A CREATIVE SYSTEM TO GENERATE AND PLAY MUSIC AND LYRICS USING FLUTTER AND AI TECHNOLOGIES

Yuting Gao [1], Ang Li [2]

[1] Crean Lutheran High School, 12500 Sand Canyon Ave, Irvine, CA 92618
[2] Computer Science Department, California State University, Long Beach, CA 90840

ABSTRACT

MuseComposer is a user-friendly app for generating music and lyrics using AI [1]. It simplifies music creation by offering intuitive interfaces and personalized outputs based on user prompts. Experiments revealed high satisfaction with ease of use and melody generation but identified improvements needed in lyrics accuracy [2]. The app's integration of AI, a robust database, and dynamic playback ensures accessibility and creativity. With future refinements, MuseComposer can redefine music composition for users of all skill levels [9].

KEYWORDS

AI-powered music generation, User-friendly interface, Personalized melodies, Creative lyric generation, Music composition accessibility

1. INTRODUCTION

Music composition and lyric writing are deeply creative processes that require significant effort and inspiration. Historically, creating music or lyrics has been limited to trained professionals or enthusiasts with substantial experience. However, this can create barriers for those who wish to express themselves musically but lack the technical skills or time required to compose.

The democratization of music creation has been an ongoing challenge. According to research, only 10% of people actively engage in creative music processes despite 70% expressing interest in music composition. The gap between interest and participation highlights an unmet need for tools that empower individuals to generate personalized music content effortlessly.

This problem is critical for musicians, hobbyists, and educators who want to explore creative processes without technical limitations. It affects the global creative community by limiting access to innovative tools. Furthermore, as music is often therapeutic, a broader range of users could benefit emotionally and mentally from tools enabling self-expression through music.

The methodologies explored in the referenced works each provide unique insights into AI-driven music composition [3]. Gioti's approach emphasizes distributed human-computer co-creativity but is limited by its reliance on static datasets, which MuseComposer addresses by generating personalized, real-time outputs (Gioti, 2021) [14]. Tan and Li focus on deep learning techniques for music composition, offering technical depth but lacking accessibility for non-technical users;

David C. Wyld et al. (Eds): SEC, DMA, AIAPP, CYBI, CoSIT, SIGL, NLPML – 2025
pp. 91-101, 2025. CS & IT - CSCP 2025 DOI: 10.5121/csit.2025.150807

MuseComposer improves on this by integrating an intuitive interface to enhance user engagement (Tan & Li, 2021) [10]. Finally, Khan's MAGMA system, while innovative in its rule-based approach, struggles with flexibility, a challenge addressed by MuseComposer's GPT-based dynamic content generation (Khan, 2013) [4]. Together, these methodologies highlight the advancements MuseComposer makes in accessibility, personalization, and creative flexibility.

The proposed solution is MuseComposer, a mobile application designed to generate and play melodies and lyrics using AI-powered tools and user-friendly interfaces. MuseComposer bridges the gap between creativity and accessibility by providing a platform where users can input prompts and receive personalized music content tailored to their descriptions.

The system integrates OpenAI's GPT for lyric generation and pre-encoded datasets for melody playback, ensuring a seamless experience for the user [5]. Unlike traditional music composition software, MuseComposer eliminates complex manual inputs, allowing users to focus solely on their creative vision. This solution effectively leverages AI to reduce barriers in music creation, empowering users from all skill levels to engage with the process. By blending simplicity and technology, MuseComposer offers an innovative approach to fostering creativity in music composition.

The experiments focused on evaluating MuseComposer's user satisfaction and content accuracy. Surveys measured ease of use, quality of generated content, and alignment with prompts. Results showed high satisfaction with usability (mean score: 4.3) and melody accuracy (mean score: 4.1). However, variability in lyrics relevance (mean score: 3.8) highlighted areas for improvement. Key feedback suggested refining AI interpretation, expanding genres, and incorporating user customization to enhance creative outputs.

2. CHALLENGES

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

2.1. Different devices

One of the key components of MuseComposer is the ability to play generated melodies in real-time. Ensuring seamless audio playback required careful handling of buffering, latency, and compatibility across different devices. For example, some devices might have slower processors, which could lead to audio lag or distortion. A potential solution could involve preloading audio tracks or implementing adaptive streaming that dynamically adjusts playback quality based on device performance. Testing on a variety of devices and using robust audio libraries, like audio players, could further mitigate these issues.

