

# Leveraging LLMs for Automated Music Generation

Corey Zhang\*, Mark Santolucito†

\* *Eastlake High School*

Sammamish, WA, USA

Email: s-cozhang@lwsd.org

† *Barnard College, Columbia University*

New York, USA

Email: msantolu@barnard.edu

**Abstract**—This paper explores the application of large language models (LLMs) for music generation, specifically focusing on generating MIDI files, JSON representations, and music-related code. We aim to leverage LLM capabilities to automate and enhance the music composition process. We systematically examine methodologies and tools for integrating LLMs in music generation with direct music creation and code-based techniques. The challenge of automating music composition using AI remains significant due to the complexity and creativity required in the process. Our findings demonstrate the potential of LLMs to innovate and streamline music composition, offering new tools and approaches for musicians and developers.

## KEYWORDS

large language models, music generation, code generation

## I. INTRODUCTION

Music composition traditionally requires significant expertise and creativity, making it a time-consuming and challenging endeavor. However, the advent of artificial intelligence (AI) and natural language processing (NLP) has introduced novel possibilities for automating aspects of music composition. Despite these advancements, the application of AI in music composition is still in its nascent stages, revealing a substantial gap in effectively harnessing these technologies for creative fields like music. Key areas for improvement include enhancing AI's ability to generate complex and creative compositions, developing tools for human-AI collaboration, creating user-friendly interfaces, increasing algorithmic efficiency and speed, and ensuring the quality and originality of AI-generated music. Addressing these gaps can democratize music creation and expand the horizons of musical innovation.

LLMs developed by OpenAI, such as GPT-3.5, have demonstrated impressive capabilities in generating structured data formats, including text and code [1]. LLMs offer the potential to pioneer new interfaces for musical expression by enabling the integration of powerful AI systems into networked devices via API calls. Accessing the capabilities of LLMs through simple network connections can facilitate novel interactions with music, such as real-time music generation and innovative engagement in live performance contexts. However, a significant challenge in leveraging LLMs for music generation lies in the insufficient understanding of how to effectively

utilize these models to produce music that is both coherent and engaging.

This research explores the potential of these models to generate music in MIDI and JSON formats, introducing an innovative approach to music creation. The need for this research is driven by the opportunity to improve the efficiency and creativity of music composition through AI, addressing the current limitations, and exploring new applications of LLMs in the music domain.

We identify the primary contributions of this work as follows:

- Development and evaluation of a framework for integrating large language models in music generation, utilizing the OpenAI API for both direct and code-based music generation in MIDI and JSON formats.
- Identification of benchmark evaluation prompts and the assessment of generated outputs based on three key metrics: correctness, compliance, and complexity, ensuring the creation of valid, standard-compliant, and sophisticated musical compositions.
- Conducting a comparative analysis between LLM-generated music and traditional code-based music generation, providing insights into the effectiveness of these models and identifying potential areas for enhancement in AI-assisted music composition.

This research seeks to advance the field of AI-driven music composition, providing a foundation for future developments and applications in both computer science and the arts.

## II. STATE OF THE ART

### A. Background

The Musical Instrument Digital Interface (MIDI) is a standardized protocol used for communication between electronic musical instruments and computers. MIDI files contain musical information such as note pitch, duration, velocity, and timing, allowing for playback across any compatible device. Unlike traditional audio files, which store sound waves, MIDI files store instructions for sound production, offering significant flexibility in musical applications [2].

JavaScript Object Notation (JSON) is a lightweight, text-based data interchange format renowned for its human readability and widespread use in web applications. Within the realm of music technology, JSON represents intricate musical structures, encompassing notes, chords, rhythms, and dynamics. This structured format enhances software applications' ability to process musical information, enabling detailed explorations and manipulations of musical data [3].

Early AI-based music composition methods primarily involved rule-based systems and algorithmic composition, which often led to repetitive and predictable outputs due to their rigid structures. Recent advancements have shifted towards machine learning techniques, offering more flexibility and creativity in music generation. JSON offers significant advantages in this context, as it can be used to create complex musical structures by precisely defining musical elements such as tempo changes, dynamic markings, and articulation patterns, which can then be easily manipulated and interpreted by software. This detailed representation allows for more nuanced and sophisticated compositions compared to earlier methods. The ability to convert JSON data into other formats, such as MIDI, further enhances its utility in music technology, allowing for seamless integration into various musical applications. Integrating MIDI and JSON within musical applications leverages MIDI's encoding of musical attributes and JSON's capacity to represent complex structures. This combination enhances music software's capabilities for composing, editing, and analyzing musical compositions. MIDI's real-time performance instructions and JSON's structured representation enable innovative approaches to music creation and analysis, advancing digital music technologies [4].

