#### **INTRODUCTION TO DEEP LEARNING LABORATORY**

#### **REPORT ON**

#### **MUSIC GENERATION project**

#### A Dissertation submitted to JNTU Hyderabad in partial fulfillment of the academic requirements for the award of the degree.

**Bachelor of Technology**

**in**

**Computer Science and Engineering**

**(Artificial Intelligence & machine learning)**

Submitted by

**S.Jyotsna (22H51A7356)**

Under the esteemed guidance of

**Mrs. A. DEEPIKA**

(Assistant Professor)



**Department of Computer Science and Engineering (Artificial Intelligence & Machine learning)**

### CMR COLLEGE OF ENGINEERING & TECHNOLOGY

(UGC Autonomous)

\*Approved by AICTE \*Affiliated to JNTUH \*NAAC Accredited with A+ Grade

KANDLAKOYA, MEDCHAL ROAD, HYDERABAD - 501401.

### 2022- 2026

**CMR COLLEGE OF ENGINEERING & TECHNOLOGY**

KANDLAKOYA, MEDCHAL ROAD, HYDERABAD – 501401

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**(ARTIFICIAL INTELLIGENCE & MACHINE LEARNING)**



### CERTIFICATE

This is to certify that the Lab Project report entitled **"MUSIC GENERATION PROJECT"** being submitted M.Vennela (22H51A7343), N.Amitha (22H51A7347), S.Jyotsna (22H51A7356) **Bachelor of Technology in Computer Science and Engineering (Artificial Intelligence & Machine learning)** is a record of bonafide work carried out his/her under my guidance and supervision. The results embodied in this project report have not been submitted to any other University or Institute for the award of any Degree.

**Mrs. A. DEEPIKA Dr. K. KIRUBAKARAN**

**Assistant Professor Associate Professor and HOD**

**Dept. of CSE(AIML) Dept. of CSE(AIML)**

**External Examiner**

### ACKNOWLEDGEMENT

With great pleasure we want to take this opportunity to express my heartfelt gratitude to all the people who helped in making this project work a grand success.

We are grateful to **Mrs. A. DEEPIKA**, **Assistant Professor**, Department of Computer Science and Engineering (Art) for his valuable technical suggestions and guidance during the execution of this project work.

We would like to thank **Dr. K. Kirubakaran**, Head of the Department of Computer Science and Engineering (Artificial Intelligence & Machine learning), CMR College of Engineering and Technology, who is the major driving force to complete my project work successfully.

We are highly indebted to **Major Dr. V A Narayana,** Principal, CMR College of Engineering and Technology, for giving permission to carry out this project in a successful and fruitful way.

We would like to thank the **Teaching & Non- teaching** staff of Department of Computer Science and Engineering (**Artificial Intelligence & Machine learning**) for their co-operation

We express our sincere thanks to **Shri. Ch. Gopal Reddy**, Secretary, CMR Group of Institutions, for his continuous care.

Finally, we extend thanks to our parents who stood behind us at different stages of this Project. We sincerely acknowledge and thank all those who gave support directly and indirectly in completion of this project work.

**M.VENNELA**

**(22H51A7343)**

**N.AMITHA**

**(22H51A7347) S.JYOTSNA**

**(22H51A7356)**

|  |  |  |
| --- | --- | --- |
| **TABLE OF CONTENTS** | | |
| **CHAPTER**  **NO.** | **TITLE** | **PAGE NO.** |
|  |  |  |
|  |  | I |
| **1** | **INTRODUCTION** |  |
|  | 1.1 Problem Statement | 2 |
|  | 1.2 Objective | 2 |
|  | 1.3 Project Scope  1.4 Limitations | 2 - 3  3 - 4 |
| **2** | **PROPOSED SYSTEM** |  |
|  | 2.1. Advantages of Proposed System | 6 |
|  | 2.2 Implementation   * 1. Design   2. Code and output | 6  7  8-13 |
| **3** | **RESULTS AND DISCUSSION** |  |
|  | 3.1 Result and Discussion | 15 |
| **4** | **CONCLUSION**  4.1 Conclusion and Future Enhancement |  |
|  | 17 |
| **5** | **REFERENCES** | 18 |

# CHAPTER 1

# INTRODUCTION

**ABSTRACT**

This project focuses on creating a music generation system using deep learning techniques. The goal is to develop a program that can compose original music in various styles, allowing users to experience new musical ideas.

