Scaling Laws for Symbolic Music Modeling

Rishi Ajith
CS-GY 6923: Machine Learning
github.com/suryaajith21/Symbolic-Music-Scaling-Project

December 15, 2025

1 Introduction

The scaling laws of neural language models, primarily established in Natural Language Processing (NLP) by Kaplan et al. (2020), describe a predictable power-law relationship between model performance (loss) and compute, dataset size, and parameter count. This project investigates whether these laws hold true for **symbolic music modeling**, a domain that shares the sequential nature of text but differs fundamentally in its hierarchical structure (rhythm, harmony, and repetition)

Using the Lakh MIDI Dataset converted into ABC notation, I trained a family of Transformer (GPT) Radford et al. (2019) and Recurrent Neural Network (LSTM) models ranging from 0.8M to 113M parameters. Our experiments confirm that symbolic music modeling follows a power-law scaling trajectory similar to natural language. Furthermore, I provide a comparative analysis showing that while LSTMs exhibit superior sample efficiency at lower parameter counts (¡5M), Transformers scale significantly better, outperforming LSTMs as model capacity increases.

In addition to the scaling study, we trained a final "BEST" model aimed at producing higher-quality symbolic music. Unlike the earlier character-level experiments, this model uses a BPE tokenizer and a stronger preprocessing pipeline designed to preserve important musical directives and remove low-quality converted files. We trained it for substantially longer than the scaling runs and evaluate it using both standard language-model metrics and the musical validity/playability of its generated ABC outputs

2 Data

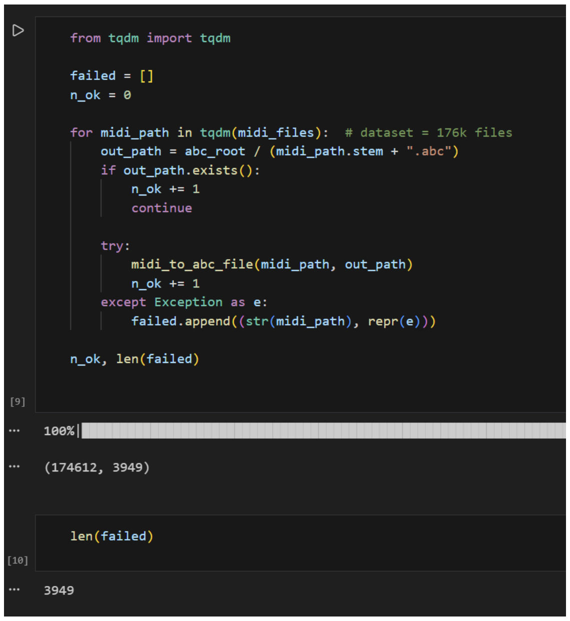
MIDI→ABC Conversion Success Rate: In total, **178,561** MIDI files were processed; **174,612** converted successfully and **3,949** failed, corresponding to a **97.79%** conversion success rate (and **2.21%** failures).

2.1 Dataset Selection

I utilized the **Lakh MIDI Dataset (LMD)**, a collection of 176,581 unique MIDI files (Raffel (2016)). MIDI was chosen for its prevalence and richness in symbolic musical information and based on the recommendation provided in the project guidelines. To facilitate training with standard language modeling architectures, I convert these binary MIDI files into **ABC notation**, a text-based format that represents notes, rhythms, and chords using ASCII characters.

2.2 Preprocessing Pipeline

This project was executed in two distinct phases. **Phase 1** was focused on evaluating the scaling law, while **Phase 2** was an attempt at creating a model capable of generating high quality symbolic music.



```
from tqdm import tqdm

failed = []
n_ok = 0

for midi_path in tqdm(midi_files):  # dataset = 176k files
    out_path = abc_root / (midi_path.stem + ".abc")
    if out_path.exists():
        n_ok += 1
        continue

    try:
        midi_to_abc_file(midi_path, out_path)
        n_ok += 1
    except Exception as e:
        failed.append((str(midi_path), repr(e)))

n_ok, len(failed)
```

[9]

```
100%|
```

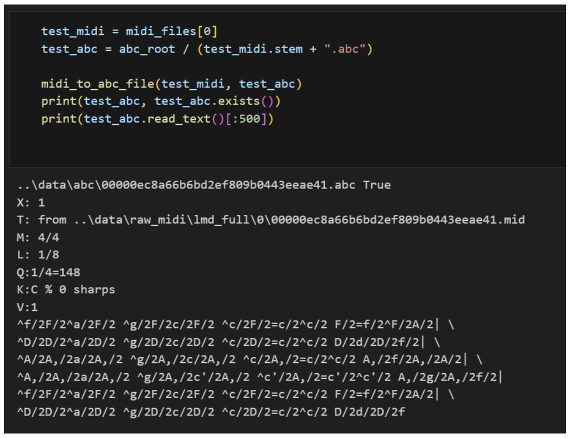
```
(174612, 3949)
```

```
len(failed)
```