LLMs, such as OpenAI's GPT-3.5, leverage extensive textual data to exhibit proficiency in generating coherent and contextually relevant text across diverse domains. These models transcend traditional text generation tasks, demonstrating competence in code synthesis, translation, and summarization. Moreover, LLMs' capability to comprehend and manipulate structured data positions them as viable tools for innovative applications in the arts, particularly in domains such as music composition. The inherent versatility of LLMs stems from their training on vast corpora, enabling them to interpret and generate complex outputs from natural language prompts. This ability holds promise for transformative applications in creative fields, including music composition, where LLMs can facilitate the synthesis of elaborate musical compositions based on textual descriptions. In the realm of computational creativity, LLMs represent a paradigm shift, offering computational methodologies that bridge the gap between textual prompts and creative outputs in domains traditionally associated with human creativity. By harnessing the latent potentials embedded in their training data, these models enable novel approaches to artistic expression and innovation. The integration of LLMs into creative processes not only expands the scope of computational tools available to musicians but also fosters interdisciplinary collaborations between computer science and artistic endeavors [5].

## B. Related Work

The application of AI in music composition has evolved significantly, encompassing diverse methodologies and technologies. Early efforts centered on employing predefined rules and algorithms to generate musical sequences. Recent advancements have leveraged machine learning techniques, particularly deep learning models, to analyze and synthesize music. Deep learning, specifically recurrent neural networks (RNNs) and convolutional neural networks (CNNs), has emerged as a powerful tool for music generation. These models learn intricate patterns from extensive datasets of musical compositions. Projects like Google's Magenta and Sony's Flow Machines are prominent applications of deep learning in producing music ranging from simplistic melodies to complex compositions [6], [7]. Transformer architectures, such as OpenAI's MuseNet and Jukedek, represent a notable advancement in music composition using AI. These models excel in capturing long-range dependencies within musical sequences, enabling the generation of diverse and cohesive musical pieces across various styles and genres [8], [9].

While there has been extensive research on generating music using AI, the specific use of JSON and MIDI formats in conjunction with LLMs is relatively new. Previous works have explored the generation of MIDI files directly from neural networks, but the integration of JSON as an intermediate representation for music generation adds a new dimension of flexibility and control. Research by Huang et al. (2018) on Music Transformer [10] and by Donahue et al. (2019) on LakhNES [11] have focused on generating MIDI files directly from AI models. These works highlight the feasibility of using deep learning models to create musically coherent MIDI sequences. JSON's use in music representation remains underexplored but will be further utilized for structured and detailed musical composition.

The application of LLMs in creative domains, such as text generation and code synthesis, has paved the way for their use in music composition. LLMs can generate creative content, including poetry, stories, and even programming code, leveraging their understanding of contextual prompts and structural nuances [12]. OpenAI's Codex model, a descendant of GPT-3, has demonstrated the capability to generate code snippets based on natural language descriptions. This ability is directly applicable to generating music-related code, such as scripts for creating MIDI files or JSON representations of musical pieces [13]. Projects like OpenAI's MuseNet have showcased that LLMs can generate music by leveraging their understanding of musical structure and styles. MuseNet can create compositions in various genres, demonstrating the potential of LLMs to contribute to music generation tasks [14].

## III. METHODOLOGY

### A. Generation Methods

We explored three generation methods for producing MIDI scores in JSON format. Specifically, we look into Direct Generation, Code Generation, Rich Code Generation. Each

of these approaches demonstrate different design choices in structuring an LLM-powered generative system.

Direct Generation involves producing MIDI scores in a JSON format that can be directly utilized without further transformation or interpretation. This method offers the advantage of providing immediate output without intermediary steps, thereby reducing the potential for errors that could arise during format translation. However, it comes with limitations, such as restricted post-generation adjustment capabilities, which may hinder fine-tuning and adaptation to specific musical requirements. Ensuring the quality of output may also necessitate more rigorous evaluation processes to validate correctness and musical integrity.

