Lexical Lullabies: LLM-Based MIDI Generation with Music Theory Constraints

Christian King
christian.king2@ucf.edu
University of Central Florida
Orlando, Florida, USA

ABSTRACT

Despite many recent advancements in LLMs and sequence modeling, there seems to exist only one text-to-MIDI model in the literature [2]. MIDI is a quantized musical interface file type that contains all the same features that one might see on a piece of sheet music. This allows computers to play and produce music through digital instruments much in the same way a musician might. This lack of text-to-MIDI models is largely due to a lack of data. This problem has since been resolved by datasets like MIDICaps [5]. This project creates a text-to-MIDI model using Llama 3.2 as a base with additional training on MIDICaps data. By learning from MIDI rather than audio the model is forced to learn hard music theory constraints like tempo, key, and chord progression as these are implicitly stated in the MIDI format. While the model trained in this paper is technically the second text-to-MIDI model trained on MIDICaps, it is however the first decoder-only LLM text-to-MIDI model. Though only 12% of model outputs passed all grammar checks (start/end markers, pitch, velocity, duration), this marks the first time a decoder-only LLM has demonstrated success in any capacity in MIDI generation from language prompts.

Project With Data (1.7 GB Compressed; 6.5 GB Extracted):
`Click here for Google Drive link`

Project Code Without Data (12.8 KB):
`Click here for Google Drive link`

1 INTRODUCTION & PROBLEM STATEMENT

Music generation is not just a niche technical task or LLM party trick; it opens the door for an entirely new way to interact with and create music. It holds the potential to give the fun of music composition to everyone, musically inclined or not. More than this, it also furthers our understanding of AI. Through these experiments, researchers gain directly transferable insights that can be used in domains like text-to-speech, multi-modal NLP, signal processing, and more. Understanding language and its relationships to different modes like music is huge and holds a lot of potential.

This interest in bridging language and music isn't new. Music generation from textual descriptions has been an active subfield of NLP for a long time, with a substantial number of publications since the introduction of the transformer. Most research in this area focuses on teaching music composition in an unsupervised manner with very few constraints. The benefits of this approach are:

(1) The LLM learns common rules and musical techniques on its own.
(2) The LLM remains highly creative, introducing its own take on music and composition.

However this approach is also not without its disadvantages. Music is very quantized in pitches and rhythm and without forcing these constraints, many of these raw audio, unsupervised models violate traditional music rules creating off beat or out of key music. This isn't typically the desired result.

As such this project takes a different approach by enforcing stricter music theory constraints. This is done primarily through the use of MIDI which is inherently stricter than a raw audio file. Instead of the earlier mentioned model that learns from raw audio and produces audio, this project will learn from MIDI and produce MIDI. MIDI is a quantized musical interface file that contains all the same features that one might see on a piece of sheet music. This means key, tempo, notes, etc. all very clearly defined. Moreover it makes it much more difficult for the model to violate music theory constraints by giving it a language/format that inherently enforces them.

The motivation behind this enforcement is the belief that musical preference is driven less by novelty and more by familiarity. By enforcing familiar rules and structures, an LLM can produce music that feels familiar and yet new, ultimately maximizing user enjoyment. The significance of this project's success is in its ability to help composers and musicians more easily write and generate new music. Moreover, it might help move the needle towards a future in which all music is generated and adapted to maximize the specific user's enjoyment. Lastly, this structure of text-to-MIDI modeling is new and could pave the way for a new approach text-to-speech modeling or similar application.

The three core objectives of this paper are:

(1) Generate MIDI files from textual description using Llama 3.2 trained on MIDICaps Dataset [5]. At the time of writing this, April 2025, there exists only one text-to-MIDI model so this will be the second and takes on a completely different approach from the other.
(2) Show how music theory constraints on music generation such as key, tempo, chord progression, etc. by explicit tokenization creates different results from previous approaches (unsupervised, text-to-audio, etc.).
(3) Provide a comprehensive set of quantitative results using BLEU-4 and ROUGE-L scores, and evaluate syntactic MIDI validity as a measure of the model's grammatical understanding. Discuss performance trends through qualitative inspection and example outputs.