By training neural networks on a large collection of existing songs, the system learns the patterns and structures of music, such as melody, harmony, and rhythm. We use advanced models like recurrent neural networks (RNNs) and transformers to generate music that sounds coherent and engaging.

The project aims to provide musicians and music enthusiasts with a creative tool that inspires new compositions. Future improvements will include features for real-time music generation and integration with music production software. Ultimately, this project seeks to merge technology with creativity, opening new avenues for musical expression.

### CHAPTER 1 INTRODUCTION

**1.1. Problem Statement**

The goal of this project is to develop a deep learning model that can generate original music compositions. The model should learn patterns and structures from a dataset of existing music and use this knowledge to create new melodies, harmonies, or full compositions that are coherent and musically pleasing. The challenge is to ensure the generated music is not only novel but also stylistically similar to the training data while avoiding repetitive or unstructured output."

This simplifies the main goals and challenges of the project:

1.learn from existing music data.

2.Generate new, coherent music.

3.Balance between novelty and familiarity.

4.Avoid producing repetitive or random noise.

**1.2.Research Objective**

The objective of a deep learning music generation project is to create a model that can autonomously compose music or assist in music creation by learning from existing compositions. The key goals include:

1.Learning Musical Patterns: Understanding melody, harmony, rhythm, and structure.

2.Music Generation: Producing original melodies, harmonies, or multi-instrument arrangements.

3.sequence prediction: predicting the note or chord based on prior ones.

4.Conditional Generation: Creating music based on genre, mood, or user inputs.

5.Style Transfer: Adapting music to different styles.

6.Evaluation: Measuring coherence, creativity, and musical quality.

**1.3. Scope**

**1.** **Data Collection and Preprocessing**:

Music Dataset: Gather a diverse set of musical compositions (e.g., MIDI files) across different genres and styles.

Preprocessing: Convert music into a machine-readable format (e.g., sequences of notes, chords, or spectrograms).

**2.** **Model Development**:

Model Type: Choose and implement suitable deep learning models like RNNs, LSTMs, Transformers, or GANs.  
Training: Train the model to generate sequences of notes, chords, or audio.

Optimization: Fine-tune the model for better creativity, coherence, and performance.

**3.** **Music Generation Features**:

Autonomous Generation: The model can generate full compositions without human input.

Conditional Generation: Allow users to influence output based on genre, mood, or initial musical themes.

Multi-Instrument Music: Generate music for multiple instruments or ensembles.

**4. Output Formats**:

Symbolic Music: Generate MIDI or notation files representing music.

Raw Audio: Advanced models may generate raw waveforms for direct audio output.

**5. User Interaction and Interface**:

Interactive Composition: Enable users to input melodies or control specific parameters.

Real-Time Feedback: Provide users with the ability to hear results in real-time.

**6. Evaluation and Testing:**

Quality Assessment: Test the generated music for creativity, coherence, and alignment with the given style or inputs.

Human Evaluation: Gather feedback from musicians or listeners to evaluate musicality and satisfaction.

**7. Extensions and Future Work**:

Explore style transfer, improvisation, or adaptive music creation for gaming, film scoring, or virtual performers.

The scope defines the project’s objectives, model capabilities, user interface, and output format, laying the groundwork for further research or development.

**1.4.Limitation:**

**1. Lack of Creativity and Originality**:

Deep learning models tend to replicate patterns from the training data, making it difficult to produce highly creative or novel compositions. The generated music may feel repetitive or lack the emotional depth of human-composed music.

**2. Difficulty Capturing Long-Term Dependencies**:

Music often requires long-term coherence, such as thematic development or structure across several minutes. Even with models like LSTMs and Transformers, capturing these long-term dependencies in a meaningful way remains challenging.