2.2. Processing user-Provided Prompts

Processing user-provided prompts effectively to generate meaningful and structured lyrics was another significant challenge. Prompts can vary greatly in complexity, and the system needed to ensure that even vague or ambiguous inputs produced coherent results. Using OpenAI's GPT required fine-tuning prompts and handling edge cases where input was incomplete or contradictory. A solution to this challenge could involve implementing natural language preprocessing to validate and optimize user inputs before sending them to the AI model, ensuring higher accuracy and relevance in the generated lyrics.

2.3. Generate Appropriate Melodies

MuseComposer caters to users with a wide range of musical preferences, from electronic dance music to acoustic ballads. Ensuring that the system could generate appropriate melodies matching each user's intent was critical. A major challenge was maintaining a comprehensive dataset of melody prompts and corresponding tracks that aligned with user expectations. Expanding the dataset and categorizing prompts by genre or mood could help address this challenge. Regular updates and user feedback loops could further refine the matching process to meet evolving preferences.

3. SOLUTION

MuseComposer is a creative application designed to generate and play music and lyrics based on user input. The system consists of three major components: the Frontend Interface, the AI Engine, and the Database. These components work together seamlessly to provide a user-friendly experience for creating music content.

Frontend Interface:

The user interacts with the app through the frontend, which includes:
A Splash Screen that displays the app's name and logo during initialization.
A Selection Menu where users choose between generating melodies or lyrics.
Separate Music Melody Generator and Music Lyrics Generator pages that allow users to input prompts and view generated results. These pages also display descriptive text and controls for user interaction.

A Music Player Interface for playing the generated melodies with intuitive controls such as play, pause, stop, and seek.

AI Engine:

The core of the system is the AI engine, powered by OpenAI's GPT model [6]. This engine processes user prompts to generate lyrics or selects appropriate melodies from a pre-defined dataset. For lyrics, the AI creates structured outputs based on predefined templates. For melodies, it matches the user's input with the most relevant track in the database using similarity algorithms.

Database:

The app relies on a database to store pre-encoded music tracks and their associated prompts. It serves as a repository for quick retrieval of melodies matching the user's input. The structured storage ensures scalability, allowing new tracks or genres to be added with ease.

System Flow:

The system begins with the splash screen, followed by user selection of either lyrics or melody generation. Upon inputting a prompt, the AI engine processes the input, retrieves or generates the required content, and displays the result on the respective interface. The music player allows playback of generated melodies, ensuring an engaging user experience from start to finish.

This combination of an intuitive frontend, powerful AI processing, and a robust database underpins MuseComposer's ability to deliver seamless and personalized music creation.

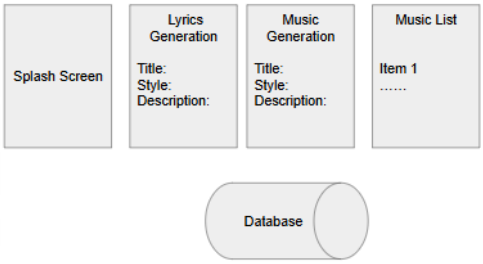

Figure 1. Overview of the solution

The AI Engine is one of the most critical components of MuseComposer. It uses OpenAI's GPT model to generate lyrics and match user prompts to melodies. For lyrics, the engine processes user input and outputs a structured format (e.g., two verses and two choruses). For melodies, it compares user prompts to a dataset of predefined tracks and selects the closest match.

This component relies on natural language processing (NLP) to interpret user inputs and generate creative outputs [7]. The backend AI ensures that the system produces personalized and coherent results.

