years, several advanced models have made significant progress in the field of deep learning-driven music generation, incorporating emotional elements and pushing creative boundaries. Despite these achievements, current solutions still have some limitations that our model seeks to address.

*1.4.1 MusicLM*

MusicLM is a recent music generation that Google Research has developed to generate music based on descriptions. By following precise rules concerning mood and style of the piece, it produces the music using highly sophisticated mathematical equations. Main advantages: text and image input processing in order to generate music, determining music continuation based on the input style with references to the high quality of output while warning against its usage for professional purposes including music scoring for films and creating specialized playlists. Nevertheless, the application of MusicLM is most useful in those cases where it is important to achieve a given emotion, for example, in creating a library of movie scores or a library of highly personalized music works. Thus, it is prone to inability to model and follow the dynamics of and the course of emotion, as well as possible adaptation there of what may be an issue in interactive or real-time as opposed to user-initiated settings.

תמונה שמכילה צילום מסך, טקסט, עיצוב

התיאור נוצר באופן אוטומטי

Figure 2: MusicLM interface visualization

*1.4.2 MuseNet*

MuseNet is another model developed by OpenAI that has massive capabilities for creating music that spans different instruments and genres. The program does not directly incorporate emotions, however, its capacity to create linguistically rich and stylistically diverse texts helps it suggest emotions indirectly. They include polyphonic composition which allows more than one melody and instruments and harmony, style versatility which can create all style of music be it classical or modern like pop and jazz, long term structure needed to create extended musical works which is important when creating emotional flow. It is worth using MuseNet for generating music in which the affective content is signified by style and orchestration, as it is not directly manipulated. Nonetheless, the flexibility of MuseNet in achieving highly nuanced musical arrangements, it is not very clear how to regulate for affective valence. Its use of style and genre to indicate emotions may not always coincide with the needed ‘emotional content’ for an application, occasionally producing rather ornamental music which may nonetheless fail to convey the desired affective density.

תמונה שמכילה טקסט, צילום מסך, שעון, עיצוב

התיאור נוצר באופן אוטומטי

Figure 3: MusNet interface visualization

*1.4.3 Additional solutions*

Other applications, like AIVA and Jukedeck, have also contributed to music generation by incorporating some level of emotional expression. However, like MusicLM and MuseNet, they struggle to offer precise control over the emotional tone of the music. This challenge is especially noticeable when music needs to shift in response to changing emotions, where accurate emotional alignment is key.

*1.4.4 Comparison of existing solution to the selected model*

Our proposed Transformer-GAN model introduces a promising approach to emotion-conditioned music generation, focusing specifically on integrating emotional metrics like valence (positivity/negativity) and arousal (intensity) directly into the music generation process. This allows the model to reflect a range of emotional states more closely, addressing some limitations of models like MusicLM that rely primarily on user-provided text prompts. One area where our model aims to make a difference is in handling dynamic emotional transitions, which can be challenging for existing models such as MusicLM and MuseNet. By combining the sequence modeling ability of Transformers with the realism feedback from GANs, our model seeks to generate music that evolves in a way that feels more naturally aligned with changing emotional inputs. In contrast to approaches that rely on general prompts or stylistic cues, we use quantifiable metrics (valence and arousal) to give more direct emotional conditioning. This provides a reproducible basis for creating music that aims to capture specific emotional shifts, allowing users to guide the emotional flow of the compositions more systematically. Additionally, our model explores real-time adaptability to emotional changes, which could enable applications where music responds dynamically, such as in adaptive soundtracks for gaming or personalized playlists. While still in the development and testing phase, this model is a step toward more interactive music experiences that adapt to users' evolving emotional contexts. In summary, although models like MusicLM and MuseNet have laid valuable groundwork in deep learning-based music generation music generation, our Transformer-GAN model offers an exploratory step toward greater emotional flexibility, aiming to provide music that reflects a more nuanced and adaptive emotional range. This project serves as an academic investigation into these possibilities, contributing to ongoing research in the intersection of deep learning and creative technologies.

1. **Background**

Understanding how emotion-driven music generation works starts with a look at the main technologies and methods that make it possible.

*2.1 Evolution of music generation technology*

Music generation technology has come a long way, from simple, rule-based systems to complex deep learning models. Earlier methods, like rule-based algorithms or Markov chains, could create predictable and structured music pattern but usually did not possess the creative touch which makes music feel human. By using deep learning, models can now learn more complex musical patterns, bringing us closer to the generation of music that reflects genuine creativity and variety.

