Machine Learning CS-GY 6923
Optional Project

Akshat Mishra (am15577)

Dec 15, 2025

Github: https://github.com/discostrangler/CS-GY6923-Final-Project

1. Introduction

Scaling laws are widely used to study how model performance changes with model size, data, and compute. Most existing work focuses on natural language, where structure is flexible and syntax errors are often tolerated. Symbolic music presents a different challenge. Music notation is discrete, highly structured, and constrained by syntax. It also contains long range dependencies related to rhythm, meter, and harmony. Studying scaling behavior in this domain helps test whether trends observed in language modeling extend to structured symbolic data.

In this project, I study empirical scaling laws for symbolic music generation using ABC notation. ABC represents music as text while preserving musical meaning such as pitch, duration, and time signature. This allows music to be modeled using language models and evaluated through syntactic validity and conversion to MIDI. I construct a complete preprocessing pipeline starting from raw ABC files, apply character level tokenization, and build train, validation, and test splits exceeding one hundred million tokens.

Using this fixed dataset and training setup, I train a family of decoder only Transformer language models spanning five model sizes. All models share the same tokenization scheme, context length, batch size measured in tokens, learning rate schedule, and training data. This controlled setup isolates the effect of model size. I measure validation loss after one epoch and fit a power law relating validation loss to parameter count.

I then perform the same scaling study using recurrent neural networks with comparable parameter counts. This enables a direct architectural comparison under identical data and optimization constraints. I analyze differences in scaling behavior, training efficiency, and memory usage between Transformers and RNNs.

Finally, I train the best performing Transformer model for extended compute and generate symbolic music samples using both unconditional and prompt based generation. I evaluate these samples using quantitative metrics such as syntactic validity and MIDI conversion success, and complement this with qualitative musical analysis. These experiments provide an empirical study of scaling laws in symbolic music modeling and highlight architectural differences between attention based and recurrent models.

2. Data

2.1 Dataset description and Preprocessing Pipeline

I built the dataset starting from raw MIDI files and converted it into a large scale text corpus in ABC notation suitable for language modeling. The full pipeline consists of MIDI to ABC conversion, corpus cleaning, quality filtering, tokenization, and dataset splitting. I report statistics at each stage to make the process transparent and reproducible.

I used the Lakh MIDI Dataset as the source of symbolic music data. The dataset was obtained from the official release at colinraffel.com/projects/lmd and contains a large collection of MIDI files spanning multiple genres and styles.

2.2 MIDI to ABC conversion

I converted the entire MIDI dataset to ABC notation using a file level conversion pipeline. Each MIDI file was mapped to a corresponding output path under /scratch/am15577/projects/ML Music Gen/abc_data, preserving the original directory structure. This ensured a one to one correspondence between input MIDI files and output ABC files.

A total of 178,561 MIDI files were discovered and processed. The conversion pipeline attempted to generate one ABC file per MIDI file and logged any conversion failures to a dedicated error log file.

The conversion produced 178,561 ABC files in total, matching the number of MIDI inputs. However, not all conversions were successful. The converter logged 4,078 failures, with common error types including malformed MIDI headers, premature end of file, invalid timing information, and MIDI files containing no note events.

Many failed conversions still produced empty placeholder ABC files with zero bytes. After conversion, I identified and removed all empty ABC files. There were 2,636 such empty files. After deleting them, 175,925 non empty ABC files remained.

Based on non empty output files, the effective MIDI to ABC conversion success rate was approximately 98.5 percent. This reflects the proportion of MIDI files that resulted in usable ABC notation after cleaning.

2.3 Quality filtering and corpus cleaning

All non empty ABC files were concatenated into a single text corpus. I applied length based quality filters to remove tunes that were either too short to contain meaningful musical structure or too long to be handled efficiently within a fixed context window.