[10]

```
3949
```

(a) Successful vs Failed conversions



```
test_midi = midi_files[0]
test_abc = abc_root / (test_midi.stem + ".abc")

midi_to_abc_file(test_midi, test_abc)
print(test_abc, test_abc.exists())
print(test_abc.read_text()[:500])
```

```
..\data\abc\00000ec8a66b6bd2ef809b0443eeae41.abc True
X: 1
T: from ..\data\raw_midi\lmd_full\0\00000ec8a66b6bd2ef809b0443eeae41.mid
M: 4/4
L: 1/8
Q:1/4=148
K:C % 0 sharps
V:1
^f/2F/2^a/2F/2 ^g/2F/2c/2F/2 ^c/2F/2=c/2^c/2 F/2=f/2^F/2A/2| \
^D/2D/2^a/2D/2 ^g/2D/2c/2D/2 ^c/2D/2=c/2^c/2 D/2d/2D/2f/2| \
^A/2A,/2a/2A,/2 ^g/2A,/2c/2A,/2 ^c/2A,/2=c/2^c/2 A,/2f/2A,/2A/2| \
^A,/2A,/2a/2A,/2 ^g/2A,/2c'/2A,/2 ^c'/2A,/2=c'/2^c'/2 A,/2g/2A,/2f/2|
^f/2F/2^a/2F/2 ^g/2F/2c/2F/2 ^c/2F/2=c/2^c/2 F/2=f/2^F/2A/2| \
^D/2D/2^a/2D/2 ^g/2D/2c/2D/2 ^c/2D/2=c/2^c/2 D/2d/2D/2f
```

(b) midi2abc example

Figure 1: MIDI2ABC Conversion statistics

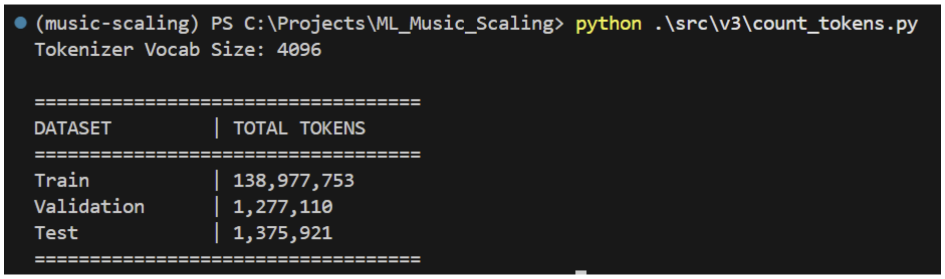
2.2.1 Phase 1: Scaling Study (Character-Level)

For the scaling study, I processed the raw ABC dataset with the goal of minimizing dataset size to exactly **100 million tokens**.

- **Tokenization:** I utilized character-level tokenization initially for the sake of simplicity. For the purpose of evaluting the scaling laws, I believed that a character-level tokenization would suffice. Although the dataset scan identified **79 unique characters** as shown in Figure:5, I constrained the vocabulary size to **64** for simplicity for the purpose of evaluating scaling laws.
- **Formatting:** I retained only structural headers (M:, L:, K:, V:) and stripped metadata like the T: header which only served to provide information on which original midi file the abc file was generated from.
- **Preprocessing Error:** A critical error was made in the preprocessing step. I stripped all lines beginning with the `%` character to remove comments. However, ABC notation relies on lines like `%%MIDI program` and `%%clef` to define instrumentation and pitch ranges. This resulted in the model's generations defaulting to abcjs's default piano instrument when I tried to listen to the generations on abcjs.net. These models also struggled in generating multi-instrument tunes.
- **Train-Test-Validation Split:** The train set had **21.89 billion tokens** but due to the massive file size, I created a **150 million token** subset of the total training set and **exactly 100 million tokens** were used during training for evaluating the scaling law as mentioned in the instructions. The dataset was split into 98% training data, 1% test data and 1% validation data.
- **Serialization format:** Each cleaned song is written as raw text into `train.txt`, `val.txt`, and `test.txt`, with songs separated by two newline characters (`\n\n`). (No explicit end-of-song token was used in Phase 1.)