2 RELATED WORKS

Music generation has been attempted many times with many different approaches over the last two decades. Early implementations (2000s) using evolutionary algorithms, more recent solutions (2010s)

being rule based CNNs, and the new cutting edge approaches (2019-present) being transformer based. The two most well known of these transformer based models are likely Google's MusicLM and OpenAI's MuseNet. While the details of MuseNet's architecture were never released, MusicLM's paper is quite extensive and will act as a reference point for much of this paper's implementation.

Google's MusicLM [1] is technically a suite of different models with the primary model being the one most similar to this project, text to audio model. This model is transformer based, taking raw audio, tokenizing it, then learning from that data. Moreover MusicLM uses Google's MuLan embeddings, a system Google developed for their speech to text models. While these projects share many similarities, this project differs from MusicLM as instead of raw audio, it learns from MIDI, instead of MuLan, it uses REMI, instead of a full transformer, it is decoder-only, and instead of an audio heavy dataset, the dataset used in this project is small audio files and frequent text descriptions.

MidiCaps [5] is the dataset used in this project. At the time of writing this it appears the MidiCaps is the largest dataset of its kind by a significant margin. Moreover the MidiCaps paper claims that as of June 2024, no text-to-MIDI models exist, steelmanning the efficacy of this project. This dataset contains 168,000 text-MIDI pairs, annotated by a clever pipeline of preprocessing and feature extraction in conjunction with Claude-3.

Llama 3 [3] is a decoder-only model made by Meta. Unlike similar decoder-only models such as GPT, Claude, and Gemini, Llama is open weight and has far fewer parameters. This means that downloading and performing finetuning of this model locally is possible. Moreover, it makes it the perfect foundational architecture for this project since it already is pretrained on a large corpus of language. This means the only additional objective is tokenizing and mapping meaningful MIDI embeddings.

3 METHODOLOGY (TECHNIQUE & APPROACH)

The overall approach can be divided into four parts: preprocessing, training, inference, and postprocessing/evaluation. The flow can be seen in greater detail in Figure 1.

3.1 Preprocessing:

3.1.1 Dataset Preprocessing: This model was exclusively trained on MIDICaps [5]. MIDICaps consist of 168,000 MIDI and text description pairs. This dataset was chosen as it is the largest and best formatted dataset for this purpose at the time of this experiment. In order to process them, the files were decompressed from many .tar formats to a single .zip then uploaded to Google Drive, transferred to Colab, then extracted locally in Colab. This pipeline allowed for efficient transfer and reading of the data as all other methods took too long or overloaded the file system due to the massive number of data files in this set.

Additionally, right after initial extraction, the separate MIDI and text paired files were merged into single files and a <MIDI> token was inserted between them to divide the string. This was done to make training easier, as this is a decoder-only model, so there will be only a single input.

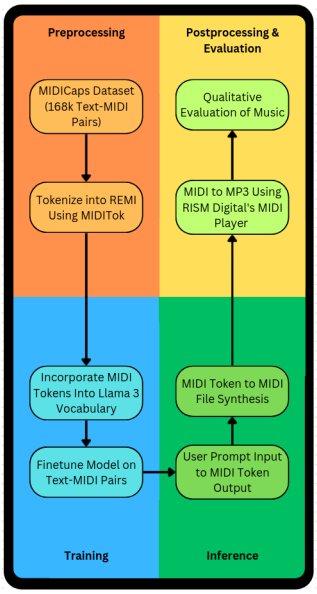

Figure 1: Overview of the Lexical Lullabies approach to training a text-to-MIDI model.

Due largely to computational constraints, this model was trained on a very unorthodox data split. Due to the large dataset and the need for regular validation, the validation set only contained 500 samples from the dataset. The test set however was given 10k samples as to get a better test post training. This was done to cut down on computation time and is discussed further in the training section.

Also due to computational limits all data sampled were fixed to 512 tokens. In cases where they were less, the input was padded. Though in the vast majority of cases the input was much longer so the data was just truncated at 512.