I retained only tunes with lengths between 200 and 8000 characters. Tunes shorter than 200 characters were typically incomplete or trivial fragments. Tunes longer than 8000 characters exceeded the intended context length and would be heavily truncated during training.

Out of 175,925 converted ABC files, 45,468 tunes satisfied this length constraint. A total of 711 tunes were dropped for being too short, and 129,746 tunes were dropped for being too long. The filtered corpus was written to all_abc_clean.txt and used for all subsequent experiments.

2.4 Sequence length distribution

The retained ABC tunes exhibited a wide range of sequence lengths. The minimum length was 200 characters by construction. The median tune length was 3,036 characters. The 95th percentile length was 7,419 characters, and the 99th percentile length was 7,879 characters. The maximum retained length was 8,000 characters. This distribution ensured both diversity and compatibility with the fixed context length used during training.

2.5 Tokenization and vocabulary construction

I constructed a character level tokenizer from the cleaned corpus. The tokenizer vocabulary was built directly from the retained ABC text without using any external music specific rules. The final vocabulary size was 103 unique tokens, including special symbols. Any character not present in the vocabulary was mapped to a dedicated unknown token.

The cleaned corpus was streamed and converted into token IDs to avoid loading the entire dataset into memory. In total, 120,000,000 tokens were generated from the filtered ABC corpus.

I kept tokenization at the character level instead of introducing higher level musical tokens, to keep the representation simple and faithful to the ABC text. This choice forces the models to learn syntax and musical structure directly from raw characters.

2.6 Dataset splits

I split the tokenized data into training, validation, and test sets. The training split contained 117,600,000 tokens, which exceeded the minimum requirement of 100 million tokens. The validation split contained 1,200,000 tokens, and the test split also contained 1,200,000 tokens. All splits were saved as NumPy arrays for efficient loading during training.

3. Methods

3.1 Model architectures

I evaluated two model families: Transformer based language models and recurrent neural networks. For both families, I trained five model sizes spanning over two orders of magnitude in parameter count. The goal was to compare scaling behavior while keeping parameter counts closely matched across architectures.

3.1.1 Transformer models

All Transformer models followed a GPT style decoder only architecture with causal self attention. Each model used learned token embeddings, learned positional embeddings, stacked Transformer blocks, and a tied output embedding. Each block consisted of multi head self attention followed by a feedforward network with GELU activation and residual connections.

Table 1: Transformer Architectures

Model	Layers	Heads	d_model	Parameters
Tiny	2	4	128	4,42,752
Small	4	8	256	32,51,456
Medium	6	6	384	1,07,85,408
Large	8	8	512	2,54,03,904
XL	12	12	768	8,53,31,712

3.1.2 RNN models

The RNN models were stacked LSTM networks with learned token embeddings and a linear output projection. All RNNs used the same vocabulary and context length as the Transformers. The hidden state size and number of layers were chosen to closely match Transformer parameter counts.

Table 2: RNN Architectures

Model	LSTM Layers	Hidden Size	Parameters	Model
Tiny	2	160	4,28,960	Tiny
Small	2	448	32,65,472	Small
Medium	2	816	1,07,52,432	Medium
Large	3	1024	2,52,97,920	Large
XL	4	1632	8,54,53,152	XL

3.2 Training setup and hyperparameters

All models were trained using the same optimization setup. I used the AdamW optimizer with a learning rate of 3e-4, beta values of 0.9 and 0.95, and weight decay of 0.1. Dropout with probability 0.1 was applied to embeddings and internal layers.

The context length was fixed at 256 tokens, and the batch size was 64 sequences. This fixed context setting also motivated the length based filtering in preprocessing, since very long tunes would be heavily truncated during training. The resulting setup keeps training consistent across runs, but it also shapes what musical material the models see. This resulted in 16,384 tokens processed per training step. I applied gradient clipping with a maximum norm of 1.0 to stabilize training.

I used a cosine learning rate schedule with linear warmup. The warmup phase covered the first 5 percent of training steps, followed by cosine decay for the remainder of the epoch.