2.2.2 Phase 2: High-Performance Model (BPE)

For the "BEST" model (206M params), I tried to correct the Phase 1 errors and implemented additional filtering and preprocessing steps:



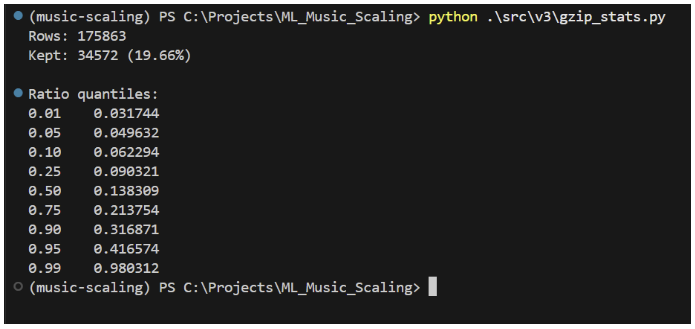
```
● (music-scaling) PS C:\Projects\ML_Music_Scaling> python .\src\v3\count_tokens.py
  Tokenizer Vocab Size: 4096

  ===================================
  DATASET          | TOTAL TOKENS
  ===================================
  Train            | 138,977,753
  Validation       | 1,277,110
  Test             | 1,375,921
  ===================================
```

Figure 2: Enter Caption

- **Preservation of MIDI Cues:** I retained `%%MIDI` headers to ensure the model could learn instrument-specific contexts. While removing all `T:` title lines and removing single-`%` comments. We drop blank lines, literal `...` lines and strip trailing backslashes that appear due to line-wrapping artifacts.
- **Gzip-ratio quality filter (entropy gate):** For each song we compute a **gzip compression ratio**

$$r = \frac{\text{gzip\_bytes}}{\text{raw\_bytes}}$$

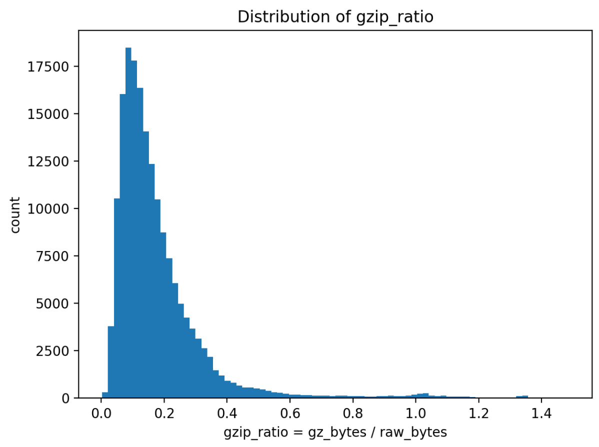
on the cleaned text, where whitespace is stripped before computing entropy so that junk whitespace cannot artificially inflate the score. Intuitively, lower ratios correspond to highly compressible (often repetitive/degenerate) files, while higher ratios indicate higher-entropy, information-dense musical content. We keep a file if $r \geq$ `MIN_RATIO` (default `0.24`). With this threshold, we retained **34,572 out of 175,863 files** (**19.66%**). The resulting ratio distribution was heavily skewed toward low values (median $r \approx 0.138$, 75th percentile $r \approx 0.214$), meaning that most raw conversions were extremely compressible and likely low-quality. While this filtering potentially improved data quality, it also reduced corpus size by $\sim$80%, which may have negatively impacted model performance by limiting diversity and total training signal.



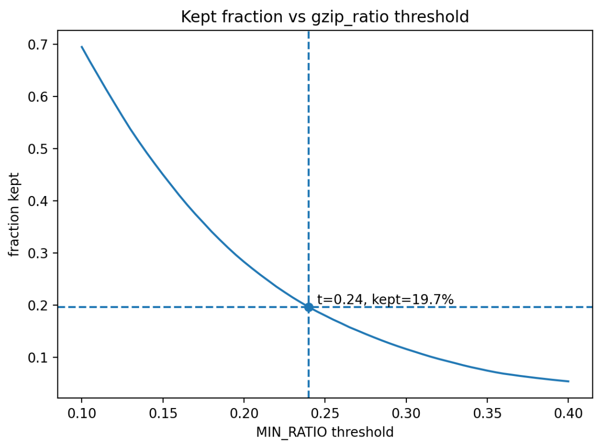
```
● (music-scaling) PS C:\Projects\ML_Music_Scaling> python .\src\v3\gzip_stats.py
  Rows: 175863
  Kept: 34572 (19.66%)

● Ratio quantiles:
  0.01    0.031744
  0.05    0.049632
  0.10    0.062294
  0.25    0.090321
  0.50    0.138309
  0.75    0.213754
  0.90    0.316871
  0.95    0.416574
  0.99    0.980312
○ (music-scaling) PS C:\Projects\ML_Music_Scaling>
```

Figure 3: Gzip stats

- **BPE Tokenization:** I trained a Byte-Pair Encoding tokenizer with a vocabulary of **4096 tokens**.
- **Extended Training:** Unlike the 100M token limit in Phase 1, the Phase 2 model was trained on the full filtered corpus of **136,393,191 tokens**.
- **Leading-silence trimming:** To reduce the "silence loop" failure mode, I trimmed leading rest-only lines at the start of each song. The main reason for the poor performance of models from Phase 1 was the large amount of silence notes present in the data. Trimming the silence substantially improved the model's capabilities to generate music.
- **Explicit end-of-song delimiter:** The final Phase 2 training corpus is written to a single file `data/V3/train.txt` where each cleaned song is followed by the delimiter `\n<|endoftext|>\n`. This gives the model a consistent "song boundary" marker during training and generation.

