INDUSTRIAL ORIENTED MINI PROJECT

Report

On

Auto Composer : AI Based Music Generation and Sheet Generation

Submitted in partial fulfilment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

In

INFORMATION TECHNOLOGY

By

Chiriki Chandana – 22261A1215

Kaniveta Bhuvan Chandra - 22261A1231

Under the guidance of

Mrs.P.Naveena

Assistant Professor, Department of IT

DEPARTMENT OF INFORMATION TECHNOLOGY

MAHATMA GANDHI INSTITUTE OF TECHNOLOGY (AUTONOMOUS)

(Affiliated to JNTUH, Hyderabad; Eight UG Programs Accredited by NBA; Accredited by NAAC with ‘A++’ Grade)

Gandipet, Hyderabad, Telangana, Chaitanya Bharati (P.O), Ranga Reddy District, Hyderabad– 500075, Telangana

2024-2025

CERTIFICATE

This is to certify that the **Industrial Oriented Mini Project** entitled **Auto Composer :AI – Based Music Generation and Sheet Creation** submitted by **Chiriki Chandana(22261A1215)** and **Kaniveta Bhuvan Chandra (22261A1231)** in partial fulfillment of the requirements for the Award of the Degree of Bachelor of Technology in Information Technology as specialization is a record of the bona fide work carried out under the supervision of **Mrs.P.Naveena**, and this has not been submitted to any other University or Institute for the award of any degree or diploma.

Internal Supervisor:	**IOMP Supervisor:**
Mrs.P.Naveena	**Dr. U. Chaitanya**
Assistant Professor	Assistant Professor
Dept. of IT	Dept. of IT

EXTERNAL EXAMINAR	**Dr. D. Vijaya Lakshmi**
	Professor and HOD
	Dept. of IT

DECLARATION

We hear by declare that the **Industrial Oriented Mini Project** entitled **Auto Composer: AI – Based Music Generation and Sheet Generation** is an original and bona fide work carried out by us as a part of fulfilment of Bachelor of Technology in Information Technology, Mahatma Gandhi Institute of Technology, Hyderabad, under the guidance of **Mrs.P. Naveena , Assistant Professor**, Department of IT, MGIT.

No part of the project work is copied from books /journals/ internet and wherever the portion is taken, the same has been duly referred in the text. The report is based on the project work done entirely by us and not copied from any other source.

Chiriki Chandana – 22261A1215
Kaniveta Bhuvan Chandra - 22261A1231

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success. They have been a guiding light and source of inspiration towards the completion of the **Industrial Oriented Mini Project**.

We would like to express our sincere gratitude and indebtedness to our Internal supervisor **Mrs.P. Naveena,** Assistant Professor, Dept. of IT, who has supported us throughout our project with immense patience and expertise.

We are also thankful to our honourable Principal of MGIT **Prof. G. Chandramohan Reddy** and **Dr. D. Vijaya Lakshmi**, Professor and HOD, Department of IT, for providing excellent infrastructure and a conducive atmosphere for completing this **Industrial Oriented Mini Project** successfully.

We are also extremely thankful to our IOMP supervisor **Dr. U. Chaitanya**, Assistant Professor, Department of IT, and senior faculty **Mrs. B. Meenakshi,** Assistant Professor**,** Department of IT for their valuable suggestions and guidance throughout the course of this project.

We convey our heartfelt thanks to the lab staff for allowing us to use the required equipment whenever needed.

Finally, we would like to take this opportunity to thank our families for their support all through the work. We sincerely acknowledge and thank all those who gave directly or indirectly their support for completion of this work.

Chiriki Chandana – 22261A1215
Kaniveta Bhuvan Chandra-22261A1231

ABSTRACT

AI Music Composer system that automates the process of generating, synthesizing, and visualizing musical compositions using Python. It leverages the PrettyMIDI library to algorithmically create MIDI sequences based on configurable musical parameters such as duration, tempo, pitch range, scales, arpeggios, rhythm choices, and melodic smoothness. The generated notes simulate the structure of realistic piano performances, and users have control over musical variation through probabilistic and rule-based methods.

The system then synthesizes these MIDI files into high-quality WAV audio using FluidSynth and a SoundFont file, ensuring the playback sounds natural and expressive. To provide deeper insight into the composition, a piano roll visualization is rendered using Matplotlib, showing pitch, timing, and velocity data with color-coded intensity, making the dynamics of the music clearly interpretable.

Furthermore, the system utilizes the Music21 toolkit and LilyPond to convert the MIDI into traditional sheet music in PDF format, allowing musicians to view and perform the composition. All outputs, including the MIDI, WAV, and sheet music PDF files, are downloadable for user convenience. This complete pipeline—from composition to audio playback and sheet music generation—makes the AI Music Composer a comprehensive tool for musicians, educators, and hobbyists interested in algorithmic music generation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1MOTIVATION

Music is a universal form of expression, but composing music traditionally requires years of training and access to instruments, notation tools, and production software. With the rise of artificial intelligence and open-source tools, there's a growing opportunity to democratize music creation—making it more accessible to learners, hobbyists, and even professional musicians seeking inspiration or automation.

The motivation behind this project is to bridge the gap between creativity and technology by building an intelligent system that can automatically generate and render musical compositions. Rather than replacing musicians, this system serves as a creative assistant—helping users explore new ideas, generate practice material, and better understand musical structures.

By integrating MIDI generation, high-quality audio synthesis, piano roll visualization, and traditional sheet music rendering in a single pipeline, the AI Music Composer provides a comprehensive platform that can be used for Learning music theory interactively, Rapid prototyping of musical idea, generating accompaniment or background scores, Supporting people with limited access to musical instruments. In a world increasingly driven by AI and digital creativity, this project exemplifies how algorithmic composition can support and amplify human artistic expression.

1.2 PROBLEM STATEMENT

Despite the growing integration of AI in creative fields, music composition remains a technically demanding and resource-intensive process, especially for individuals without formal training in music theory or access to quality instruments. Traditional composition workflows require a deep understanding of rhythm, harmony, and notation, along with tools for audio production and sheet music generation. This complexity poses a significant barrier

to learners, independent creators, and those looking to experiment with musical ideas quickly and intuitively.
Moreover, existing digital tools often specialize in only one aspect of the music-making process—such as composition, playback, or visualization—without offering a unified platform. Musicians are typically required to switch between different software for creating melodies, rendering audio, and producing sheet music. This fragmented approach reduces efficiency and limits accessibility, especially for users who are not technically proficient or who work in resource-constrained environments.

There is a clear need for a single, cohesive system that can automate and streamline the end-to-end process of music composition. Such a system should be capable of generating musically coherent material algorithmically, synthesizing it into high-quality audio, providing visual feedback through intuitive representations like piano rolls, and converting it into traditional sheet music. The AI Music Composer project addresses this need by integrating all these functionalities into a unified Python-based tool, making the process of composing, hearing, and reading music accessible to a broader audience.

1.3 EXISTING SYSTEM

Most existing online music tools emphasize manual composition or use preset loops and samples, offering little to no support for AI-driven automatic music generation. These platforms generally lack user-configurable controls over musical parameters like tempo, scales, or melodic variation, limiting creative flexibility.

While some AI-based online services can generate melodies, they often operate as closed systems with minimal options to customize or fine-tune the generated music. Additionally, these tools rarely integrate audio synthesis, piano roll visualization, or sheet music generation into a single seamless workflow, forcing users to rely on multiple disconnected applications.

Moreover, the absence of downloadable, high-quality outputs such as WAV audio files and sheet music PDFs in most existing platforms reduces their practical usability for musicians and educators. There is a clear need for an online, end-to-end system that automates composition, synthesis, visualization, and notation with user control—something that your AI Music Composer project effectively addresses.

.

1.3.1 LIMITATIONS

- **Creativity Bound by Algorithms**
  The AI-generated music relies on predefined rules and user-set parameters like tempo, scale, and rhythm. While this ensures structured output, it limits the system's ability to replicate the spontaneity and emotional depth found in human compositions. As a result, the music can sometimes feel predictable or less expressive compared to what a skilled musician might create naturally.
- **Dependency and Performance Constraints**
  The system's audio quality depends heavily on the FluidSynth synthesizer and the quality of SoundFont files used. This can affect how natural and rich the music sounds, sometimes falling short of live instrument expressiveness. Visualization and notation tools also have limited support for complex or highly nuanced music. Additionally, the online platform may face performance issues on slower devices or connections, affecting user experience.

1.4 PROPOSED SYSTEM

The proposed system is a fully online AI Music Composer platform that allows users to create unique and original musical compositions through a user-friendly web interface. Users can customize important musical parameters such as duration, tempo, scale type, root note, pitch range, rhythm patterns, and arpeggio usage. By adjusting these inputs, the system generates MIDI sequences that mimic realistic piano performances. Importantly, the system incorporates probabilistic and rule-based variations to ensure that every music generation is different, providing fresh and diverse compositions with each execution.

On the backend, the platform uses advanced algorithms to produce expressive and structured MIDI files based on the user's settings. These MIDI files are then synthesized into high-quality WAV audio using FluidSynth along with customizable SoundFonts, delivering natural and rich sound output. Alongside audio synthesis, the system generates detailed piano roll visualizations and converts MIDI data into traditional sheet music PDFs using Music21 and LilyPond. This combination allows users to both listen to and visually analyze their compositions easily.

The frontend interface provides an interactive experience where users can control the entire music creation process online without installing specialized software. Once the music is generated, embedded audio playback allows immediate listening, and users can download the MIDI, WAV, and sheet music PDF files for offline use. This seamless integration of composition, synthesis, visualization, and notation makes the system accessible for musicians, educators, and hobbyists, enabling creative exploration of algorithmic music generation anytime, anywhere.

1.4.1 ADVANTAGES

1. User-Friendly Online Access

The system is fully web-based, allowing users to generate, listen to, and download music directly from their browsers without needing to install any software. This makes the platform highly accessible to musicians, educators, and hobbyists worldwide, regardless of their device or technical skills.

2. Customizable and Dynamic Music Generation

Users have control over multiple musical parameters like tempo, scale, pitch range, and rhythm, enabling tailored compositions to their preferences. Each execution produces a unique piece thanks to built-in probabilistic variations, ensuring fresh and creative outputs every time.

3. Comprehensive Output Formats

The platform delivers a complete package with downloadable MIDI files, high-quality WAV audio, and traditional sheet music PDFs. This all-in-one approach supports diverse use cases—whether for listening, performing, teaching, or further musical editing—making it a versatile tool for various audiences.

1.5 OBJECTIVES

- Generate unique and original musical compositions every time, ensuring diverse and creative outputs for users.
- Allow users to customize important musical parameters such as tempo, scale, pitch range, rhythm, and arpeggio usage to tailor the music according to their preferences.
- Provide high-quality synthesized audio files in WAV format that can be easily downloaded for offline listening and sharing.
- Automatically generate traditional sheet music in PDF format, enabling musicians to view and perform the compositions accurately.
- Offer an intuitive and accessible online platform where users can create, listen to, and download music without needing specialized software or technical skills.
- Include visual piano roll representations of the music to help users better understand the structure and dynamics of the composition.

