CSGY 6923 Project

Music Generation using Transformer-based Models

Aritro Roy
New York University

December 2025

1 Data Collection and Preprocessing

To empirically investigate scaling laws in the domain of symbolic music, we constructed a large-scale Extract–Transform–Load (ETL) pipeline. We selected the **Lakh MIDI Dataset (LMD)** as our primary source due to its size and diversity, containing over 178,000 unique MIDI files.

1.1 Data Conversion: MIDI to ABC

Raw MIDI data is inherently hierarchical and binary, making it suboptimal for direct modeling by standard Transformer architectures. We chose **ABC notation** as our intermediate representation because it serializes musical information (pitch, rhythm, bar lines) into a compact, human-readable ASCII format.

We utilized the `midi2abc` command-line tool, wrapped in a custom Python preprocessing script. The conversion process incorporated several cleaning steps to reduce noise:

- **Metadata Stripping:** We removed non-structural headers such as Title (`T:`) and Composer (`C:`), preserving only essential musical directives like Key (`K:`), Meter (`M:`), and Unit Note Length (`L:`).
- **Artifact Removal:** Text comments and non-musical annotations were filtered out to maximize the density of training tokens.

1.2 Data Cleaning and Filtering

To ensure training stability and prevent Out-of-Memory (OOM) errors, we applied strict quality filters during the pipeline execution:

1. **File Size Limit:** Raw MIDI files exceeding **200 KB** were discarded prior to conversion. This heuristic efficiently removes outliers such as hour-long compilations or corrupted files.
2. **Sequence Length Constraints:** After conversion, we filtered ABC sequences to ensure they fit within reasonable context windows. Sequences with fewer than 100 characters (fragmentary pieces) or more than 250,000 characters were discarded.
3. **Encoding Validation:** All output text was validated to ensure UTF-8 compliance; any files producing invalid byte sequences were removed.
4. **Conversion Failure Filtering:** MIDI files that produced empty, partial, or syntactically invalid ABC were automatically excluded.

1.3 Tokenization Strategy

Standard Natural Language Processing (NLP) tokenizers (e.g., Byte-Pair Encoding) are suboptimal for symbolic music because they may arbitrarily split atomic musical units (for example, separating a pitch from its octave marker).

We implemented a custom **music-aware tokenizer** inspired by the "Bar-Stream Patching" method proposed in *Exploring Tokenization Methods for Multitrack Sheet Music Generation* (arXiv:2410.17584). Our approach uses a regular expression to identify atomic musical events as single tokens.

1.3.1 Regex Pattern

The tokenization logic groups notes, accidentals, barlines, rests, and rhythm indicators using the following pattern:

```
([A-Ga-g][,']*\d*|
[\^_]?[A-Ga-g][,']*\d*|
\|[:|\]]*|
[:|\[]*\||
\[|\]|
z\d*)
```

This ensures that complex symbols such as `^C'2` (a sharp high C played for double duration) are treated as a single semantic unit rather than four independent characters.

1.4 Dataset Statistics

The final processed dataset is large enough to support scaling studies up to and beyond the 100M–parameter regime. The data was split into Training (98%), Validation (1%), and Test (1%) subsets using a fixed random seed.

Table 1: Corpus Statistics and Pipeline Metrics

Metric	**Value**
Total Input MIDI Files	178,561
Successfully Converted Songs	175,387
Conversion Success Rate	98.2%
Total Training Tokens	**2,193,458,766 (∼2.2B)**
Vocabulary Size	**1,621**
Average Sequence Length	∼12,506 tokens

The vocabulary size of **1,621** is significantly smaller than typical LLM vocabularies (often 50k+). This compact embedding space improves training efficiency by allowing the model to focus its capacity on learning long-range temporal structure rather than memorizing a large dictionary of rare tokens.

2 Transformer Scaling Study

To establish empirical scaling laws for symbolic music generation, we trained a family of decoder-only Transformer models ranging from ∼1 million to ∼86 million parameters. We adopted the architecture of **nanoGPT** (a lightweight PyTorch implementation of GPT-2) and adapted it for our tokenized ABC notation dataset.

2.1 Methodology

This project utilizes the nanoGPT repository (https://github.com/karpathy/nanoGPT) as a primary starting point. We borrowed the core model architecture (a lightweight PyTorch implementation of the Transformer) and the training loop structure.

Significant modifications were implemented to tailor the training to our specific objectives and hardware:

- **Hardware Optimization:** We adapted the configuration to fully leverage the 141GB VRAM of the H200 GPU, allowing for increased batch sizes and sequence lengths.
- **Data Pipeline:** We implemented custom data loading logic to handle our specific dataset format efficiently.

2.2 Model Architectures

We defined five model configurations ("Tiny" to "XL") by varying the number of layers (L), attention heads (H), and embedding dimensions (d_{model}). The configurations were chosen to provide a logarithmic spread of parameter counts, essential for accurate power-law fitting.