*2.2 The role of Deep Learning and Neural Networks*

Deep learning has revolutionized music generation by enabling models to identify and learn from implicit patterns in music data. RNNs (Recurrent Neural Networks) and CNNs (Convolutional Neural Networks) have been used traditionally to process sequential data, but our project uses a Transformer-based approach. This method is especially suited for extracting the long-range dependencies that are inherent in music and produces compositions that sound more cohesive and dynamic. By moving beyond simple repetition or fixed templates, deep learning architectures allow the system to generate music that is stylistically consistent but with imaginative variations, therefore producing pieces that are interesting and expressive.

*2.3 Combining GANs and Transformers*

Combining Generative Adversarial Networks (GANs) and Transformers proved to be highly effective for music generation. GANs utilize the feedback loop, with one component generating music and another part evaluating its quality, which helps generate more polished and realistic outputs. Transformers, on the other hand, capture the relationships between musical notes and patterns over time, which is important for generating music that sounds coherent and flows naturally. Together, they create a system that can produce music that is both emotionally expressive and structurally sound.

1. **Model Architecture**

*3.1 Transformer-GAN Architecture*

*3.1.1 Transformer*

The Transformer model plays a central role in generating music by handling sequence modeling, which is especially important in capturing the long-term patterns and relationships found in music. Using self-attention, the Transformer analyzes how different parts of a musical sequence relate to each other, allowing it to grasp intricate musical structures. This setup helps the model create compositions that develop logically and naturally, keeping elements like harmony, melody, and rhythm in sync.

*3.1.2 GANs (Generative Adversarial Networks)*

To make the generated music feel more realistic and emotionally aligned, we integrate GANs into the architecture. This is how it operates: the Generator creates music sequences guided by the structure the Transformer provides. These sequences are then evaluated by the Discriminator, which acts as a quality check, determining whether the music sounds authentic and matches the intended emotional tone (using metrics like valence and arousal). Through this back-and-forth process, the Generator learns to refine its outputs, gradually producing music that becomes harder and harder to distinguish from real, human-composed pieces, thanks to the constant feedback from the Discriminator.

A diagram of a process

Description automatically generatedA diagram of a block diagram

Description automatically generated*3.2 Generator and Discriminator Models*

Figure 5: Discriminator architecture diagram

Figure 4: Generator architecture diagram

*3.2.1 Generator(a):*

The Generator is responsible for creating new musical sequences based on specified conditions like emotional states (e.g., happiness or sadness). It works as follows:

Embedding layer: Converts the input sequence (notes or musical tokens) into high-dimensional vectors. A Positional Embedding is also added to retain the order of elements, essential for maintaining musical structure.

Attention block: Here, the Transformer architecture uses self-attention to capture relationships between sequence elements, even those far apart. This capability allows the Generator to maintain coherence and recognize recurring themes or motifs across the composition.

Fully Connected (FC) layer: Refines the processed information from the attention mechanism to enhance musical detail.

Conditioning and output: Integrates Condition Embedding, enabling the model to tailor its output to specific emotional inputs. The final musical sequence reflects both the original context and the chosen emotional state.

*3.2.2 Discriminator(b):*

The Discriminator evaluates whether the music generated by the model is realistic or authentic. It operates as follows:

Input embedding: Similar to the Generator, it embeds the sequence (real or generated) in a high-dimensional space.

Positional and CLS Embedding: Positional Embedding helps with sequence order, while CLS (classification) Embedding helps differentiate input categories.

Attention block: Examines relationships within the musical sequence to assess coherence and consistency, identifying long-range dependencies that suggest realism.

Fully Connected layer and prediction maps: A Fully Connected layer refines the sequence analysis, generating two types of predictions: Local and Global Prediction Maps. These provide judgments on the smaller, detailed sections of the music and the overall structure, respectively, enabling a comprehensive assessment of authenticity.

*3.2.3 Interaction between Generator and Discriminator*

The integration of Transformer and GAN architecture brings a unique advantage to music generation:

Sequence modeling: The Generator leverages the self-attention of the Transformer to grasp complex, long-range dependencies in music, producing sequences that are coherent and dynamically varied.

Adversarial process: The Generator and Discriminator engage in an adversarial setup, where the Generator creates music aiming to sound realistic, while the Discriminator assesses its authenticity. Through this iterative process, the Generator refines its output to better "fool" the Discriminator, achieving increasingly realistic results.

Emotional conditioning: By conditioning the output on emotional inputs (such as valence and arousal), the Generator can produce music that not only sounds authentic but aligns with specific emotional contexts, a valuable feature for adaptive or personalized music applications.

Realism and feedback loop: With each iteration, the Discriminator provides feedback to the Generator, helping it improve both the quality and emotional accuracy of its musical sequences. This feedback loop enhances the ability of the Generator to produce music that closely resembles human compositions.

The Transformer-GAN architecture combines the strengths of Transformers for capturing complex musical sequences with the iterative refinement of GANs, producing music that is both emotionally expressive and structurally coherent. The Transformer models long-term dependencies, while the GAN enhances the realism, making this architecture highly effective for generating quality, emotion-driven music.