Each model was trained for exactly one epoch over the full training token stream. The token order was fixed and contiguous, and no reshuffling was performed. This ensured that all models saw the same number of tokens under identical conditions.

3.3 Experimental design for scaling studies

The scaling experiments were designed to isolate the effect of model capacity. Across all model sizes and architectures, I held the dataset, token budget, batch size, context length, optimizer, and learning rate schedule constant. The only variable was model architecture and parameter count.

For each model size, I ran a single training run and recorded validation loss after completing one full epoch. I also recorded wall clock training time and peak GPU memory usage. These measurements allowed direct comparison of efficiency and scaling trends.

To quantify scaling behavior, I fit a power law with offset of the form $L = aN^{-\alpha} + c$, where N is the number of trainable parameters and L is the final validation loss. The fitting was performed using a grid search over alpha and offset values using only the observed data points.

3.4 Evaluation metrics

For language modeling performance, I reported validation cross entropy loss after one epoch. This metric was used for scaling analysis and architecture comparison.
For efficiency, I recorded wall clock training time per epoch and peak GPU memory usage during training.

For generation quality, I evaluated structural validity of generated samples. I generated both unconditional samples and conditional samples using ABC prefixes. Generated ABC files were tested for parse validity and for successful conversion back to MIDI format. I reported parse success rate and ABC to MIDI conversion success rate as indicators of syntactic and structural correctness. Qualitative assessment was performed by listening to successfully converted MIDI files and inspecting musical structure. No external perceptual metrics were used.

4. Results

4.1 Transformer Scaling Behavior

Figure 1 shows the training loss curves for five Transformer sizes. All models show a rapid drop in loss during the first few hundred steps, followed by a slower improvement phase. Larger models train more smoothly and reach lower loss values throughout training. The tiny model remains clearly worse than the others and plateaus at a higher training loss.

Figure 2 shows validation loss after one epoch versus model size on a log scale, with a power law fit. Validation loss decreases consistently with scale. The fitted exponent is $\alpha \approx 0.443$, which indicates strong scaling benefits in this regime.

Scaling improves validation loss, but the marginal gains shrink at the top end. For example, moving from large to xl reduces validation loss from 0.5936 to 0.5731, while wall clock time increases from 616.9 seconds to 1,793.4 seconds and peak GPU memory increases from 7,157.12 MB to 16,261.19 MB. This shows a clear tradeoff: the xl model is best in loss, but it is much more expensive in compute and memory.

Table 3: Transformer Comparison

model_name	final_val_loss	wall_clock_seconds	gpu_mem_peak_mb
tiny	1.0237	39.9	693.18
small	0.7293	139.2	2447.25
medium	0.634	318.7	4067.59
large	0.5936	616.9	7157.12
xl	0.5731	1793.4	16261.19

