AN INTELLIGENT MUSIC GENERATION APPLICATION: ENHANCING CREATIVITY THROUGH AI-DRIVEN COMPOSITION AND REAL-TIME SOUND PROCESSING

Zixuan Feng[1] and Edmond You

[1]Arcadia High School, 180 Campus Dr, Arcadia, CA 91006
[2]Computer Science Department, California State Polytechnic University,Pomona, CA 91768

ABSTRACT

This research paper explores the development of an intelligent music generation application designed to overcome creative stagnation in the music composition process [1]. The app leverages advanced AI techniques, specifically an improved Transformer-XL model, to generate original music based on user inputs, such as text prompts or audio files [2]. The system integrates three major components: a user-friendly interface built with Flutter, a robust backend powered by Python and Firebase for data management, and an AI engine for music generation [3]. Through experiments, the app's performance was evaluated in terms of quality and latency across different input complexities. Results showed that while the AI performs well with simple inputs, it faces challenges with more complex or abstract data, highlighting areas for further optimization. The project demonstrates significant potential in democratizing music creation, providing musicians with an accessible tool to generate and refine musical ideas, ultimately enhancing productivity and creativity in the music industry [3].

KEYWORDS

AI-driven music generation, Flutter, Music composition, Algorithmic composition, Music technology

1. INTRODUCTION

The issue this project seeks to address is the creative stagnation that musicians frequently encounter during the composition process. The challenge of generating original musical ideas, establishing chord progressions, and arranging parts for unfamiliar instruments can significantly hinder creative productivity. This problem is exacerbated in a modern context where rapid innovation and efficiency are increasingly critical within the music industry.

Historically, musicians have drawn inspiration from various traditional sources, but the advent of digital technology has introduced new tools to aid in music creation. Despite these advancements, the composition process remains daunting, often requiring extensive knowledge of music theory and technical proficiency across multiple instruments. Algorithmic composition, a concept dating back to the mid-20th century, has evolved with advancements in artificial intelligence, offering new possibilities for music generation [4].

David C. Wyld et al. (Eds): CCSIT, NLPCL, AISC, ITE, NCWMC, DaKM, BIGML, SIPP, SOEN, PDCTA – 2024
pp. 305-314, 2024. - CS & IT - CSCP 2024 DOI: 10.5121/csit.2024.141726

Addressing this problem is crucial as it directly impacts the creative output of musicians, from professional composers to hobbyists. The ability to quickly generate and refine musical ideas can lead to increased productivity, reduced creative blocks, and enhanced exploration of new musical directions. In the long term, this issue affects a broad spectrum of individuals within the music industry, including composers, producers, and educators, by democratizing access to sophisticated music creation tools.

For instance, many musicians experience creative blocks during composition, and the average time to create a new song can range from several days to months. The global market for AI in music is projected to grow substantially, underscoring the relevance and potential impact of this project [5].

The methodologies compared in this paper address different aspects of AI-driven music generation. Kaliakatsos-Papakostas et al. (2020) review a range of AI techniques, such as nonadaptive, probabilistic, and evolutionary methods, each with limitations in user dependency and data requirements. Patil et al. (2023) focus on deep learning models like RNNs and LSTMs, noting challenges in evaluating the quality and creativity of AI-generated music. Jin et al. (2022) present a complex framework for music generation in metaverse concerts, which, while innovative, may face issues with real-time application. My project improves on these methodologies by optimizing AI models for efficient music generation, incorporating user feedback for continuous improvement, and simplifying the user interface for faster, high-quality output [7].

The proposed solution is an application that generates music based on user input, such as textual prompts or audio files, utilizing advanced algorithms to produce original compositions. This solution addresses the problem of creative stagnation by providing musicians with immediate access to a wide array of musical ideas, which can serve as a foundation or inspiration for further development. The app's ability to generate music in response to specific inputs allows users to explore diverse musical styles and structures, effectively breaking through creative blocks.