1. **Challenges and Solutions**

*4.1 Performance and Computation Challenges*

*4.1.1 Training Time and Resources*

Training on a CPU was a critical problem as each epoch lasted an unacceptably long time, which drastically reduced our development and experimentation cycles. We addressed this problem by shifted the training processes to GPUs and leveraged Google Colab Pro+, enabling us to run experiments much faster and accelerate our development cycle.

*4.1.2 Resource Management*

During training, we encountered significant limitations regarding computational resources and access to compute units. Google Colab imposed usage restrictions that prevented us from having unlimited resource access, leading to delays and a complex scheduling process for training sessions. To overcome this challenges, we meticulously planned the training schedule to ensure that every available resource window was utilized to its fullest potential. We implemented dynamic resource management strategies, which involved efficiently distributing training tasks based on the available workload. Additionally, we carefully managed compute units by applying advanced optimization techniques, enabling us to maximize system performance within the existing constraints.

*4.2 Model and Data Size Challenges*

Our initial model was large and complex and this led to high resource consumption and extremely long training times. This made it difficult to perform real time experiments and adjustments, thereby hindering the refinement of the model. To address this problem, we redesigned the model into a smaller, more efficient version. This efficient model preserved the core capabilities required while significantly reducing the number of parameters, thus reducing resource consumption and training time. In addition, we trained the model on a representative subset of the data (approximately 820 songs), enabling faster experiments with still acceptable accuracy.

*4.3 Engineering Challenges*

*4.3.1 Algorithm Selection and Optimization*

we needed to extract musical features from various elements such as tempo, musical key, notes, and durations from MIDI files. Default algorithms were not exactly tailored for this task, and they failed to capture the unique musical characteristics. Also, without precise tuning of parameters such as the learning rate and batch size, the model struggled to generalize well and tended to overfit the training data. To overcome these challenges, we customized the algorithms to give higher priority to extracting specific musical features. We also meticulously fine-tuned key parameters (such as the learning rate and batch size) through experimental trials to determine the optimal values. Finally, we introduced regularization techniques to prevent overfitting, so that the model can learn in a balanced and stable manner.

*4.3.2 System Design and Integration*

As we implemented the system, integration issues arose among the process components- data processing, feature extraction, and model training. Each stage needed to handle massive data quickly and reliably but lacked a unified and modular system design. This led to delays, inefficient resource usage and difficulties in tracking processes in real-time. To address these challenges, we designed the system in a modular way where each component had a clear purpose and shared data with the others through defined interfaces. This enabled efficient memory and resources management in order to maintain smooth data flow between the data processing, feature extraction, and model training processes. As a result, we enhanced system performance and minimized issues from component compatibility.

*4.4 Technical Challenges*

*4.4.1 Virtual Environment and Dependency Management*

During development, we encountered conflicts between library versions as different packages required specific versions to function properly. This led to instability in the working environment and issues in executing the code. To address these issues, we utilized tools such as virtualenv and conda to create isolated environments. This allowed us to precisely manage the versions of libraries and avoid conflicts and the working environment became stable and consistent for the project.

*4.4.2 Server Setup and API Connectivity*

Setting up a server to manage API calls and ensure smooth interaction between the user interface and the backend system is challenging, especially with regards to load management and data security. Without proper configuration, the system can become unstable and insecure. We set up the server using Flask, a stable and popular framework, and incorporated best practices for load management and data security. This configuration enabled communication between the user interface and the backend system to be reliable, fast, and secure.

*4.4.3 Implementation of the User Interface*

Users expect an intuitive and seamless interface, but unifying the UI and the model can also be challenging and lead to issues such as CORS issues and inadequate session management. The user experience can be negatively impacted and lead to communication errors in the system. We designed a RESTful API to ensure seamless integration between the user interface and the model, addressing issues such as CORS and session management. This solution ensured seamless communication between the interfaces, significantly enhancing the user experience and security.

1. **Development Process**

*5.1 Data Preprocessing*

The code uses the Maestro-v1.0.0 dataset, which contains MIDI files of musical performances. Imports required libraries:   
os- for manage file paths and interact with the local file system, numpy for manage numerical arrays where the processed data is stored. pickle: Used to save the collected features into a .pkl file. Music21- for using the functions converter, note, and chord to process and parse MIDI files. Google Drive- when working in a Google Colab environment.

first mounts Google Drive to enable access to the folder containing the MIDI files, then uses music21 to load each MIDI file and flatten it into a musical stream while skipping corrupted or invalid files using a try/except block. While parsing, it extracts the average tempo (or defaults to 120), analyzes the key (for example, C major), determines the time signature (defaulting to 4/4 if none is found), and iterates over all musical elements to check whether each element is a note, a chord, or a rest, recording for each note its pitch, offset, and duration, for each chord a string representing its pitches alongside offset and duration, and for each rest the offset and duration; finally, for every MIDI file, the code creates a dictionary that includes the file name, the tempo, the key, the time signature, and a “notes” list containing all these extracted events, and appends this dictionary to a list called features, which is then saved in a .pkl file for future use.