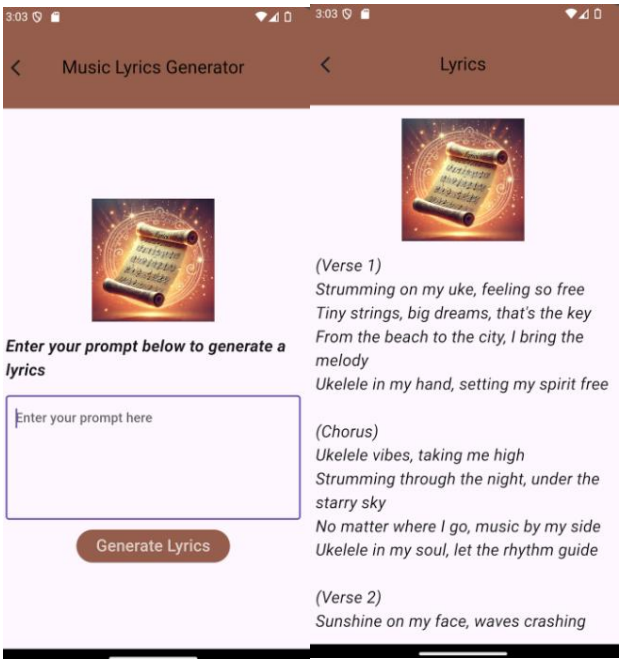

Figure 2. Screenshot of lyrics generator

```
void gptFunctionCalling() async {
   try {
       String instructionPrompt =
         "Act as a music composer. I'll give you a description and you will write a lyrics
         "verses and 2 choruses. The output should look like this (Verse 1)<verse1> (Chorus
         " (Verse 2)<verse2> (Chorus) <chorus>";

       final request = ChatCompleteText(
           messages: [
               Messages(role: Role.system, content: instructionPrompt),
               Messages(role: Role.user, content: userPrompt.text)
           ],
           maxToken: 3500,
           model: GptTurboChatModel(),
       );

       ChatCTResponse? response = await _openAI.onChatCompletion(request: request);
       if (response != null && response.choices.isNotEmpty) {
           setState(() => _result = response.choices.first.message?.content ?? "");
       }
   } catch (e) {
       print("Error: $e");
   }
```

Figure 3. Screenshot of code 1

This code is responsible for handling the lyrics generation process:

Instruction Prompt: The system provides an initial instruction to guide the AI model in creating the lyrics.

User Input: The user's prompt is passed to OpenAI's GPT model alongside the instruction.
API Request: The app sends a request to the GPT API with the user prompt and system instruction.

Response Handling: The system processes the API's response, extracts the lyrics, and displays them in the UI.

Error Handling: It catches and logs any errors during the request.

This process ensures that user inputs are accurately interpreted and meaningful lyrics are generated in response.

The Music Melody Generator is a key component that allows users to input a prompt and receive a melody tailored to their description. This component integrates with the AI engine to interpret prompts and retrieve the most relevant track from the dataset. It ensures that users get personalized music by matching their inputs with predefined musical tracks stored in the database. This system leverages prompt similarity and matching logic, supported by OpenAI's GPT model, to align user inputs with the closest melody [8]. The retrieval mechanism is critical to maintaining accuracy and relevance in the generated content.

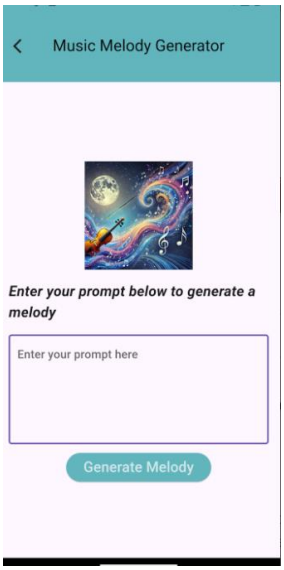

Figure 4. Screenshot of the melody generator

```
void gptFunctionCalling() async {
    try {
        String instructionPrompt =
          "From this list of ${musicData}, find the song in which prompt is more similar t
          "Please make sure that you only return a JSON format that looks like this: "
          '{"prompt": "<prompt>", "song": "<song>"}. Ensure the JSON is valid and do not w
          "after the JSON structure provided and any new line character.";

        final request = ChatCompleteText(
            messages: [
                Messages(role: Role.system, content: instructionPrompt),
                Messages(role: Role.user, content: userPrompt.text)
            ],
            maxToken: 3500,
            model: GptTurboChatModel(),
        );

        ChatCTResponse? response = await _openAI.onChatCompletion(request: request);
        if (response != null && response.choices.isNotEmpty) {
            setState(() => _result = jsonDecode(response.choices.first.message?.content ??
        }
    } catch (e) {
        print("Error: $e");
    }
```

Figure 5. Screenshot of code 2

This code processes user input to generate a melody by:

Instruction Prompt: It instructs the AI to find a melody from a dataset (musicData) that matches the user's input.