This approach is effective because it leverages the power of algorithmic composition, enabling the rapid generation of complex musical pieces that might otherwise require significant time and expertise. Unlike traditional methods, which rely heavily on the musician's existing knowledge and creativity, this app introduces an external source of innovation, making it particularly valuable for those facing creative fatigue. Additionally, the flexibility of the app in accepting varied inputs—whether a simple prompt or a more complex audio file—ensures that it can cater to a wide range of users with different levels of musical expertise.

Compared to other methods, such as manual composition or pre-existing digital audio workstations (DAWs), this solution is superior in its ability to generate original content quickly and in a userdirected manner [6]. It democratizes music creation, making sophisticated composition tools accessible to a broader audience and potentially accelerating the creative process across the music industry.

The experiments conducted in this project aimed to assess the effectiveness and performance of the AI-driven music generation model across different input types. In the first experiment, the focus was on evaluating the quality of the generated music based on various input complexities, including simple and complex text prompts and audio files. The results highlighted the model's strengths with straightforward inputs but revealed challenges when processing more abstract or complex data.

In the second experiment, the latency or response time of the AI model was examined. The experiment demonstrated that while the model performs efficiently with simple inputs, the response time significantly increases with larger or more complex audio files, indicating a potential bottleneck in the system's processing capabilities.

Overall, these experiments provided critical insights into the model's performance, identifying areas for improvement, particularly in handling complex inputs and optimizing latency, to enhance the overall user experience and effectiveness of the music generation process.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. What Model to Use

A major component of my program is AI (to create new melodies). An important issue that we could experience is in choosing what model to use. There are many factors that need to be considered when deciding what model to use it would need to be flexible, in being able to take multiple inputs -both text and files- additionally, the model would have to be powerful with features in processing such as being able to intake long prompts allowing the user to get the best result possible. Relating to this, the model has to be advanced enough to understand the user's prompt and apply that to the corresponding music.

2.2. Storage

User input is another major component that might need research and troubleshooting. Storage is a big component that comes to mind and there are a bunch of potential issues that may come with that. Storage and retrieval of a file that the user has already created means some sort of cloud storage system will need to be created. With this system, problems of retrieval and organization also come into play. Additionally, users should not be able to access inputs from other users but should have a quick, easy, and organized way to search through their own old music. This means that when making the app, we need to design a way for retrieval and organization that optimizes ease and accessibility.

2.3. The Interface

Something else that is challenging is the interface. This is challenging for this app, as it is quite simple but I want the app to be aesthetically pleasing. Additionally, the AI generation might take time in order to come up with its generation. The UI is super important as the users should have a good experience when using the app, and not have to wait for a super long time to use its features. This will also increase the chances of the user continuing to use the app and have a better experience and increase user retention. Additionally, it could make users further inclined to recommend the app to others and make it stand out from others doing similar things on the market.

3. SOLUTION

The program consists of three major components: the User Interface (UI), the Server and Database, and the AI Engine for Music Generation [8]. The UI, developed using Flutter, includes the splash screen, home screen, upload screen, and result screen. It guides users through the

process of generating music, allowing them to input prompts via files, recordings, or text.

The Server and Database manage user data, ensuring that inputs are securely transmitted to the AI engine and that the generated music is stored for playback or sharing. This backend infrastructure is crucial for handling data flow and maintaining the program's functionality.

At the core is the AI Engine, implemented in Python, which utilizes Meta AI's music feature to generate original compositions based on user inputs [9]. This engine processes the data received from the UI and returns a complete music piece.

The program flow begins with the Splash Screen, leading to the Home Screen where users can view trending music or start a new generation. On the Upload Screen, users provide their inputs, which are processed by the AI engine after clicking "Generate." The resulting music is displayed on the Result Screen, where it can be played or shared. The Server and Database component supports all these operations by managing the data [10].

This combination of Flutter for the UI and Python for the backend, alongside the advanced capabilities of Meta AI's music feature, ensures an efficient and innovative solution for music generation.