**3. Contextual Understanding**:

Music generation models struggle with understanding the broader context of composition, such as mood, cultural significance, or narrative, leading to music that may lack depth or coherence.

**4. Limited Musical Structure**:

While models can generate short sequences of notes or chords, creating larger, well-structured forms like sonatas, symphonies, or pop song structures is more difficult due to the complexity of musical structure and transitions.

**5. Data Quality and Availability**:

High-quality datasets, especially multi-instrument music, may be limited. Models often rely on MIDI files, which don't capture the nuance of human performance (e.g., dynamics, expression).

**6. Overfitting to Training Data**:

There’s a risk of overfitting, where the model generates music that is too similar to the training data, limiting its ability to create diverse or unexpected compositions.

**7. Evaluation Challenges**:

Assessing the quality of generated music is subjective and difficult to quantify. Current metrics struggle to fully capture musicality, creativity, or emotional impact.

**8. Computational Resources**:

Training deep learning models, especially for raw audio generation (like WaveNet), requires significant computational resources and time.

**9. Lack of Human-Like Expressiveness**:

Models generally produce music that lacks the expressiveness and subtle timing variations that human performers bring to a composition, leading to music that can sound mechanical or rigid.

**10. Complexity of Real-Time Interaction**:

Implementing real-time interaction with a music generation system (e.g., for live improvisation or collaboration) is technically complex and may not yet be fully responsive or practical.

These limitations highlight the gap between machine-generated and human-composed music, especially in terms of creativity, structure, and expressiveness.

# CHAPTER 2

## PROPOSED

## SYSTEM

### CHAPTER 2

**PROPOSED SOLUTION**

**CHAPTER 2**

**2.1 Advantages of proposed System**

* **Creativity**: They create unique, novel music by blending styles and breaking conventional music rules.
* **Efficiency**: They generate music quickly, automating tasks like melody or chord creation.
* **Personalization**: Music can be tailored to user preferences or adapt in real-time (e.g., in video games).
* **Learning from data**: Models analyze vast music datasets, enabling innovation across genres.
* **Accessibility**: They make music creation easier for non-experts and are cost-effective.
* **Collaboration**: They assist human composers, providing creative ideas or full compositions.

**2.2 Implementation:**

**1.Project Setup**

**Choose a framework:** Popular deep learning frameworks like TensorFlow, Keras, or PyTorch are commonly used.

**Programming language:** Python is the most widely used for deep learning due to its libraries and ecosystem.

**2. Data Collection**

**Get MIDI files:** Music is often represented in MIDI (Musical Instrument Digital Interface) format, which captures musical notes, durations, and instruments in a structured way.

**Available datasets:** You can use datasets like Lakh MIDI Dataset, Classical Archives, or Magenta’s open MIDI dataset.

**Preprocessing:** Convert the MIDI data into a format suitable for the model (e.g., sequences of note values, durations, and other features like velocity).

**3. Data Preprocessing**

**Tokenization:** Convert musical notes, rhythms, or chords into sequences of tokens (like word tokenization in NLP). You might represent each note or chord as a separate token.

**Sequence generation:** Create input-output pairs for training by sliding a window over the sequence of notes. For example, predict the next note based on a sequence of preceding notes.

**Encoding:** One-hot encoding or embedding layers can be used to represent each note/chord.

**Normalizing:** Normalize time signatures, tempos, or other features that affect the musical flow.

**4. Model Architecture**

**Recurrent Neural Networks (RNNs):** Because music has sequential dependencies, models like Long Short-Term Memory (LSTM) or Gated Recurrent Units (GRU) are widely used.

**Transformer models:** More recent models like Transformers (used in language models) can be adapted for music generation because they handle long-range dependencies well.

**Variational Autoencoders (VAEs) or GANs:** VAEs and GANs can be used to generate music by learning latent representations of musical patterns.

**Example architecture (LSTM-based):**

* **Input:** Sequence of notes/chords (encoded).
* **Layers:**
* **Embedding layer (optional): To** convert input notes/chords into a dense representation.
* **LSTM layers:** For learning the temporal sequence in the data.