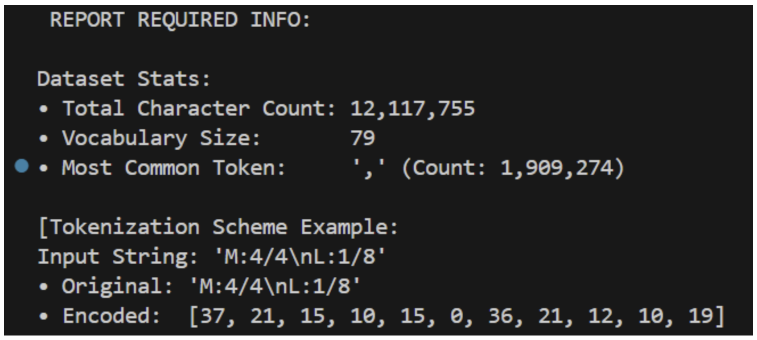


(a) Gzip ratio histogram



(b) Kept fraction vs threshold

Figure 4: Gzip-ratio filtering diagnostics



```
 REPORT REQUIRED INFO:

Dataset Stats:
• Total Character Count: 12,117,755
• Vocabulary Size:       79
• Most Common Token:     ',' (Count: 1,909,274)

[Tokenization Scheme Example:
Input String: 'M:4/4\nL:1/8'
• Original: 'M:4/4\nL:1/8'
• Encoded:  [37, 21, 15, 10, 15, 0, 36, 21, 12, 10, 19]
```

Figure 5: Character Level Tokenization with Example and Dataset statistics

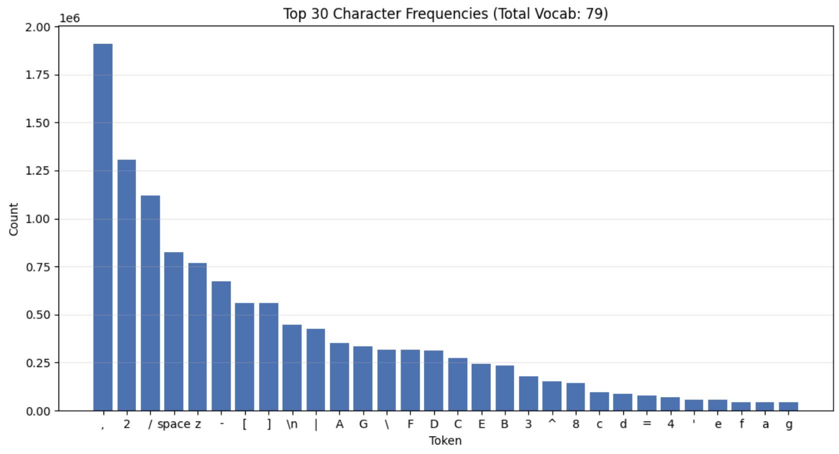


Figure 6: Histogram of vocab distribution

3 Methods

3.1 Model Architectures

I compared two distinct architecture families: **Decoder-only Transformers (GPT)** and **Long Short-Term Memory (LSTM) RNNs**.

3.1.1 Transformer Configurations

I trained a total of 6 GPT-2–style decoder-only language model (masked self-attention Transformer) (Radford et al. (2019)). We implemented and trained it using the nanoGPT codebase by (Karpathy (2022)), with minor modifications for ABC tokenization and dataset preprocessing. The *Scaling Series* (Tiny through XL) utilized character-level tokenization to verify the power law. The *BEST* model utilized the Phase 2 BPE vocabulary and expanded width/depth.

Model	Phase	Vocab	Params	Layers	Embed Dim	Heads
Tiny	1	64	0.8M	4	128	4
Small	1	64	4.8M	6	256	8
Medium	1	64	21.4M	12	384	6
Large	1	64	50.5M	16	512	8
XL	1	64	113.5M	16	768	12
BEST	**2**	**4096**	**206M**	**16**	**1024**	**16**

Table 1: Transformer Model Configurations across both phases.

3.1.2 LSTM Configuration

For the recurrent baselines we used a stacked LSTM language model implemented with PyTorch `nn.LSTM`.

- **Embedding + Hidden size:** the token embedding dimension equals the LSTM hidden size ($d_{\text{emb}} = h = n\_embd$).
- **Depth:** we fixed the number of LSTM layers to $n\_\text{layer} = 2$ for all sizes (standardizing depth while scaling width).
- **Dropout:** 0.1 applied between LSTM layers (PyTorch LSTM dropout is active when $n\_\text{layer} > 1$).
- **Tied embeddings:** input embedding weights are tied with the output projection layer to match standard GPT practice and avoid parameter bloat.

