CALIFORNIA STATE UNIVERSITY
BAKERSFIELD

LLM-Based Music Generation Using Beat-Standardized MIDI Tokens

Huaiyu Zhang, Braden Stitt, Dominic Flores, Michael Kausch, Chengwei Lei

Introduction

This project explores symbolic music generation using beat-standardized MIDI tokens and a large language model (LLM). MIDI data is preprocessed into rhythm-aligned sequences with consistent tempo and structure, allowing the model to better capture musical patterns. We train GPT-2 to predict these beat-based tokens, enabling it to generate coherent melodies that can be rendered into audio.

We believe the beat alignment improves temporal consistency and helps the model generalize across styles. Output quality is evaluated using Fréchet Audio Distance (FAD) and SHA-256 hashing to ensure novelty. This work also serves as a comparative baseline against waveform-based models such as AudioLM.

GPT2 Model

- *Autoregressive Learning*: Ideal for generating temporally coherent music by predicting one token at a time.
- *Transformer Architecture*: Captures long-range dependencies crucial for musical structure.
- *Lightweight & Customizable*: Models offer a balance between performance and resource efficiency.
- *Benefit*: reduced training overhead while maintaining sequence quality.

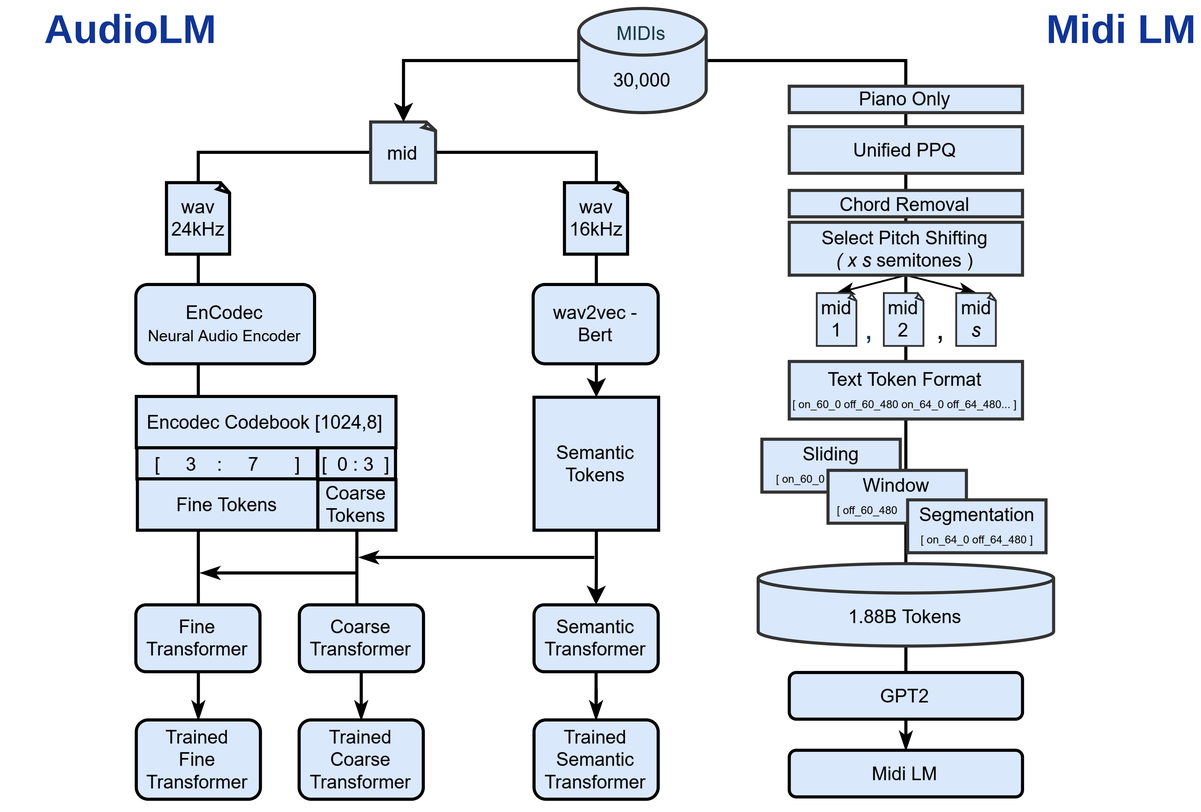
Hyperparameter	Value
Layers	6
Heads	8
Hidden Dim.	768
Token Size	1024
Vocab Size	4050