**Dense layer:** To predict the next note or chord.

* **Softmax layer:** Output probability distribution over all possible notes.

**5. Training the Model**

**Loss function:** Use categorical cross-entropy to measure the difference between predicted and actual notes.

**Optimizer:** Choose an optimizer like Adam or SGD for training.

**Batch size & epochs:** Tune hyperparameters like batch size, number of epochs, and learning rate for optimal training.

**Data augmentation:** Augment the dataset by transposing notes to different keys to create more training data.

**6. Generating Music**

After training, use the model to generate music by giving it a seed sequence (a few starting notes or chords). The model will predict the next note, which is then fed back into the model to predict the next note, continuing the process. You can apply temperature sampling to control the diversity of the generated music (lower temperature = more predictable, higher temperature = more random).

**7. Post-Processing**

Convert the generated sequence of notes back into MIDI format. Use libraries like mido or pretty midi to synthesize MIDI files. Optionally, use DAWs (Digital Audio Workstations) like FL Studio or Ableton to improve the sound quality or apply additional effects.

**8. Evaluation**

**Subjective evaluation:** Listen to the generated music to assess its quality.

**Quantitative evaluation:** Use metrics like pitch distribution, note duration distribution, and transition probabilities to compare the generated music with real music.

**9. Enhancements**

**Polyphony:** Handle polyphonic music (multiple notes at once) by extending your tokenization to capture chords or multiple instruments.

**Conditional generation:** Allow the model to generate music based on style, mood, or a given melody.

**Interactive systems:** Build an interactive app where users can input melodies or chords, and the model generates the rest.

**Example Project Libraries:**

* **Magenta (by Google):** A framework for music and art generation using deep learning.
* **PrettyMIDI:** For processing MIDI files.
* **Music21:** For music theory and symbolic music analysis.
* **Mido:** To work with MIDI files.

**2.3 DESIGN:**

Designing a music generation system involves several key steps:

1. **Requirements Gathering:** Define the user needs, music styles (e.g., classical, pop), and system objectives, such as generating background music or melodies with emotional undertones.

2. **Data Collection and Integration:** Gather diverse music data, such as MIDI files, audio samples, and sheet music. Integrate data from sources like public music datasets and licensed compositions.

3. **Model Selection:** Choose appropriate algorithms, such as recurrent neural networks (RNNs), transformers, or variational autoencoders (VAEs), based on the desired complexity of music generation.

4. **Feature Engineering:** Extract relevant features like tempo, pitch, harmony, and rhythm from the input data to help the model understand music structure and patterns.

5. **Model Training and Evaluation:** Train the model on historical musical compositions and evaluate its performance using metrics like creativity, harmony, and musicality, often judged by human experts or through quantitative measures like perplexity.

6. **Deployment:** Integrate the music generation model into an interactive platform (e.g., a music app or DAW) for real-time music creation, ensuring scalability and responsiveness for users.

7. **Monitoring and Maintenance**: Continuously monitor the quality of the generated music, update the model as needed to enhance creativity, and ensure that the generated outputs respect licensing and ethical standards regarding music originality and ownership.

**2.4 Code:**

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Activation, Embedding

from tensorflow.keras.optimizers import Adam

import pretty\_midi

# Helper function to read MIDI files and convert them into note sequences

def midi\_to\_note\_sequence(midi\_file):

midi\_data = pretty\_midi.PrettyMIDI(midi\_file)

notes = []

for instrument in midi\_data.instruments:

if not instrument.is\_drum:

for note in instrument.notes:

notes.append((note.start, note.pitch, note.end))

return sorted(notes)

# Tokenize the notes into sequences

def tokenize\_notes(notes, seq\_length):

pitches = [note[1] for note in notes]

unique\_pitches = sorted(list(set(pitches)))

pitch\_to\_int = {pitch: i for i, pitch in enumerate(unique\_pitches)}

sequences = []

targets = []

for i in range(0, len(pitches) - seq\_length):

seq\_in = pitches[i:i + seq\_length]

seq\_out = pitches[i + seq\_length]

sequences.append([pitch\_to\_int[note] for note in seq\_in])

targets.append(pitch\_to\_int[seq\_out])

return np.array(sequences), np.array(targets), len(unique\_pitches), pitch\_to\_int