Table 1: Examples of MIDI Tokens in REMI Format

Token Type	Example
Bar Marker	Bar_1
Position Index	Position_0
Pitch Event	Pitch_64
Velocity Level	Velocity_80
Duration Value	Duration_4

3.1.2 Tokenization: The above table shows a few examples of MIDI tokens in REMI format. MIDI files when encoded in REMI are formatted mostly using English words. For example, a MIDI file might contain: ["Bar_1", "Position_1", "Pitch_100"]. This presents a serious problem to an LLM that has been trained on English text, so to prevent ambiguity and confusion, it is much easier to tokenize these MIDI sequences into their own vocabulary by expanding the vocabulary of the model.

To do this, the Python library MIDITok was used [4]. MIDITok using the REMI scheme converts MIDI into a series of tokens. The

model's tokenizer is then expanded to recognize this new REMI vocabulary.

In addition to the REMI vocabulary, three extra tokens were added to the tokenizer: "<MIDI_START>", "<MIDI_END>", "<MIDI>". Recall that the model being trained, Llama, is a decoder-only model. As such the text description and the MIDI are passed as a single string of tokens. MIDI start and end tokens define where the MIDI sequence begins and ends. Moreover, the MIDI token is used to divide the two sections. The reason this additional dividing token is used is to make multiple phases of curriculum training easier. That is explained in more detail in the training section.

3.2 Training:

3.2.1 Model Architecture: The meta-llama/Llama-3.2-1B model was chosen for this project largely because its one of the best performing LLMs within this parameter range and it has been pretrained on English text. This gave the benefit of only having to perform finetuning in order to teach the model the relationships between English text and MIDI tokens. This is also the primary justification for decoder-only: by using the Llama model, pretraining can be leveraged which greatly saves on compute as the only training performed is finetuning for MIDI comprehension.

3.2.2 Fine-tuning Approach: The actual training of the model was performed in two phases. The first phase was 1 epoch and pure MIDI data as input. This phase was done as a form of curriculum training so that the model could be introduced to MIDI structure prior to learning its relationship to English text. This phase was notably shorter than phase two, but seemed to significantly help the model in early training.

Phase 2 of the training, captions were rejoined with their MIDI counterparts. This training was done for 4 epochs resuming from the final weights of phase 1. In this phase, the model learned to build relationships between English and the MIDI tokens. This is the phase where the majority of the learning was done.

3.2.3 Hyperparameters:

(1) Learning Rate: 2e-4 - For both phases of this training a learning rate of 2e-4 was used. This was decided on by trial and error and general Llama 3 training ranges.
(2) Weight decay: 0.01
(3) Warmup Steps: 500 - The learning rate also was given a 500 step warmup period and a weight decay of 0.01. This was to prevent large jumps in the very early steps of learning, when the loss is very high.
(4) Max gradient norm: 1.0 - Gradient clipping was applied to prevent overshooting minima and exploding gradients while the learning rate is high.
(5) Optimizer: AdamW - Standard for LLM training.
(6) Batch size: 32 - Technically the batch size was 8, but with 4 gradient accumulation steps, so it was essentially a batch size of 32. This was chosen based off GPU limits.

All experiments were run on a single NVIDIA L4 20GB GPU with roughly 20 GPU-hours per total model training.

3.2.4 Loss Function: The loss function is a cross entropy loss, but the model has been pretrained on English text, so the MIDI is the part that is more difficult for the model to learn. To help the model, the loss was divided into a text term and a MIDI term, where the MIDI loss is doubled. This is done to encourage the model to learn more heavily from the MIDI part of the sequence since that is the important part during inference.

This is another part where the aforementioned <MIDI> token becomes useful as this is the token that allows for the creation of a MIDI mask. This MIDI mask denotes which tokens loss should be doubled and which should not.

3.3 Inference:

Once the model is finetuned, it can take in prompts and outputs a sequence of MIDI tokens in the Llama REMI token vocabulary. This prompt is given five times to get a set of samples, then a valid sample is picked randomly from the set. This is done to allow for the chance that certain samples may not follow MIDI grammar perfectly. From there MIDITok is used to decode these tokens into a valid MIDI file. This MIDI is then passed to FluidSynth [6], a MIDI to audio library, to create an mp3 file.