Code Generation refers to the production of executable code, such as JavaScript, which generates MIDI scores. This approach provides enhanced control over customization and manipulation of musical elements post-generation. Modifications to the generated code can be made to optimize performance or tailor compositions to different stylistic preferences. Nevertheless, it introduces the risk of errors in code execution, particularly if scripts are not meticulously crafted or if unexpected issues arise during runtime. Proficiency in coding is essential to effectively modify scripts and ensure they meet the desired musical criteria.

Rich Code Generation involves the creation of sophisticated scripts that incorporate complex data structures and detailed instructions for generating intricate musical pieces. This method excels in providing a higher level of musical nuance and detail, facilitated by annotations that enhance understanding and modification of the generated code. Despite this, the increased complexity in both generation and execution phases demands substantial computational resources. Execution may require advanced hardware capabilities to handle the processing demands of complex algorithms and extensive musical data structures effectively.

## B. Evaluation Metrics

Correct measures the adherence of generated MIDI pieces to predefined correctness criteria, assigning a binary value (0 or 1) based on whether the output meets these criteria. This metric provides a straightforward method for comparing the quality of outputs across a dataset, ensuring consistency in evaluation. However, subjectivity in defining what constitutes "correctness" can introduce variability in ratings, impacting the reliability of assessments. Moreover, Correct does not consider the compilation success of generated music, focusing solely on qualitative aspects. It is best suited for scenarios with well-defined criteria and minimal subjective bias, where multiple evaluators can validate adherence to compositional rules or intentions.

Compiled refines the Correct metric by evaluating only those MIDI pieces that compile successfully into executable outputs, assigning a binary value (0 or 1) based on compilation success. This metric enhances practical usability assessment by ensuring that evaluated pieces are not only correct but also executable in real-world applications. However, Compiled

may overlook high-quality pieces that fail compilation due to minor, non-musical errors, potentially diminishing its utility in scenarios where frequent compilation issues occur without informative value. It remains ideal for evaluating the practical execution of music alongside its aesthetic or structural quality.

Complexity quantifies the intricacy of generated music by measuring the number of individual notes in the MIDI representation. This metric offers an objective measure of musical detail and variation, which correlates with the creative richness of compositions. However, it is important to note that high complexity does not always equate to superior quality or musicality; excessively complex pieces can be chaotic and less aesthetically pleasing. Therefore, Complexity is best utilized in conjunction with other metrics to ensure that increased complexity enhances rather than detracts from the overall musical experience.

Each metric contributes a distinct perspective to the evaluation of generated MIDI music, collectively forming a comprehensive assessment framework. Correct serves as a foundational criterion, assessing alignment with compositional rules or intentions. Compiled focuses on practical usability by evaluating executable outputs, ensuring feasibility in real-world applications. Complexity adds depth by quantifying creative richness, highlighting the intricacy and variation within compositions. Together, these metrics provide a nuanced understanding of the strengths and limitations of generation models, offering insights into both qualitative and practical aspects of generated MIDI music.

## C. Evaluation Process

We used three specific prompts to generate MIDI scores in JSON format. Each prompt was tested across 10 trials:

Prompt 1: Generate MIDI scores in JSON notation that can be played back by Tone.js

This baseline prompt tests the model's ability to generate MIDI scores compatible with Tone.js, evaluating its most fundamental capability for practical application in web-based music playback and interactive environments.

Prompt 2: Generate a MIDI JSON Bach chorale adhering counterpoint rules with four voices following a chord progression and counterpoint harmony rules

This prompt assesses the model's skill in generating standard four-part structure adhering to strict counterpoint rules, gauging its proficiency in classical music composition.

Prompt 3: Generate a MIDI JSON complex piano piece with two hands with syncopated rhythms and varied notes

This prompt challenges the model to create more sophisticated, multi-layered music, demonstrating its capacity for intricate musical composition and rhythmic complexity.

Each of the three prompts was executed 10 times using each generation method. This means there were a total of 30 trials per method, resulting in 90 trials in total. For each trial:

- 1) The generation method was applied to the prompt.
- 2) The output MIDI score in JSON format was evaluated based on predefined metrics (correctness, compilation success, and complexity).
- 3) The results were recorded and analyzed to determine the performance and consistency of each method.

For generating the MIDI scores, we used the OpenAI API’s gpt-3.5-turbo model for all three methods [1].