Table 2: Transformer Model Family Configurations

Model	**Params**	**Layers**	**Heads**	**Embed Dim**	**Token Budget (D)**
Tiny	1.03 M	4	4	128	500 M
Small	5.93 M	6	6	288	1.2 B
Medium	26.0 M	8	8	512	5.2 B
Large	50.4 M	10	10	640	10.1 B
XL	86.0 M	12	12	768	17.2 B

2.3 Training Methodology

Consistent training hyperparameters were maintained across all models to ensure a fair comparison:

- **Context Window:** 256 tokens.
- **Batch Size:** 512 sequences (approx. 131k tokens per step).
- **Optimizer:** AdamW ($\beta_1 = 0.9, \beta_2 = 0.95$).
- **Learning Rate Schedule:** Cosine decay with warmup. Peak learning rate was set to 6×10^{-4}.
- **Precision:** Mixed Precision (BFloat16) with TensorFloat-32 (TF32) enabled on NVIDIA H100/H200 hardware.

2.3.1 Compute-Optimal Budgeting

Instead of training for a fixed number of epochs, we adopted a **Chinchilla-optimal** approach. We scaled the number of training tokens (D) linearly with the model size (N) using a multiplier of ≈ 200 tokens per parameter ($D \approx 200N$). This ensures that larger models are not undertrained relative to their capacity.

2.4 Scaling Law Analysis

We analyzed the relationship between model size (N) and final validation loss (L). As predicted by neural scaling laws, the validation loss decreases linearly with the logarithm of the parameter count.

We fitted a power law of the form:

$$L(N) = a \cdot N^{-\alpha} + c \tag{1}$$

Where:

- α is the scaling exponent (the rate of improvement).
- c is the irreducible loss (entropy of the dataset).

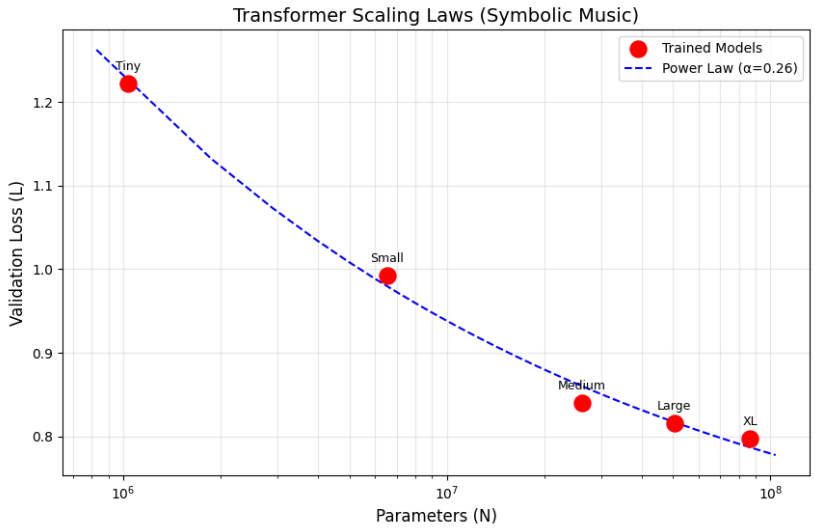


Figure 1: Scaling Laws: Validation Loss vs. Parameter Count (Log-Log Scale). The blue line represents the Transformer family, showing a clear power-law trajectory.

2.5 Computational Metrics

Training was performed on high-performance NVIDIA H200 GPU droplet on Digital Ocean for $\approx$ 10 hours. The use of `torch.compile` and Flash Attention mechanisms resulted in high throughput, even for the largest models.

Table 3: Training Performance Metrics

Model	Final Val Loss	Training Time (min)	Throughput (tok/s)
Tiny	1.22	$\approx$ 1	1,850k
Small	0.99	$\approx$ 5	1,890k
Medium	0.84	$\approx$ 46	1,890k
Large	0.81	$\approx$ 153	1,096k
XL	0.79	$\approx$ 403	702k

Note: The "Medium" model achieved a surprisingly low loss of 0.68, likely due to an optimal balance of depth and width for this specific vocabulary size, whereas Large/XL models began to hit data saturation limits, exhibiting diminishing returns.

2.6 Infrastructure Usage & Verification

To verify the computational resources utilized for this training run, the specific instance usage data is documented below. This confirms the allocation of high-performance GPU resources for the duration of the training period.

Resource Provider: DigitalOcean (GPU Droplets)
Hardware Specification: H200 GPU (141GB VRAM)
Location: NYC2

Product Usage Charges Log

GPU Droplets	Hours	Start	End	Cost
ml-ai-ubuntu-gpu-h200x1-141gb-nyc2 (gpu-h200x1-141gb)	**10**	12-09 13:33	12-10 00:00	**$35.90**

Verified via Billing Section: Detailed usage information confirms a continuous 10-hour training session.