3.3.1 Prompt Engineering: Experiments were done with two different prompting styles: zero-shot and three-shot. In zero-shot a prompt was given, followed by a "<MIDI_START>" token. From there the model generates the remainder of the MIDI file. Three-shot was the same except with three-shot the prompt gave the MIDI three priming tokens that followed the "<MIDI-START>" token. All together such a prompt might look like: "A beautiful, fast, and energized piano melody. <MIDI-START> Program_0 Pitch_60 Duration_8". These tokens not only create a strong signal that the MIDI section has begun, but also breaks the model out of its desire for repetitious MIDI token hacking.

3.3.2 Inference Parameters:

(1) Temperature: 1.0 - This temperature was chosen as a happy medium, as it encourages creativity and reduces repetition while keeping the model on track/producing MIDI tokens only.
(2) P-value: 0.95 - Due to the small number of MIDI tokens in the vocabulary, higher P-values worked better here. However, above 0.95 seemed to increase the chance of invalid tokens.
(3) Max tokens generated: 256 - Longer generations increased the likelihood of mistakes or compounding hallucinations.

3.4 Postprocessing & Human Evaluation:

3.4.1 Postprocessing. The postprocessing is done in large part with the Python library, FluidSynth, which allows for easy MIDI to mp3 conversion. So once a given inference is run, the LLM vocabulary is converted from tokens to MIDI using MIDITok, then MIDI to mp3 using FluidSynth. At that point, the MIDI file can be heard as it is played by virtual instruments.

3.4.2 Human Evaluation. The valid MIDI files processed were listened to by a human. At best the generated files played a single note. Most of them contained valid MIDI information, but didn't have it organized in a way that created any audible music. For example, many files would repeat a token like "velocity_60" and while this is valid, it doesn't produce sound.

3.5 End-to-End Generation Pipeline

After training, the model ideally is capable of taking English language prompts and producing valid, playable music through the following pipeline:

(1) **Prompt Encoding:** The English language description is tokenized alongside a "<MIDI_START>" token and few-shot example tokens.
(2) **Token Generation:** The model generates up to 256 tokens using top-p sampling and repetition penalties. Five outputs are generated per prompt, and the first syntactically valid output is selected.
(3) **REMI Decoding:** Generated tokens are converted back into a MIDI sequence using MIDITok, ensuring format consistency.
(4) **MIDI to Audio Conversion:** Valid MIDI sequences are rendered into audio using FluidSynth, enabling direct playback and inspection.

This full pipeline allows for testing various prompt styles and inference parameters in a practical setting. While most outputs were musically trivial, this approach demonstrates a fully functional language to music generation pipeline that is fully modular, allowing for improvements to be made to the model without change or disruption to the pipeline itself.

4 EVALUATION & RESULTS

4.1 Experimental Setup

The model was evaluated on 10,000 text-MIDI pairs from the test portion of the data split. Inference used the following hyperparameters along with the 3-shot prompt engineering and grammar constraints described in the previous section:

- Temperature: 1.0
- Top-p: 0.95
- Max new tokens: 256
- Repetition penalty: 1.2
- Repeat n-gram penalty size: 3
- Batch size: 16

4.1.1 *Metrics Used:*

- **BLEU-4:** 4-gram precision with clipping and brevity penalty (0–100). This quantifies the exact overlap of short token sequences between generated and reference MIDI.
- **ROUGE-L:** Longest common subsequence based F1 measure between hypothesis and reference (0–100). ROUGE L captures longer patterns, indicating whether or not the model has the ability to reproduce extended MIDI structures.

4.2 Results

Table 3: Performance on 10,000 test samples (three-shot)

Metric	Score
BLEU-4	10.70
ROUGE-L	22.86

4.3 Validity of Outputs

Even slight token shifts can cause big changes in n-gram metrics, so to get a good idea of how many outputs were valid, MIDI grammar checks were run across all 10,000 test samples. Table 4 shows how many outputs passed all validity tests (start and end markers plus at least one pitch, duration, and velocity token).

Table 4: MIDI Grammar Validity on Generated Sequences

Outcome	Count (%)
Valid MIDI (Three-shot)	1185 (11.85%)
Invalid MIDI (Three-shot)	8815 (88.15%)
Valid MIDI (Zero-shot)	0 (0.0%)
Invalid MIDI (Zero-shot)	10000 (100.0%)

Table 2 shows two contrasting cases: a sequence that is syntactically valid but musically trivial, and one that fails basic MIDI grammar checks.