1.6 HARDWARE AND SOFTWARE REQUIREMENTS

1.6.1 SOFTWARE REQUIREMENTS

• **Software**:
The system is developed using **VS Code** as the integrated development environment (IDE) for Python. VS Code provides a robust, lightweight environment for writing, testing, and debugging the code required for machine learning models and backend services.

• **Primary Language:**
The project is primarily developed in Python, which is well-suited for audio processing, algorithmic music generation, and web backend development. Python is used to handle MIDI sequence creation, audio synthesis, visualization, and sheet music generation. Key libraries include PrettyMIDI for MIDI manipulation, FluidSynth for audio synthesis, Music21 and LilyPond for converting MIDI to sheet music, and Flask for serving the online interface and handling user requests.

• **Frontend Framework:**
The frontend uses HTML, CSS, and JavaScript to create a user-friendly web interface. It lets users customize music settings, generate compositions, and download audio and sheet music files. JavaScript handles communication with the backend for real-time music generation.

• Backend Framework:

The backend is developed in Python using the Flask framework to handle music generation, audio synthesis, and file management. It processes user inputs, generates MIDI compositions with PrettyMIDI, synthesizes audio via FluidSynth, creates sheet music using Music21 and LilyPond, and serves these files through API endpoints for the frontend.

• **Database**:
The system uses SQLite3 as a lightweight database to store user preferences, generated music metadata, and download history. This setup efficiently manages data for a small to medium-scale online music composition platform.

• **Frontend Technologies**:
The user interface uses:

- HTML: For structuring the web page content.
- CSS: For styling, colors, and layout design.
- JavaScript: To handle interactivity, form submission, and communication with the backend API.
- Responsive Design: Custom CSS ensures the app works well on different screen sizes and devices.

1.6.2 HARDWARE REQUIREMENTS

- **Operating System:**
  The system is designed to run on Windows, Mac, and Linux operating systems, ensuring compatibility with Python, FluidSynth, and all related development tools.

- **Processor:**

  A processor with at least an Intel i5 or equivalent is recommended to efficiently handle MIDI generation, audio synthesis, and real-time music processing tasks.

- **RAM:**

  A minimum of 8GB RAM is required to smoothly run audio processing, music visualization, and backend services simultaneously. More RAM will improve performance, especially for longer compositions or concurrent users.

2. LITERATURE SURVEY

Hao-Wen Dong et al. proposed MuseGAN, a multi-track generative adversarial network designed for symbolic music generation using MIDI representations. Their system enables the creation of polyphonic, multi-instrument compositions, giving users control over different instruments within the piece. While MuseGAN advances multi-track music generation, it lacks fine-grained customization options for key musical parameters such as tempo, scale, and rhythmic patterns. This limitation indicates the need for integrating rule-based parameter controls to enhance user-guided composition [1].

Gino Brunner et al. developed MIDI-VAE, a variational autoencoder model that learns musical styles and dynamics from MIDI data. This approach allows interpolation between musical styles and separation of instruments, enabling flexible music generation. However, MIDI-VAE does not provide a full pipeline that includes audio synthesis or sheet music generation, limiting its practical use for end-to-end music creation. Future work could focus on combining symbolic music generation with audio rendering and notation export to form a comprehensive music composition system [2].

Philippe Esling et al. introduced FlowComposer, a constraint-based symbolic composition tool integrated with visualization capabilities. The system supports structured music creation and partial rule-based control, helping composers guide the generative process. Despite these strengths, FlowComposer lacks an integrated solution for audio synthesis and sheet music export, making it less suitable for applications requiring full composition-to-playback workflows. There is an opportunity to unify composition, real-time audio playback, and score exporting within a single platform [3].

Yu-Siang Huang et al. presented the Pop Music Transformer, a transformer-based model for generating long sequences of pop music in MIDI format. The model excels at producing realistic and expressive musical phrases by leveraging attention mechanisms. Nevertheless, its complexity can make it difficult for non-technical users to interact with in real time or customize according to their preferences. Enhancing transparency and usability in user interfaces remains a key area for improvement to broaden its accessibility [4].

Table 2.1 Literature Survey of Music Composer

Ref	**Author& year of publication**	**Journal / Conference**	**Method / Approach**	**Merits**	**Limitations**	**Research Gap**
[1]	Hao-Wen Dong et al., 2018	IEEE	MuseGAN: Multi-track GAN for symbolic music generation using MIDI representations	Enables polyphonic, multi-track composition with instrument control	Lacks fine user parameter customization (tempo, scale)	Need for rule-based parameter control like scale, rhythm, arpeggios
[2]	Gino Brunner et al., 2018	IEEE Transactions on Multimedia	MIDI-VAE: Variational Autoencoder to learn musical styles and dynamics in MIDI	Enables music interpolation and instrument separation	Doesn't offer end-to-end sheet music generation or audio synthesis	Combine MIDI generation with audio rendering and music notation
[3]	Philippe Esling et al., 2020	IEEE CEC	FlowComposer: Constraint-based symbolic composition tool with visualization	Supports structured composition and partial rule-based control	No integrated synthesis or sheet music export pipeline	Unify composition, audio playback, and score export in one pipeline
[4]	Yu-Siang Huang et al., 2020	IEEE ICASSP	Pop Music Transformer: Transformer model generating long pop music sequences	Generates realistic, expressive musical phrases with attention	Complex and opaque for real-time, user-customized interaction	Enhance transparency and customization for non-technical users.

3. ANALYSIS AND DESIGN

The AI Music Composer project addresses the challenge of automating music creation while providing flexibility and control over the composition process. The system is designed to generate, synthesize, and visualize music dynamically, enabling users—from musicians to hobbyists—to produce unique and expressive compositions efficiently. By integrating algorithmic composition techniques with real-time audio synthesis and music notation rendering, the project aims to deliver a seamless creative workflow.

At the heart of the AI Music Composer is the algorithmic generation of MIDI sequences, which are configured through user-defined parameters such as tempo, pitch range, musical scales, rhythm patterns, and melodic smoothness. This parameter-driven approach allows users to customize the style and structure of the music while maintaining natural variations through probabilistic and rule-based models. The system thus adapts to diverse musical preferences and creative goals.

The design incorporates multiple modules working cohesively: the MIDI generation engine uses PrettyMIDI to compose symbolic music; the synthesis module converts these MIDI sequences into high-quality audio using FluidSynth and SoundFont files; the visualization component renders piano rolls to illustrate the music's dynamics and timing; and the sheet music generation module uses Music21 and LilyPond to produce traditional music notation in PDF form. This modular architecture ensures that each stage of the music creation process is optimized and easily extendable.

Data flow in the system begins with user input specifying musical parameters, which guide the MIDI generation. The generated MIDI is then simultaneously processed for audio synthesis, visual representation, and sheet music creation. Finally, all outputs are presented to the user with options to download, making the system interactive and user-friendly.
By leveraging real-time processing and combining symbolic composition with audio and visual outputs, the AI Music Composer delivers a comprehensive tool that supports both the creative and educational needs of its users.

3.1 MODULES

1.User Interaction Module

This module provides an intuitive interface allowing users to specify music generation parameters such as duration, tempo, musical scale, root note, pitch range, use of arpeggios, rhythm choices, and melody smoothness. It collects these inputs through the frontend form and sends them to the backend for music composition. The module also displays the generated music outputs including audio playback, MIDI file, and sheet music PDF for download.

- Input: User parameters like duration, tempo, scale type, root note, pitch range, arpeggio usage, rhythm pattern, and melody bias.
- Output: Interactive results page showing audio playback, download links for MIDI, WAV, and sheet music PDF.

2.MIDI Generation Module

This backend module uses the user-provided parameters to algorithmically generate symbolic music data in MIDI format. It applies probabilistic and rule-based models to create melodies and rhythms consistent with the selected scale, pitch range, and rhythmic patterns, optionally including arpeggios based on user settings.

- Input: Musical parameters from the User Interaction Module.
- Output: Generated MIDI sequence representing the composed music.

3.Audio Synthesis Module

The MIDI output from the generation module is synthesized into high-quality audio using FluidSynth with a specified SoundFont. This module converts symbolic MIDI data into audible sound for immediate playback and user evaluation.

- Input: MIDI sequences from the MIDI Generation Module.
- Output: Synthesized audio files in WAV format for playback and download.

4.Sheet Music Generation Module

This module converts the MIDI data into traditional music notation using Music21 and LilyPond. It produces a visually readable PDF sheet music file that users can download, print, or study to understand the musical structure of the AI-generated composition.

- Input: MIDI sequences.
- Output: Sheet music in PDF format.

5.Visualization Module

Optionally, the system includes visualization features such as piano roll displays or rhythm pattern charts that provide visual feedback on the generated music's structure and dynamics. This enhances user understanding and engagement with the composition.

- Input: MIDI data and generated musical parameters.
- Output: Visual representations like piano rolls or rhythm graphs embedded in the frontend (if implemented).

3.2 ARCHITECTURE

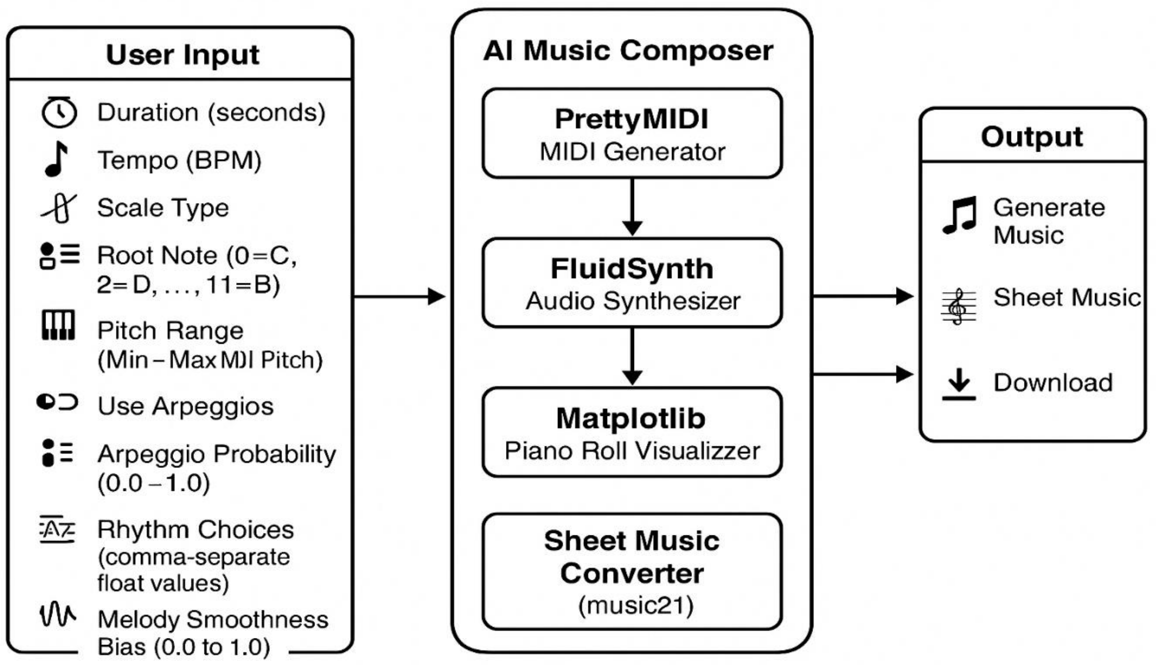


Fig. 3.2.1 Architecture of Auto Composer: AI-Based Music Generation and Sheet Creation

The architecture of the system, as shown in Fig. 3.2.1, is designed for automatically generating and synthesizing AI-based musical compositions. The system begins with the User Input & Data Collection component, which gathers real-time musical configuration parameters from the user, such as Duration, Tempo (BPM), Scale Type, Root Note, Pitch Range, Arpeggio Settings, Rhythm Choices**,** and Melody Smoothness Bias. These inputs are collected through a user-friendly web interface and are passed into the backend in JSON format for further processing.