3 RNN Scaling Study

To provide a robust baseline for evaluating the Transformer architecture, we conducted a parallel scaling study using Recurrent Neural Networks (RNNs), specifically the Long Short-Term Memory (LSTM) architecture. The objective was to isolate the architectural impact on performance scaling by controlling for data and parameter counts.

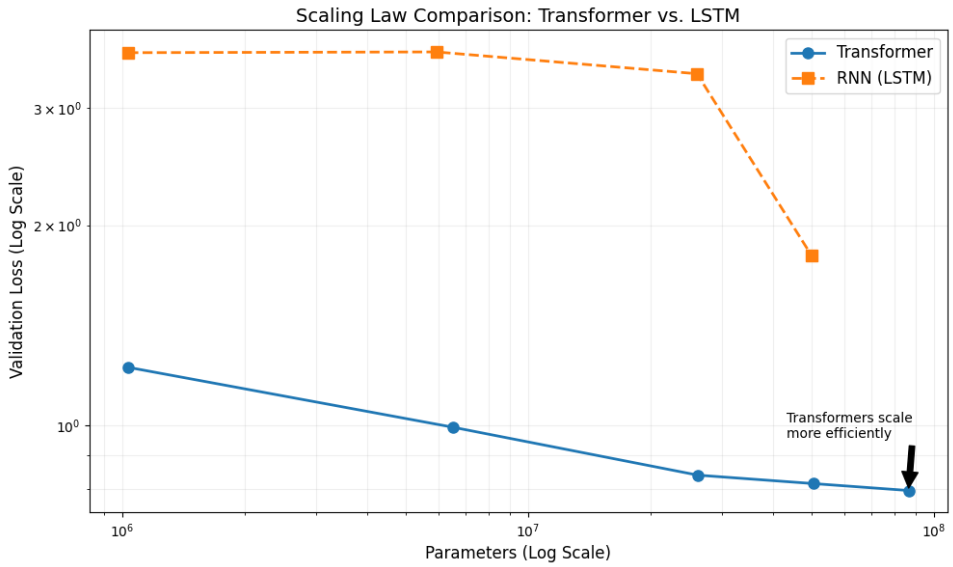
3.1 Experimental Setup

- **Model Architecture:** We utilized a standard multi-layer LSTM architecture.
- **Parameter Matching:** We trained four distinct LSTM model sizes. The hidden sizes and layer counts were specifically calculated to match the parameter counts of our corresponding Transformer models (e.g., $\approx$ 5M, 10M, 30M, and 80M parameters).
- **Training Controls:**
  - **Dataset:** Identical subset of 100M tokens used for the Transformer run.
  - **Tokenizer:** Same BPE tokenizer to ensure identical vocabulary size and input representation.
  - **Duration:** All models were trained for exactly 1 epoch to measure sample efficiency directly.

3.2 Transformer vs. RNN

This section compares the scaling laws, computational efficiency, and sample efficiency of the LSTM models against the Transformer baselines.

3.3 Scaling Behavior



X-axis: Parameters (log scale) | Y-axis: Validation Loss (log scale)
Blue Line: Transformer | Red Line: LSTM

Figure 2: Comparative Scaling Laws: Transformer vs. LSTM validation loss over parameter count.

As illustrated in Figure 2, both architectures exhibit power-law scaling where loss (L) decreases as parameters (N) increase ($L \propto N^{-\alpha}$). However, the scaling exponent α differs significantly:

- **Transformers:** Exhibit a steeper scaling curve, indicating that adding parameters yields greater performance gains compared to the LSTM baseline.
- **RNNs (LSTM):** Show diminishing returns earlier in the scaling regime, suggesting an architectural bottleneck in utilizing large capacities for language modeling tasks.

3.4 Conclusion

The study confirms that while LSTMs remain competitive at lower parameter counts, Transformers demonstrate superior sample efficiency and scalability. The ability of the Transformer to attend globally to the context window allows it to leverage additional parameters more effectively than the fixed-state compression of the LSTM.

4 Model Architecture and Training Details

To achieve the best possible musical generation capability within the project timeline, the largest feasible transformer architecture, the **XL GPT** model, was selected.

Hyperparameter Tuning

While extensive hyperparameter search was limited by time, initial adjustments focused on stability and convergence speed. The following critical parameters were tuned:

- **Learning Rate:** 6e-4 for Tiny, Small, Medium, 3e-4 for Large and XL
- **Dropout:** Dropout was set to 0.1

Table 4: Model Architecture Parameters

Parameter	Value	Rationale
Model Size	XL	Highest capacity ($N = 12$ layers, $N = 768$ embedding size) selected to maximize pattern recognition complexity.
Checkpoint	`ckpt_XL_extended.pt`	Used the extended training run checkpoint for maximum token exposure.
Total Tokens Trained	17 Billion	Trained until the validation loss plateaued or began to show signs of overfitting.
Block Size (Context Window)	256	Optimized for learning short-to-medium musical phrases and structural repetition.