*5.2 Model Architecture*

The code utilizes a toolset and infrastructure centered around PyTorch, a Transformer architecture, and GANs. Below are the main components:

PyTorch- The primary library for building neural network models, featuring modules that allow you to define layers, loss functions, and optimizers. Attention/Transformer Modules- Leverages attention-based layers, either using built-in PyTorch classes or custom implementations, to construct the Transformer architecture. music21- convert notes (tokens) into the required format of the model and streamline the process of generating MIDI files.

The code constructs a neural network model using PyTorch based on a hybrid model of Transformer architecture for music generation and a Generative Adversarial Network (GAN) training process. The process begins by importation of PyTorch modules functions, attention functions, and any additional utilities required for handling musical sequences. Following the definition of the generator and discriminator, they are integrated into a GAN framework: the generator produces new musical tokens sequences, and the discriminator evaluates these sequences and real data. The objective of the generator is to fool the discriminator, while the objective of the discriminator is to correctly classify real or generated music. During configuration, the model is set up with loss functions appropriate for adversarial training (such as binary cross-entropy for real/fake classification) and optimizers (Adam) for each network. This adversarial setup continuously refines the generator’s output quality while making the discriminator more robust in detecting artificial sequences. Through the dynamic computation graph and attention-based mechanisms in PyTorch, the model is able to learn complex musical patterns and generate coherent compositions with greater realism.

*5.3 Model Training*

PyTorch- For building and training the neural network model, defining layers, and executing the training process. NumPy- For handling numerical data, managing arrays, and creating mini-batches for efficient computation. Adam Optimizer- For updating model weights based on the computed gradients during backpropagation.

Training the Transformer-GAN model means training the Generator and Discriminator on the preprocessed music data. We built the model using PyTorch, which simplifies building neural networks and enables GPU training for efficiency. During training, we load song snippets in batches using a DataLoader instead of one huge sequence, which is more efficient and helps the model generalize. For each batch, the Generator attempts to produce new musical sequences within the given input conditions, and the Discriminator evaluates both real sequences from the dataset and synthetic ones generated by the Generator. We calculate two losses: one for the ability of the Generator to fool the Discriminator, and one for how well the Discriminator distinguishes real from fake, then update their weights with backpropagation and the Adam optimizer. This process repeats for many epochs, where each epoch is a complete pass over the dataset. Over time, the Generator refines its music creation, and the Discriminator becomes better at detecting flawed sequences. We monitor metrics such as the loss of the Generator (lower indicates more convincing music) and potentially a measure of the accuracy of the Discriminator to ensure training remains stable. If the Discriminator becomes too strong too quickly, the loss of the Generator might spike, indicating that we may need to adjust hyperparameters like the learning rate or revise the training schedule.

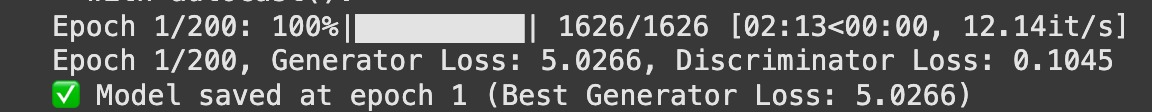
At this early stage, the Generator loss (5.0266) is quite high, indicating that it is still learning how to produce convincing music sequences. The Discriminator loss (0.1045) is extremely low, suggesting that it is able to easily separate real sequences from those generated by the model.

Figure 6: Training Progress at Epoch 1

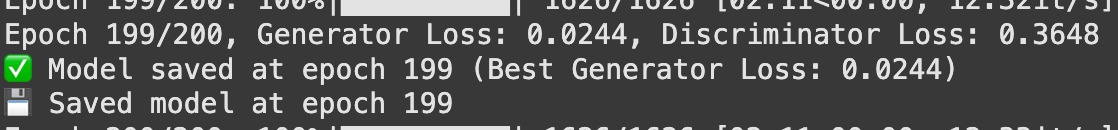


Figure 7: Training Progress at Epoch 199

At this point, the Generator loss has dropped dramatically (0.0244), showing it can now produce far more realistic sequences. In contrast, the Discriminator loss (0.3648) has increased, reflecting that it struggles more to differentiate real sequences from those created by the Generator.