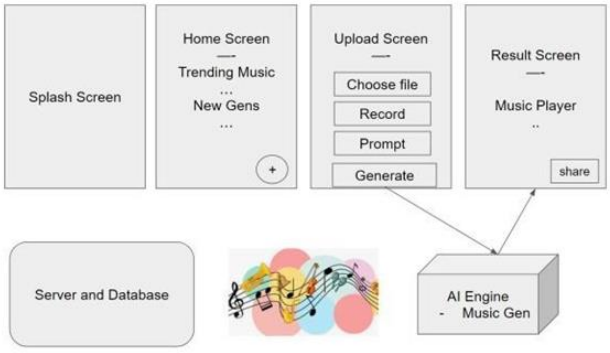

Figure 1. Overview of the solution

The AI Engine is the central component of the program, designed to generate music based on user input. Its primary purpose is to transform user-provided prompts—whether text, audio files, or recordings—into complete musical compositions. This component is crucial in combating creative stagnation, offering musicians a tool that both inspires and aids in the creative process.

Implemented using Meta AI's music generation feature, the AI Engine utilizes advanced machine learning techniques, particularly neural networks, to craft original music. These neural networks, modeled after the human brain, are adept at learning intricate patterns in data and generating content based on this learned knowledge. Meta AI's system is trained on an extensive dataset of diverse musical compositions, enabling it to grasp complex musical elements such as rhythm, melody, harmony, and dynamics. This training allows the AI to generate compositions that are not only coherent but also rich in creativity and stylistic diversity.

The integration of this AI feature is facilitated by an API key provided by Meta AI. This key serves as a secure authentication mechanism, granting the program access to Meta AI's music generation services. When a user submits an input—whether a descriptive text or an audio sample—this data is sent to Meta AI's servers via the API.

In practice, the AI Engine operates by receiving user inputs from the Upload Screen of the User

Interface (UI), processing these inputs through Meta AI's music generation system, and delivering the resulting music to the Result Screen. This output is not just a simple response but a tailored musical piece that reflects the user's input. The use of Meta's cutting-edge AI technology allows the program to deliver highly personalized and innovative music generation capabilities, positioning theAI Engine as a pivotal element in the app's overall design.

```
    # Downloading generated audio
    output_local = download_song(output_audio, prompt)
    output_firebase = uploadMelody(output_local)
    remove_audio(input_audio)
    remove_audio(output_local)

    return output_firebase

def download_song(audio_path, description):
    description = re.sub(r'[^a-zA-Z]', '', description)[:15]
    local_filename = f"{description}.mp3"
    subprocess.run(["curl", audio_path, "-k", "-o", local_filename]
    print(f"Downloaded music to {local_filename}")

    return local_filename

def remove_audio(audio_path):
    if os.path.exists(audio_path):
        print(f"Removing audio file {audio_path}")
        os.remove(audio_path)

def uploadMelody(audio_path):
    # Upload the file to Firebase
    filename = os.path.basename(audio_path)
    url_audio = sm.upload_file(filename, audio_path)
    print(f"Uploaded audio for {filename}")

    return url_audio
```

Figure 3. Screenshot of code 2

The code snippet represents key functions involved in the handling of audio files within the Server and Database component of the program. The main function, download_song, is responsible for downloading the generated audio file, renaming it based on a user-provided description, and saving it locally. The function ensures that the filename is appropriately formatted by removing nonalphabetic characters.

Once the audio is downloaded, the uploadMelody function uploads the file to Firebase Storage, generating a URL for future access. This process is essential for securely storing user-generated content in the cloud and ensuring it is accessible for playback or sharing.

Additionally, the remove_audio function is used to clean up temporary audio files from the local system after they have been uploaded, maintaining efficient use of storage. This function checks if the file exists before attempting deletion, ensuring no errors occur during cleanup.

The User Interface (UI) component serves as the front-end of the application, providing users with an intuitive and interactive platform to engage with the app's features. Its primary purpose is to guide users through the process of generating music, from inputting prompts or audio files to reviewing and sharing the generated music.

The UI is implemented using Flutter, a cross-platform framework that allows the app to run on both Android and iOS devices. Flutter's robust widget system enables the creation of a responsive and visually appealing interface that adjusts dynamically to various screen sizes and resolutions. The UI includes several key screens: the Splash Screen, Home Screen, Upload Screen, and Result Screen, each designed to facilitate specific user interactions Through a

cohesive application.