Figure 1: Transformer Training

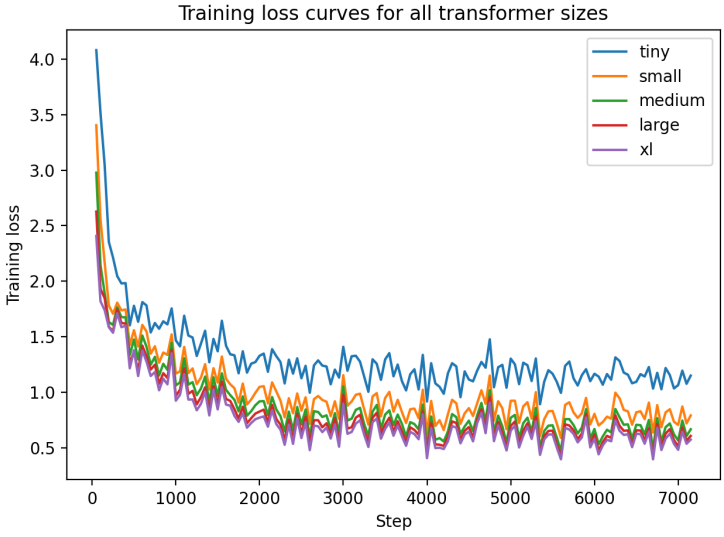
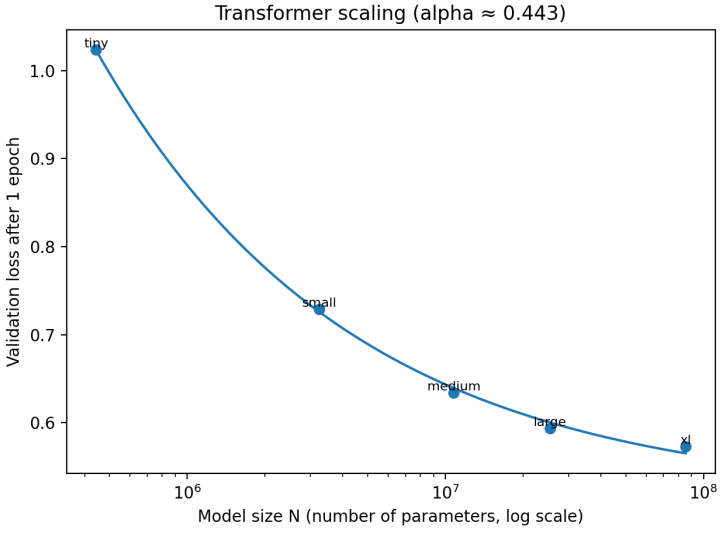


Figure 2: Transformer Scaling Curve



4.2 RNN Scaling Behavior

Figure 3 shows the training loss curves for all RNN model sizes. All RNNs exhibit a sharp drop in loss during the early training steps, followed by slower convergence. Larger models consistently achieve lower training loss than smaller ones, but the gap between model sizes remains modest. Training curves show noticeable noise and oscillations throughout training, including for the xl model, indicating less stable optimization as scale increases.

Figure 4 shows validation loss after one epoch plotted against model size on a logarithmic scale, along with a power law fit. Validation loss decreases monotonically with increasing parameter count, from 1.025 for the tiny model to 0.702 for the xl model. The fitted scaling exponent is $\alpha \approx 0.129$, indicating weak scaling behavior. Although increasing model size improves performance, gains diminish rapidly at larger scales, suggesting early saturation in the RNN family.

Table 4: RNN Comparison

model_name	final_val_loss	wall_clock_seconds	gpu_mem_peak_mb
tiny	1.0254	76.7	472.35
small	0.8816	108.1	819.06
medium	0.7956	146.4	1491.44
large	0.7508	235.5	2455.39
xl	0.7022	561.2	5346.2