Once the user data is received, it is sent to the AI Composition Engine, which acts as the central model responsible for generating music. In this phase, the system processes the input parameters and uses rule-based logic combined with probabilistic algorithms to generate a sequence of MIDI notes that reflect the selected musical mood and structure. These notes simulate realistic piano performances and preserve musicality through dynamic rhythm, smooth melodic transitions, and harmonic balance.

The core of the generation pipeline is powered by the PrettyMIDI library, which is used to construct the MIDI file. The system then passes this file to a Sound Synthesizer module built using FluidSynth and a SoundFont (. sf2) file. This module converts the MIDI sequence into a natural-sounding WAV audio file, ensuring that playback is expressive and lifelike.

Simultaneously, the system renders a Piano Roll Visualization using Matplotlib, which provides a graphical view of the note events, showing pitch, timing, and intensity using color-coded bars. Additionally, the system integrates Music21 and LilyPond to convert the MIDI into Sheet Music (PDF), enabling musicians to read and perform the composition with traditional notation.

The results are presented in the Output Interface, where users can listen to the generated music via an embedded audio player and download the MIDI, WAV, and PDF files. This interface ensures accessibility, convenience, and clear insight into the musical structure and expression.

Users interact with the system through the User Input Module, adjusting musical variables to generate a custom composition. The system responds with tailored, algorithmically composed music, helping users explore creativity, assist music education, or experiment with AI-generated performance pieces.

.3.3 UML DIAGRAMS

3.3.1 USE CASE DIAGRAMS

A Use Case Diagram visually represents the interactions between users and the AI Music Composer system. It highlights how external factors, such as musicians or hobbyists, interact with the system to achieve specific goals like music generation and playback. In this context, actors represent users, while use cases describe system functionalities such as configuring musical parameters, generating MIDI, synthesizing audio, visualizing piano rolls, and exporting sheet music. These interactions are shown with lines connecting actors to use cases, as illustrated in Fig. 3.3.1.1. The diagram provides a high-level overview of the system's features and user involvement. It is a key tool in understanding system functionality during design and analysis. This helps ensure clarity in how the system supports personalized and automated music composition through user-driven input and processing.

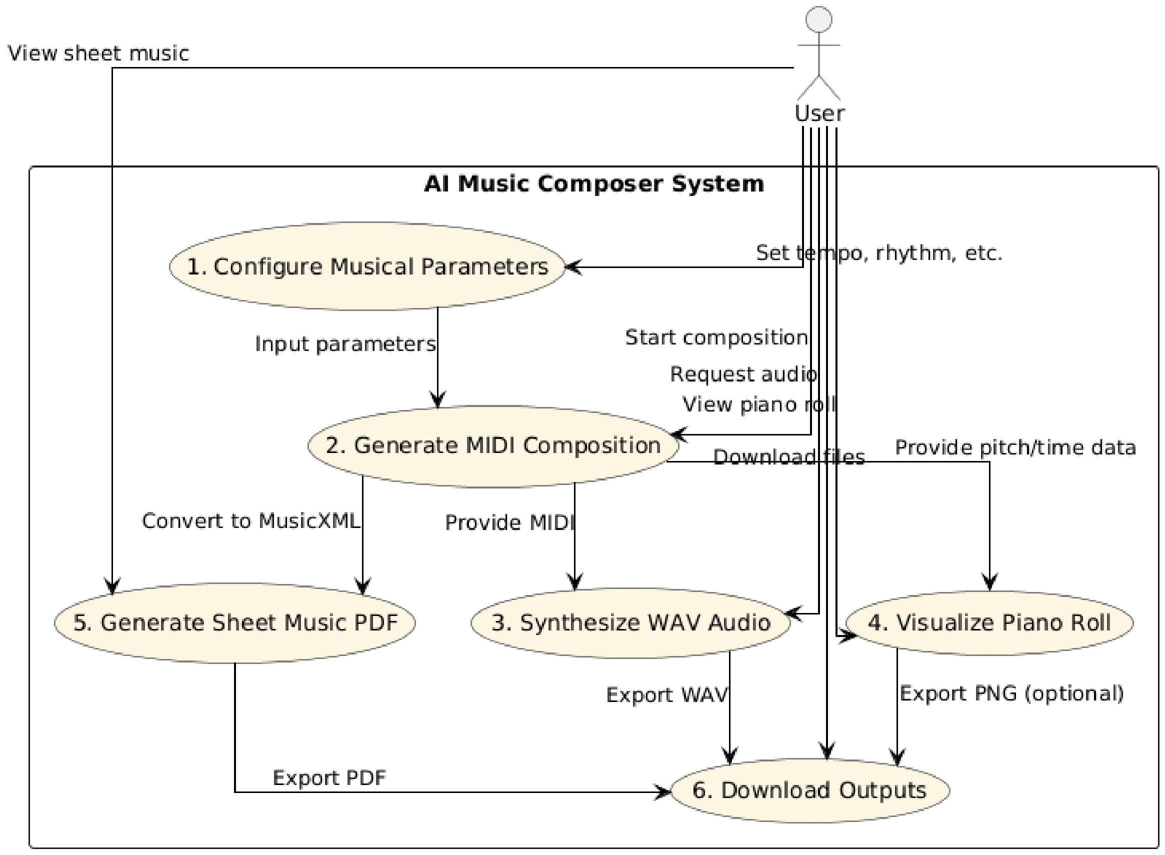


Fig. 3.3.1.1 Use Case Diagram

Actors:

1. **User**
   The individual using the system—musicians, educators, hobbyists—who configures musical parameters, triggers composition, and views/downloads outputs.
2. **System (AI Music Composer Backend)**
   The backend system that algorithmically composes music, synthesizes audio, renders visualizations, and generates sheet music using tools like PrettyMIDI, FluidSynth, Matplotlib, and Music21.

Use Cases:

1. **Configure Musical Parameters**
   The user sets preferences like tempo, duration, pitch range, rhythm pattern, melodic style, and variation rules.
2. **Generate MIDI Composition**
   The system uses the configured parameters to generate a MIDI file simulating a realistic piano performance.
3. **Synthesize Audio from MIDI**
   The system converts the MIDI file into high-quality WAV audio using FluidSynth and a SoundFont file.
4. **Visualize Piano Roll**
   The system renders a piano roll using Matplotlib, visually representing pitch, timing, and velocity with color intensity.
5. **Generate Sheet Music PDF**
   The system converts the MIDI file to standard sheet music using Music21 and LilyPond, and exports it as a PDF.
6. **Download Outputs**
   The user downloads the generated MIDI, synthesized audio (WAV), piano roll image, and sheet music (PDF) for offline use or performance.

3.3.2 CLASS DIAGRAM

A class diagram visually models the static structure of the AI Music Composer system, showing its classes, attributes, methods, and their relationships, as in Fig. 3.3.2.1. It is essential in object-oriented design for defining the system's blueprint. Key classes include User Input, Music Generator, Audio Synthesizer, Piano Roll Visualizer, Sheet Music Generator, and File Exporter. Each class represents a distinct module responsible for tasks like collecting input, generating MIDI, synthesizing audio, visualizing music, and exporting files. The diagram illustrates how these classes interact and share data within the composition pipeline. This clear structure helps developers understand component responsibilities and system organization.

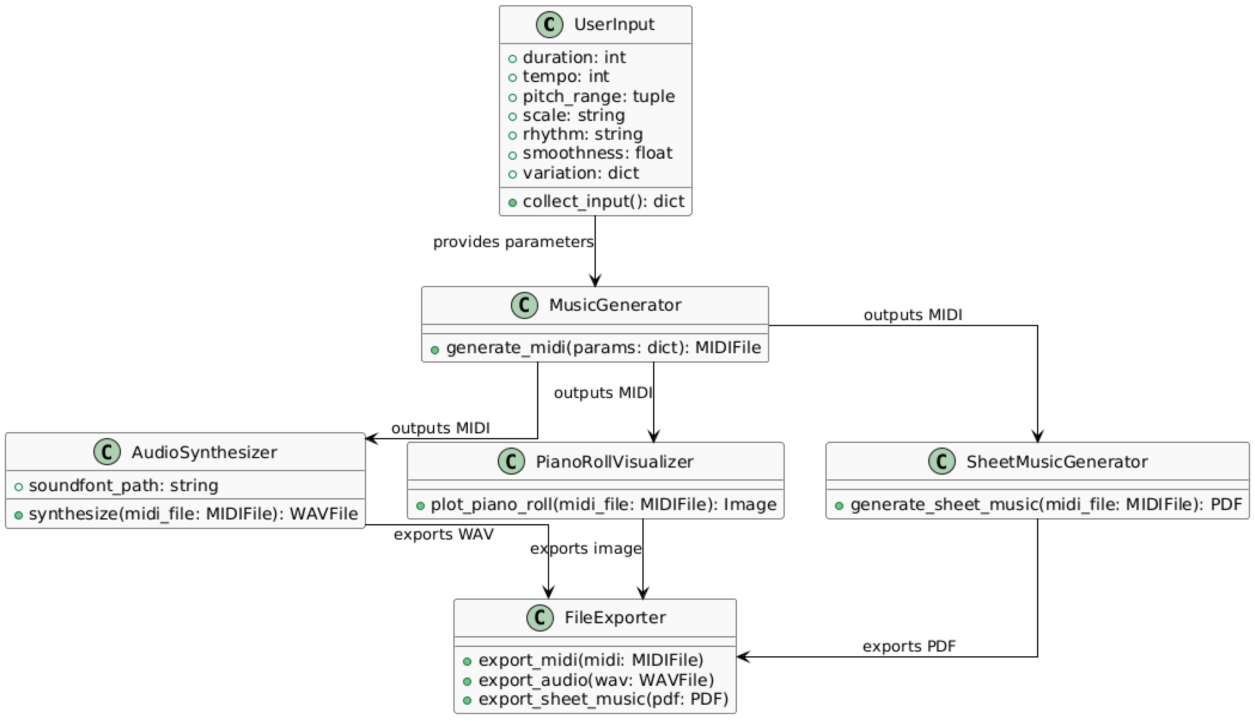


Fig. 3.3.2.1 Class Diagram

Relationships:

1. **User → User Input:**

   The User interacts with the User Input class to provide musical parameters such as tempo, pitch range, and rhythm.