Our evaluation methodology investigates the generation of MIDI scores using LLMs across three methods: Direct Generation, Code Generation, and Rich Code Generation. We evaluated the generated outputs using the Correct, Compiled, and Complexity metrics, providing a comprehensive analysis of the effectiveness and intricacy of the generated music and code. All evaluation results are derived from averaging the scores obtained across 10 trials to provide a comprehensive analysis of the musical and coding outputs.

#### IV. RESULTS

In this evaluation, we measure three dimensions of our generational approach to assess our system, namely Correct, Compiled, and Complexity, along with a new metric we introduced, Conceptual Correctness, to tackle shortcomings.

##### A. Correct

Correct	Generation Methods		
	Direct Generation	Code Generation	Rich Code Generation
Prompt 1	0.5	0.9	0.2
Prompt 2	0.7	0.6	0
Prompt 3	0.7	0.5	0

Figure 1. Comparison of calculated results for the three generation methods for each of the three prompts for Correct benchmark.

Direct Generation shows moderate consistency across prompts, achieving scores ranging from 0.5 to 0.7. Code Generation performs well with a high score of 0.9 for one prompt but exhibits variability, scoring lower on others (0.6 and 0.5). Rich Code Generation consistently scores lower across all prompts (0.2, 0, and 0), indicating challenges in producing musically accurate outputs using complex data structures. These results suggest that while Code Generation excels in certain scenarios, all methods require refinement to enhance overall correctness in MIDI score generation.

##### B. Conceptual Correctness

Traditional correctness metrics for evaluating Rich Code Generation, such as melodic and harmonic coherence, rhythmic consistency, and tonal balance, often fall short and yield very low results when assessing code generated through methods involving randomization and iterative constructs like `Math.random()` and `for` loops. The nature of these data structures can lead to the creation of incoherent and extremely dissonant music, which may not align with traditional musical conventions. For example, when using `Math.random()` to generate pitches and durations, there is no guarantee that the resulting musical phrases will adhere to recognized melodic or harmonic patterns. Similarly, employing `for` loops to iterate

over musical elements may produce repetitive or non-musical sequences that lack expressive or structural integrity.

To address these limitations, we propose a modified metric: Conceptual Correctness. This metric assesses whether the generated code adheres to the compositional instructions involving randomization and iterative structures, rather than focusing solely on traditional musical qualities. Conceptual Correctness ensures that the generated music aligns with the intended compositional approach, evaluating the presence of recognizable musical forms, specified harmonic progressions, rhythmic patterns, and overall coherence and expressiveness.

Consider a prompt instructing the generation of a piece using random pitches within a specified scale and repeated rhythmic patterns via `for` loops. Traditional metrics might rate the output poorly due to lack of harmonic progression or melodic development. However, Conceptual Correctness would evaluate if the generated code correctly uses randomization to select pitches from the specified scale and employs `for` loops to create the repetitive patterns as described. For instance, if the prompt specifies generating a melody using random pitches from a C major scale and repeating a rhythmic pattern four times, the traditional metrics might penalize the output for not having a clear melodic contour or harmonic resolution. On the other hand, Conceptual Correctness would give a positive score if the code produces notes within the C major scale and correctly implements the repetitive rhythmic structure.

Using the same binary assessment as correct, Conceptual Correctness determines if the generated code qualifies as rich code per the prompt’s intentions. A score of 1 indicates the code meets the criteria for Rich Code Generation, while a score of 0 indicates it does not. This metric prioritizes procedural fidelity and structural guidelines over traditional musical aesthetics, providing a nuanced evaluation of Rich Code Generation methods.

Experiments were conducted in the same manner as the other metrics to compare the performance of Rich Code Generation using both traditional and Conceptual Correctness metrics. MIDI scores were generated based on three prompts, with 10 trials for each prompt, and results were obtained through averaging the 10 trials.

Prompt	Complexity
Prompt 1	0.6
Prompt 2	0.8
Prompt 3	1

Figure 2. Comparison of calculated Conceptual Correctness results for each of the three prompts.

The results presented in Figure 2 underscore the need for a more context-sensitive approach, such as Conceptual Correctness, to effectively evaluate Rich Code Generation techniques.

