**AI-powered music generation system**

This plan will guide you through building an AI-powered music generation system from scratch, deploying it live, and leveraging GitHub to improve your skills. The project will be broken into **3 phases**, each lasting roughly a month. You will also learn to set up **GitHub workflows** to collaborate efficiently and deploy your code.

**Phase 1: Research and Setup (Week 1-4)**

**1. Project Planning & Repository Setup**

* **Create a GitHub Repository**: Set up a private or public repository for the project.
  + **Repository Name**: ai-music-generation
  + **Branches**: Main, Development, Features (for new ideas or features)
  + **GitHub Skills**:
    - Add a README.md with an initial project overview.
    - Create a .gitignore file to avoid committing large or irrelevant files (e.g., audio, data files).
* **Collaborate on GitHub**: Both of you will create branches and submit pull requests to the Development branch.

**Resources**:

* [GitHub for Beginners Guide](https://docs.github.com/en/get-started/quickstart)
* [GitHub Actions for CI/CD](https://docs.github.com/en/actions)

**2. Model Selection & Research**

* **MuseNet** or **Jukebox** (OpenAI): These models can generate high-quality music. Research both models and decide which fits your project.
* **Training/Pretrained Models**: Since training from scratch is complex, you can use pretrained models to save time.

**Resources**:

* [MuseNet on OpenAI](https://openai.com/blog/musenet/)
* [Jukebox on GitHub](https://github.com/openai/jukebox)
* Transformer Models for Music Generation

**3. Music Dataset Collection**

* Look for public datasets that include different music genres, tempos, and moods. Although you won't train from scratch, you may need a small dataset for post-processing and testing.

**Resources**:

* [Free Music Archive](https://freemusicarchive.org/)
* GTZAN Music Genre Dataset

**4. Environment Setup**

* **Python Libraries**:
  + Install required libraries like torch, tensorflow, librosa, pydub for handling audio, and streamlit for building your UI.
* **Model Hosting**: Set up cloud GPU (e.g., Google Colab, AWS, GCP) for training/testing.
* **GitHub Actions**:
  + Set up a basic CI workflow for testing and linting your Python code using pytest and flake8.

**Resources**:

* Librosa for Audio Processing
* Google Colab Setup

**Phase 2: Development & Real-Time Controls (Week 5-8)**

**1. Model Integration**

* **MuseNet/Jukebox Integration**: Integrate the chosen generative model into your Python application.
  + **Input Parameters**: Enable users to choose tempo, genre, and mood.
  + **GitHub Usage**:
    - Push the initial integration and track all changes using pull requests.
    - Keep your branches small and specific (e.g., feature-musenet-integration).
* **Code Testing**: Use unit testing on smaller functions (e.g., audio input/output handling).

**Resources**:

* OpenAI Jukebox Colab
* Python Unit Testing Guide

**2. Real-Time Music Customization**

* **Dynamic Controls**: Integrate sliders or input boxes for users to adjust the mood, tempo, and instruments in real time.
* **Streamlit for UI**: Create a simple UI using Streamlit to allow users to input preferences and listen to generated music instantly.

**GitHub Tasks**:

* Work on a feature-branch for real-time control integration.
* Create an issue and assign each of you tasks to work on (collaborating using GitHub Issues).

**Resources**:

* Streamlit for UI

**3. Post-Processing and Audio Refinement**

* Use librosa or pydub to refine the generated audio (e.g., improving quality or applying filters).
* **Enhancements**: Use effects like reverb, equalization, and dynamic compression to polish the output.

**GitHub Skills**:

* Add continuous integration checks to automatically test the audio output functionality.
* Use GitHub Actions to run the audio quality tests in your CI pipeline.

**Resources**:

* [Pydub for Audio Manipulation](https://pydub.com/)
* [Audio Post-Processing Guide](https://audiomasterclass.com/)

**Phase 3: Deployment & Live System (Week 9-12)**

**1. Deploying the Application**

* **Cloud Deployment**: Deploy your system using platforms like **Heroku**, **AWS**, or **Streamlit Cloud**.
* **Dockerize the App**: Containerize your Python app for easier deployment.
* **Deployment Pipeline**: Set up a CI/CD pipeline with GitHub Actions to automatically deploy the latest version of the app after successful tests.

**GitHub Tasks**:

* Create a feature-docker branch and containerize your application.
* Integrate GitHub Actions to auto-deploy the application after every push to the Main branch.

**Resources**:

* Streamlit Deployment
* Docker for Beginners

**2. Real-Time Interaction Testing**

* Test the real-time controls by adjusting tempo, genre, and mood. Optimize for performance.
* Use A/B testing to gather feedback from users on the generated music.

**GitHub Tasks**:

* Create an issue for user feedback and track the performance optimizations.

**3. Final Testing & Polishing**

* Conduct thorough testing, including stress tests to see how well the app handles high traffic or multiple users.
* Polish the UI for a professional, user-friendly experience.
* Ensure your generated music meets quality standards and can adjust dynamically during live streams.

**4. Documentation & Final Deployment**

* Write detailed documentation (in the README.md) on how to use the system, its features, and how to contribute.
* Deploy the final version live and share it with potential users.

**GitHub Tasks**:

* Finalize all documentation and merge all feature branches into Main.

**Key Tools & Resources**

* **Generative Models**:
  + [MuseNet](https://github.com/openai/musenet)
  + [Jukebox](https://github.com/openai/jukebox)
* **Audio Processing**:
  + [Librosa](https://librosa.org/)
  + [Pydub](https://pydub.com/)
* **Web App & Deployment**:
  + Streamlit
  + Heroku Deployment Guide
* **GitHub Skills**:
  + [GitHub Basics](https://docs.github.com/en/get-started)
  + [GitHub Actions for CI/CD](https://docs.github.com/en/actions)