*5.4 Generate Music*

Once the training process is complete and the final model is saved, the program transitions into generating music. It begins by loading the trained model, which contains the learned weights and configurations. Next, a snippet of notes used during training is selected to initiate the generation process. With the seed, the model predicts the next note by passing the sequence through its embedding, recurrent, and output layers. The newly predicted note is appended to the seed, thereby updating the input for the next prediction. The iterative loop continues until composition is of desired length or composition meets specific criteria defined in the code. After the entire sequence of notes is formed, the code utilizes music21 to convert the numerical tokens into a musical stream. This step maps each token onto actual musical elements such as notes, chords, or rests. The musical stream is then written to a MIDI file, reflecting the newly generated composition. The code then directly converts the MIDI file to WAV using the midi2audio library and then converts the resulting WAV file to MP3 using the pydub library such that the final output is an MP3 file ready for playback or distribution. Finally, the MP3 file is saved for future use. Here, the users can listen to the new composition, download it, or integrate it into other projects, demonstrating the ability of the model to create original musical pieces based on patterns learned during training.

*5.5 Web Interface*

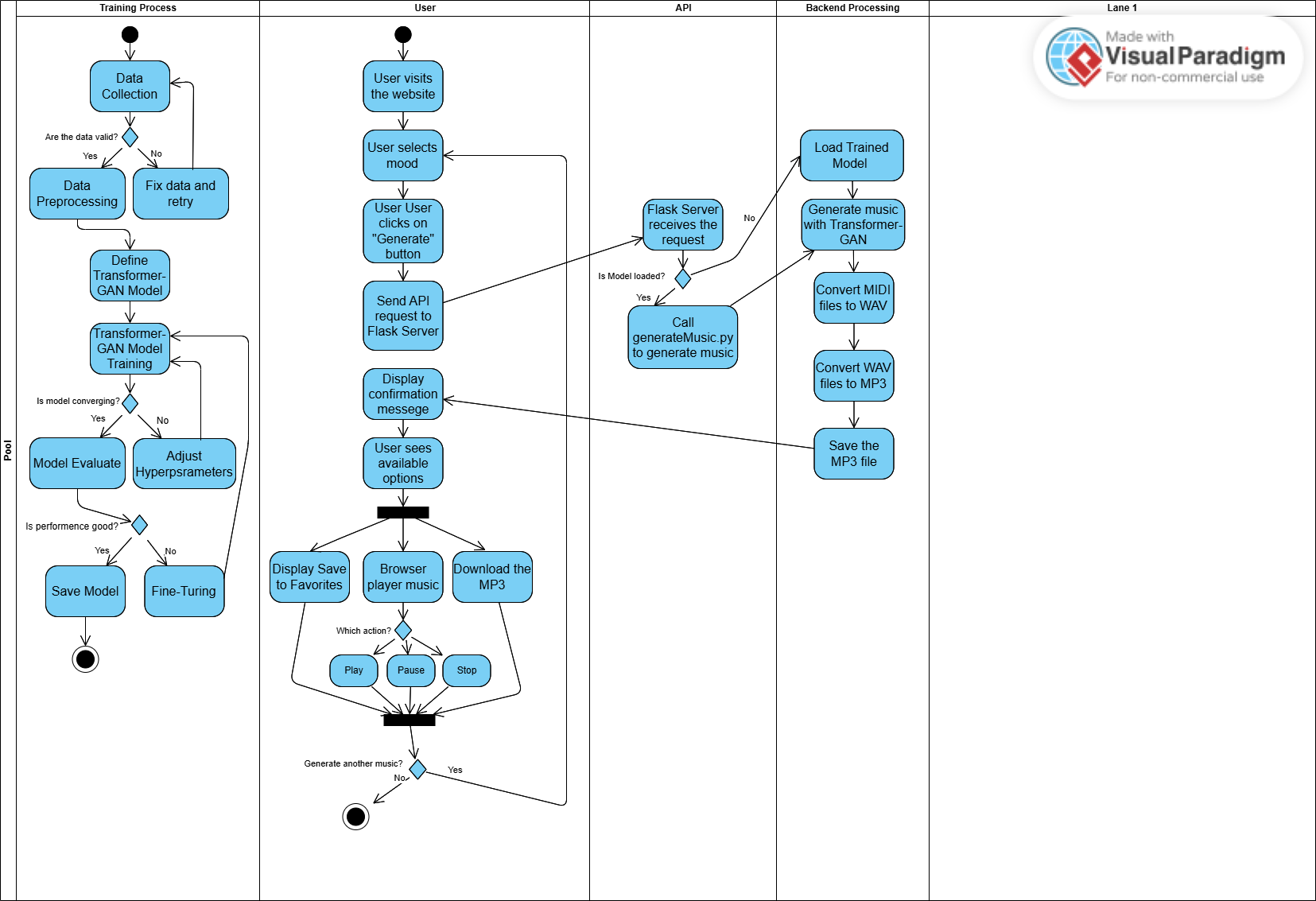
At this phase, a web interface was developed to allow users to generate music and listen to it directly in their browser. A web server is configured to run on a certain port, displaying an HTML page that includes a number of necessary tools: CSS library (TailwindCSS)- for building a modern, responsive layout. UI component library (DaisyUI)- provides ready-made, customizable elements. audio management library (Amplitude.js)- for handling playback controls and playlist management. localStorage- for preserving user data such as search history or favorite tracks.