Datasets

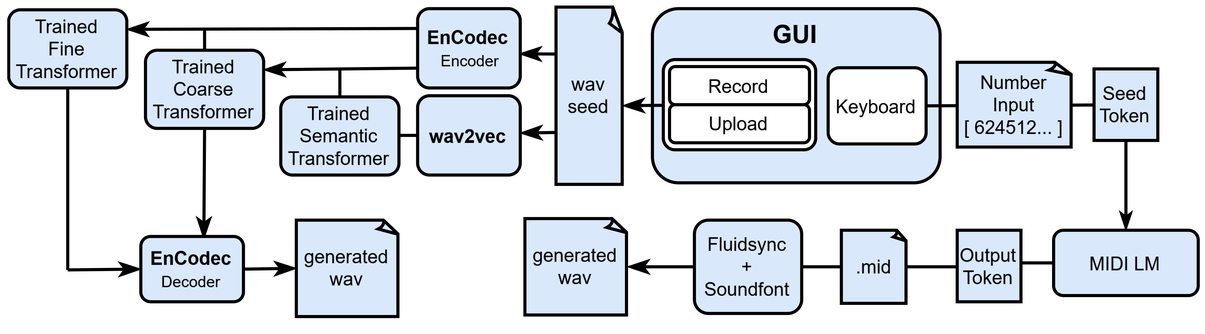
- MidiLM: ADL Piano (~11,000 piano MIDI)
- AudioLM: Dopeloop (~30,000 procedurally generated MIDI)

Tokenization

To transform musical input into a format suitable for language modeling, we implemented a custom tokenization pipeline for MIDI's. This approach converts simplified numerical notation into a structured sequence of text-based tokens representing musical events and was compared against AudioLM, a state-of-the-art hierarchical transformer architecture developed by Google.

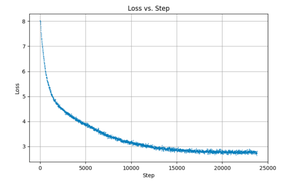


User Interaction / Data Flow

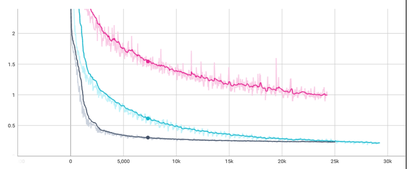


Training

GPT2 Training Loss

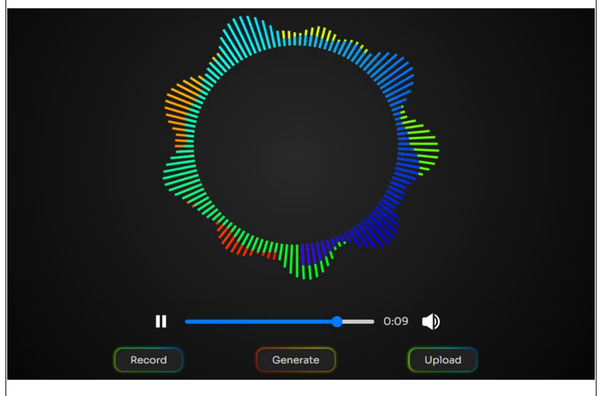


AudioLM



User Interface

We developed a browser-based graphical user interface (GUI) using ReactJS and Vite. Users can: Upload or record audio in the browser, trigger symbolic or waveform-based generation, and playback and download generated results.



Node.js + ExpressJS server routes client requests and triggers Python scripts to process audio within isolated environments, ensuring compatibility and reproducibility.

Acknowledgements

Dr. Lei for his excellent guidance, feedback and inspiration throughout this project. Dr. Cruz for allowing us use of his research server.