

# CS-GY 6923: Machine Learning – Optional Project

Michael Blanchette

## 1. Introduction

Scaling laws have become an important tool for understanding and modeling the behavior of neural language models. Seminal work by OpenAI has demonstrated that the performance of language models can be described by predictable power-law relationships with respect to factors such as model size, dataset size, and computing power (Kaplan et al., 2020).

Although scaling laws have been extensively researched in the natural language domain, their application to other types of sequential data has remained underdeveloped. Symbolic music represents one such domain and is a widely compelling area for the exploration of this phenomenon. Unlike natural language, the notation of music is governed by explicit (and sometimes implicit) structural rules, including but not limited to time signatures, key, and rhythm. Even so, music still exhibits the hierarchical and long-range dependencies that are present in language modeling, making the combination of formal and creative complexity of symbolic music an interesting area to examine scaling laws beyond text.

This project aims to investigate scaling laws in symbolic music using transformer and RNN-based models. We convert a large corpus of MIDI files from the Lakh MIDI Dataset (Raffel, n.d.) into ABC notation, a method to represent musical notes and rhythms using readable, text-based symbols (Benini, 2025), which is then used for training. We then train families of transformer and LSTM models across a variety of parameter counts to derive the scaling law relationships.

Our main contributions are as follows:

1. We develop a complete preprocessing pipeline for converting MIDI files to tokenized ABC notation, producing a corpus of over 13.9 billion tokens.
2. We empirically derive scaling laws for transformer models in the symbolic music domain, fitting power-law curves to characterize the relationship between validation loss and model size.
3. We conduct a comparative scaling study between transformer and LSTM architectures, analyzing differences in scaling efficiency and computational cost.
4. We select our best-performing model for generating music samples, evaluating both syntactic validity and musical coherence.

## 2. Data

### 2.1 Dataset Selection

We use the Lakh MIDI Dataset (LMD) as our primary data source, as recommended by the project guidelines. The LMD contains 178,561 unique MIDI files spanning a variety of musical styles, genres, and instrumentation. We proceeded with this dataset for several reasons: its large scale provided more than sufficient data for training models of increasing orders of magnitude in parameter count, its diversity

in genre provided the wide range of exposure the models needed to learn varied musical patterns and structures, and its common use in symbolic music research made it more convenient to compare our findings to prior work.

## 2.2 Preprocessing Pipeline

Our preprocessing pipeline converts raw MIDI files into ABC notation, encoding musical information including pitch, duration, key signatures, time signatures, and structural markers using ASCII characters, making it directly applicable for character-level language modeling.

The conversion pipeline consists of:

1. **Extraction and Cleanup:** We first extract the compressed dataset and clean up artifacts from the archive structure, including removing macOS-specific metadata directories that can interfere with file processing (Since the system we used to download and transfer the data was a MacBook Pro, which was then uploaded to Google Colab to carry out the project).
2. **Parallel MIDI-to-ABC Conversion:** We convert each MIDI file to ABC notation using the midi2abc command-line tool. To accelerate processing, we parallelize conversion across multiple CPU workers. Each conversion is wrapped in a try-except block with a 10-second timeout to take note of and handle corrupted or problematic files. Of the 178,561 MIDI files in the dataset, 175,961 were successfully converted to ABC notation, while 2600 files (approximately 1.46%) failed conversion due to corruption, formatting issues, or timeouts. Regardless, this 98.54% success rate provides sufficient data for our experiments.
3. **Tokenization:** We scan through all converted ABC files to identify every unique character (Further discussed in Section 2.3).
4. **Validation and Analysis:** We identify empty or invalid ABC files by checking if they contain at least one ABC header field ('X:', 'T:', 'M:', or 'K:') and contain notes.
5. **Length-Based Filtering:** We analyze the token length distribution of the valid files (Appendix A) from the last step to identify an appropriate filtering threshold. We found significant variability during this step, with some files ranging from 120 tokens to 549 million tokens. We implemented a threshold of between 200 and 100,000 tokens to remove outliers, including 1083 files that are too short to contain meaningful content, and remove 1226 extreme outliers that would overly dominate training.