When users select a mood and click the “Generate” button, an API call is sent to the server, which runs a script to create an MP3 file. The server provides a URL which the interface loads for instant playing or download. Search history and favorite songs are stored in in localStorage in the browser so that users can revisit older creations to replay. The result is a user-friendly interface where one can choose a music style, listen to the newly generated composition, and download the resulting MP3 file with one click.

**6. Visualizing the music generation process**

תמונה שמכילה טקסט, צילום מסך, גופן, עיצוב

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.*6.1 Deployment Diagram*

*6.2Activity Diagram*

**7. Product Requirement**

*7.1 Functional Requirements*

**Emotion-Based Music Generation:**

* Users must be able to select emotional parameters like “happy”,“calm”, “elegant”.
* The system should generate a MP3 filethat aligns with the specified emotional criteria.

**User-Friendly Interface**

* A web-based UI should allow users to click a “Generate” button and receive a result.
* The interface should include an embedded audio player and provide an option to download the file.

**Automatic File**

* After generating the note sequence (MIDI), the system should convert it to WAV and then to MP3 without user intervention.
* The final output must be an MP3 file ready for playback.

**User Management**

* The system should store the generation history or favorite tracks of the user.
* Users must be able to revisit and replay previously generated pieces.

*7.2 Non-Functional Requirements*

**Performance**

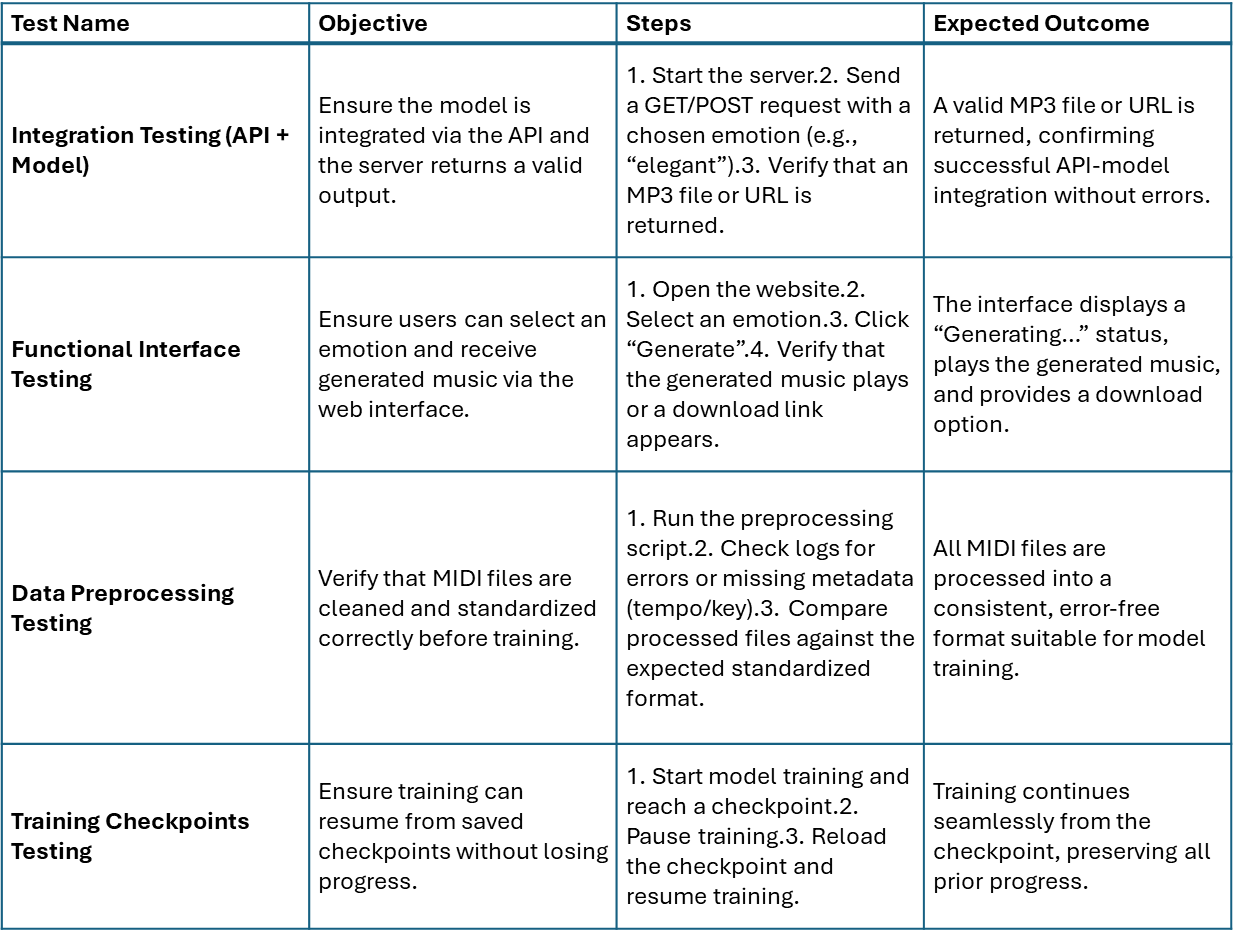
* The system should generate a short music clip (30–60 seconds) within a reasonable timeframe, depending on GPU availability.
* The system should handle multiple requests concurrently, limited by server/cloud capabilities.