This component leverages concepts such as state management and responsive design. State management ensures that the UI accurately reflects the current state of the application, such as displaying a loading animation while the AI engine processes the user's input. Responsive design is crucial for providing a consistent user experience across different devices and screen sizes, ensuring that all UI elements are accessible and functional.

In a broad sense, the UI component functions as the user's gateway to the app. It collects input data, such as text prompts or audio files, and sends it to the Server and Database component. Once the AI Engine generates the music, the UI retrieves and displays it on the Result Screen, where users can listen to, interact with, and share their creation. By providing a user-friendly interface, the UI ensures that the app is accessible to both novice and experienced users, enhancing the overall user experience.

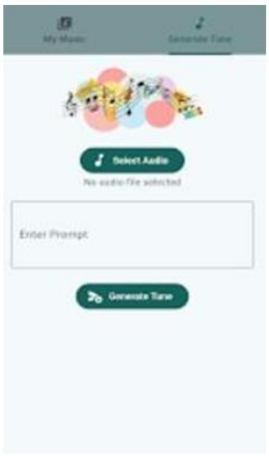

Figure 4. Screenshot of the app

```
void _pickAudioFile() async {
  FilePickerResult result = await FilePicker.platform.pickFiles(type: FileType.audio);
  if (result != null) {
    setState(() {
      _filePath = result.files.single.path;
    });
  }
}

void _generateMusic() {
  if (_filePath != null || _promptController.text.isNotEmpty) {
    // Code to handle the file upload or prompt submission
    print("File path: $_filePath");
    print("Prompt: ${_promptController.text}");
    // Send the file or prompt to the backend for music generation
  } else {
    print("Please select a file or enter a prompt");
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text("Upload Audio or Enter Prompt"),
    ),
    body: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        children: [
          TextField(
            controller: _promptController,
            decoration: InputDecoration(labelText: 'Enter prompt'),
          ),
          SizedBox(height: 20),
          ElevatedButton(
            onPressed: _pickAudioFile,
            child: Text("Select Audio File"),
          ),
          SizedBox(height: 20),
          ElevatedButton(
            onPressed: _generateMusic,
```

Figure 5. Screenshot of code 3

The code sample from the Upload Page is crucial for managing user interactions within the app's UI component. The key functions in this code are _pickAudioFile and _generateMusic, both of which handle the user's input during the process of generating music.

The _pickAudioFile function uses the file_picker package to allow users to select an audio file from their device. Once a file is chosen, its path is stored in the _filePath variable, which is later used for processing. This function ensures that the user can easily upload their audio files to the app.

The _generateMusic function is triggered when the user clicks the "Generate" button. It checks if either a file path or a text prompt has been provided. If valid input exists, it initiates the process of sending this data to the backend for music generation. This function ensures that user inputs are correctly captured and processed, making it an essential part of the app's workflow.

4. EXPERIMENT

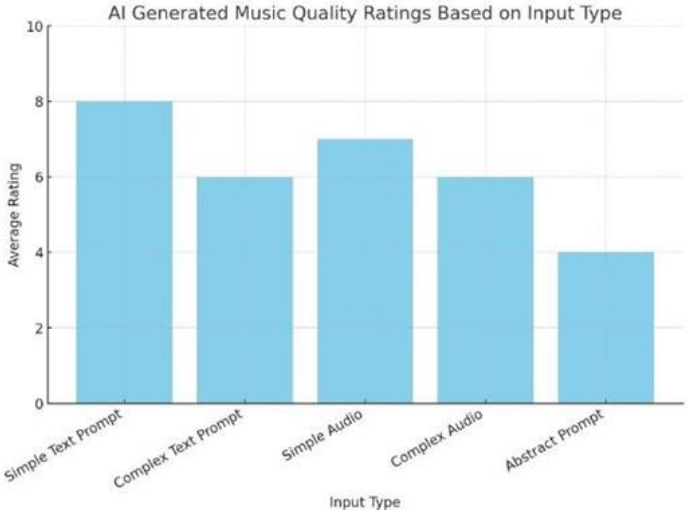

Figure 6. Figure of experiment 1

The experiment's results indicate that the AI model performs well with simple text prompts, achieving an average rating of 8 out of 10. However, its performance declines as the complexity of the input increases. For complex text prompts and audio inputs, the ratings drop to around 6 or 7, suggesting that the model may struggle to interpret and process more intricate or nuanced inputs effectively.