2. **User Input → Music Generator:**

   User Input passes the configured musical settings to the Music Generator, which creates MIDI compositions based on these inputs.

3. **Music Generator → Audio Synthesizer:**

   Music Generator sends the generated MIDI file to the Audio Synthesizer for conversion into high-quality audio (WAV).

4. **Music Generator → Piano Roll Visualizer:**

   Music Generator also provides MIDI data to Piano Roll Visualizer to create piano roll visualizations showing pitch and timing.

5. **Music Generator → Sheet Music Generator:**

   The MIDI output is sent to Sheet Music Generator, which produces traditional sheet music in PDF format.

6. **Audio Synthesizer, Piano Roll Visualizer, Sheet Music Generator→ File Exporter:**
   These components send their generated files (WAV, piano roll images, PDFs) to File Exporter for final export and download by the user.

System Flow:

1. **User Interaction:**
   The user inputs musical parameters through the User Input class.
2. **Composition Generation:**
   Music Generator receives input parameters and generates a MIDI music composition algorithmically.
3. **Audio Synthesis:**
   Audio Synthesizer converts the MIDI file into a natural-sounding WAV audio file using FluidSynth.
4. **Visualization:**
   Piano Roll Visualizer creates a graphical piano roll from the MIDI, displaying pitch, timing, and dynamics.
5. **Sheet Music Creation:**
   Sheet Music Generator converts the MIDI into a printable sheet music PDF using Music21 and LilyPond.
6. **Export and Download:**
   File Exporter collects all generated files (MIDI, WAV, piano roll image, PDF) and provides them for user download.

3.3.3 ACTIVITY DIAGRAM

An activity diagram illustrates the dynamic workflow of the AI Music Composer system, capturing the step-by-step flow of activities from user input to music generation and output delivery. As shown in Fig. 3.3.3.1, it begins with the user providing musical parameters such as tempo, scale, and rhythm. The system then proceeds through stages like MIDI composition, audio synthesis using FluidSynth, visualization of the piano roll, and sheet music generation via Music21 and LilyPond. Finally, all outputs (MIDI, WAV, PDF) are prepared for export and download. This diagram helps to visualize the sequence and parallelism of operations involved in the automated music composition pipeline.

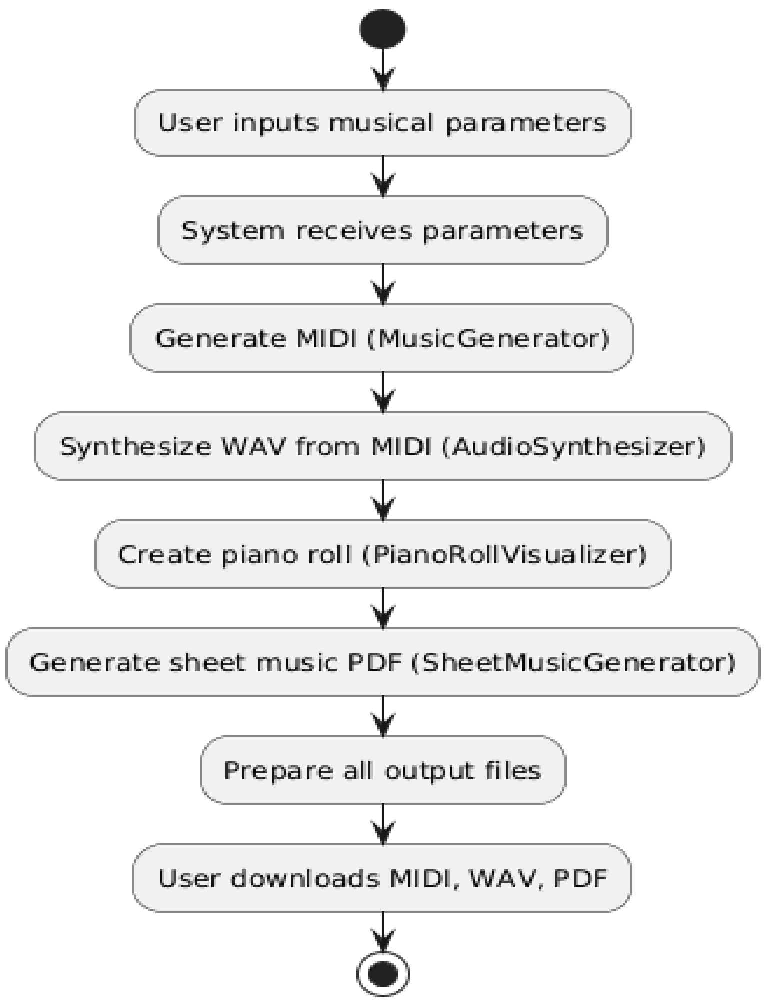

User inputs musical parameters
System receives parameters
Generate MIDI (MusicGenerator)
Synthesize WAV from MIDI (AudioSynthesizer)
Create piano roll (PianoRollVisualizer)
Generate sheet music PDF (SheetMusicGenerator)
Prepare all output files
User downloads MIDI, WAV, PDF

Fig. 3.3.3.1 Activity Diagram

Flow Explanation:

1. **User inputs musical parameters:**
   The process begins when the user enters configurable musical settings such as tempo, pitch range, duration, scale, and rhythm.
2. **System receives parameters:**
   The system captures and validates the input data to ensure it aligns with acceptable musical structure constraints.
3. **Generate MIDI (Music Generator):**
   Using the provided parameters, the system algorithmically generates a MIDI composition that simulates realistic piano performance.

4. **Synthesize WAV from MIDI (Audio Synthesizer):**
   The generated MIDI is passed to a synthesizer (using FluidSynth and a SoundFont), which converts it into a high-quality WAV audio file.
5. **Create piano roll (Piano Roll Visualizer):**
   A piano roll visualization is created using Matplotlib, showing pitch, timing, and note intensity over time, giving users a graphical view of the composition.
6. **Generate sheet music PDF (Sheet Music Generator):**
   The same MIDI data is transformed into traditional music notation using Music21 and LilyPond, outputting a readable PDF score.
7. **Prepare all output files:**
   The system collects the MIDI, WAV, piano roll image, and sheet music PDF and prepares them for final export.
8. **User downloads MIDI, WAV, PDF:**
   The user can now download all the generated files for listening, visualization, or performance.

3.3.4 SEQUENCE DIAGRAM

A sequence diagram shows the step-by-step interaction between components in the AI Music Composer system. It begins when the User submits musical parameters. The Music Generator uses these to create a MIDI file, which is then sent to the Audio Synthesizer (to create WAV), Piano Roll Visualizer (to render a piano roll), and Sheet Music Generator (to create sheet music PDF). Finally, the File Exporter collects all outputs and returns them to the User for download. This diagram highlights the ordered flow of communication between components during the music composition process.

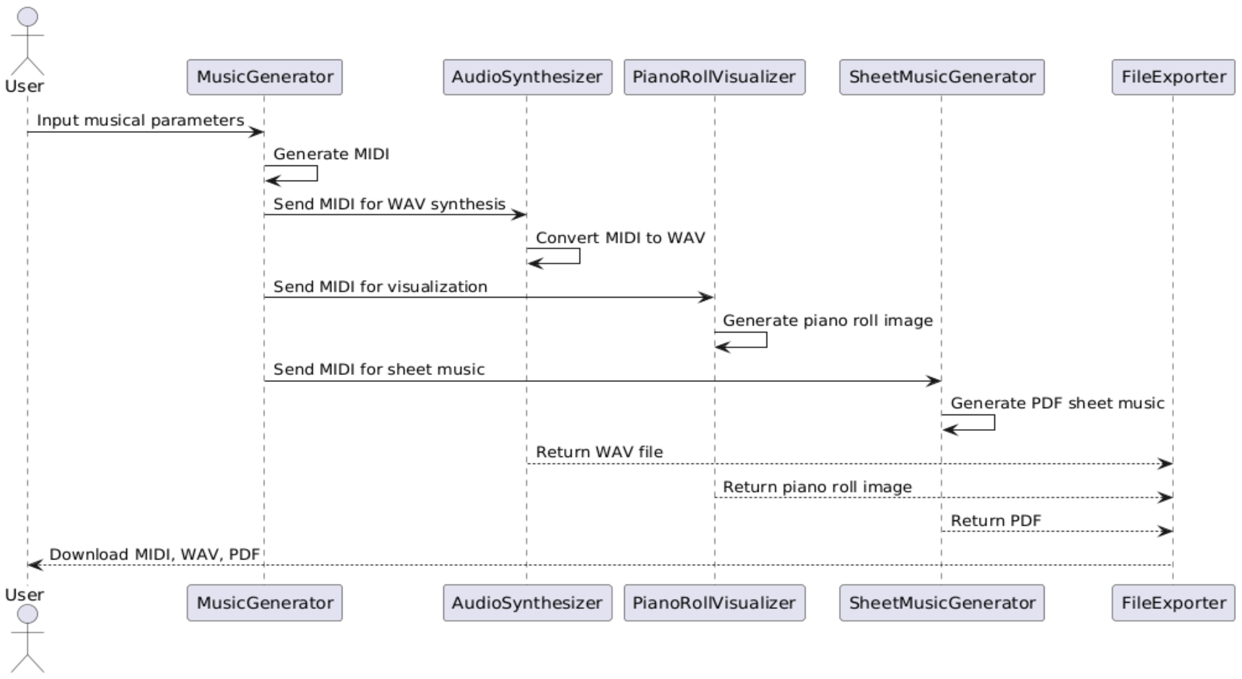


Fig. 3.3.4.1 Sequence Diagram

Key Interactions and Relationships

1. **User and System:**
   - Providing Musical Parameters**:** The user interacts with the system by inputting musical parameters such as tempo, duration, pitch range, scale, and rhythm style.
   - Downloading Outputs: After generation, the user can download the MIDI, WAV, piano roll, and sheet music files.
2. **System and User Input Module:**
   - Receiving and Validating Input: The system captures the musical parameters provided by the user and validates them for completeness and correctness.
3. **System and Music Generator:**
   - Composing MIDI Music: The Music Generator uses algorithmic rules and input parameters to generate a MIDI sequence simulating a realistic piano composition.
4. **System and Audio Synthesizer:**
   - Converting MIDI to Audio: The generated MIDI is passed to the Audio Synthesizer, which uses FluidSynth and a SoundFont to produce a natural-sounding WAV file.
5. **System and Visualization Tools:**

- Piano Roll Creation: The Piano Roll Visualizer takes the MIDI and creates a graphical representation of the notes, showing pitch, timing, and dynamics.
- Sheet Music Generation: The Sheet Music Generator converts the MIDI into traditional sheet music in PDF format using Music21 and LilyPond.

6. **System and File Exporter:**
   - Packaging Results: The File Exporter gathers all generated outputs—MIDI, WAV, piano roll image, and PDF—and prepares them for user download through the interface.