**Maintainability and Extensibility**

* The codebase should be modular to facilitate updates and changes (for adding more emotional states or swapping out the model).
* Training procedures, parameters, and configurations should be documented to streamline future maintenance and model enhancements.

**Reliability & Stability**

* The system must run consistently without crashes, particularly during music creation and file conversion.
* Proper error handling and informative messages should be provided to the user in case of failure.

**8. Test Planning**

**9. Result**

*9.1 Conclusions*

Throughout this project, our goal was to classify music according to emotional content and generate a unique composition for each selected mood, enabling users to choose the desired emotion and receive an accessible audio file. During the research phase, we identified several potential datasets, but most either unavailable or came in incompatible formats (for instance, MP3 rather than MIDI). Further, our primary dataset (Maestro-v1.0.0) included about 1,200 solo piano pieces, which limited the stylistic and emotional diversity and therefore of generalizability by the model. With these constraints, we were only able to fully implement the “elegant” mood, producing original music but not yet supporting additional moods. While working in Google Colab, we also faced resource constraints that required optimizing batch sizes and epochs to avoid memory crashes. Our development strategy prioritized simplicity: rather than attempting to support multiple moods with partial data, we focused on “elegant” to ensure stable performance and high-quality output. As a result, we created a user-friendly web interface built on design and audio libraries, allowing users to easily generate and listen to a musical composition. Though we did not finish the initial multiple moods scheme, focusing on one mood enhanced the system’s stability and demonstrated its core capabilities.

*9.2 Reflections and Insights*

We followed a structured approach throughout the project, starting with data exploration and goal-setting, then moving through model development and user interface design. In retrospect, however, we would have invested more time refining our plan for data acquisition and setting up the development environment. A more extensive, diversified dataset and a well-prepared training infrastructure would have facilitated easier experimentation and supported multiple moods from the out set. Nevertheless, we succeeded in delivering a functional system that generates original music and provides a user-friendly interface, thereby meeting the core goal of producing an emotion-aligned composition. While we did not fully implement all the moods we initially envisioned, our work demonstrates the feasibility of this approach and lays the groundwork for future expansion. We consider this partial fulfillment of our metrics to presenting a working prototype for music generation and an accessible interface, while recognizing that mode data, resources, and time would be necessary to achieve a more complete range of emotional classifications.

* 1. **User Guide**

“Music Generation” provides a quick and intuitive way to generate and listen to custom music clips at the click of a button. The user interface is designed to be straightforward, allowing you to select moods or categories, play the music, and bookmark favorite tracks for later listening.

*10.1 Selecting a Mood*

From the sidebar or top menu, click on a mood:

Then, click on "Generate" button to produce new music.

*10.2 Song Generation Confirmation*

תמונה שמכילה טקסט, צילום מסך, מולטימדיה, תכונות מולטימדיה

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.After clicking "Generate", the system creates and loads the new music track, displaying a "Loading…" message until the process is complete:

After clicking "Generate" and seeing the "The output is ready" message, click "OK" to close the notification and listen to your newly generated track:

תמונה שמכילה טקסט, צילום מסך

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי.

Clicking “OK” closes the notification, and you can then find the newly generated track in the player interface, ready for playback.

תמונה שמכילה טקסט, צילום מסך, מולטימדיה, גופן

תוכן שנוצר על-ידי בינה מלאכותית עשוי להיות שגוי. *10.3 Music Playback*

* Click on the track name or a “Play” button to start audio playback.
* Use “Pause” to halt the music.
* A progress bar may allow you to seek a specific point in the track.
* If there are multiple tracks, “Next” or “Previous” buttons allow you to switch between them.