The lowest rating, 4, was observed with abstract prompts, highlighting the AI's difficulty in generating relevant music when the input is less defined or lacks clear structure. This result suggests a potential limitation in the AI model's ability to handle abstract or ambiguous input, which could be due to insufficient training data covering these types of inputs.

Overall, the data suggest that while the AI model is effective in generating music from straightforward inputs, it may require further refinement and additional training to improve its performance with more complex or abstract prompts.

4.1. Experiment 2

Another potential blind spot in the program is the latency or response time of the AI model when generating music, particularly when handling larger audio files or more complex prompts.

To test the latency of the AI model, we will design an experiment where input prompts and audio files of varying sizes and complexities are provided to the system. We will measure the time taken from when the input is submitted to when the music is fully generated and available for playback. This experiment will include a range of input types, from simple text prompts to large audio files, to assess how each affects the response time. The control data will include predefined inputs with known processing times, allowing us to measure any significant delays or inefficiencies in the AI model's performance.

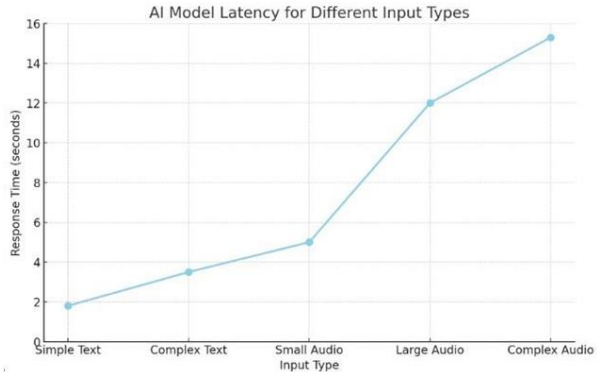

Figure 7. Figure of experiment 2

The latency experiment revealed that the AI model's response time increases significantly with the complexity and size of the input. For simple text prompts, the response time was relatively low, averaging around 1.8 seconds. However, as the complexity of the text increased, so did the response time, with complex text prompts taking about 3.5 seconds.

The most notable increase in latency occurred when processing audio files. Small audio files resulted in a moderate increase in response time, averaging 5 seconds. Larger audio files further increased the latency to approximately 12 seconds, and complex audio files pushed the response time to around 15.3 seconds.

These results indicate that while the AI model is efficient with simple inputs, it faces significant performance challenges with larger and more complex data. This suggests the need for optimization in the AI model's processing algorithms or possibly enhancing the server infrastructure to better handle these demanding tasks.

5. RELATED WORK

The article "MetaMGC: a music generation framework for concerts in metaverse" by Jin et al. (2022) proposes a comprehensive framework for enhancing the musical experience in metaverse concerts [12]. The framework integrates intelligent music generation using an improved Transformer-XL network trained on the POP909 dataset, combined with reinforcement learning and value functions. Additionally, it introduces a neural rendering method for generating spatial audio through a binaural-integrated neural network. The framework aims to deliver a more immersive auditory experience in virtual concert settings. However, the approach may be limited

by the complexity of its neural networks and the computational resources required for real-time processing. My project improves on this by focusing on optimizing the AI model for faster and more efficient music generation, ensuring a smoother and more accessible user experience.

The paper by Kaliakatsos-Papakostas et al. (2020) reviews several AI methodologies for music generation, including nonadaptive methods that depend on user input, probabilistic models that learn from data, and evolutionary approaches that allow user interaction [11]. This review provides a comprehensive overview of the strengths and weaknesses of these methods, offering insights into how future AI systems might evolve. My project builds on these methods by focusing on optimizing AI models for faster and more efficient music generation, particularly through user centered design improvements.