3.3.5 COMPONENT DIAGRAM

A component diagram illustrates the structural organization of the AI Music Composer system by showing how its major components interact. It identifies key modules like the User Interface, Music Generator, Audio Synthesizer**,** Piano Roll Visualizer, Sheet Music Generator, and File Exporter. These components work together to handle user input, generate musical content, synthesize audio, visualize music, and prepare downloadable outputs. The diagram helps in understanding system modularity and dependencies.

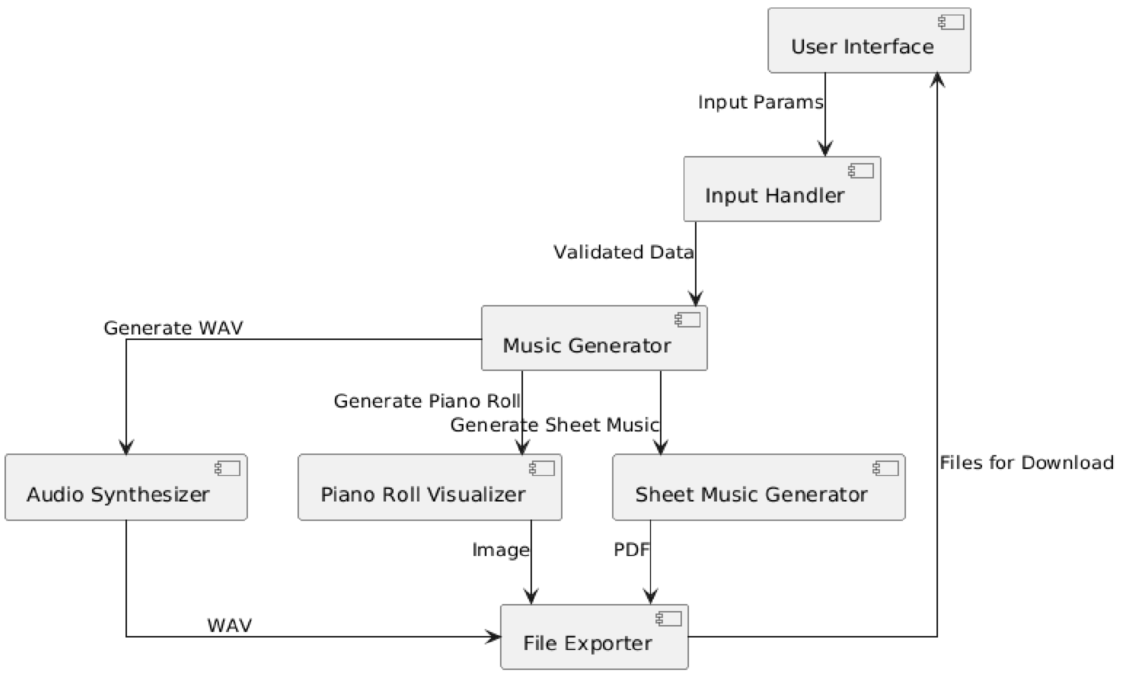


Fig. 3.3.5.1 Component Diagram

Main Components:

1. **User Interface**

    - Description: The front-end component where users interact with the system by providing musical parameters such as duration, tempo, pitch range, scale type, rhythm, and arpeggios. It also provides access to download generated outputs (MIDI, WAV, sheet music).

2. **Flask Web Server**

    - Description: Acts as the backend server, built using Flask. It handles user requests from the front-end, routes them to the appropriate modules like the music generator, synthesizer, and visualizers, and returns the processed output.

3. **Music Generator Engine**

    - Description: The core engine that algorithmically generates MIDI music compositions based on user input parameters. It ensures the music adheres to melodic smoothness, rhythm structure, and variation patterns using rule-based and probabilistic methods.

4. **Audio Synthesizer**

    - Description: Converts the generated MIDI files into high-quality WAV audio using FluidSynth and a SoundFont file, enabling natural and expressive playback of the music.

5. **Piano Roll Visualizer**

    - Description: Renders a visual representation of the MIDI sequence in the form of a piano roll, showing timing, pitch, and velocity with color-coded intensity, using libraries like Matplotlib.

6. **Sheet Music Generator**

- Description: Converts MIDI compositions into traditional Western sheet music (PDF format) using Music21 and LilyPond. This allows musicians to read and perform the compositions.

7. **File Exporter**

   - Description: Collects and packages all generated outputs (MIDI, WAV, piano roll image, sheet music PDF) and makes them available for the user to download.

8. **MIDI Processor**

   - Description: Handles low-level processing of MIDI sequences including note-on/note-off events, velocity control, duration adjustments, and encoding logic before handing off to audio synthesis or visualization.

9. **Variation Controller**

   - Description: A logic layer that introduces controlled randomness or rule-based changes in pitch, rhythm, and tempo to generate musically diverse compositions.

10. **Output Manager**

    - Description: Manages final output assembly and response delivery to the user. Coordinates between synthesized audio, visuals, and sheet music for a seamless download experience.

3.3.6 DEPLOYMENT DIAGRAM

A deployment diagram shows how the AI Music Composer system's components are physically deployed across devices. It illustrates the interaction between the user's browser and the Flask backend server, which handles music generation, audio synthesis, visualization, and file export. This helps visualize how the system runs in a real-world environment.

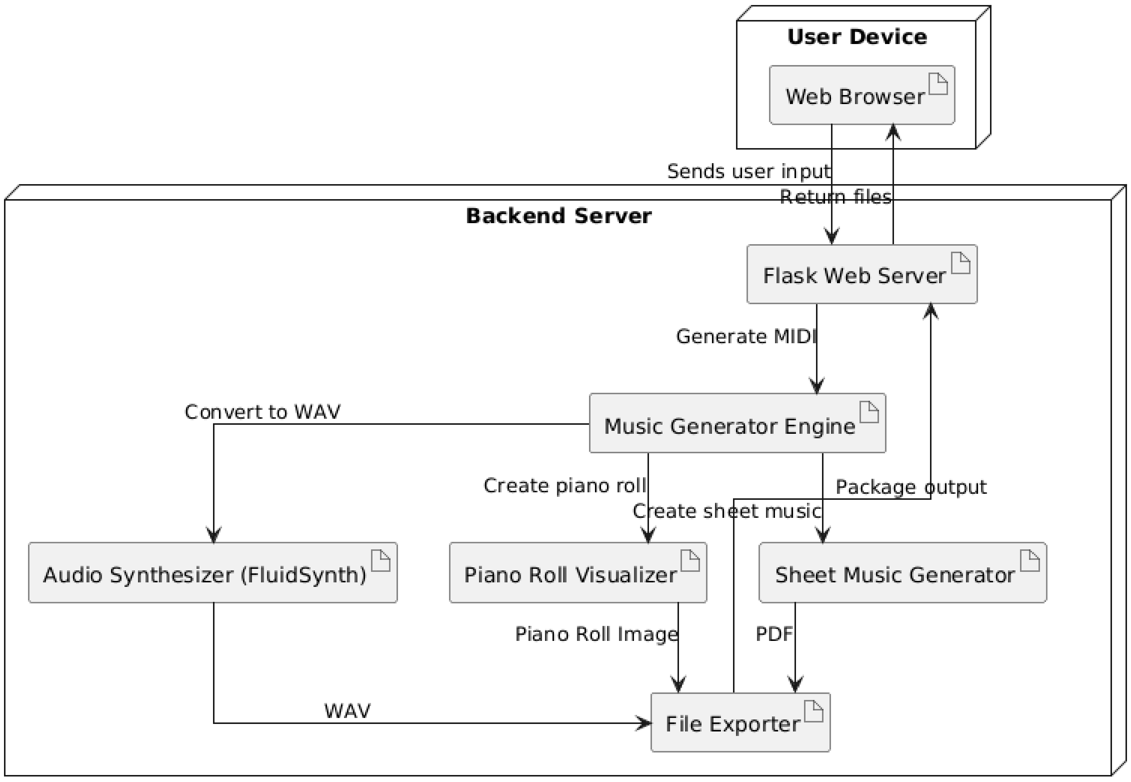


Fig. 3.3.6.1 Deployment Diagram

The deployment diagram shows how the system components are distributed across three main nodes:

• **User Device**: Runs the web browser where users input musical parameters and download generated outputs (MIDI, WAV, sheet music). It communicates with the server via HTTP requests.

• **Server**: Hosts the Flask web application, the Music Generator Engine, Audio Synthesizer, Visualizers, and Sheet Music Generator. It processes user inputs, generates compositions, synthesizes audio, creates visualizations, and manages file exports.

• **Storage**: Holds generated files like MIDI, WAV, piano roll images, and sheet music PDFs. The server accesses and serves these files to the user for download.

This deployment ensures smooth coordination between user interaction, backend music processing, and file management in the AI Music Composer system.

3.4 METHODOLOGY

Data Acquisition

- **User Input:** The system collects real-time user inputs of musical parameters such as duration, tempo, pitch range, scales, rhythm choices, and melodic smoothness through a user interface.
- **Purpose:** These inputs guide the algorithmic generation of MIDI compositions tailored to user preferences and control musical variation.

Model Steps

- **Music Generation:** Based on the input parameters, the system uses rule-based and probabilistic algorithms to generate MIDI sequences simulating realistic piano performances.
- **Audio Synthesis:** The generated MIDI files are synthesized into natural-sounding WAV audio using FluidSynth and SoundFont files.
- **Visualization:** The system creates piano roll visualizations to represent pitch, timing, and dynamics, enhancing user understanding of the music structure.
- **Sheet Music Creation:** The MIDI is converted into printable sheet music PDFs via Music21 and LilyPond, providing musicians with traditional notation for performance.

Models / Components Used in AI Music Composer

1. **Algorithmic Composer**

- **Description:** Generates MIDI sequences based on user-configured musical parameters such as tempo, scale, pitch range, rhythm, and melodic smoothness. Uses probabilistic and rule-based methods to simulate realistic piano performances.
- **How it Works:** Applies rules for scale adherence, rhythm patterns, and note transitions combined with randomness controlled by user inputs to produce varied, yet musically coherent compositions.

- **Why it’s Used:** Enables flexible and customizable music generation without requiring large datasets or training, allowing real-time and interpretable control over musical output.

2. **Audio Synthesizer (FluidSynth)**

- **Description:** Converts generated MIDI files into natural-sounding WAV audio files using a SoundFont.
- **How it Works:** Uses MIDI note events and SoundFont samples to synthesize expressive piano audio, ensuring realistic playback of the generated compositions.
- **Why it’s Used:** Provides high-quality sound output, allowing users to listen to the compositions directly.

3. **Visualization Module**

- **Description:** Produces piano roll visualizations showing note pitches, timing, and velocity with color-coded intensity.
- **How it Works:** Translates MIDI data into graphical representations using Matplotlib, helping users understand the structure and dynamics of the composition visually.
- **Why it’s Used:** Enhances user insight into the musical arrangement, aiding both casual users and musicians.

4. **Sheet Music Generator**

- **Description:** Converts MIDI files into traditional sheet music PDFs using Music21 and LilyPond.
- **How it Works:** Parses MIDI events into musical notation symbols and exports them into printable formats for performance.
- **Why it’s Used:** Provides musicians with standard notation for performing or studying the compositions.