##### Example 1: Passing Conceptual Correctness

Prompt: Generate MIDI scores in JSON notation that can be played back by `Tone.js`

Code:

```
function generateMIDIScore() {
  const tracks = [];
  for (let i = 0; i < 4; i++) {
    const notes = [];
    for (let j = 0; j < 4; j++) {
      notes.push({
        pitch: Math.floor(Math.random() * 1000)
          + 100,
        duration: Math.floor(Math.random() * 4)
          + 1,
        startTime: Math.floor(Math.random() * 4)
      });
    }
    tracks.push({ notes });
  }
  return { tracks };
}

generateMIDIScore();
```

This example passes the Conceptual Correctness criteria because it adheres to the prompt’s instructions by using `for` loops and `Math.random()` to generate the MIDI score. The generated code reflects the intended use of iterative constructs and randomization to produce musical elements, demonstrating rich Code Generation as specified.

Example 2: Failing Conceptual Correctness

Prompt: Generate a MIDI JSON Bach chorale following counterpoint rules with four voices following a chord progression and counterpoint harmony rules

Code:

```
function generateMIDIScore() {
  const midiScore = {
    tracks: [
      {
        notes: [
          { pitch: Math.floor(Math.random() *
            (1000 - 100)) + 100, duration:
            Math.floor(Math.random() * 4) + 1,
            startTime: 0 },
          { pitch: Math.floor(Math.random() *
            (1000 - 100)) + 100, duration:
            Math.floor(Math.random() * 4) + 1,
            startTime: 1 },
          // ... more notes for voice 1
        ]
      },
      ...
    ]
  };
}

generateMIDIScore();
```

This example fails the Conceptual Correctness criteria because the segment of the generated code shown below does not fully adhere to the prompt’s instructions. Although it generates a MIDI score with four voices, it does not follow counterpoint rules, chord progressions, or harmony rules as specified. Additionally, it lacks the use of iterative constructs like `for` loops to generate the musical elements, and the comment within the

code indicates incomplete implementation. Thus, it does not meet the criteria for rich Code Generation as intended by the prompt.

### C. Compiled

Compiled	Generation Methods		
	Direct Generation	Code Generation	Rich Code Generation
Prompt 1	0.8	0.9	0.8
Prompt 2	0.7	0.6	0.9
Prompt 3	0.7	0.6	1

Figure 3. Comparison of calculated results for the three generation methods for each of the three prompts for Compiled benchmark.

Prompt 3 achieves the highest scores across all methods, with Rich Code Generation achieving a perfect score of 1, indicating optimal compilation success in generating MIDI scores. Prompt 2 also performs well, particularly with Rich Code Generation, achieving a score of 0.9, highlighting the method’s capability to consistently produce correctly compiled outputs. Prompt 1 shows strong performance overall, with Code Generation achieving the highest score of 0.9, closely followed by Direct Generation and Rich Code Generation at 0.8. These results showcase the effectiveness of Rich Code Generation in ensuring accurate compilation of MIDI scores across diverse compositional prompts.

### D. Complexity

Complexity	Generation Methods		
	Direct Generation	Code Generation	Rich Code Generation
Prompt 1	6.3	6.4	13.5
Prompt 2	16.7	15.7	40.4
Prompt 3	12	9.4	52.4

Figure 4. Comparison of calculated results for the three generation methods for each of the three prompts for Complexity benchmark.

The results indicate varying levels of complexity generated by each method for different prompts. Rich Code Generation consistently produces the most complex musical outputs across all prompts, with scores of 13.5, 40.4, and 52.4 for Prompts 1, 2, and 3, respectively. This method leverages detailed instructions and complex data structures to introduce nuanced musical elements, resulting in compositions that exhibit higher levels of intricacy and sophistication. In contrast, Direct Generation and Code Generation exhibit lower complexity scores, indicating less intricate musical compositions. These findings show the capability of Rich Code Generation to generate musically complex outputs, demonstrating its potential for creating sophisticated musical compositions that align closely with predefined complexity metrics.

### E. Evaluations

The Direct Generation approach revealed several recurring issues in the generated sequences. Length was a significant concern, as sequences were often excessively short, typically spanning only four beats. Harmonic consistency was generally maintained within a C major chord framework, with both voices playing scales—one starting from C and the other from G—creating a cohesive harmonic structure. However,

these sequences occasionally diverged from traditional chord patterns. Voice interaction showed that while the voices generally complemented each other, note durations were overly prolonged, limiting sequences to just four notes. Chords from common keys, such as C major and E minor, were used, with note durations of either one or two beats. Yet, there were problems with numerical pitches leading to odd-sounding low-pitched notes. A consistent duration and pitch range with increasing pitch frequency were observed, but sequences often deviated from typical chord patterns, using a scale structure with random pitch increments. The chord progression typically followed a I-I-ii-V pattern in C major, although some deviations, like moving from a C major chord to an A minor chord, introduced interesting tensions but were not always diatonic to the key.