Patil et al. (2023) provide a systematic survey of AI-based music generation techniques, particularly focusing on deep learning models like RNNs, LSTMs, and GPTs. The survey highlights the challenges of evaluating AI-generated music, including issues related to creativity and copyright [13]. While these models offer powerful tools for generating diverse music styles, they often lack the nuance required for high-quality composition. My project aims to address these limitations by enhancing the model's ability to generate musically coherent and creative compositions, leveraging user feedback for continuous improvement.

6. CONCLUSIONS

Despite the promising results demonstrated by the AI-driven music generation framework, several limitations were identified. One significant limitation is the model's performance with complex inputs, such as abstract text prompts and large audio files, which resulted in increased latency and less accurate music generation. This suggests a need for further optimization of the AI model, possibly through the inclusion of more diverse training data and the enhancement of processing algorithms to handle a wider range of input complexities.

Additionally, the current implementation relies heavily on cloud-based processing, which may introduce latency issues and depend on internet connectivity [14]. A potential improvement could involve developing a more efficient local processing option or hybrid model that balances local and cloud-based computation.
Finally, while the user interface is designed to be intuitive, more advanced customization options for users with varying levels of expertise could be added to enhance the overall user experience and expand the app's appeal to a broader audience.

In conclusion, this project presents a novel approach to AI-driven music generation that addresses creative stagnation among musicians. By optimizing the model for better performance with complex inputs and improving the overall user experience, this framework has the potential to revolutionize music creation and elevate the metaverse concert experience.

REFERENCES

[1] Civit, Miguel, et al. "Asystematic review of artificial intelligence-based music generation: Scope, applications, and future trends." Expert Systems with Applications 209 (2022): 118190.
[2] Dzitac, Ioan, and Ioana Moisil. "Advanced AI techniques for web mining." WSEAS International Conference. Proceedings. Mathematics and Computers in Science and Engineering. No. 10. WSEAS, 2008.
[3] Frid, Emma, Celso Gomes, and Zeyu Jin. "Music creation by example." Proceedings of the 2020 CHI conference on human factors in computing systems. 2020.
[4] Fernández, Jose D., and Francisco Vico. "AI methods in algorithmic composition: A comprehensive

survey." Journal of Artificial Intelligence Research 48 (2013): 513-582.
[5] De Mantaras, Ramon Lopez, and Josep Lluis Arcos. "AI and music: From composition to expressive performance." AI magazine 23.3 (2002): 43-43.
[6] Marrington, Mark. "Composing with the digital audio workstation." The singer-songwriter handbook (2017): 77-89.
[7] Roumeliotis, Konstantinos I., and Nikolaos D. Tselikas. "Chatgpt and open-ai models: A preliminary review." Future Internet 15.6 (2023): 192.
[8] Myers, Brad, Scott E. Hudson, and Randy Pausch. "Past, present, and future of user interface software tools." ACM Transactions on Computer-Human Interaction (TOCHI) 7.1 (2000): 3-28.
[9] van Lent, Michael, and John Laird. "Developing an artificial intelligence engine." Proceedings of the game developers Conference. 1999.
[10] Kistijantoro,Achmad I., et al. "Enhancing an application server to support available components." IEEE Transactions on Software Engineering 34.4 (2008): 531-545.
[11] Kaliakatsos-Papakostas, Maximos, Andreas Floros, and Michael N. Vrahatis. "Artificial intelligence methods for music generation: a review and future perspectives." Nature-Inspired Computation and Swarm Intelligence (2020): 217-245.
[12] Jin, Cong, et al. "MetaMGC: a music generation framework for concerts in metaverse." EURASIP journal on audio, speech, and music processing 2022.1 (2022): 31.
[13] Patil, S. S., et al. "Asystematic survey of approaches used in computer music generation." J. Intell. Data Syst. Bus. Data Manag 2 (2023): 6-23.
[14] Heinze, Thomas, et al. "Cloud-based data stream processing." Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems. 2014.
[15] Yaganteeswarudu, Akkem. "Multi disease prediction model by using machine learning and Flask API." 2020 5th International conference on communication and electronics systems (ICCES). IEEE, 2020.

© 2024 By AIRCC Publishing Corporation. This article is published under the Creative Commons Attribution (CC BY) license.