**Milestone P1 — Initial Experiment and Evaluation Setup**

**Project Title: Symbolic Music Generation using Transformer-Based Architectures**

**Author:**

Cade Stocker  
University of South Carolina  
Email: [cstocker@email.sc.edu](mailto:cstocker@email.sc.edu)

**System and Dataset Setup**

The system is developed in **Python 3.11** using **PyTorch 2.9.0+** on a **MacBook Air M1 (8GB RAM)**. The Apple Silicon platform enables lightweight GPU acceleration through **Metal Performance Shaders (MPS)**. All dependencies are managed using uv with a locked environment (uv.lock and pyproject.toml), ensuring reproducibility across systems.

The dataset used is the **Nottingham Dataset**, a collection of approximately 1,200 British and American folk melodies. Each song is represented in **MIDI format**, making it ideal for symbolic music generation tasks due to its simplicity and uniform structure. The smaller dataset size allows for quick experimentation and iteration — a crucial factor when training and evaluating multiple model architectures on limited hardware.

In future experiments, I plan to explore scaling to larger datasets such as the **Lakh Piano Roll Dataset** (LPD), which contains over 21,000 multitrack songs and would enable multi-instrument modeling and evaluation of scalability.

Two distinct tokenization schemes are implemented for sequence modeling:

1. **Naive Tokenization:**  
   Each note is represented by a single integer corresponding to its pitch, simplifying sequence length and reducing vocabulary size.
2. **MidiTok Tokenization [1]:**  
   Utilizes multi-token events to encode pitch, duration, velocity, and timing. This provides richer context but results in a larger vocabulary and longer sequences.

The **Nottingham dataset’s simplicity** makes it well-suited for comparing these two symbolic representations under different neural architectures.

**Baseline Implementation**

The baseline model follows the **Musenet architecture** introduced by Pandey et al. [2], which separates the compositional process into two interacting modules:

1. **Discriminator (Harmonic Selector):** Predicts the most probable harmonic or chord progression for upcoming measures.
2. **Generator (Melody Composer):** Produces new musical notes conditioned on both the previous measures and the discriminator’s harmonic output.

To maintain modularity and comparability, the system employs a **factory design pattern** for both components, allowing flexible substitution between architectures:

|  |  |
| --- | --- |
| **Component** | **Available Architectures** |
| Discriminator | MLP, LSTM, Transformer |
| Generator | GRU, LSTM, Transformer |

The initial baseline pipeline is implemented as:

**MIDI Files → Tokenization → Single LSTM Generator → Output Tokens → MIDI Reconstruction**

This forms the foundation for early experiments before introducing the full Musenet-style generator–discriminator pair.

All code and training scripts are version-controlled and reproducible via:

uv sync

python training/train\_generator.py --model\_type lstm --epochs 5

This configuration trains a simple LSTM generator as the primary baseline for comparison against more complex models.

**Preliminary Experiment**

Early experiments focused on evaluating the effects of tokenization strategies on training dynamics and musical quality.

**Experimental Setup:**

* **Model:** LSTM Generator
* **Epochs:** 5
* **Dataset:** Nottingham MIDI (train/test split: 80/20)
* **Tokenizations:** Naive vs. MidiTok
* **Evaluation Metrics:** Polyphony, note density, pitch range, duration (via PrettyMIDI)

**Pipeline Overview:**

1. Raw MIDI files are preprocessed and tokenized.
2. The generator learns next-note prediction from token sequences.
3. Generated token sequences are decoded back into MIDI.
4. Output MIDI files are automatically analyzed for structural metrics and logged into CSVs for each model type.

**Results Summary:**

* **Naive Tokenization:**
  + Faster training due to smaller vocabulary and shorter sequences.
  + Generated melodies were longer and more varied, sometimes producing creative variations of seed inputs.
  + However, occasional harmonic inconsistencies occurred (e.g., misplaced chords).
* **MidiTok Tokenization:**
  + Required significantly more compute and training time.
  + Generated outputs often consisted primarily of chords, with limited melodic movement — possibly due to data sparsity and the higher-dimensional token space.
  + Future tuning (e.g., sequence length adjustment, embedding size) may improve results.

Overall, the naive tokenization baseline provided **more coherent musical output** under constrained training conditions. This supports the hypothesis that simpler symbolic encodings perform better on small datasets and models, consistent with observations in MidiNet [3] and Briot et al. [4].

**Documentation and Reproducibility**

All experiments are fully documented within the project repository ([GitHub: csce585-midi](https://github.com/csce585-mlsystems/csce585-midi)).

The README.md provides:

* **Environment setup instructions** (using uv sync and Python 3.11).
* **Dataset preprocessing commands** (preprocess\_naive.py, preprocess\_miditok.py).
* **Model training scripts** (train\_generator.py, train\_discriminator.py).
* **Reproduction instructions** for baseline experiments.
* **Evaluation utilities** to compute musical metrics automatically.

Each experiment is logged with:

* Fixed **random seeds** for reproducibility.
* Clear directory organization (/training/logs/, /outputs/midi/, /data/preprocessed/).
* Version-locked dependencies (uv.lock).

Together, these ensure reproducible execution and evaluation, satisfying the project’s documentation requirements.

**References**

[1] Fradet, L. (2023). **MidiTok: A MIDI Tokenization Library for Deep Learning-based Music Generation.** GitHub repository.  
[2] Pandey, R., Kaur, G., & Mehra, P. (2022). **Musenet: Music Generation using Abstractive and Generative Methods.** ResearchGate preprint. [Online]. Available: <https://www.researchgate.net/publication/363856706_Musenet_Music_Generation_using_Abstractive_and_Generative_Methods>  
[3] Yang, L., Chou, S., & Yang, Y. (2017). **MidiNet: A Convolutional Generative Adversarial Network for Symbolic-Domain Music Generation.** arXiv:1703.10847  
[4] Briot, J., Hadjeres, G., & Pachet, F. (2017). **Deep Learning Techniques for Music Generation — A Survey.** arXiv:1709.01620