## 2.3 Tokenization Scheme

We adopt character-level tokenization for our ABC notation corpus. This was chosen for several reasons:

- **Simplicity:** No parsing logic is required, making the pipeline straightforward to debug.
- **Vocabulary Sizing:** The number of characters in our vocabulary would be minimal.
- **Detailed Modeling:** Highly meaningful information can be captured at the character level. For example, the difference between 'C' and 'c' denotes different octaves, and characters like '^' describe a sharp tone while '\_' describes a flat tone that modifies the following note.

Our vocabulary consists of 100 unique characters, including:

- Letters A-Z and a-z represent notes across octaves.
- Digits 0-9 represent note durations.
- ABC notation syntax of punctuation and special characters.

The vocabulary is as shown below:

```
!"#$%& ' () *+, -./0123456789: ;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcde
fghijklmnopqrstuvwxyz{|}~
```

## 2.4 Dataset Statistics

Appendix B summarizes the complete data preprocessing pipeline from raw MIDI files to the final cleaned dataset. The final dataset retains 97.3% of the original LMD files, where our cleaned corpus contains approximately 3.7 billion tokens, substantially exceeding the 100 million token minimum specified in the project requirements. The final train/validation/test split (98%/1%/1%) is outlined in Appendix C.

## 3. Methods

### 3.1 Model Architectures

We implement two neural language model architectures for our study: a decoder-only transformer model and a recurrent LSTM model.

#### 1. Transformer

The transformer implementation is derived from the GPT-2 architecture as implemented in nanoGPT (Karpathy, 2022). The core architectural components of causal self-attention, feed-forward MLP, residual blocks with pre-normalization, and weight initialization are borrowed directly from the nanoGPT repository. For our study, we implemented the following components that augment the transformer model:

- Parameter counting utilities that exclude position embeddings (which scale with context length rather than model capacity).
- The training loop includes metrics collection, checkpointing, and logging.
- Early stopping with patience-based termination.
- GPU memory tracking and throughput measurement.
- The model configuration system for defining multiple model sizes.

#### 2. LSTM

On the other hand, we implement a standard LSTM model with learned token embeddings, stacked LSTM layers with forget gate bias initialization, and a linear output projection. Weight tying is applied when embedding and hidden dimensions match.

We train five transformer configurations (Appendix D.1) and four LSTM configurations (Appendix D.2) spanning approximately two orders of magnitude in parameter count.

## 3.2 Training Setup

All models were trained on the NVIDIA A100 GPU and NVIDIA L4 GPU via Google Colab. The A100 GPU was primarily used, but training needed to be shifted to the L4 GPU for some of the LSTM models due to computation and cost constraints.

### Optimization

We applied the AdamW optimizer with  $\beta_1 = 0.9$ ,  $\beta_2 = 0.95$ , and weight decay of 0.1. A cosine-scheduled learning rate is used with a linear warmup, where its peak learning rate reaches  $3 \times 10^{-4}$  from a minimum learning rate of  $3 \times 10^{-5}$ , including 100 warmup iterations. All gradients are clipped to a maximum norm of 1.0.

### Batching

We use a batch size of 64 sequences with a context length of 256, containing 16,384 tokens per batch. Training data is sampled randomly with replacement from the tokenized corpus.

### Regularization

All models use a dropout of 0.1 applied to attention weights, MLP outputs, and residual connections (Transformer) or between LSTM layers and before the output projection (LSTM).

## 3.3 Experimental Design

We investigate neural scaling laws to see how they describe model performance improvement as a power-law function of compute, model size, or dataset size, as mentioned in Section 1. Specifically, we will examine the relationship between cross-entropy loss and non-embedding parameter count, expecting a relationship of the form  $L = a \cdot N^{-\alpha} + c$ , where  $N$  is the parameter count.

To trace this scaling curve, we train each model in Section 3.1 and report the validation loss at the best checkpoint for each model.

## 3.4 Evaluation Metrics

For each model, we will measure the following metrics:

- **Validation loss:** Cross-entropy loss on held-out data, averaged over 50 batches.
- **Training throughput:** Tokens processed per second.
- **Peak GPU memory:** Maximum GPU memory allocated during training.
- **Wall-clock time:** Total training duration.

We will then perform scaling analysis as mentioned in Section 3.3 to fit the power-law relationships and examine how validation loss scales with parameter count  $N$  across different model architectures.

## 4. Results

### 4.1 Transformer Scaling Behavior

Appendix E presents the main scaling results of the transformer models: the fitted power-law relationship (Appendix E.1), model/training comparison (Appendix E.2), wall-clock time (Appendix E.3), and GPU memory usage (Appendix E.4).