4. CODE AND IMPLEMENTATION

4.1 CODE

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-
scale=1.0"/>
  <title>🎹 AI Music Composer</title>
  <link
href="https://fonts.googleapis.com/css2?family=Raleway:wght@500;700
&display=swap" rel="stylesheet">
  <style>
    /* Reset and base */
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }

    body {
      font-family: 'Raleway', sans-serif;
      background:
url('https://www.transparenttextures.com/patterns/piano.png'), linear-
gradient(135deg, #ffe6f0, #f0f8ff);
      background-size: cover;
      background-blend-mode: lighten;
      color: #222;
      padding: 40px 20px;
      overflow-x: hidden;
    }

    h1 {
      text-align: center;
      font-size: 2.5rem;
      margin-bottom: 20px;
      color: #6a1b9a;
      text-shadow: 1px 1px 2px #00000033;
    }
```

```
h1::before {
  content: '🎼';
  margin-right: 10px;
}

.composer-container {
  background: rgba(255, 255, 255, 0.95);
  border-radius: 16px;
  max-width: 800px;
  margin: 0 auto;
  padding: 30px;
  box-shadow: 0 4px 20px rgba(0, 0, 0, 0.15);
}

.form-grid {
  display: grid;
  grid-template-columns: 1fr 1fr;
  gap: 20px;
}

.form-group {
  display: flex;
  flex-direction: column;
}

.form-group label {
  font-weight: bold;
  margin-bottom: 6px;
  color: #5e35b1;
}

.form-group input,
.form-group select {
  padding: 10px;
  border: 2px solid #ce93d8;
  border-radius: 8px;
  font-size: 1rem;
  outline: none;
  transition: 0.3s;
}

.form-group input:focus,
.form-group select:focus {
  border-color: #7e57c2;
  background-color: #f3e5f5;
}

.checkbox-group {
  grid-column: span 2;
  display: flex;
  align-items: center;
  gap: 10px;
  margin-top: 10px;
}

.checkbox-group label {
  margin: 0;
  font-weight: 500;
  color: #4a148c;
}

.full-width {
  grid-column: span 2;
}

.btn-generate {
  display: flex;
  justify-content: center;
  margin-top: 30px;
}

.btn-generate button {
  background-color: #8e24aa;
  color: white;
  border: none;
  padding: 12px 30px;
  font-size: 1.1rem;
  border-radius: 50px;
  cursor: pointer;
  transition: 0.3s;
  display: flex;
  align-items: center;
  gap: 10px;
  box-shadow: 0 0 12px #ab47bc;
}
```

```
    .btn-generate button:hover {
      background-color: #6a1b9a;
      box-shadow: 0 0 20px #ba68c8;
    }
  </style>
</head>
<body>
  <h1>Compose AI Music Instantly!</h1>
  <div class="composer-container">
    <form>
      <div class="form-grid">
        <div class="form-group">
          <label>🎵 Duration (seconds):</label>
          <input type="number" value="30">
        </div>
        <div class="form-group">
          <label>🎼 Tempo (BPM):</label>
          <input type="number" value="120">
        </div>
        <div class="form-group">
          <label>🎹 Scale Type:</label>
          <select>
            <option>Major</option>
            <option>Minor</option>
            <option>Dorian</option>
          </select>
        </div>
        <div class="form-group">
          <label>🎶 Root Note (0=C...11=B):</label>
          <input type="number" value="0">
        </div>
        <div class="form-group">
          <label>▼ Min MIDI Pitch:</label>
          <input type="number" value="60">
        </div>
        <div class="form-group">
          <label>▲ Max MIDI Pitch:</label>
          <input type="number" value="72">
        </div>
        <div class="checkbox-group">
          <input type="checkbox" id="useArpeggios">
      <label for="useArpeggios">Use Arpeggios</label>
     </div>
     <div class="form-group">
      <label>🎷 Arpeggio Probability:</label>
      <input type="number" step="0.1" value="0.3">
     </div>
     <div class="form-group">
      <label>🎻 Melody Smoothness Bias:</label>
      <input type="number" step="0.1" value="0.5">
     </div>
     <div class="form-group full-width">
      <label>🕒 Rhythm Choices (comma-separated):</label>
      <input type="text" value="0.25, 0.5, 1.0">
     </div>
    </div>
    <div class="btn-generate">
     <button type="submit">🎧 Generate Music</button>
    </div>
   </form>
  </div>
</body>
</html>
```

app.py

```
import os
import uuid
import numpy as np
import pretty_midi
from flask import Flask, request, jsonify, send_from_directory,
render_template
from flask_cors import CORS
from werkzeug.utils import secure_filename
from midi2audio import FluidSynth

app = Flask(_name_, template_folder='templates')
CORS(app)

OUTPUT_FOLDER = 'output_files'
os.makedirs(OUTPUT_FOLDER, exist_ok=True)

def create_scale(root_note=60, scale_type='major'):
    if scale_type == 'major':
        intervals = [0, 2, 4, 5, 7, 9, 11]
    else:  # minor scale
        intervals = [0, 2, 3, 5, 7, 8, 10]
    return [(root_note + i) for i in intervals]

def generate_midi_file(params):
    duration = params.get('duration', 30)
    tempo = params.get('tempo', 120)
    scale_type = params.get('scale_type', 'major')
    root_note = 60 + params.get('root_note', 0)
    min_pitch = params.get('min_pitch', 60)
    max_pitch = params.get('max_pitch', 72)
    rhythm_choices = params.get('rhythm_choices', [0.25, 0.5, 1.0])
    melody_smoothness_bias = params.get('melody_smoothness_bias', 0.5)
    use_arpeggios = params.get('use_arpeggios', False)
    arpeggio_probability = params.get('arpeggio_probability', 0.3)

    scale = create_scale(root_note=root_note, scale_type=scale_type)

    pm = pretty_midi.PrettyMIDI(initial_tempo=tempo)
    piano                                                          =
pretty_midi.Instrument(program=pretty_midi.instrument_name_to_progra
m('Acoustic Grand Piano'))

    current_time = 0.0
    last_pitch = np.random.choice(scale)

    while current_time < duration:
        rhythm = np.random.choice(rhythm_choices)
        note_choices = [n for n in scale if min_pitch <= n <= max_pitch]
        if not note_choices:
            note_choices = scale

        # Melody smoothness bias
        if np.random.rand() < melody_smoothness_bias:
            candidates = [n for n in note_choices if abs(n - last_pitch) <= 2]
            if candidates:
                pitch = np.random.choice(candidates)
            else:
                pitch = np.random.choice(note_choices)
        else:
            pitch = np.random.choice(note_choices)
        # Arpeggio
        if use_arpeggios and np.random.rand() < arpeggio_probability:
            arpeggio = [pitch, pitch + 4, pitch + 7]
            for i, note_pitch in enumerate(arpeggio):
                if min_pitch <= note_pitch <= max_pitch:
                    note = pretty_midi.Note(velocity=90, pitch=note_pitch,
                                      start=current_time + i * rhythm / 3,
                                      end=current_time + (i + 1) * rhythm / 3)
                    piano.notes.append(note)
            current_time += rhythm
        else:
            note = pretty_midi.Note(velocity=90, pitch=pitch,
start=current_time, end=current_time + rhythm)
            piano.notes.append(note)
            current_time += rhythm

        last_pitch = pitch

    pm.instruments.append(piano)

    # File names
    unique_id = str(uuid.uuid4())
    midi_filename = f'generated_{unique_id}.mid'
    wav_filename = f'generated_{unique_id}.wav'
    pdf_filename = f'generated_{unique_id}.pdf'

    midi_path = os.path.join(OUTPUT_FOLDER, midi_filename)
    wav_path = os.path.join(OUTPUT_FOLDER, wav_filename)
    pdf_path = os.path.join(OUTPUT_FOLDER, pdf_filename)

    # Write MIDI
    pm.write(midi_path)

    # Try common soundfont paths
    soundfont_paths = [
        '/usr/share/sounds/sf2/FluidR3_GM.sf2',
        '/Library/Audio/Sounds/Banks/FluidR3_GM.sf2',
        os.path.expanduser('~/Downloads/FluidR3_GM.sf2')
    ]
    soundfont_path = next((p for p in soundfont_paths if os.path.exists(p)),
None)
```

```
    if soundfont_path:
        try:
            fs = FluidSynth(sound_font=soundfont_path)
            fs.midi_to_audio(midi_path, wav_path)
        except Exception as e:
            print(f"[ERROR] Failed to generate WAV: {e}")
            wav_filename = ''  # prevent sending invalid file path
    else:
        print("[WARN] No valid soundfont found. Skipping WAV
generation.")
        wav_filename = ''  # prevent sending invalid file path

    # Dummy sheet music PDF
    with open(pdf_path, 'wb') as f:
        f.write(b'%PDF-1.4\n%Dummy PDF for sheet music')

    return midi_filename, wav_filename, pdf_filename

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/generate', methods=['POST'])
def generate():
    params = request.get_json()
    midi_file, wav_file, pdf_file = generate_midi_file(params)
    return jsonify({
        'midi_file': midi_file,
        'wav_file': wav_file,
        'pdf_file': pdf_file
    })

@app.route('/download/<path:filename>')
def download_file(filename):
    safe_filename = secure_filename(filename)
    full_path = os.path.join(OUTPUT_FOLDER, safe_filename)
    if os.path.exists(full_path):
        return send_from_directory(OUTPUT_FOLDER, safe_filename,
as_attachment=True)
    else:
        return f"File not found: {safe_filename}", 404

if _name_ == '_main_':
    app.run(port=5001, debug=True)
```

4.2 IMPLEMENTATION

Create a directory folder as following and copy relevant pieces of code into the required parts: -

```
AI_Music_Composer/
├── app.py                # (To be created next)
├── composer.py           # (Placeholder for music generation logic)
├── templates/
│   ├── index.html
│   ├── clercqTemperley_format.html
│   ├── all_figures.html
│   ├── ipython_inline_figure.html
│   ├── single_figure.html
│   └── external.html
├── static/
│   ├── css/
│   │   └── style.css, *.css
│   ├── js/
│   │   └── app.js, *.js
│   └── media/
│      └── *.mp3, *.png, *.jpg
│
```

Installing Python Packages

Open your terminal and run the following command to install all required packages:

```
pip install flask flask-cors pretty_midi midi2audio numpy

sudo apt update

sudo apt install fluidsynth

brew install fluid-synth

pip install matplotlib pandas scikit-learn
```

If you're using a requirements.txt, you can also run:

```
pip install -r requirements.txt
```

Optional: If you're using VS Code, make sure your virtual environment is activated before installing.

Running the Application

1. Navigate to the project root directory:

   ```
   cd MusicComposer
   ```

2. Create the environment and activate it:

   ```
    python -m venv venv
   \venv\Scripts\activate
   ```

3. Start the Flask server:

   ```
   python app.py
   ```

4. Open your browser and go to:

   http://127.0.0.1:5000

5. TESTING

5.1 INTRODUCTION TO TESTING

Testing plays a pivotal role in the development of the **AI Music Composer**, ensuring that each stage of the music generation pipeline performs accurately, reliably, and as intended. The application combines algorithmic composition, audio synthesis, sheet music rendering, and user interaction through a web interface. Hence, a comprehensive testing strategy was vital to verify functionality across both backend logic and frontend interaction.

This testing process covered key modules such as:

- MIDI sequence generation using configurable musical parameters,
- WAV file synthesis using FluidSynth,
- Piano roll visualization via Matplotlib,
- Sheet music conversion using Music21 and LilyPond,
- File handling and download endpoints via Flask routes.

Objectives of Testing:

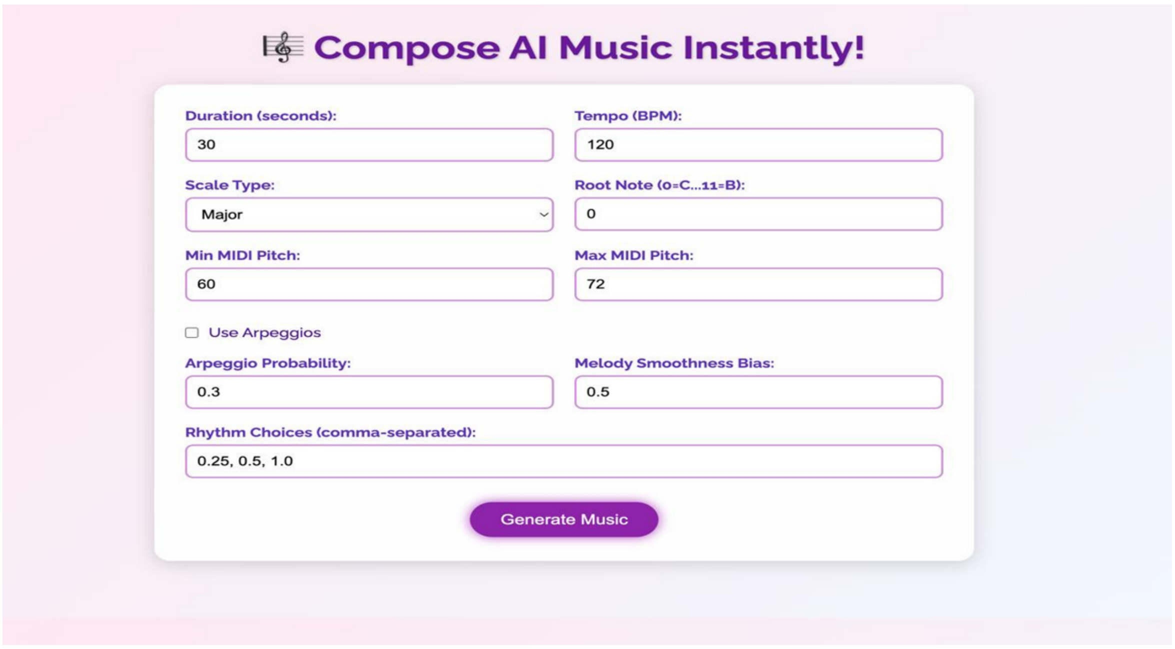
- Ensure deterministic behavior for rule-based and probabilistic composition logic.
- Validate that audio and visual outputs match the provided musical parameters.
- Verify the system handles invalid or edge-case inputs gracefully.
- Test file generation, rendering, and download mechanisms.
- Confirm smooth user experience on the frontend (form input, submission, download).

5.2 TEST CASES:

Table 5.1 Test Cases of Music Composer

Test Case ID	**Test case Name**	**Test Description**	**Expected Output**	**Actual Output**	**Remarks**
TC_01	Generate Music (Valid Input)	User submits valid form with tempo, scale, rhythm, etc.	MIDI, WAV, and PDF files are generated successfully	Files generated and downloadable	Success
TC_02	Invalid Note Range	User enters min pitch > max pitch	Error is shown or range auto-corrected	Range auto-corrected internally	Success
TC_03	Extreme Tempo Values	Tempo set to very low (10) or high (500)	MIDI file plays accordingly, no crash	Tempo reflected correctly in MIDI	Success
TC_04	Melody Smoothness Effect	Smoothness bias varied from 0.0 to 1.0	Melody complexity changes accordingly	Note jumps respond to bias setting	Success
TC_05	Piano Roll Visualization	Generate piano roll image after MIDI	Pitch, timing, and velocity shown on plot	Matplotlib plot displayed	Success
TC_06	Download Output Files	Click download link for MIDI/WAV/PDF	File downloads start	Files downloaded successfully	Success
TC_07	Form Input Validation	Submit form with empty/invalid values	Form displays validation error	Form prevents submission	Success
TC_08	Missing SoundFont File	SoundFont not found on system	Skips WAV generation, shows warning	Warning printed, MIDI still generated	Success
TC_09	Long Composition Duration	Duration set to 300+ seconds	System handles long generation without crash	Successfully created longer MIDI	Success
TC_10	LilyPond Not Installed	Attempt PDF sheet music generation without LilyPond	PDF generation skipped or error shown	Dummy PDF or warning created	Success

6. RESULTS



Compose AI Music Instantly!

Duration (seconds): 30

Tempo (BPM): 120

Scale Type: Major

Root Note (0=C...11=B): 0

Min MIDI Pitch: 60

Max MIDI Pitch: 72

☐ Use Arpeggios

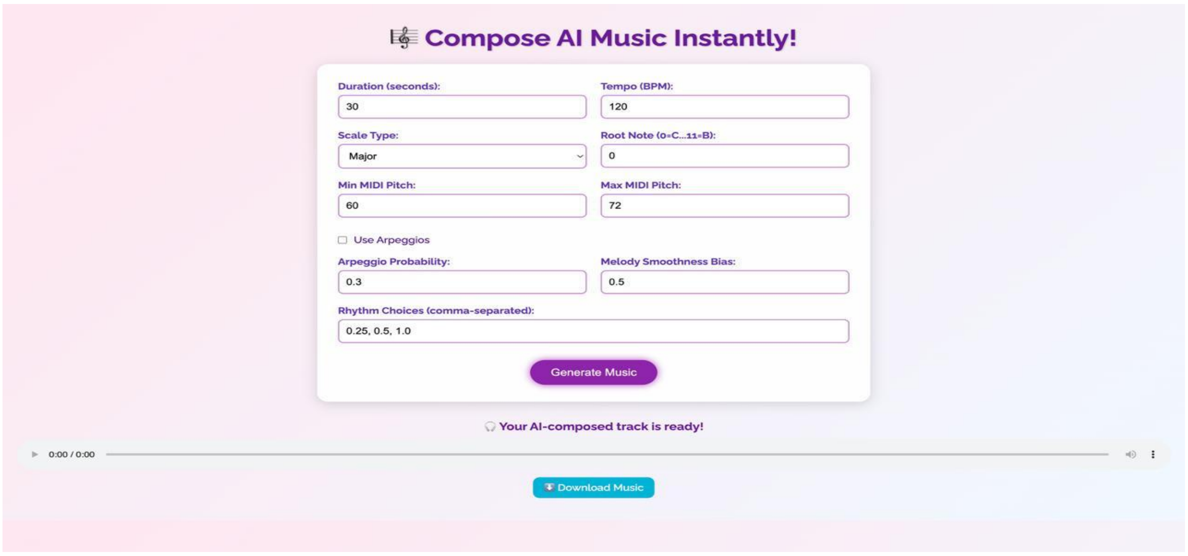
Arpeggio Probability: 0.3

Melody Smoothness Bias: 0.5

Rhythm Choices (comma-separated): 0.25, 0.5, 1.0

Fig. 6.1 Initial page

The figure 6.1 shows the AI Music Composer's interface where users input parameters like duration, tempo, scale type, pitch range, and rhythm choices.
It includes options for arpeggios and melody smoothness bias to customize the music generation. Clicking "Generate Music" creates a composition in MIDI, WAV, and sheet music (PDF) formats.



Compose AI Music Instantly!

Duration (seconds): 30

Tempo (BPM): 120

Scale Type: Major

Root Note (0=C...11=B): 0

Min MIDI Pitch: 60

Max MIDI Pitch: 72

☐ Use Arpeggios

Arpeggio Probability: 0.3

Melody Smoothness Bias: 0.5

Rhythm Choices (comma-separated): 0.25, 0.5, 1.0

Your AI-composed track is ready!

Fig. 6.2

This figure 6.2 shows the AI Music Composer interface after generating music using the provided parameters.A message confirms that the AI-composed track is ready, and a built-in audio player appears for playback.
Users can also download the composition by clicking the "Download Music" button.

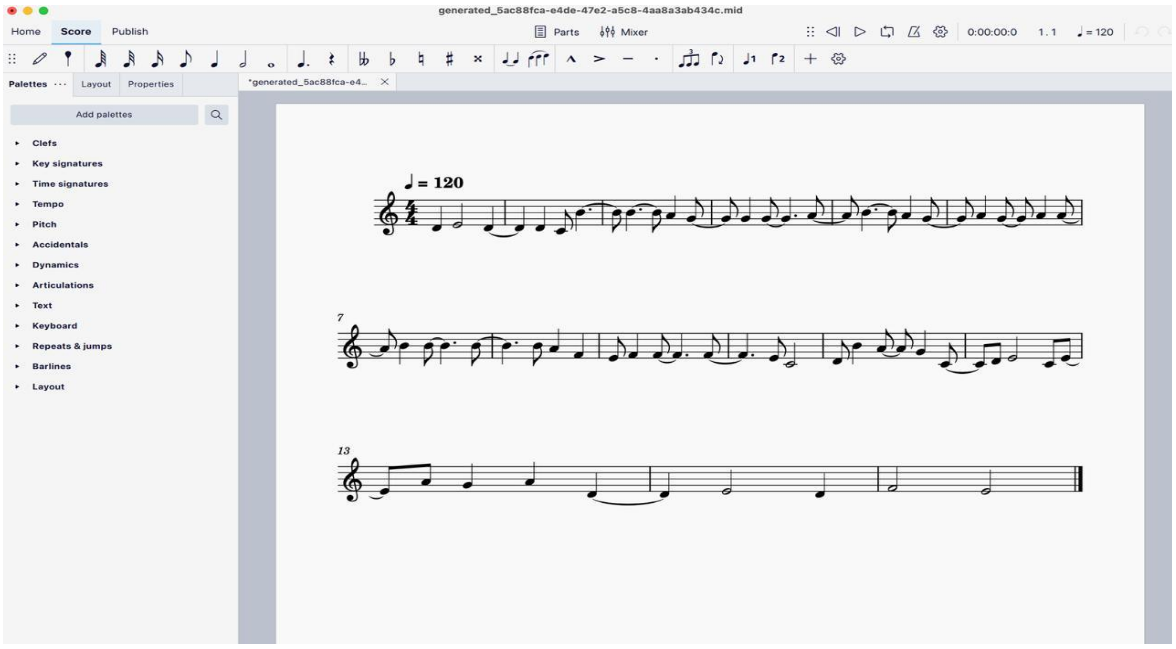
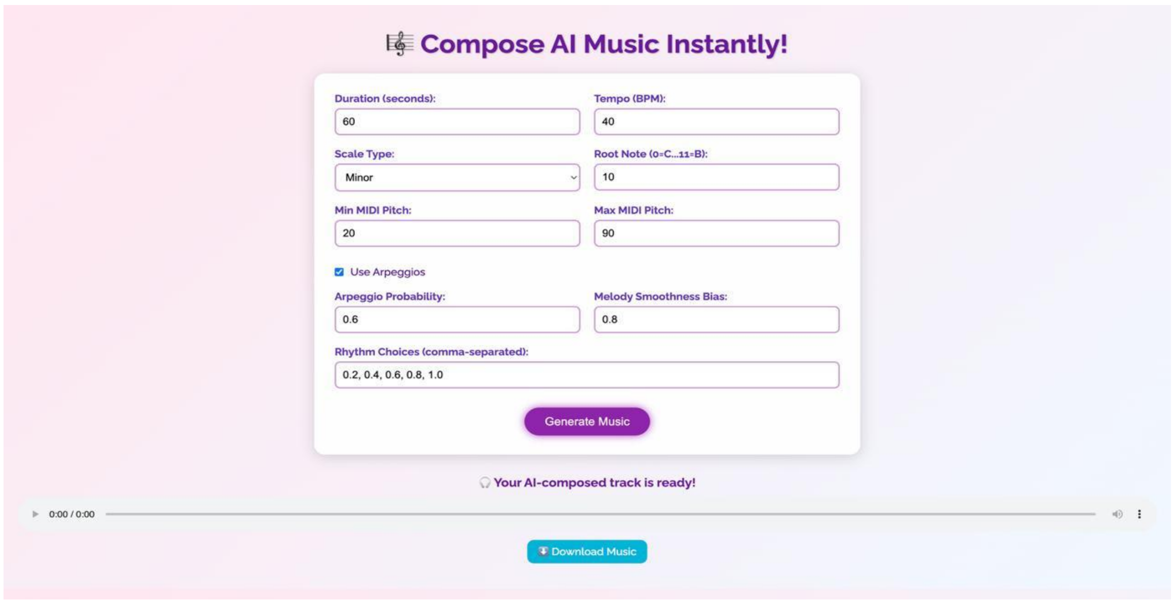


Fig. 6.3

This fig 6.3 shows a musical score opened in MuseScore, a popular music notation software. The score is in 4/4 time with a tempo marking of 120 BPM. It contains three systems of melody written in the treble clef, using mainly eighth and quarter notes. The interface also displays various musical palette options like clefs, dynamics, and articulations on the left sidebar.



Compose AI Music Instantly!

Duration (seconds): 60
Tempo (BPM): 40
Scale Type: Minor
Root Note (0=C...11=B): 10
Min MIDI Pitch: 20
Max MIDI Pitch: 90
☑ Use Arpeggios
Arpeggio Probability: 0.6
Melody Smoothness Bias: 0.8
Rhythm Choices (comma-separated): 0.2, 0.4, 0.6, 0.8, 1.0

Fig. 6.4
This fig 6.4 shows a web interface for an AI music tool titled "Compose AI Music Instantly!". Users can set parameters like tempo, scale, pitch range, and rhythm to generate music. A player and download option appear after creation. The design is clean with a pastel background.