Model	Hidden Size (h)	Layers	Embedding Dim	Total Params
LSTM-Tiny	256	2	256	1.07M
LSTM-Small	560	2	560	5.06M
LSTM-Medium	1100	2	1100	19.45M
LSTM-Large	1750	2	1750	49.14M
LSTM-XL	2500	2	2500	100.20M

Table 2: Exact LSTM baseline configurations used in the scaling study (width-scaled, fixed depth).

3.2 Hyperparameters & Training Knobs

To ensure fair comparison in the scaling study, we fixed the token budget and batch/context settings across model sizes. The Phase 2 "BEST" model used a larger context window and longer training to prioritize generation quality.

Phase 1 (Scaling Study) Fixed Budget:

- **Training token budget:** `MAX_TRAIN_TOKENS` = 100,000,000.
- **Batch size:** `BATCH_SIZE` = 64 sequences.
- **Context window:** `BLOCK_SIZE` = 256.
- **Tokens per step:** $64 \times 256 = 16{,}384$.
- **Total steps (integer budget):** $\lfloor 100{,}000{,}000/16{,}384 \rfloor = 6{,}103$ steps.

Phase 2 ("BEST" Model) Training Knobs:

- **Precision:** mixed precision via `torch.autocast` with `dtype=torch.bfloat16`.
- **Optimizer:** AdamW with $(\beta_1, \beta_2) = (0.9, 0.95)$ and weight decay 0.1.
- **Gradient clipping:** global norm clipped at 1.0.
- **Dropout:** 0.1 (embeddings/attention/MLP residual stream).
- **Micro-batch size:** 16 sequences, with **gradient accumulation** = 4.
- **Effective batch size:** $16 \times 4 = 64$ sequences/step.
- **Context window:** 1024 tokens.
- **Tokens per step:** $16 \times 4 \times 1024 = 65{,}536$.
- **Learning rate:** peak 3×10^{-4}, cosine decay to `MIN_LR` $= 3 \times 10^{-5}$ with `WARMUP_STEPS`=200.
- **Total steps:** `MAX_STEPS` = 11,000 (multiple passes over the filtered corpus).

3.3 Training Setup

All models were trained with the following consistent hyperparameters:

- **Loss Function:** Cross-Entropy Loss
- **Context Window:** 256 tokens
- **Optimization:** AdamW optimizer
- **Duration:** 1 epoch (on 100M tokens)
- **Hardware:** NVIDIA A100 GPU/ Nvidia RTX 5070 Ti [The A100 was primarily used to train the larger models]

3.4 Training Setup

I utilized the AdamW optimizer across all experiments. However, the training duration varied by phase to serve different experimental goals:

- **Phase 1 (Scaling Laws):** To strictly control compute, all scaling models (Transformer Tiny–XL and LSTM Tiny–XL) were trained for exactly **1 epoch on 100 million tokens**.
  - *Steps:* ≈ **6,103 steps**.
  - *Batch Size:* 64 sequences × 256 context length.
- **Phase 2 ("BEST" Model):** To maximize quality, the 206M parameter model was trained on the larger Phase 2 dataset.
  - **Total Tokens:** 136,393,191 tokens.
  - **Steps:** Trained for **11,000 steps** to ensure convergence.

4 Results: Scaling Laws

4.1 Transformer Scaling Study

Here we can observe a clear power-law relationship between the number of parameters (N) and the validation loss (L). Plotting the loss on a log-log scale revealed a near-linear trajectory.

- **Scaling Exponent (α):** By fitting the power law equation $L(N) \approx CN^{-\alpha}$, I derived a scaling exponent of $\alpha \approx 0.0828$.

This result aligns well with the theoretical range for natural language text (typically $0.05 - 0.07$) as shown by Kaplan et al. (2020), suggesting that symbolic music shares similar fundamental information-theoretic scaling properties with language.