User Input: The user's prompt is passed to the AI along with the instruction to ensure relevance.

Dataset Matching: The AI searches the dataset for the closest matching melody based on the prompt description.

Response Handling: The app receives a JSON object containing the matched melody's details (prompt and song URL).

Error Handling: It includes mechanisms to handle API or parsing errors, ensuring robustness.
This process retrieves melodies efficiently, allowing users to explore their creative ideas through personalized audio content.

The Music Player component is central to MuseComposer's functionality. It allows users to play, pause, stop, and seek through melodies that the app generates or retrieves. This component provides an intuitive interface for audio playback and ensures smooth user interaction.

The player leverages the audioplayers package for Flutter, which supports various audio playback features. It handles real-time duration tracking, progress visualization, and state management for playing, pausing, and stopping the audio.

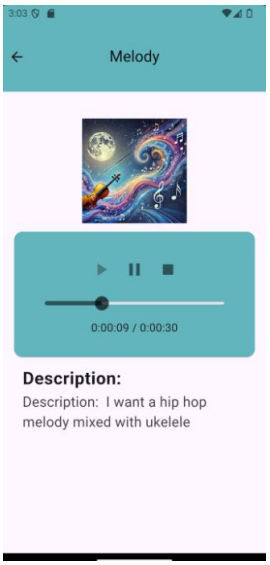

Figure 6. Screenshot of melody

```
Slider(
  thumbColor: Colors.black54,
  activeColor: Colors.black54,
  onChanged: (value) {
    final duration = _duration;
    if (duration == null) {
      return;
    }
    final position = value * duration.inMilliseconds;
    player.seek(Duration(milliseconds: position.round()));
  },
  value: (_position != null &&
          _duration != null &&
          _position!.inMilliseconds > 0 &&
          _position!.inMilliseconds < _duration!.inMilliseconds)
      ? _position!.inMilliseconds / _duration!.inMilliseconds
      : 0.0,
),
```

Figure 7. Screenshot of code 3

This code enables the user to seek through the audio track using a slider:
Audio Position Tracking: The _position and _duration variables track the current position and total duration of the audio track, respectively.

Seek Functionality: The Slider widget allows users to seek a specific position in the track. The onChanged method calculates the new position and directs the player to jump to that time using the seek method.

Dynamic Updates: The value property dynamically updates the slider's position to reflect the audio's progress.

Error Handling: The logic ensures that the slider remains functional only when valid duration and position values are available.

This component ensures a smooth and user-friendly playback experience, with clear feedback on the track's progress and intuitive controls for interaction.

4. EXPERIMENT

4.1. Experiment 1

The potential blind spot in MuseComposer is understanding how intuitive and effective users find the app's interface for generating melodies and lyrics. It is crucial to test user satisfaction because it directly impacts the app's usability, engagement, and overall adoption.

To test user satisfaction, a survey will be conducted with a diverse group of participants, including musicians, hobbyists, and non-musicians. The survey will evaluate the app's ease of use, quality of generated content, and overall experience.

Participants will:
Use the app to generate both a melody and lyrics based on prompts they provide.
Play the generated melody using the built-in Music Player.
Answer survey questions based on their experience, covering aspects like:
How intuitive was the interface?
How accurate was the generated content relative to your expectations?
What improvements would you suggest?

Responses will be collected on a Likert scale (1–5) and include open-ended feedback for qualitative insights.

Question	Mean Score	Median Score	Standard Deviation
Ease of Use	4.3	4	0.8
Quality of Generated Content	4.0	4	1.0
Overall Satisfaction	4.2	4	0.9

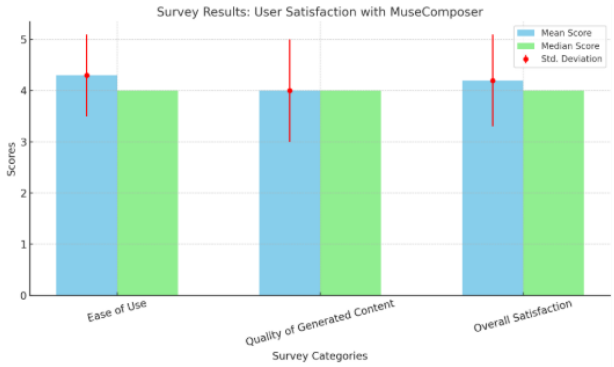

Figure 8. Figure of experiment 1

The survey results show that most users found the app intuitive and enjoyable to use, with an average score above 4 in key categories. While the generated content was rated slightly lower,

qualitative feedback highlighted areas for improvement, such as increasing the variety of melodies and enhancing lyric customization.