The first example in Table 2 passes all grammar checks. It starts and ends correctly and contains pitch, duration, and velocity tokens, but simply plays a repetitive scale rather than a "funky" groove. The second example illustrates a common invalid case: the missing end marker causes the sequence to fail parsing into a complete MIDI file. This comparison shows that validity is necessary, but not sufficient for quality music outputs.

4.4 Analysis

Table 3 shows the results of the trained model on 10,000 test samples from the dataset. A BLEU-4 score of 10.70 shows that there were a few exact 4-gram matches between generated and reference sequences, but not many. The low ROUGE-L score (22.86) confirms minimal overlap in longer subsequences, suggesting that while the model sometimes captures individual tokens, it fails to consistently reproduce longer, continuous MIDI patterns. Together these scores show that the model is beginning to learn how to map text-to-MIDI, but hasn't learned completely which is likely a testament to just how different these two languages are.

Despite these low scores, it's worth noting that 12% of generated responses passed validity checks. This means that 12% of outputs are within valid MIDI grammar, they just are simply suboptimal. The above text metrics penalize small shifts in token order and do not directly measure whether a sequence is syntactically valid as a MIDI file so this is not reflected.

Moreover, the three-shot prompting setup clearly outperforms zero-shot in terms of MIDI validity, with 11.85% of generations passing the grammar checks compared to just 0% under zero-shot. This gain suggests that providing the model with a few example MIDI tokens not only primes it to recognize the <MIDI_START> and <MIDI_END> tokens, but also helps it internalize basic event sequencing (pitch, duration, velocity) more reliably. In practice, the three-shot context appears to reduce the frequency of truncated or invalid outputs, even though overall musical coherence remains low.

After manually inspecting the output of 100+ model generations, several recurring patterns of failure emerged:

Table 2: Valid but Poor vs. Invalid Generation Examples

Prompt	Generated Sequence (truncated)	Valid	Notes
Simple ascending scale in G major	<MIDI_START> Program_0 Pitch_55 Duration_4 Velocity_64 Bar_1 Position_1 Pitch_57 . . . <MIDI_END>	Yes	Musically trivial
Funky bass groove with swing feel	<MIDI_START> Program_33 Pitch_36 Duration_4 Velocity_80 Bar_None Position_0 Pitch_38	No	Missing <MIDI_END>

- **Stuck token loops:** Sequences like "Pitch_60 Pitch_60 Pitch_60..." were frequently repeated, especially when no repeat n-gram size was low or when temperature was set below 1.0.
- **Floating durations:** Some sequences used durations without surrounding pitch or velocity, leading to silent or musically nonsensical outputs.
- **Truncated endings:** Roughly 10% of valid outputs still ended prematurely, often before a full bar structure was completed.

Understanding and addressing these cases are likely just as important as tuning the architecture or data pipeline. These cases highlight both weaknesses in the training data and limitations in the model's ability to handle long term musical structure. Moreover, these results establish a baseline and highlight clear directions for improvement:

(1) Better data curation and filtering to improve training quality.
(2) Grammar aware modified loss during training.
(3) Use of music theory based metrics like key adherence and rhythmic stability.

Each of these is discussed in further detail in the conclusion (section 6).

5 DISCUSSION & CHALLENGES

5.1 Discussion

While this project never quite achieved its goal of producing beautiful MIDI masterpieces, it does act as a proof of concept. It is clear that with 12% of generated responses being valid, the model is learning something. It's just a matter of how to clean the data and train the model in such a way that it truly learns the new grammar, especially since MIDI grammar is so different from that of any other written language on which the model was pretrained.

Beyond serving as a proof of concept, several insights have emerged. First, it is essential to curate data and check its quality thoroughly before spending money on compute and scratching your head wondering why your loss is so high. Just because a dataset exists does not mean it is of good quality. Moreover, becoming very familiar with the data format beforehand makes it possible to write effective MIDI specific tests. Doing these tests upfront saves a lot of time later.