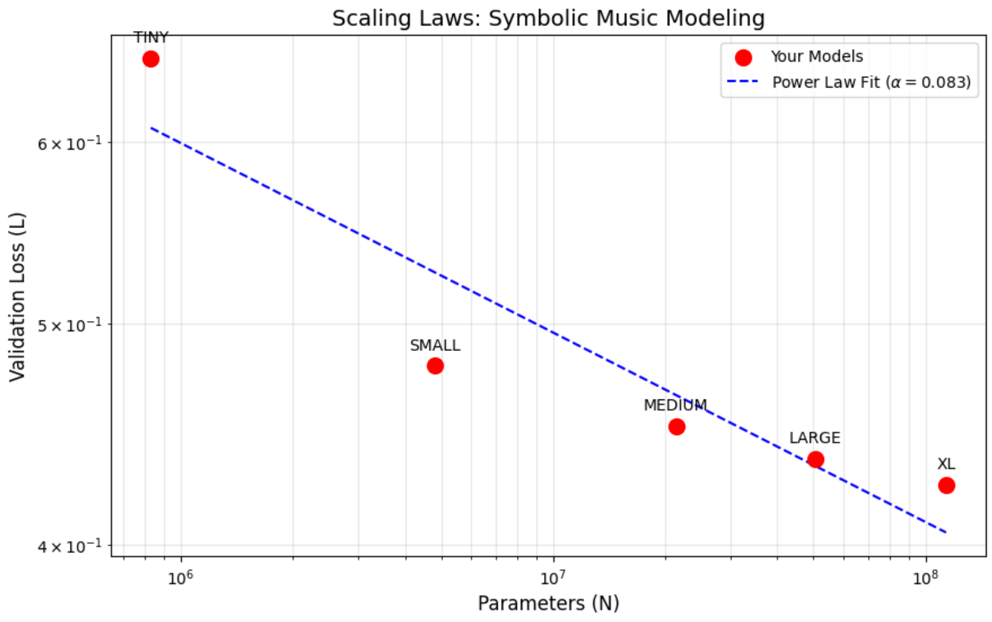


Figure 7: Transformer Scaling Plot: Validation Loss vs. Parameters (Log-Log Scale).

4.2 RNN vs. Transformer Comparison

Comparing the two architectures we can see that:

1. **Low Parameter Regime ($<$ 5M):** The LSTM baseline outperformed the Transformer. The inductive bias of the RNN appears more efficient for learning short-range musical syntax when capacity is limited.
2. **High Parameter Regime ($>$ 20M):** The LSTM performance saturated quickly ($\alpha \approx 0.03$). In contrast, the Transformer continued to improve steadily ($\alpha \approx 0.08$).
3. **Crossover Point:** At approximately 5 million parameters, the Transformer overtook the LSTM. By the XL scale, the Transformer was decisively performing better than the respective LSTM.

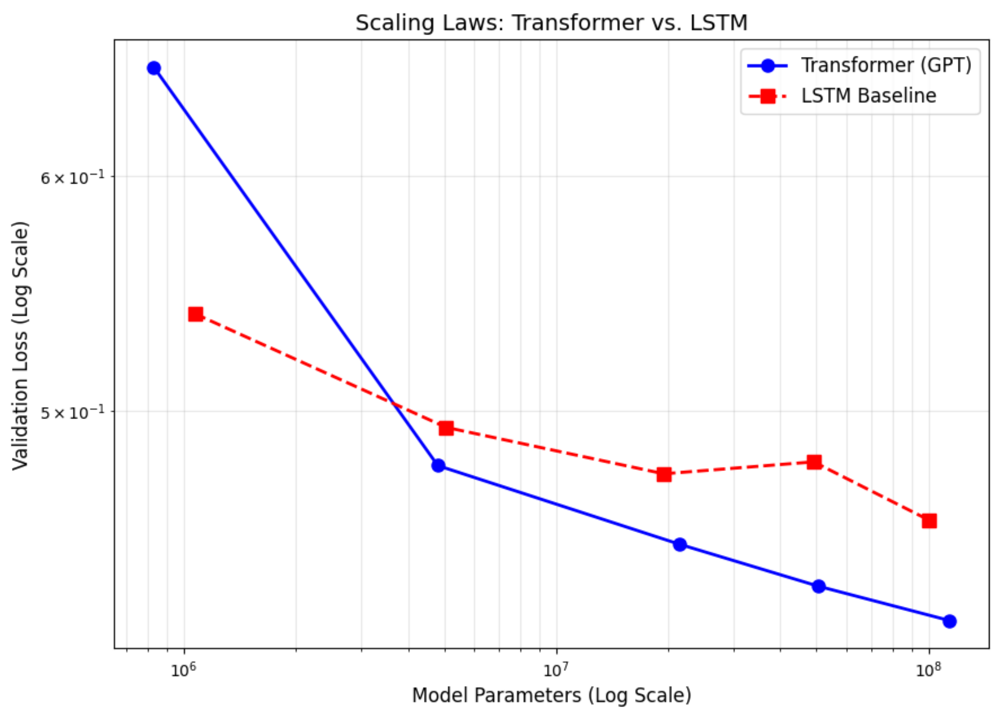


Figure 8: Comparison of Transformer and LSTM scaling behaviors.