*10.4 Actions After Playback*

* **Download:** The user can save the MP3 file locally.
* **Play the Next Track:** If there is a playlist or additional recommendations, the user can move on to the next track.
* **Choose another emotion:** You can choose another emotion or mood to generate a new piece of music

**11. Maintenance Guide**

*11.1 Operating Environment and Installation*

It is recommended to use a machine capable of running Python 3.10 or 3.11, as Python 3.13 may cause compatibility issues with certain libraries.

Optional GPU support (for re-training the model), typically via Google Colab or a CUDA-compatible environment.

*11.1.1 Python Package Dependencies*

* numpy<2
* torch, torchvision, torchaudio (CPU or GPU versions, depending on requirements)
* music21
* midi2audio (requires separate FluidSynth installation)
* pydub (requires ffmpeg)

*11.1.2 External Dependencies*

* FluidSynth (required for converting MIDI files to WAV)
* SoundFont (used by FluidSynth to render audio samples)
* ffmpeg (required for converting WAV files to MP3)

*11.2 Installation Instructions*

Open your terminal (or command prompt) and navigate to your project directory.

**Install Python Dependencies:** Inside the activated virtual environment, install the required packages using pip.

**Install External Tools:**

* FluidSynth- On macOS: brew install fluidsynth On Linux: sudo apt-get install fluidsynth On Windows: download and install FluidSynth or use a package manager like Chocolatey if available.
* SoundFont- Obtain a valid SoundFont file and place it in a known directory.
* Ffmpeg- On macOS: brew install ffmpeg On Linux: sudo apt-get install ffmpeg On Windows: download the latest FFmpeg build and add it to your PATH.

**Project File Configuration:**

Download the folder from Google Drive (which contains the following files:

* best\_model\_Gloss\_0.0319.pth – The trained model weights.
* note\_to\_int.pkl and int\_to\_note.pkl – Mapping files for converting notes to indices and vice versa.
* training\_target.pkl – Target data for training.
* FluidR3\_GM.sf2 – The SoundFont file required for converting MIDI to WAV.

After downloading, place the entire folder directly into the mymusic directory of your project. This ensures that the file paths align with the existing code, eliminating the need for further modifications. If you change the folder name or its location, remember to update the paths in your scripts accordingly.

**Run the Application:**

Start the web server or the relevant Python scripts, then open your browser and navigate to the specified port to access the web interface, generate music, and download or play the resulting audio files.

**GPU Support:**

If you plan to train or retrain models on a GPU, ensure that you have installed the correct CUDA and cuDNN libraries, and use the GPU-enabled version of PyTorch. By following these steps, you will have a fully functional environment to run the music generation project, train models, and serve the application via a web interface.

*11.3 Enhancements and Updates:*

**Updating or Retraining the Model**:

* Replace the weights file with the new one created after additional training.
* If you have changed the file name, update the path in the loading script.
* If you have added notes or changed the data structure, also update the mapping files.

**Adding Functionality to the Web Interface:**

* To modify API logic (like adding a new endpoint), edit the server code (Flask).
* To add new buttons or visual elements, update the HTML file.

**Expanding the Dataset:**

* You can add new MIDI files to the same data folder and rerun the preprocessing and training steps.
* Ensure that the new files conform to the current input format (music21).

**Regression Testing:**

* After any changes, verify that the system still generates valid music files and that the interface functions as expected.
* Run automated or user tests to ensure no new issues have been introduced.

**Expanding the Range of Emotions:**

 Collecting and preparing a dedicated dataset – Gather MIDI files that musically represent the desired emotion and preprocess them for consistency.

 Training a separate model – A new Transformer-GAN is trained exclusively on the selected emotional dataset.

 Integrating the new model – The system is updated to load the correct model when the user selects an emotion.

 Validating the results – The generated music is tested to ensure it accurately reflects the intended emotion.

**12. References:**

1. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. Advances in Neural Information Processing Systems, 2672–2680. <https://arxiv.org/abs/1406.2661>
2. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), 4171–4186. <https://arxiv.org/abs/1810.04805>
3. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780. <https://doi.org/10.1162/neco.1997.9.8.1735>
4. Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. Proceedings of the 3rd International Conference on Learning Representations (ICLR). <https://arxiv.org/abs/1412.6980>
5. Neves, P. L. T., Fornari, J., & Florindo, J. B. (2022). Generating music with sentiment using transformer-GANs. arXiv preprint. <https://arxiv.org/pdf/2212.11134>
6. Radford, A., Metz, L., & Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint. <https://arxiv.org/abs/1511.06434>
7. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 5998–6008. <https://arxiv.org/abs/1706.03762>
8. Zhang, H., Xu, T., Li, H., Zhang, S., Wang, X., Huang, X., & Metaxas, D. (2017). StackGAN: Text to photo-realistic image synthesis with stacked generative adversarial networks. Proceedings of the IEEE International Conference on Computer Vision, 5907–5915. <https://arxiv.org/abs/1612.03242>