The validation loss of the transformer models follows a clear power-law relationship across the different model sizes (Appendix E.1). We can fit a power-law function of the form  $L = 9.67 \cdot N^{-0.377} + 0.021$ , achieving an  $R^2 = 0.999$ , indicating that the power-law model can capture nearly all the variance in the data.

Training times ranged from 421 seconds for the Tiny model to 10,086 seconds for the XL model (Appendix E.3), and GPU memory usage ranged from 1.24 GB to 17.11 GB (Appendix E.4) accordingly. The final validation loss decreased from 0.078 for the tiny model (0.8M parameters) to 0.032 for the XL model (85M parameters), representing a 60% reduction in loss over a 106x increase in parameter count (Appendix E.1). However, the diminishing returns characteristic of power-law scaling are evident: the final three models (Medium, Large, and XL) achieve similar losses despite substantial differences in size, suggesting that these models are approaching the irreducible entropy of the musical corpus.

Appendix E.5 presents the same data from Appendix E.1 but on a log-log axis, where we can interpret the power-law relationship as a linear function. Overall, training has revealed that the larger models converge to lower losses but require proportionally more computational power. All transformer models completed training without early stopping, indicating that none had fully converged, and additional training and data could potentially yield further improvements. Throughput decreased substantially with model size, from approximately 238,000 tokens per second for the smallest model to 10,000 tokens per second for the largest (Appendix E.6).

## 4.2 LSTM Scaling Behavior (Comparison with Transformers)

Appendix F presents the main scaling results of the LSTM models with a comparison to the previous transformer models: the fitted power-law relationship (Appendix F.1), wall-clock time (Appendix F.2), GPU memory usage (Appendix F.3), and training efficiency comparison (Appendix F.4).

As seen in Appendix F.1, both architectures exhibit power-law scaling behavior, but with slightly different characteristics. The LSTM achieves a steeper scaling exponent ( $\alpha = 0.663 \pm 0.071$ ) compared to its transformer counterpart ( $\alpha = 0.377 \pm 0.047$ ), meaning that LSTM performance generally improves more rapidly with additional parameters. However, the transformer scaling curve achieves a lower loss floor ( $c = 0.021$  versus the LSTM's  $c = 0.032$ ), indicating superior asymptotic performance. This difference may reflect the transformer model's ability to use attention to capture long-range dependencies in musical structure more distributedly, such as recurring themes, melodic boundaries, and harmonic progressions that span many tokens.

The computational tradeoffs between architectures are substantial. As shown in Appendix F.2, the LSTM models train approximately 7.7x faster than the transformer models in aggregate (2859 seconds total versus 21,882 seconds) and require 5.7x less peak GPU memory (3.0 GB versus 17.1 GB in Appendix F.3). Average throughput for LSTMs (305,551 tokens per second) exceeds transformers (75,162 tokens

per second) by a factor of four. These efficiency advantages likely stem from the LSTM's linear complexity in sequence length, compared to the transformer's quadratic attention mechanism.

However, when considering wall-clock compute, transformers remain competitive. The Training Efficiency plot (Appendix F.4) shows that while LSTMs achieve strong performance quickly, the transformer models ultimately reach lower losses given sufficient compute. Specifically, the XL transformer model achieves a final loss of 0.032, outperforming the best LSTM loss of 0.036 despite requiring substantially more training time (Appendix F.1).

A high-level summary comparison between the transformer and LSTM model scaling properties can be viewed in Appendix G.

### 4.3 Generated Music Samples

To assess the models beyond cross-entropy loss and computational usage, we generated ABC notation sequences using the best-performing model: The XL transformers model (85M parameters), which achieved a validation loss of 0.032. Quantitative evaluation of the model on the test set yielded a test loss of approximately 0.1883 and a test perplexity of 1.21, indicating strong ability for the model to generalize to unseen musical sequences (Appendix H).

10 unconditional samples and 10 conditional samples were generated. For unconditional sample generation, the model received a minimal ABC header as a prompt (Appendix I). The model then generated up to 512 tokens using temperature 0.9 and top-k sampling with k=40.