The second prevailing insight is to be very careful with what gets tokenized. As discussed in the challenges section, tokenizing in the wrong way or not extending the vocabulary far enough can result in catastrophic failure.

The third and final major takeaway is the importance of choosing the correct loss scheme. Initially, training treated the English text data the same as the MIDI data in terms of loss. The oversight was that the model had been pretrained on English but had never seen MIDI. To correct this, two key steps were taken:

(1) Curriculum training was implemented, with the first epoch using only MIDI and text and MIDI pairs introduced from epoch two onward.
(2) A higher loss weight was applied to the MIDI portion of each sample.

These changes pushed the model toward actually learning MIDI instead of defaulting to English text.

Overall, this work was extremely educational and provided enormous insight into how these models learn and how to best approach, train, and work with them. It also lays the first stepping stones towards a successful text-to-MIDI project. Many of the methods used here will undoubtedly contribute to a successful pipeline in the future. While likely only foundational, the pipeline presented in this work has proven to be a solid first step for obtaining valid outputs.

5.2 Challenges

This project idea had a lot of potential but was limited by a variety of challenges. Each issue was new to us, this being our first experience training an LLM, so every problem brought many days of headache and wasted compute. In the end, these experiences proved valuable and made the project worthwhile even if the results were suboptimal. The main challenges were the following:

5.2.1 Data Quality. The MIDICaps 168k text-to-MIDI dataset sounded very promising in the early stages of review. The novelty of its publication and the allure of training the first MIDI text Transformer model were appealing. However, this turned out not to be as perfect as it seemed. While MIDICaps is indeed a very large MIDI dataset, its quality is not optimal. Many of the MIDI files (roughly 3%) are invalid because they do not contain a single note. Many more are simply not of high quality; these files contained only a single note or silence. Together, these issues produced poor training data that went unnoticed for weeks. Once addressed, the invalid files were removed, but filtering out low quality music proved much less straightforward and this is not to mention instances where captions did not match the MIDI file. Overall, the data issues only compounded throughout the project and in the end was likely the largest contributor to the model's poor performance.

Although less severe, the data were initially separated and mapped together via JSON. For the decoder-only model, these needed to be merged and separator tokens inserted around the MIDI section of

every sample. While doable, this process was time consuming and made the dataset less than ideal for the decoder-only approach.

5.2.2 Vocabulary Gaps. MIDI is composed of keywords followed by numbers that correspond with that keyword. For example, "Pitch_60" means that the pitch is set to 60. Initially, the vocabulary added to the LLM included only the keywords: Pitch, Velocity, Duration, etc. This meant the model could not learn the numeric suffixes. This was later corrected by adding a specific token for each valid number in the range (Pitch_1, …, Pitch_120).

5.2.3 Compute Constraints. Due to the size of the MIDICaps dataset and the need for repeated evaluations, training was restricted to relatively short runs with a small model size. The model used, LLaMA 3.2 1B, while accessible and efficient, may simply not have enough capacity to learn both the structure of language and the structure of music in a meaningful way. Larger decoder-only models or encoder-decoder architectures might better capture long term dependencies and complex relationships between text and MIDI. It could very well still be feasible with the model used, but it would have likely been easier with more compute and more parameters. In addition, long generation times during evaluation made it difficult to iterate quickly. This slowed development and limited the ability to fine tune inference parameters or fully explore alternative loss methods.

5.2.4 Token Imbalance. MIDI files do not have even token distributions. Certain tokens like "Pitch_60" often appear at the start of MIDI sequences, and tokens like "Bar_None" are also highly prevalent. This caused a tendency for the model to repeat these tokens in every prompt to achieve a decent loss. Dropout, punishing repeated n grams, and batch normalization helped, but the issue persisted, making it challenging to balance regularization and underfitting.

6 CONCLUDING REMARKS

This work set out with the goal of making the first decoder-only text-to-MIDI model and while it didn't reach anywhere near the performance we had hoped for at the onset, in many ways it still accomplished its goal. With 12% of the outputs being valid, a BLEU-4 score of 10.70, and a ROUGE-L of 22.86, this model shows that this pipeline is really possible. It's clear that a lot more time and obstacles must be overcome to turn this project into a high performing model or anywhere close to SOTA music generation model, but the results here definitely show promise.