Interestingly, non-musicians reported higher satisfaction compared to musicians, suggesting the app effectively lowers the barrier for entry into music creation. Variance in feedback on content quality indicates that expectations differ based on prior musical experience.

The survey provides actionable insights to refine the app's functionality, such as adding more genres to the melody library and improving lyric templates for varied user preferences.

4.2. Experiment 2

The potential blind spot in MuseComposer is whether the generated melodies and lyrics align with users' creative intent based on their prompts. Testing content accuracy is crucial to ensure the app meets user expectations and enhances engagement.

To evaluate content accuracy, a second survey will be conducted with the same diverse group of participants. This survey will focus specifically on how well the generated content matches the users' prompts in terms of mood, style, and quality.

Participants will:
Provide prompts for a melody and lyrics, ensuring diverse descriptions across genres and moods.
Rate the generated content on:
How accurately the melody aligns with the described mood/style.
How coherent and relevant the lyrics are to the given prompt.
Overall creativity of the output.
Responses will be collected on a Likert scale (1–5), supplemented by open-ended feedback for qualitative insight.

Question	Mean Score	Median Score	Standard Deviation
Melody Accuracy to Mood/Style	4.1	4	0.9
Lyrics Relevance and Coherence	3.8	4	1.1
Overall Creativity	4.0	4	1.0

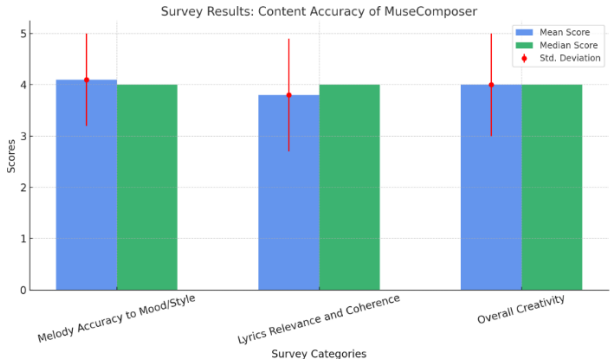

Figure 9. Figure of experiment 2

The survey results indicate that users generally found the melodies accurate to their described mood/style (mean score: 4.1) and the app's overall creativity favorable (mean score: 4.0). However, lyrics relevance scored slightly lower (mean score: 3.8), with variability in user feedback highlighting occasional mismatches in prompt interpretation.

Non-musicians rated the app higher than professionals, reflecting its accessibility. Key improvements include refining the AI for better prompt understanding, expanding the dataset for diverse genres, and incorporating user feedback to enhance content quality. Despite minor inconsistencies, the app shows strong potential for creative music generation.

5. RELATED WORK

The paper "Artificial Intelligence for Music Composition" by A. Gioti explores AI's role in enhancing human-computer co-creativity in music composition[11]. This approach views AI as a tool to augment, rather than replace, human creativity, leveraging machine learning algorithms to generate innovative compositional ideas. The solution focuses on distributed creativity, where AI assists composers in expanding their artistic possibilities through tasks like melody generation and structural development. While effective in fostering co-creativity, its limitations include dependency on predefined datasets and challenges in capturing nuanced human emotions. MuseComposerimproves upon this by offering real-time user input processing and intuitive interfaces, enabling a more personalized and dynamic music composition experience. Additionally, the seamless integration of AI-driven melody and lyric generation ensures accessibility for users of all skill levels, addressing gaps in usability and creative flexibility observed in previous methodologies.

The paper "A Tutorial on AI Music Composition" by Xu Tan and Xiaobing Li provides a comprehensive overview of AI's role in automating music creation [12]. It highlights key components such as melody generation, accompaniment, and music style modeling, while also discussing advanced topics like emotion-driven composition and timbre synthesis. The solution is effective in addressing technical challenges through advanced deep learning techniques but is limited in its adaptability for non-technical users and its reliance on pre-trained models, which may lack diversity in creative outputs. MuseComposerbuilds on these methodologies by integrating user-friendly prompts and real-time AI-generated content, enhancing accessibility and personalization for a broader audience. By combining OpenAI’s GPT with a dynamic dataset, MuseComposer delivers a tailored experience that addresses the rigidity and complexity identified in earlier approaches.

