PROJECT DOCUMENTATION

Task 3: AI Music Generation Studio

Submitted by: Muneeb

Information Technology | The University Of Chakwal

AI Intern | CodeAlpha

February 2026

1. Executive Summary

This document details the development of an autonomous music composition engine using Deep Learning, fulfilling Task 3 of the CodeAlpha AI Internship. The system leverages Recurrent Neural Networks (RNNs) to predict musical sequences, bridging algorithmic probability and creative expression to generate original piano and orchestral motifs.

2. Methodology: Data Engineering

2.1 Symbolic Representation

Unlike raw audio files, this project utilizes MIDI (Musical Instrument Digital Interface) data. This allows the AI to learn the fundamental language of music, such as pitch, duration, and rhythm, rather than processing complex sound waves.

2.2 Preprocessing Pipeline

Using the 'music21' Python library, complex MIDI files were parsed to extract individual notes and chords. These musical events were then vectorized and mapped to unique integer IDs, creating a distinct vocabulary of sounds that the neural network can process mathematically.

3. Neural Network Architecture

3.1 Long Short-Term Memory (LSTM)

The core engine utilizes an LSTM network. Standard neural networks struggle to remember long sequences, but LSTMs use memory gates to retain long-term dependencies. This ensures that the generated melody maintains a consistent key and rhythmic structure over time, rather than devolving into random noise.

3.2 Network Topology

The model features multiple dense LSTM layers (256 to 512 units) combined with Dropout layers (set at 30 percent). This regularization prevents overfitting, ensuring the AI composes truly original music rather than memorizing and regurgitating the training dataset.

4. Implementation & Full-Stack Deployment

4.1 Flask Backend API

A Python Flask server handles the inference requests. When a user prompts the system, the AI generates a sequence of 100 to 150 notes. These integer predictions are then mapped back to their musical string equivalents and re-serialized into a downloadable .mid file.

4.2 Glassmorphism Web UI

The frontend is built with modern HTML/CSS/JS, featuring a 'Midnight Silk' glassmorphism design. It integrates MidiPlayer.js and Soundfont-player to provide real-time audio synthesis directly in the browser, allowing users to preview their AI-generated tracks using different digital instruments (e.g., Grand Piano, Orchestral Violin).

5. Technical Challenges & Optimizations

5.1 Overcoming the 'Repetitive Tune' Problem

Initial greedy-search predictions resulted in repetitive and uncreative melodies. This was resolved by implementing 'Temperature Sampling'. Instead of always selecting the single highest-probability note, the AI introduces calculated randomness (temperature = 0.8). This stochastic approach allows the model to take creative risks, resulting in highly varied and natural-sounding compositions.

5.2 State Management & Auto-Refresh Conflicts

During local deployment, dynamic file generation caused IDE live-servers to forcefully refresh the DOM, clearing the user interface state. This was mitigated by configuring the server environment to ignore the 'static' output directory, ensuring a seamless and uninterrupted user experience.