# Build the model architecture

def build\_model(seq\_length, num\_pitches):

model = Sequential()

model.add(Embedding(num\_pitches, 100, input\_length=seq\_length))

model.add(LSTM(256, return\_sequences=True))

model.add(LSTM(256))

model.add(Dense(num\_pitches))

model.add(Activation('softmax'))

optimizer = Adam(learning\_rate=0.001)

model.compile(loss='sparse\_categorical\_crossentropy', optimizer=optimizer)

return model

# Generate music using the trained model

def generate\_music(model, seed\_sequence, num\_notes, pitch\_to\_int, int\_to\_pitch):

generated\_sequence = list(seed\_sequence)

for \_ in range(num\_notes):

input\_seq = np.reshape(generated\_sequence[-len(seed\_sequence):], (1, len(seed\_sequence)))

prediction = model.predict(input\_seq, verbose=0)

next\_note = np.argmax(prediction)

generated\_sequence.append(next\_note)

return [int\_to\_pitch[i] for i in generated\_sequence]

# Save the generated sequence to a MIDI file and print the saved location

def sequence\_to\_midi(sequence, output\_file):

midi = pretty\_midi.PrettyMIDI()

instrument = pretty\_midi.Instrument(program=0)

start\_time = 0

for pitch in sequence:

note = pretty\_midi.Note(velocity=100, pitch=pitch, start=start\_time, end=start\_time + 0.5)

instrument.notes.append(note)

start\_time += 0.5

midi.instruments.append(instrument)

midi.write(output\_file)

# Print confirmation message to ensure the file was saved

print(f'MIDI file saved at: {output\_file}')

# Main function to run the project

def main():

midi\_file = 'C:/Users/jyots/Downloads/d-callme.mid' # Replace with your MIDI file path

notes = midi\_to\_note\_sequence(midi\_file)

seq\_length = 20

sequences, targets, num\_pitches, pitch\_to\_int = tokenize\_notes(notes, seq\_length)

int\_to\_pitch = {i: pitch for pitch, i in pitch\_to\_int.items()}

model = build\_model(seq\_length, num\_pitches)

model.fit(sequences, targets, epochs=10, batch\_size=64)

seed\_sequence = sequences[0]

generated\_notes = generate\_music(model, seed\_sequence, 100, pitch\_to\_int, int\_to\_pitch)

# Specify where you want the file to be saved

output\_file = 'C:/Users/jyots/Downloads/generated\_music.mid' # Specify path to save generated music

sequence\_to\_midi(generated\_notes, output\_file)

if \_\_name\_\_ == '\_\_main\_\_':

main()

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import LSTM, Dense, Activation, Embedding

from tensorflow.keras.optimizers import Adam

import pretty\_midi

# Helper function to read MIDI files and convert them into note sequences

def midi\_to\_note\_sequence(midi\_file):

midi\_data = pretty\_midi.PrettyMIDI(midi\_file)

notes = []

for instrument in midi\_data.instruments:

if not instrument.is\_drum:

for note in instrument.notes:

notes.append((note.start, note.pitch, note.end))

return sorted(notes)

# Tokenize the notes into sequences

def tokenize\_notes(notes, seq\_length):

pitches = [note[1] for note in notes]

unique\_pitches = sorted(list(set(pitches)))

pitch\_to\_int = {pitch: i for i, pitch in enumerate(unique\_pitches)}

sequences = []

targets = []

for i in range(0, len(pitches) - seq\_length):

seq\_in = pitches[i:i + seq\_length]

seq\_out = pitches[i + seq\_length]

sequences.append([pitch\_to\_int[note] for note in seq\_in])

targets.append(pitch\_to\_int[seq\_out])

return np.array(sequences), np.array(targets), len(unique\_pitches), pitch\_to\_int