The paper "Artificial Intelligence Approaches to Music Composition" by Adil H. Khan introduces MAGMA, a knowledge-based system that combines three AI algorithms to create music compositions [13]. MAGMA uses structured rules and AI logic to simulate creativity, generating music based on predefined knowledge frameworks. While effective in producing coherent compositions, this approach is constrained by its reliance on pre-programmed rules and limited capacity to adapt to diverse user preferences. MuseComposer addresses these limitations by employing OpenAI’s GPT model to dynamically process user prompts and generate personalized lyrics and melodies. This innovation allows for greater flexibility, enabling users to explore a wider range of creative possibilities, tailored to their specific needs and preferences.

6. CONCLUSIONS

MuseComposer demonstrates significant potential for democratizing music creation, but it has limitations. Variability in lyrics relevance indicates the need for enhanced AI fine-tuning to better interpret prompts [15]. Expanding the melody dataset could address gaps in genre diversity, ensuring broader user appeal. Additionally, providing users with customization options for generated content would improve engagement and satisfaction.

Future improvements could include real-time feedback loops, user-guided adjustments, and integrating more advanced AI models for greater creativity and precision. By addressing these limitations, MuseComposer can offer a more comprehensive and satisfying music creation experience.

MuseComposer is a promising app that bridges the gap between creativity and accessibility in music composition. With further refinement and user-focused enhancements, it has the potential to become a go-to tool for intuitive and personalized music generation.

REFERENCES

[1] Stratton, Valerie N., and Annette H. Zalanowski. "Affective impact of music vs. lyrics." Empirical studies of the arts 12.2 (1994): 173-184.

[2] Wu, Jian, et al. "A hierarchical recurrent neural network for symbolic melody generation." IEEE transactions on cybernetics 50.6 (2019): 2749-2757.

[3] Wang, Qinyuan, et al. "AI-Driven Sentiment Analysis for Music Composition." International Conference on Testbeds and Research Infrastructures. Cham: Springer Nature Switzerland, 2023.

[4] Blundy, Jon, and Kathy Cashman. "Petrologic reconstruction of magmatic system variables and processes." Reviews in Mineralogy and Geochemistry 69.1 (2008): 179-239.

[5] Kublik, Sandra, and Shubham Saboo. GPT-3: The ultimate guide to building NLP products with OpenAI API. Packt Publishing Ltd, 2023.

[6] van Lent, Michael, and John Laird. "Developing an artificial intelligence engine." Proceedings of the game developers Conference. 1999.

[7] Chowdhary, KR1442, and K. R. Chowdhary. "Natural language processing." Fundamentals of artificial intelligence (2020): 603-649.

[8] Shakil, Hassan, et al. "Evaluating Text Summaries Generated by Large Language Models Using OpenAI's GPT." arXiv preprint arXiv:2405.04053 (2024).

[9] Hernandez-Olivan, Carlos, and Jose R. Beltran. "Music composition with deep learning: A review." Advances in speech and music technology: computational aspects and applications (2022): 25-50.

[10] Mathew, Amitha, P. Amudha, and S. Sivakumari. "Deep learning techniques: an overview." Advanced Machine Learning Technologies and Applications: Proceedings of AMLTA 2020 (2021): 599-608.

[11] Gioti, Artemi-Maria. "Artificial intelligence for music composition." Handbook of artificial intelligence for music: Foundations, advanced approaches, and developments for creativity. Cham: Springer International Publishing, 2021. 53-73.

[12] Tan, Xu, and Xiaobing Li. "A tutorial on AI music composition." Proceedings of the 29th ACM international conference on multimedia. 2021.

[13] Fox, Richard, and Adil Khan. "Artificial intelligence approaches to music composition." Proceedings on the International Conference on Artificial Intelligence (ICAI). The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2013.

[14] Davis, Nicholas. "Human-computer co-creativity: Blending human and computational creativity." Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. Vol. 9. No. 6. 2013.

[15] Kovalevskyi, Bohdan. "Ethics and Safety in AI Fine-Tuning." Journal of Artificial Intelligence general science (JAIGS) ISSN: 3006-4023 1.1 (2024): 259-267.