Figure 3: RNN Training

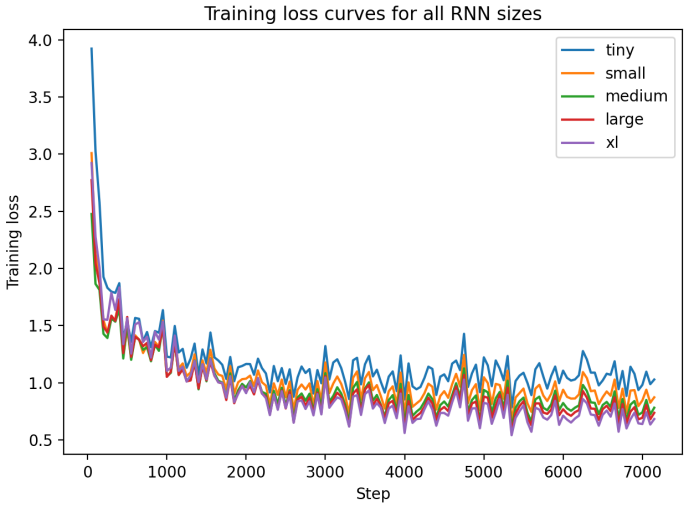
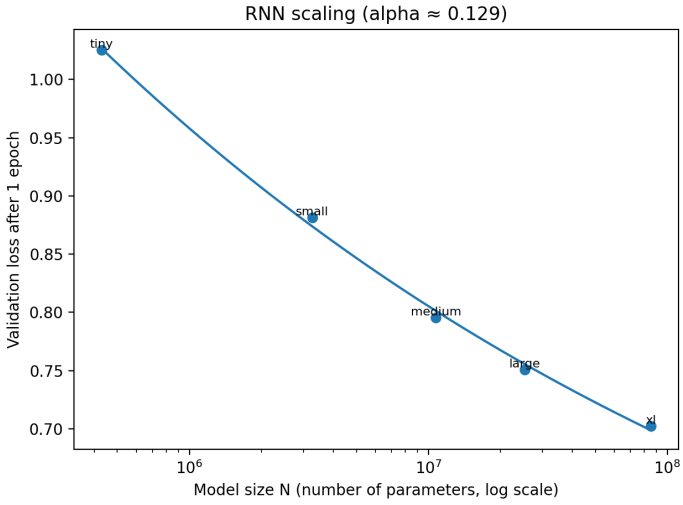


Figure 4: RNN Scaling Curve



4.3 Comparison

The combined scaling plot shows both architectures on the same axes, and it highlights a clear gap in how validation loss improves with model size. The Transformer curve drops faster and stays below the RNN curve across the full range from about 4e5 to 8.5e7 parameters. The fitted power law exponent is steeper for Transformers (alpha about 0.443) than for RNNs (alpha about 0.129), so Transformers scale better in the sense that adding parameters yields larger reductions in validation loss.

In compute terms, the RNN is cheaper per epoch and uses less GPU memory at the same parameter scale. For the xl models, the RNN reaches one epoch in about 561 seconds with a peak memory of about 5.3 GB, while the Transformer takes about 1793 seconds and peaks around 16.3 GB. This pattern matches what I see across the table: Transformers trade more time and memory for better loss. If I compare efficiency per parameter, the RNN is more computationally efficient, but the Transformer is more performance efficient because it achieves lower validation loss at the same model size.

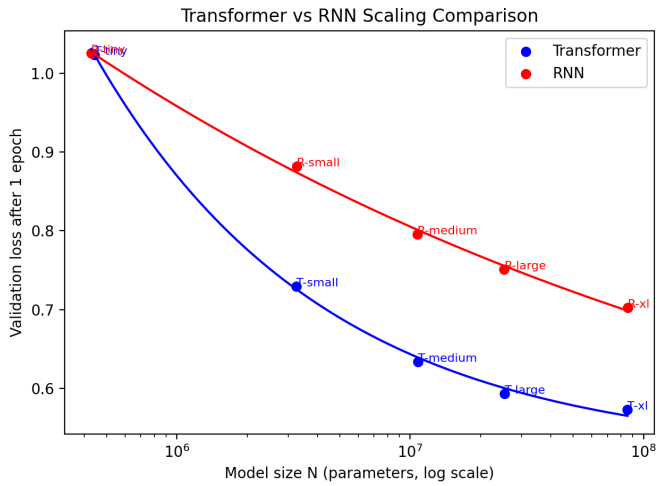
I think the difference comes from how the two architectures use capacity. Self attention lets the Transformer directly connect distant tokens and represent long range musical structure without compressing everything into a fixed size hidden state. In contrast, the RNN must carry information through the hidden state over many steps, which creates an information bottleneck and makes long range dependencies harder to learn. This shows up as weaker scaling for the RNN, where gains from increasing size saturate earlier.