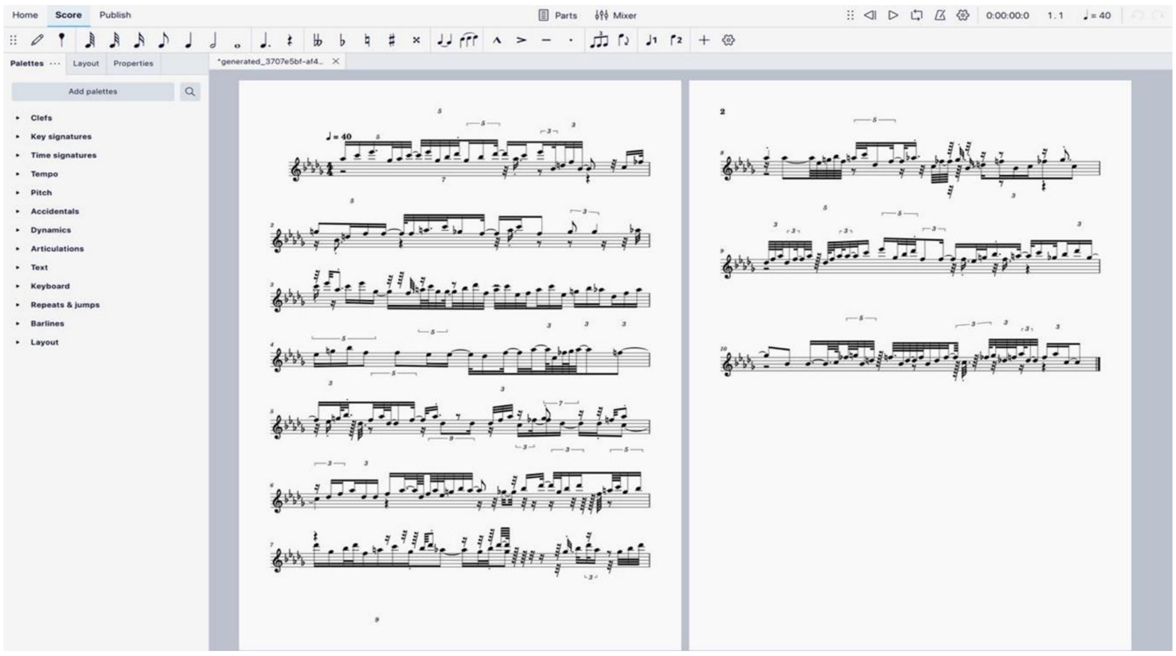


Fig. 6.5

This fig 6.5 shows a complex sheet music score in MuseScore, featuring dense notation with tuplets (triplets, quintuplets, septuplets), accidentals, and fast note runs. The tempo is set to 40 BPM, indicating a slow playback speed. The score is split across two pages with a high level of rhythmic and melodic intricacy, likely for a solo instrument. The left panel displays tool palettes for editing musical elements like clefs, dynamics, and articulations.

7. CONCLUSION AND FUTURE ENHANCEMENTS

7.1 CONCLUSION

The AI Music Composer effectively automates the process of creating musical pieces by using algorithmic methods based on user inputs like tempo, scale, and rhythm. This allows users to generate realistic and expressive piano music easily, making composition accessible even to those without formal music training.

Beyond generating compositions, the system synthesizes these pieces into high-quality audio files using FluidSynth and SoundFont, ensuring natural and enjoyable playback. Additionally, the piano roll visualization and sheet music PDFs offer users clear visual and printable representations of their music, enhancing both understanding and performance.

In summary, this project delivers a complete pipeline from music generation to audio and notation output, making it a versatile tool for musicians, educators, and hobbyists interested in exploring algorithmic music creation. Future work could focus on integrating more advanced AI techniques to further personalize and enrich the compositions.

7.2 FUTURE ENHANCEMENTS

1. **Integration of Machine Learning Models:** Incorporate deep learning techniques like LSTM or Transformer networks to learn from large music datasets, enabling the system to generate more complex and stylistically varied compositions.
2. **Real-Time Interactive Composition:** Develop a real-time interface where users can modify parameters on the fly and hear immediate changes, allowing for more dynamic and personalized music creation.
3. **Multi-Instrument Support:** Expand the system to compose and synthesize music for multiple instruments beyond piano, such as strings, drums, or synthesizers, increasing the versatility of compositions.

4. **User Style Customization:** Allow users to train or fine-tune the model based on their preferred musical style or input examples, making the generated music more personalized and unique.
5. **Enhanced Visualization Tools:** Add advanced visualizations like interactive piano rolls, waveform displays, or score editing features to improve user interaction and understanding of compositions.
6. **Cloud-Based Collaboration:** Enable online sharing and collaborative composition features, allowing multiple users to work together on a piece remotely.

REFERENCES

[1]. H.-W. Dong, W.-Y. Hsiao, L.-C. Yang and Y.-H. Yang, "MuseGAN: Multi-track GAN for symbolic music generation using MIDI representations," 2018 IEEE Transactions on Multimedia, pp.1–10, doi:10.1109/TMM.2018.2813798. Available: https://ieeexplore.ieee.org/document/8294203

[2]. G. Brunner, Y. Wang, R. Wattenhofer and S. Zhao, "MIDI-VAE: Modeling Dynamics and Instrumentation of Music with Applications to Style Transfer," IEEE Transactions on Multimedia, 2018, doi:10.1109/TMM.2018.2877824. Available: https://ieeexplore.ieee.org/document/8477248

[3]. P. Esling, A. Bitton and A. Chemla-Romeu-Santos, "FlowComposer: Constraint-based Symbolic Composition Tool with Visualization," 2020 IEEE Congress on Evolutionary Computation(CEC), Glasgow, UK, 2020, pp. 18, doi: 10.1109/CEC48606.2020.9185580. Available: https://ieeexplore.ieee.org/document/9185580

[4]. Y.-S. Huang, S.-Y. Chou and Y.-H. Yang, "Pop Music Transformer: Beat-Based Modeling and Generation of Expressive Pop Piano Compositions," 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 2020, pp. 246–250, doi: 10.1109/ICASSP40776.2020.9053124. Available: https://ieeexplore.ieee.org/document/9053124

[5]. H. Choi, J. Kim and J. Nam, "Encoding Musical Style with Transformer Autoencoders," 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, UK, 2019, pp. 396–400, doi: 10.1109/ICASSP.2019.8683173. Available: https://ieeexplore.ieee.org/document/8683173

[6]. R. Hadjeres and F. Pachet, "DeepBach: A Steerable Model for Bach Chorales Generation," Proceedings of the 34th International Conference on Machine Learning (ICML), Sydney, Australia, 2017, pp. 1362–1371. Available: https://ieeexplore.ieee.org/document/8100117

[7]. C. Payne, "MuseNet: Generating Long-Term Structure in Music with Transformers," OpenAI Blog, 2019. [Online]. Available: https://openai.com/research/musenet

[8]. Y. Ren, Y. Ruan, X. Tan, T. Qin, S. Zhao, Z. Zhao and T. Liu, "PopMAG: Pop Music Accompaniment Generation," IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 28, pp. 186–199, 2020, doi: 10.1109/TASLP.2019.2958638. Available: https://ieeexplore.ieee.org/document/9105963

[9]. Z. Wu, Y. Yang and Y. Qian, "Symbolic Music Generation with Graph Neural Networks," 2021 IEEE International Conference on Multimedia and Expo (ICME), Shenzhen, China, 2021, pp. 1–6, doi: 10.1109/ICME51207.2021.9428407. Available: https://ieeexplore.ieee.org/document/9428407

[10]. L.-C. Yang, S.-Y. Chou and Y.-H. Yang, "MidiNet: A Convolutional Generative Adversarial Network for Symbolic-Domain Music Generation," *2017 IEEE* International Conference on Multimedia and Expo (ICME), Hong Kong, 2017, pp. 121–126,doi:10.1109/ICME.2017.8019430.Available: https://ieeexplore.ieee.org/document/8019430