The evaluation of Code Generation focused on both functionality and musicality. Functionally, the code reliably ran a function containing the MIDI score in JS format, playable in Tone.js, and returned the object for playback. Musically, a simple C major chord structure was common, with one voice playing an ascending C major scale and the other a descending F dominant 7th chord. Despite moments of harmonic dissonance, interactions between voices generally resonated, though sustained note durations restricted sequences to merely four notes. Some sequences included a 2+ octave scale with varied note durations (0.5, 1, 1.5, 2 beats), and voices often used notes from common chords with C major as the tonic. The generated sequences displayed a I-I6-V-IV7 chord progression in C major, but these progressions sometimes resulted in dissonance and clashing chords. Additionally, some compositions were notably shorter, often containing only one chord.

Rich Code Generation produced more complex and varied results but also introduced several challenges. The generated melodies were often extremely dissonant, with no variation in note duration or discernible trends in pitches, a dissonance that persisted in both singular and dual melodies. The pitch range could be altered through prompt engineering, particularly the numerical ranges used. The use of for loops and Math.random() enabled more complex and longer compositions. Despite this complexity, the melodies often lacked a common theme, with two voices failing to harmonize. Some sequences started with dissonant melodies but branched into multiple voices, showing moderate success in replicating two-voice structures typical in piano music. There was minimal variation in note duration, and no clear trends or structures in the pitches. In some instances, the notes' durations were strictly syncopated, adding rhythmic interest but maintaining harmonic dissonance. The generated sequences did not accurately represent the structure of classical music or Bach chorales, as they lacked the expected harmonic and melodic patterns.

#### F. Summary of Findings

The analysis reveals significant performance variation across generation approaches. Direct Generation produced cohesive sequences but struggled with length and pitch. Code Gen-

eration, while functional, often resulted in harmonic clashes and simplistic structures. Rich Code Generation provided complexity and length but compromised musicality and coherence. The results the need to evaluate multiple dimensions of generated code, including harmonic consistency, melodic structure, duration, and pitch range, to develop robust and efficient code generation systems.

### V. DISCUSSION

In this section, we explore the potential applications of our findings to the development of LLM-assisted tools for music generation and composition, and outline future research directions in this domain.

#### A. Enhancing Music Composition Tools

The insights offered from our evaluation of LLM-generated music sequences can significantly enhance existing music composition tools. By understanding the strengths and limitations of different generation approaches, developers can create more robust and user-friendly interfaces that empower composers to produce music more efficiently and creatively. For instance, integrating direct generation methods can provide users with quick and intuitive means to generate musical ideas, while incorporating code generation techniques can offer more granular control over the composition process [15].

#### B. Exploring New Musical Styles and Genres

The flexibility and adaptability of LLMs open exciting possibilities for exploring new musical styles and genres. Training models on diverse datasets that span different musical traditions and cultures can foster cross-cultural exchange and collaboration, potentially leading to the emergence of innovative hybrid genres and experimental compositions. Moreover, incorporating user feedback and preferences into the training process can enable LLMs to adapt and evolve over time, reflecting changing trends and tastes in the music industry [16].

#### C. AI Music Cognition

AI in music within therapeutic contexts offer personalized interventions by analyzing patients' responses to music, enhancing treatment for conditions like anxiety and depression. Real-time adaptation based on biometric data optimizes therapy effectiveness, while expanding accessibility through mobile platforms benefits remote and immobile populations. AI complements traditional methods by providing data-driven insights and improving therapist-patient interactions [17].

#### D. Future Directions

Several promising avenues for future research in LLM-assisted music generation can be identified. First, utilizing JSON as an intermediary step in music generation can enhance control over the composition process, incorporating domain-specific knowledge and optimizing JSON-to-MIDI conversion to generate music that transcends conventional stylistic boundaries. Integrating additional modalities such as lyrics, images, and videos can enrich the music generation process, creating

more immersive and interactive user experiences. Developing collaborative composition tools that allow multiple users to interact with LLM-generated content in real-time can facilitate remote collaboration among musicians and composers. Addressing ethical considerations such as bias, fairness, and cultural representation is crucial to ensure that LLM-generated music is inclusive and respectful of diverse communities and perspectives. Lastly, engaging end-users, including musicians, composers, educators, and music enthusiasts, in the design and evaluation of LLM-assisted tools is essential for creating solutions that meet their needs and preferences [18], [19], [20].