From a sample efficiency view, both models are trained on the same corpus and I measure validation loss after one epoch, so lower loss means the model extracts more from the same amount of data. The Transformer is more sample efficient because it achieves lower validation loss throughout the scaling range. From a computational efficiency view, the RNN is more efficient because it trains faster and uses less memory per epoch, but the Transformer converts extra compute into stronger quality gains. Overall, Transformers scale better and are more sample efficient, while RNNs are more compute and memory efficient, and this supports choosing the Transformer xl for final training and sample generation.

Table 5: RNN vs Transformer Comparison

size	trans_val_loss	rnn_val_loss	trans_time_sec	rnn_time_sec	trans_gpu_MB	rnn_gpu_MB
tiny	1.0237	1.0254	39.9	76.7	693.18	472.35
small	0.7293	0.8816	139.2	108.1	2447.25	819.06
medium	0.634	0.7956	318.7	146.4	4067.59	1491.44
large	0.5936	0.7508	616.9	235.5	7157.12	2455.39
xl	0.5731	0.7022	1793.4	561.2	16261.19	5346.2

Figure 5: Transformer vs RNN Scaling Curve



4.4 Generated samples and evaluation

60 percent of generated ABC outputs were syntactically correct and were successfully convert to MIDI, which I treat as the main end to end validity metric for sample quality. The remaining outputs fail conversion due to ABC formatting issues, most commonly malformed headers or missing default note length fields, rather than obviously random musical content.

On the held out test set, the model achieves a final perplexity of approximately 1.65, indicating strong predictive performance on unseen symbolic music sequences.

Qualitatively, the converted MIDI samples sound musically coherent. They typically maintain a stable meter, show repeated motifs with small variations, and exhibit phrase level structure. Many samples suggest a consistent tonal center and use common melodic contours such as stepwise motion with occasional leaps. Rhythm and bar level grouping are usually respected in the successful conversions, which supports that the model learned both syntactic conventions of ABC and higher level musical patterns. I did not run a systematic size by size qualitative study, so I do not claim a sharp phase transition in musical capabilities across scales. Instead, I report the musical patterns that were consistently present in successfully converted samples, and treat formatting failures as the main practical bottleneck.

Generated Audio Files: https://drive.google.com/drive/folders/1jlab9-a48PIxS7ueMbuf7JydStF9Jwsw?usp=share_link

5. Discussions

5.1 Key insights from the scaling experiments

Across both model families, increasing parameter count improved validation loss after one epoch, but the strength of that improvement depended heavily on architecture. Transformers showed strong scaling, with validation loss decreasing consistently and a fitted scaling exponent around 0.443. RNNs also improved with size, but much more weakly, with an exponent around 0.129 and signs of earlier saturation. The power law fits align with the scaling law motivation described earlier in the report, showing predictable loss improvement with scale in this symbolic music setting. The fitted exponents here summarize behavior under this specific one epoch, fixed token budget design.

The architecture gap is visible across the whole parameter range studied. The Transformer curve stays below the RNN curve from the smallest to the largest models, meaning that at matched scale, Transformers achieved better validation loss under the same data and optimization constraints.

Efficiency measurements show a clear tradeoff. RNNs were cheaper per epoch and used less peak GPU memory at similar parameter counts, while Transformers spent more time and memory but translated that extra compute into lower loss. For the largest models, the RNN finished an epoch in about 561 seconds with about 5.3 GB peak memory, while the Transformer took about 1793 seconds and peaked around 16.3 GB. Under a fixed token budget and identical optimization, these curves highlight a compute tradeoff: RNNs are cheaper per epoch, while Transformers convert extra time and memory into larger reductions in validation loss. In this setting, allocating capacity to the Transformer yields stronger returns than allocating the same scale to the RNN.

5.2 Interpreting the results for symbolic music modeling

These results fit the core challenge of symbolic music modeling described in the report: ABC is discrete, highly structured, and constrained by syntax, and the sequences contain long range dependencies tied to rhythm, meter, and harmony.