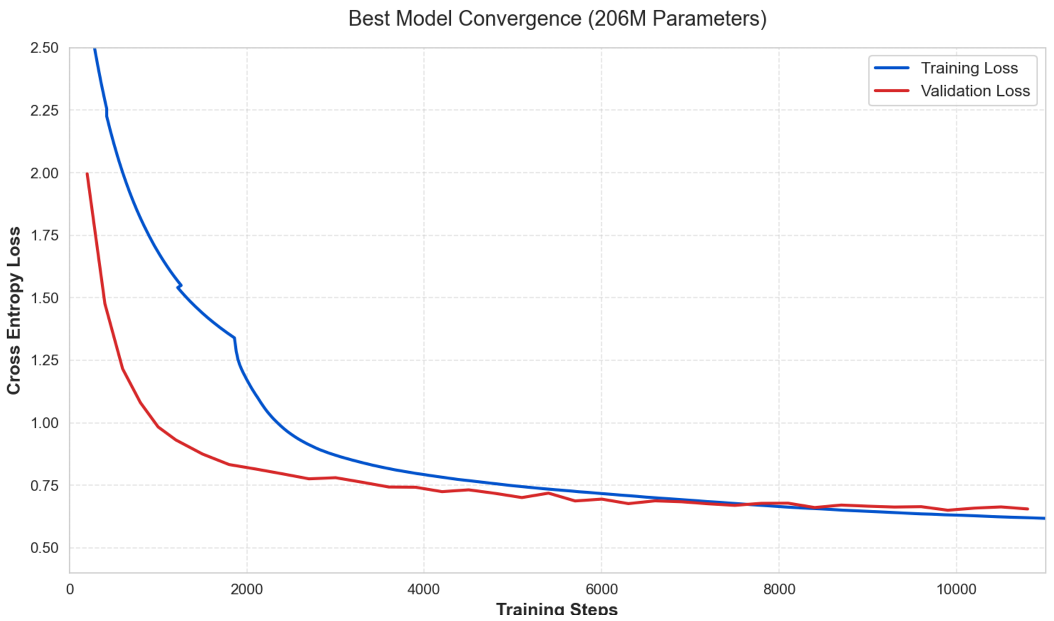


Figure 9: Best model's Training and Validation Loss

4.3 Sample Evaluation

4.3.1 Quantitative Metrics

We evaluated the trained BEST model using three objective measures:

- **Final perplexity on test set: 1.9336**.
- **Syntactic validity rate: 68.0%** of generated samples were successfully parsed as ABC by `music21`.
- **ABC→MIDI success rate: 58.8%** of generated samples successfully converted to MIDI via `music21`.

The gap between syntactic validity and MIDI success indicates that a portion of samples are parseable ABC text but still fail MIDI export due to subtle structural issues.

4.3.2 Qualitative Analysis of Musical Structure

Overall, the model reliably produces correct global structure markers such as `X:`, `M:`, `L:`, `Q:`, and `K:`, and most samples maintain consistent barlines and rhythmic grouping under the declared meter (typically 4/4). Harmonically, many samples show coherent local progressions and repeated motifs consistent with short-form musical phrases.

However, we also observed common failure modes:

- **Excessively dense chord clusters:** Some samples overuse bracketed chords (e.g., many stacked notes per timestep), which can sound noisy or mechanically "blocky."
- **Repetition/local loops:** Some generations fall into short repeating patterns, especially when the texture is sparse or rest-heavy.
- **Instrument/voice inconsistency (when present):** Multi-voice generations are possible, but occasionally voices drift in role or produce unnatural coordination.

Examples from unconditional generations. These abc files along with a few of the best generations can be found in the **github** link attached under **samples**.

- `GREAT.abc`: A strong multi-voice sample with multiple `V:` sections (including a percussion channel) and clear repeating bass/harmony structure, producing the most "arranged" sound among the examples.
- `Good_1.abc`: Single-voice output in key of F with steady tempo; musically plausible phrasing, but more chromatic/dense chord usage.
- `Decent_1.abc`: Single-voice in A♭; coherent rhythmic flow, but tends toward heavy chordal textures and occasionally less idiomatic voice-leading.

5 Discussion and Design Analysis

5.1 Interpretation of Scaling Laws

Our experiments confirm that symbolic music modeling exhibits power-law scaling properties analogous to natural language processing. We derived a scaling exponent of $\alpha \approx 0.08$, which aligns closely with the range of $0.05 - 0.07$ observed in textual language models by Kaplan et al. (2020). This suggests that despite the structural differences between music (hierarchical, rhythmic) and language (semantic, syntactic), the fundamental information-theoretic difficulty of predicting the next token scales similarly with model capacity.

A key finding of our comparative study is the distinct point at approximately 5 million parameters. Below this threshold, LSTMs exhibited better performance due to their inductive bias which handles sequential dependencies without the quadratic complexity of self-attention. However, as model capacity increased beyond 5M parameters, Transformers started to show much better performance. This confirms that while RNNs are efficient for lightweight applications, Transformer architectures are better for scalable, high-performance music generation and similar NLP tasks.