By pursuing these research directions and fostering interdisciplinary collaborations between experts in machine learning, musicology, cognitive science, and human-computer interaction, we can unlock the full potential of LLMs to revolutionize music creation and appreciation in the digital age.

#### REFERENCES

- [1] OpenAI, “Gpt-3.5 turbo,” <https://beta.openai.com/docs/guides/gpt>, 2021.
- [2] D. M. Huber, *Modern Recording Techniques*, 9th ed. Routledge, 2020.
- [3] “Json: Javascript object notation,” Available at <https://www.json.org/>, accessed: Date accessed.
- [4] A. Baratè and other authors, “Advances and perspectives in web technologies for music representation,” *DigitCult-Scientific Journal on Digital Cultures*, vol. 1, no. 2, pp. 1–18, 2016.
- [5] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, J. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, “Language models are few-shot learners,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 18 714–18 732, 2020.
- [6] A. DuBreuil, *Hands-On Music Generation with Magenta: Explore the Role of Deep Learning in Music Generation and Assisted Music Composition*. Packt Publishing, 2020.
- [7] F. Ghedini, F. Pachet, and P. Roy, “Creating music and texts with flow machines,” in *Multidisciplinary Contributions to the Science of Creative Thinking*, ser. Creativity in the Twenty First Century, G. Corazza and S. Agnoli, Eds. Singapore: Springer, 2016. [Online]. Available: [https://doi.org/10.1007/978-981-287-618-8\\_18](https://doi.org/10.1007/978-981-287-618-8_18)
- [8] T. Sexton, “MuseNet,” *Music Reference Services Quarterly*, vol. 26, no. 3–4, pp. 151–153, 2023. [Online]. Available: <https://doi.org/10.1080/10588167.2023.2247289>
- [9] D. Herremans and C.-H. Chuan, “The emergence of deep learning: new opportunities for music and audio technologies,” *Neural Computing and Applications*, vol. 32, pp. 913–914, 2020. [Online]. Available: <https://doi.org/10.1007/s00521-019-04166-0>
- [10] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, N. Shazeer, C. Hawthorne, A. M. Dai, M. D. Hoffman, and D. Eck, “Music transformer: Generating music with long-term structure,” in *Proceedings of the International Conference on Machine Learning (ICML)*, 2018, pp. 2410–2419.
- [11] F. Colombo, N. Chen, C. Donahue, Z. Li, and B. Pardo, “Lakhnes: Improving multi-instrumental music generation with cross-domain pre-training,” *Transactions of the International Society for Music Information Retrieval (TISMIR)*, vol. 2, no. 1, pp. 97–108, 2019.
- [12] L.-C. Lu *et al.*, “Llm discussion: Enhancing the creativity of large language models via discussion framework and role-play,” *arXiv preprint arXiv:2405.06373*, 2024.
- [13] OpenAI, “Openai codex,” <https://openai.com/index/openai-codex/>, 2021.
- [14] OpenAI, “Openai musenet,” <https://openai.com/index/musenet/>, 2019.
- [15] Y. Zhang, “Utilizing computational music analysis and ai for enhanced music composition: Exploring pre- and post-analysis,” *Educational Administration: Theory and Practice*, vol. 30, no. 5, pp. 269–282, 2024. [Online]. Available: <https://doi.org/10.53555/kuvey.v30i5.2841>
- [16] E. V. Epure, G. Salha, M. Moussallam, and R. Hennequin, “Modeling the music genre perception across language-bound cultures,” in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, B. Webber, T. Cohn, Y. He, and Y. Liu, Eds. Online: Association for Computational Linguistics, Nov. 2020, pp. 4765–4779. [Online]. Available: <https://aclanthology.org/2020.emnlp-main.386>
- [17] J. Sun, J. Yang, G. Zhou, Y. Jin, and J. Gong, “Understanding human-ai collaboration in music therapy through co-design with therapists,” in *Proceedings of the CHI Conference on Human Factors in Computing Systems*, 2024.
- [18] S. Oramas and et al., “Multimodal deep learning for music genre classification,” *Transactions of the International Society for Music Information Retrieval*, vol. 1, no. 1, pp. 4–21, 2018.
- [19] M. Newman, L. Morris, and J. H. Lee, “Human-ai music creation: Understanding the perceptions and experiences of music creators for ethical and productive collaboration,” in *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, 2023.
- [20] E. Kaasinen *et al.*, “A user-centric view of intelligent environments: User expectations, user experience and user role in building intelligent environments,” *Computers*, vol. 2, no. 1, pp. 1–33, 2012.