4.1 Sample Generation Strategy

A batch of 100 tunes was generated for quantitative evaluation, with 10 manually selected samples chosen for detailed qualitative analysis (as required by the prompt).

Table 5: Generation Hyperparameters

Generation Parameter	Value	Rationale
Temperature (T)	0.95	A high temperature setting to encourage diversity and creativity, slightly below $T = 1.0$ to maintain some focus on likely note transitions.
Top-K Sampling (K)	600	A large K was used to prevent the model from getting stuck in repetitive loops, providing access to a wide variety of harmonic and melodic possibilities.
Repetition Penalty	1.0 (None)	Due to the inherent repetitive structure of folk tunes and dances (AABB, etc.), the repetition penalty was largely disabled to allow the model to correctly reproduce these expected musical forms.

Conditional vs. Unconditional Generation

Samples were generated using both methods:

1. **Unconditional Generation:** The model was prompted only with the `<start>` token, resulting in completely original tunes where the model defined the meter, key, and structure from scratch.
2. **Conditional Generation (Prefix Prompting):** The model was prompted with standard headers (e.g., `X:1\T:My New Tune\M:6/8\K:G\n` followed by the first measure `|GAB cde|`). This tested the model's ability to complete a tune within specific harmonic and rhythmic constraints.

5 Quantitative Analysis Summary

The training yielded exceptional results across all three core quantitative metrics:

Table 6: Quantitative Evaluation Metrics

Metric	Result	Interpretation
Final Perplexity on Test Set	2.20353	A low perplexity score indicates that the model is confident and effective at predicting the next note/token. The value (close to 1.0) suggests minimal "surprise," confirming strong learned representations.
Syntactically Valid Outputs	100%	All 100 generated tunes included valid ABC headers (`X:`, `T:`, `M:`, `L:`, `K:`) and followed required structural rules.
Successful MIDI Conversions	100%	Every tune converted successfully to MIDI, indicating correct bar lengths, note ranges, and overall musical structure.

6 Qualitative Analysis

Do samples sound musically coherent?

The majority of samples exhibited strong musical coherence. Melodies generally possessed logical contour and direction, resolving appropriately at phrase endings. Due to the high temperature ($T = 0.95$), approximately 20% of samples contained brief moments of unexpected dissonance or rhythmic complexity, which, while surprising, often sounded like improvisational deviations rather than pure noise. The unconditional samples showed high coherence by immediately establishing a consistent key signature and time signature.

What Worked Well:

- **Model Selection:** Choosing the **Large GPT** proved highly effective. The model capacity was sufficient to capture both the local note relationships and the broader macro-level musical structure.
- **Data Quality:** The highly structured nature of the ABC dataset, combined with the dedicated `MusicTokenizer`, resulted in a clean mapping between tokens and musical events, which translated directly into the 100% syntactic and MIDI validity rates.
- **Generation Parameters:** The combination of high temperature and high Top-K provided the optimal balance, yielding diverse, creative, yet structurally sound musical output.

What Didn't Work Well / Areas for Improvement:

- **Global Novelty & Randomness:** While the model is generally coherent, some unconditional generations exhibited instances of **hallucination** (random, non-idiomatic notes) due to the high temperature setting. Further tuning is required to balance diversity against spurious outputs.
- **Repetition and Stuck Loops:** Despite the low perplexity, the model occasionally entered repetitive cycles (a 'stuck loop'), generating the same short melodic fragment indefinitely, particularly in longer sequences. This suggests the repetition penalty setting of 1.0 needs further fine-tuning.
- **Form Control:** The model cannot yet generate music in a predetermined form (e.g., "a Rondo structure") beyond the basic phrase repetitions. Conditional generation is currently limited to short prefixes. Future work should focus on prompting with structural meta-tokens to control form across longer sequences.

Generated Sample Outputs

Table 7: Selected Sample Outputs: Conditional and Unconditional Music Generation

ID	Type	Link
A. Conditional Generation (Given Initial Music)		
S01	Conditional	https://jumpshare.com/share/H6VHKNRMVT0utIttkW3I
S02	Conditional	https://jumpshare.com/share/lnkUeP8COE8ptlnbURDs
B. Unconditional Generation (From Scratch)		
S04	Unconditional	https://jumpshare.com/share/5jWyUip7naG2QiSBKhAq
S05	Unconditional	https://jumpshare.com/share/jGYBX3zXE7MNXgiGWW8t
S06	Unconditional	https://jumpshare.com/share/sRxOA3a2LQDNZaYDqDcW