5.2 Design Decisions and Impact

Tokenization Strategy: In Phase 1, we utilized character-level tokenization (Vocab Size: 64) to allow the model to learn the raw syntax of ABC notation. While this minimized the vocabulary embedding overhead, it significantly increased the sequence length required to represent meaningful musical ideas. In Phase 2, we shifted to a Byte-Pair Encoding (BPE) tokenizer with a vocabulary of 4096. While this improved expressivity and the model's capabilities to produce music using multiple instruments rather than sticking to one particular instrument, I believe the vocabulary size of 4096 was too small to capture complex common musical structures (e.g., full chords or recurring motifs) as single tokens. Consequently, the model often required 5-6 tokens just to output a single valid chord. This fragmentation effectively reduced the model's context window, causing it to lose track of global attributes like key signature and time signature more rapidly than it would have with a larger vocabulary.

Handling Data Artifacts (The "Silence Loop"): Qualitative analysis of early checkpoints of models used for the scaling study trained using a character-level tokenizer (Phase 1 models) revealed a "Silence Loop" failure mode, where models would generate infinite strings of rests (e.g., `z8|z8|`). This was traced to the prevalence of multi-track MIDI files in the training data, where accompaniment tracks often contain long periods of silence. By implementing a heuristic to trim leading rest-lines in Phase 2, we forced the model to predict musical onsets earlier in the context window, significantly improving generation quality.

The preprocessing methods I used, such as trimming leading silence and filtering based on gzip compression ratios, prioritize dataset density over completeness. While necessary for efficient training within the time constraints, these steps may have introduced data artifacts or filtered out valid minimalist compositions, a trade-off that could potentially be explored in the future for the sake of completeness.

5.3 Limitations

- **Context Window constraints:** We utilized a context window of 256 tokens due to compute constraints for Phase 1. In the domain of ABC notation, this captures roughly 4-8 bars of music. This limited context prevented the model from learning long-term musical structures, such as ABA form or recurring themes that span an entire composition. Our Phase-2 model used used a context window of 1024 tokens and was capable of producing non-repetitive tracks unlike the models from Phase 1.

- **Overfitting and Memorization:** Our "Best" model (206M parameters) was trained for approximately 5 epochs (11,000 steps) on a relatively small filtered corpus of $\approx$136M tokens. I believe this could have potentially led to over-parameterization, where the model tended to memorize specific training examples rather than learning generalizable rules of improvisation.

5.4 Future Work

Future iterations of this work would benefit from three key improvements:

1. **Expanded Context and Vocabulary:** Increasing the context window to 1024+ tokens and the BPE vocabulary to $\approx$20,000 would allow the model to capture full musical phrases and reduce the "forgetting" of key signatures.

2. **Multi-Modal Evaluation:** While we relied on loss metrics and text-based inspection, integrating a direct MIDI-to-Waveform rendering pipeline during validation would allow for better perceptual evaluation of the generated music.

3. **Data Augmentation:** To combat overfitting, techniques such as pitch transposition (augmenting the dataset by transposing songs to different keys) could artificially increase the effective dataset size and improve the model's key-invariance.

4. **Better Data Preprocessing Pipeline:** A significant amount of the work was done in preprocessing the data; in the end we discarded roughly 80% of the data in an attempt to fix the repetitive nature and the silence loop problems. A better preprocessing pipeline could be constructed taking into account diversity of instruments, and implementing granular track-level filtering. Instead of discarding entire

files, future pipelines should dynamically isolate high-entropy tracks (melody and harmony) while ignoring sparse accompaniment layers, thereby maximizing data utilization without reintroducing silence artifacts.

6 Conclusion

This project successfully established that symbolic music modeling follows neural scaling laws similar to those in natural language processing. By training a family of models ranging from 0.8M to 113M+ parameters, we demonstrated a predictable power-law improvement in validation loss with a scaling exponent of $\alpha \approx 0.08$.

Our results highlight a critical architectural trade-off: while LSTMs offer superior efficiency at the "Tiny" scale (< 5M parameters), they lack the scaling properties required for high-fidelity generation. Transformers, despite their computational cost, emerge as the clear choice for scalable music modeling. However, our analysis of the "Best" model reveals that scaling parameters alone is insufficient; data quality, tokenizer granularity, and context length are equally critical bottlenecks. Future work lies not just in making models larger, but in better representing the hierarchical language of music itself.

7 Appendix

All code and generation samples along with information on the various scripts used etc can be found at: github.com/suryaajith21/Symbolic-Music-Scaling-Project

References

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. (2020). Scaling laws for neural language models.

Karpathy, A. (2022). NanoGPT. `https://github.com/karpathy/nanoGPT`.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Raffel, C. (2016). *Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching*. PhD thesis, Columbia University.