#### APPENDIX

Prompt	Trial	Correct		
		Direct Generation	Code Generation	Rich Code Generation
Prompt 1	Trial 1	1	1	0
	Trial 2	0	1	0
	Trial 3	0	0	0
	Trial 4	1	1	0
	Trial 5	1	1	0
	Trial 6	0	1	1
	Trial 7	1	1	1
	Trial 8	0	1	0
	Trial 9	0	1	0
	Trial 10	1	1	0
Prompt 2	Trial 1	1	1	0
	Trial 2	1	0	0
	Trial 3	0	0	0
	Trial 4	1	1	0
	Trial 5	0	0	0
	Trial 6	1	1	0
	Trial 7	1	1	0
	Trial 8	1	1	0
	Trial 9	0	0	0
	Trial 10	1	1	0
Prompt 3	Trial 1	0	1	0
	Trial 2	1	1	0
	Trial 3	0	0	0
	Trial 4	1	1	0
	Trial 5	1	1	0
	Trial 6	1	0	0
	Trial 7	1	0	0
	Trial 8	1	0	0
	Trial 9	0	0	0
	Trial 10	1	1	0

Figure 5. Table of trials for the three prompts and ten trials each for Correct benchmark.

Prompt	Trials
Prompt 1	0
	1
	0
	1
	1
	1
	0
	1
	0
	1
Prompt 2	1
	1
	1
	1
	0
	1
	1
	0
	1
	1
Prompt 3	1
	1
	1
	1
	1
	1
	1
	1
	1
	1

Figure 6. Table of trials for the three prompts and ten trials each for Conceptual Correctness benchmark.

Prompt	Trial	Direct Generation	Code Generation	Rich Code Generation
Prompt 1	Trial 1	1	1	1
	Trial 2	0	1	1
	Trial 3	1	0	0
	Trial 4	1	1	1
	Trial 5	1	1	1
	Trial 6	1	1	1
	Trial 7	1	1	1
	Trial 8	0	1	0
	Trial 9	1	1	1
	Trial 10	1	1	1
Prompt 2	Trial 1	1	1	1
	Trial 2	1	0	1
	Trial 3	0	0	1
	Trial 4	1	1	0
	Trial 5	0	0	1
	Trial 6	1	1	1
	Trial 7	1	1	1
	Trial 8	1	1	1
	Trial 9	0	0	1
	Trial 10	1	1	1
Prompt 3	Trial 1	0	1	1
	Trial 2	1	1	1
	Trial 3	0	0	1
	Trial 4	1	1	1
	Trial 5	1	1	1
	Trial 6	1	1	1
	Trial 7	1	0	1
	Trial 8	1	0	1
	Trial 9	0	0	1
	Trial 10	1	1	1

Figure 7. Table of trials for the three prompts and ten trials each for Compiled benchmark.

Prompt	Trial	Direct Generation	Code Generation	Rich Code Generation
Prompt 1	Trial 1	6	4	6
	Trial 2	5	6	3
	Trial 3	6	8	12
	Trial 4	8	8	6
	Trial 5	8	6	22
	Trial 6	6	6	16
	Trial 7	4	8	5
	Trial 8	6	7	32
	Trial 9	8	6	24
	Trial 10	6	5	9
Prompt 2	Trial 1	16	16	64
	Trial 2	16	25	64
	Trial 3	16	16	16
	Trial 4	16	16	12
	Trial 5	23	8	64
	Trial 6	16	16	32
	Trial 7	16	16	8
	Trial 8	16	16	16
	Trial 9	16	12	64
	Trial 10	16	16	64
Prompt 3	Trial 1	24	16	100
	Trial 2	10	8	32
	Trial 3	24	8	32
	Trial 4	8	14	16
	Trial 5	12	8	32
	Trial 6	9	10	32
	Trial 7	8	8	16
	Trial 8	8	10	200
	Trial 9	8	4	32
	Trial 10	9	6	32

Figure 8. Table of trials for the three prompts and ten trials each for Complexity benchmark.