# Build the model architecture

def build\_model(seq\_length, num\_pitches):

model = Sequential()

model.add(Embedding(num\_pitches, 100, input\_length=seq\_length))

model.add(LSTM(256, return\_sequences=True))

model.add(LSTM(256))

model.add(Dense(num\_pitches))

model.add(Activation('softmax'))

optimizer = Adam(learning\_rate=0.001)

model.compile(loss='sparse\_categorical\_crossentropy', optimizer=optimizer)

return model

# Generate music using the trained model

def generate\_music(model, seed\_sequence, num\_notes, pitch\_to\_int, int\_to\_pitch):

generated\_sequence = list(seed\_sequence)

for \_ in range(num\_notes):

input\_seq = np.reshape(generated\_sequence[-len(seed\_sequence):], (1, len(seed\_sequence)))

prediction = model.predict(input\_seq, verbose=0)

next\_note = np.argmax(prediction)

generated\_sequence.append(next\_note)

return [int\_to\_pitch[i] for i in generated\_sequence]

# Save the generated sequence to a MIDI file and print the saved location

def sequence\_to\_midi(sequence, output\_file):

midi = pretty\_midi.PrettyMIDI()

instrument = pretty\_midi.Instrument(program=0)

start\_time = 0

for pitch in sequence:

note = pretty\_midi.Note(velocity=100, pitch=pitch, start=start\_time, end=start\_time + 0.5)

instrument.notes.append(note)

start\_time += 0.5

midi.instruments.append(instrument)

midi.write(output\_file)

# Print confirmation message to ensure the file was saved

print(f'MIDI file saved at: {output\_file}')

# Main function to run the project

def main():

midi\_file = 'C:/Users/jyots/Downloads/d-callme.mid' # Replace with your MIDI file path

notes = midi\_to\_note\_sequence(midi\_file)

seq\_length = 20

sequences, targets, num\_pitches, pitch\_to\_int = tokenize\_notes(notes, seq\_length)

int\_to\_pitch = {i: pitch for pitch, i in pitch\_to\_int.items()}

model = build\_model(seq\_length, num\_pitches)

model.fit(sequences, targets, epochs=10, batch\_size=64)

seed\_sequence = sequences[0]

generated\_notes = generate\_music(model, seed\_sequence, 100, pitch\_to\_int, int\_to\_pitch)

# Specify where you want the file to be saved

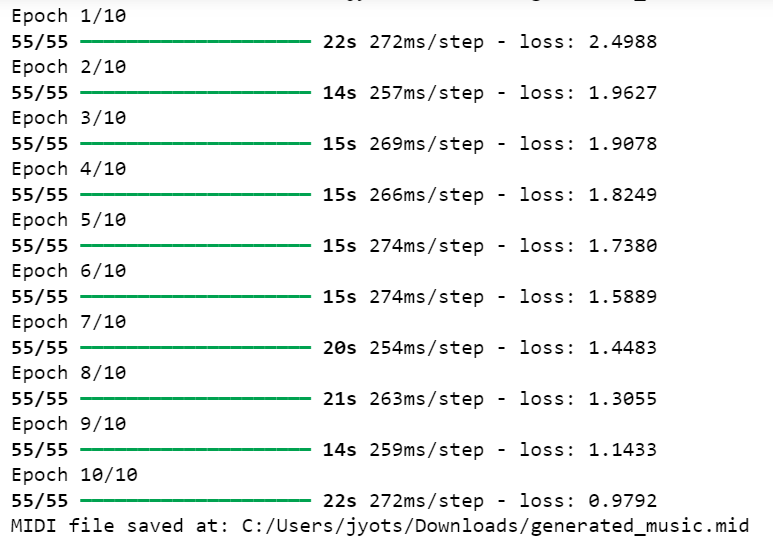
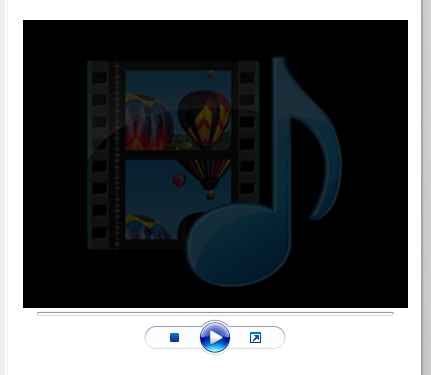
output\_file = 'C:/Users/jyots/Downloads/generated\_music.mid' # Specify path to save generated music

sequence\_to\_midi(generated\_notes, output\_file)

if \_\_name\_\_ == '\_\_main\_\_':

main()