In that context, the Transformer advantage is consistent with how attention can connect distant tokens directly, making it easier to represent long range musical structure without forcing everything through a fixed size hidden state. The report contrasts this with the RNN bottleneck, where information must be carried step by step through the hidden state, making long range dependencies harder and leading to weaker scaling.

From a sample efficiency point of view, both architectures trained on the same corpus and were evaluated after one epoch, so the model with lower loss is extracting more from the same amount of data. Under this definition, the Transformer was more sample efficient across the full scaling range.

The generation results are broadly consistent with the modeling results. The report notes that about 60 percent of generated ABC outputs successfully converted back to MIDI, and that failures were mainly formatting issues like malformed headers or missing default note length fields rather than obviously random content. This suggests the model learned substantial structure, but that strict syntax is still a major constraint in end to end usability.

5.3 Design decisions and their impact

A major strength of the experimental design is how tightly variables were controlled. The dataset, token budget, batch size, context length, optimizer, and learning rate schedule were held constant across model sizes and across architectures, with parameter count and architecture as the main changing factors. This makes the scaling and comparison conclusions easier to attribute to architecture rather than training differences.

Several preprocessing decisions were closely tied to the fixed context setting. The corpus was filtered to keep tunes between 200 and 8000 characters, explicitly motivated by avoiding trivial fragments and avoiding sequences that would be heavily truncated. This helped produce a consistent training setup but also shaped what kinds of musical material the models saw.

Character level tokenization kept the representation simple and faithful to the ABC text, producing a compact vocabulary of 103 tokens and a total of 120,000,000 tokens after cleaning. This choice supports straightforward language modeling on ABC, and it also means the models must learn syntax and musical structure directly from raw characters rather than from higher level musical tokens.

Finally, training each model for exactly one epoch with fixed contiguous token order and no reshuffling made the scaling measurements consistent and comparable, which is ideal for the controlled scaling study goal.

5.4 Limitations and future work

Several limitations follow directly from the chosen setup. First, each model size used a single training run and results were recorded after one epoch, which provides a clean snapshot for scaling analysis but limits conclusions about longer training behavior or run to run variation. Another limitation is that the RNN family showed noisier and less stable training behavior, and its scaling

gains saturated earlier in this setup. This reinforces that the main conclusions are about relative scaling trends under controlled conditions, not about fully optimized training for each architecture.

Second, the fixed context length and the length based filtering simplify training but restrict coverage of very long pieces, and the report explicitly notes that very long tunes would be heavily truncated during training.

Third, evaluation of generation quality was intentionally limited. Quantitatively, the main end to end metric was ABC to MIDI conversion success, and qualitatively the report relied on listening to successfully converted MIDI files, with no external perceptual metrics. This keeps the evaluation aligned with the project scope, but it also leaves room to broaden evaluation in future iterations.

Finally, the 60 percent MIDI conversion rate shows that strict ABC formatting remains a practical bottleneck for generation, even when the musical content in successful conversions sounds coherent and shows stable meter, repeated motifs, and phrase level structure. Improving structural validity within ABC would directly raise the fraction of samples that can be rendered and listened to, which is crucial for symbolic music generation workflows.

Appendix A: Generated Samples

Generated Audio Files: https://drive.google.com/drive/folders/1jlab9-a48PIxS7ueMbuf7JydStF9Jwsw?usp=share_link

References

Benini, D. (2025). ABC notation: The basics. Nota ABC. https://notabc.app/abc/basics/

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., & Amodei, D. (2020). Scaling laws for neural language models. arXiv:2001.08361. https://arxiv.org/abs/2001.08361

Karpathy, A. (2022). nanoGPT: The simplest, fastest repository for training and fine tuning medium sized GPTs [GitHub repository]. https://github.com/karpathy/nanoGPT/tree/master

Raffel, C. (n.d.). The Lakh MIDI Dataset v0.1. https://colinraffel.com/projects/lmd/