6.1 Future Steps for Lexical Lullabies:

Key insights gained from this project were discussed in section 5, so they won't be repeated here. However it's worth discussing how these insights in conjunction with the results can be applied back to this project to shape its future trajectory:

- **Improved data curation and augmentation:** The dataset used in this project was full of invalid MIDI files, bad prompts that didn't match the output, and some valid but just poor quality/musical uninteresting MIDIs. Taken altogether these things made for a poor dataset, that ultimately prioritized quantity over quality. To mitigate this, preprocessing could be extended to detect long silences, musical complexity via note density, and strange or suspicious key changes and patterns. Likely even a simple MLP could be trained to make such distinctions. This would allow for the filtering of a much better and still very large dataset to train on.
- **Vocabulary and Tokenization:** The approach taken to tokenization in this project was very trial and error without much planning or understanding at the onset. In retrospect it seems obvious that constructing a vocabulary through a combination of dataset extracted MIDI tokens and a method like byte pair encoding would result in much better performance. However it wasn't until late in this project's development that we became aware of these methods so they are definitely things to be implemented in the future.
- **Curriculum Training:** Curriculum training (pure MIDI first, then text with MIDI) and up weighting the MIDI part of the data loss proved beneficial. However, it is clear that this could be taken further with MIDI data being sorted by complexity. For example, single instruments early in the training to later introducing whole MIDI orchestras. This of course would too involve some additional level of preprocessing, but things like sorting by instrument count are computationally trivial.
- **Grammar Enforcing Loss:** While it's not clear the best way to proceed with this, it is evident that the training needs more deliberate enforcement of grammar, as the model with its 12% validity metric, seems perfectly content with not adhering to the grammar. Some sort of modified loss that penalizes inserting invalid tokens given the sequence may improve training. More experimentation is definitely needed here.
- **Model Architecture:** While Llama 3.2 1B was chosen for its accessibility, performance, and pre-training on English, architectures like encoder–decoder transformers, may be generally more well suited for this task. decoder-only models certainly could work, but it is very possible that an encoder-decoder could work better given the two very distinct natures of English vs MIDI.
- **Music Specific Metrics:** Music metrics weren't ever added to this project, primarily due to the lack of actual valid music produced. However, these metrics will play a big role in any further made improvements here as they keep track of things like tempo and key adherence. So this will definitely be added to this project.

Taken all together, these improvements lay out a plan for a robust and high-quality text-to-MIDI model. By combining better quality data, smarter tokenization, improved curriculum training, grammar aware loss and music specific metrics, the next version of this pipeline should achieve much better performance across the board. We hope this proof of concept sparks further exploration at the intersection of LLMs and music generation and are excited for where this project will take us next.

REFERENCES

[1] Andrea Agostinelli, Timo I Denk, Zalán Borsos, Jesse Engel, Mauro Verzetti, Antoine Caillon, Qingqing Huang, Aren Jansen, Adam Roberts, Marco Tagliasacchi, et al. 2023. Musiclm: Generating music from text. *arXiv preprint arXiv:2301.11325* (2023).

[2] Keshav Bhandari, Abhinaba Roy, Kyra Wang, Geeta Puri, Simon Colton, and Dorien Herremans. 2024. Text2midi: Generating Symbolic Music from Captions. *arXiv preprint arXiv:2412.16526* (2024).

[3] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[4] Nathan Fradet, Jean-Pierre Briot, Fabien Chhel, Amal El Fallah Seghrouchni, and Nicolas Gutowski. 2023. MidiTok: A python package for MIDI file tokenization. *arXiv preprint arXiv:2310.17202* (2023).

[5] Jan Melechovsky, Abhinaba Roy, and Dorien Herremans. 2024. MidiCaps: A large-scale MIDI dataset with text captions. arXiv:2406.02255 [eess.AS] https://arxiv.org/abs/2406.02255

[6] Jan Newmarch. 2017. *FluidSynth*. Apress, Berkeley, CA, 351–353. https://doi.org/10.1007/978-1-4842-2496-0_20