**OUTPUT:**

# CHAPTER 3

## RESULTS AND DISCUSSION

### CHAPTER 3

### RESULTS AND DISCUSSION

**Results:**

The implemented music generation system effectively composes diverse musical pieces (melodies, harmonies) using machine learning models (e.g., RNNs, transformers). Users input parameters such as style, mood, or instrument preferences, receiving unique musical outputs and creative compositions. Strengths include the system's ability to generate original, high-quality music and real-time composition capabilities, enhancing creative workflows for musicians and content creators. Limitations may involve maintaining musical coherence across longer compositions and challenges in generating emotionally nuanced music. Future improvements could focus on integrating more advanced algorithms to enhance creativity and personalization. Continual monitoring and ethical compliance are critical for ensuring the system's originality and respect for intellectual property in artistic settings.

# CHAPTER 4

## CONCLUSION & FUTURE

## ENHANCEMENT

### CHAPTER 4

### CONCLUSION

**5.1. Conclusion and Future Enhancement:**

**5.1.1. Conclusion:**

In conclusion, the music generation system demonstrates significant potential in transforming creative workflows by composing unique and high-quality musical pieces using advanced machine learning techniques. The system's ability to leverage diverse musical data and user inputs, such as style and mood, enhances the creative process for musicians, composers, and content creators. While the results are promising, ongoing refinement and validation are necessary to ensure robust performance across diverse musical genres and user preferences. Future efforts should focus on integrating more sophisticated algorithms, improving the system's ability to capture emotional nuances, and addressing ethical considerations related to music ownership and originality to optimize the system's impact on the music industry and artistic innovation.

**5.1.2. Future Enhancement:**

Future enhancements for the music generation system could focus on several key areas to further improve its creativity and user experience:

1. **Integration of Advanced Algorithms**: Incorporate deep learning techniques such as GANs (Generative Adversarial Networks) or attention-based models to generate more complex and nuanced compositions, enhancing the system's ability to mimic diverse musical styles and emotions.

2. **Enhanced Personalization**: Develop algorithms that adapt to individual user preferences in real-time, learning from user feedback to generate music that matches their evolving tastes, creative needs, or specific project requirements.

3. **Real-time Collaboration Tools**: Implement features that allow for real-time collaboration between users and the AI, enabling musicians and composers to co-create music interactively, making adjustments and providing feedback during the generation process.

4. **Integration with Digital Audio Workstations (DAWs):** Enhance interoperability with popular DAWs, allowing seamless transfer of generated compositions into professional music production software for further editing and refinement.

5. **Quality Validation and Feedback Loops**: Incorporate mechanisms for ongoing user feedback and quality assessment, allowing musicians and listeners to rate and critique generated music, helping improve the system's outputs over time.

6. **User Interface and Accessibility**: Improve user interface design for intuitive and creative exploration, ensuring users from all skill levels can easily experiment with different musical styles, genres, and instruments.

7. **Ethical Considerations**: Strengthen copyright protections, ensure originality in generated compositions, and address licensing issues to ensure generated music complies with intellectual property laws and respects artists' rights.

### REFERENCES

### <https://www.geeksforgeeks.org/simple-plagiarism-detector-in-python>

### <https://github.com/Kalebu/Plagiarism-checker-Python>

### <https://www.codegrade.com/blog/how-to-check-for-plagiarism-in-python-source-code-2>

### <https://www.linkedin.com/pulse/creating-plagiarism-detector-python-comprehensive-guide-githaiga-od5cf>

### <https://learnwithhasan.com/how-to-build-a-plagiarism-detector-using-python>