For conditional sample generation, we provided 10 distinct prompts specifying different musical styles, meters, and keys to test the model's ability to respect stylistic constraints. These prompts included waltzes (3/4 time), jigs (6/8 time), marches, reels, slow airs, polkas, and hymns across various keys, including G major, D major, A minor, E major, C major, G mixolydian, D minor, F major, and B-flat major. Conditional samples used temperature 0.85 and top-k=50.

All 20 generated samples were successfully converted to MIDI using abc2midi, demonstrating that the model produces structurally coherent ABC notation that external parsers recognize as valid music. However, syntactic validity, which we define as containing required header fields, note characters, barlines, and balanced brackets, was achieved by only 65% of samples. The gap between MIDI conversion success and our stricter syntactic validity metric suggests that abc2midi tolerates minor formatting irregularities that our heuristic checks flag as invalid. In other words, while we are still able to play the musical sample generated by the model, the sample the model was expected to generate may not be complete or fully coherent, producing malformed syntax, unbalanced repeat signs, or invalid note sequences. Five of these samples (3 unconditional, 2 conditional) can be viewed in Appendix J.

## 5. Discussion

### 5.1 Qualitative Evaluation

Most of our samples exhibit musically coherent results with discernible rhythmic structure and adherence to the specified key signature. When playing the samples, we can identify music that is generally organized in that phrases align with the boundaries of the bars, note durations follow logical patterns, and melodies remain within the expected ranges for the given key.

However, closer inspection also reveals several limitations. The XL transformer model demonstrates a strong tendency to insert extended rests at the beginning of many generated sequences, as seen in Sample 2 of Appendix J, suggesting that the training corpus may have contained significant leading silence that the model has learned to reproduce. To account for this, future work may require the trimming of overly empty spaces within its samples if the goal is to produce samples that begin with musical content immediately.

Furthermore, many generated samples exhibit limited melodic diversity, often rigidly sticking to a narrow range of notes rather than exploring the full melodic combinations within the specified key (such as Sample 4, Sample 5, and the final portion of Sample 2 in Appendix J). While it maintains a rhythmically coherent nature, it results in music that lacks the creativity and variation that we typically may expect from human-composed music.

### 5.2 Key Insights from Scaling Experiments

Our scaling experiments reveal important findings for symbolic music modeling. First, both the transformer and LSTM architectures exhibit clear power-law scaling behavior, providing strong evidence that neural scaling laws can be extended beyond natural language applications to structured musical representations. The scaling exponent of the transformer model ( $\alpha = 0.377 \pm 0.047$ ) is smaller than the  $\alpha \approx 0.5$  reported by Kaplan et al. (2020) for natural language, which may be due to the structural differences between ABC notation and writing. Specifically, musical notation is inherently defined by highly regular patterns, repeated motifs, and a much more constrained vocabulary.

The transformer model's lower loss floor ( $c = 0.021$  versus  $c = 0.032$  for the LSTM) provides evidence that attention mechanisms provide a meaningful advantage for modeling musical sequences, likely due to their ability to directly attend to distant but musically relevant notation, such as a repeated musical idea or a motif that has not been repeated for a while.

Finally, the convergence of validation losses as we increased the number of parameters in our transformer models signals that we may be approaching a fixed barrier, suggesting that future improvements may require further expansion of the training data or refinement of the preprocessing representation before adding more parameters.

### **5.3 Interpretation of Results in the Context of Music Modeling**

The low perplexity of 1.21 reflects ABC notation’s constrained structure. With a vocabulary of only 100 characters and highly rigid formatting, a large amount of each sequence is generally quite predictable. This is a strong contrast with natural language, where perplexities are generally higher, reflecting the greater entropy in open-ended text.

Furthermore, the 100% MIDI conversion rate demonstrates that character-level tokenization transformer models can learn ABC notation and create MIDI outputs successfully without explicit supervision. However, the 65% syntactic validity we achieved suggests that abc2midi overly tolerates notation errors, which leads to many notations being incomplete and many MIDI files being unable to play any sound despite having some notation.

### **5.4 Design Decisions and Their Impact**

The decision to use character-level tokenization, while highly efficient and convenient, may have limited the ability of our models to abstract musical notation at a higher level. Tokenizing at a less granular level and taking into account notes and chords as larger units may have helped provide more versatile results in our models’ performance. Additionally, our fixed content length of 256 tokens also suffers from the same tradeoff between efficiency and performance. While sufficient for shorter notations, this window may limit the learning of structures that span multiple sections, possibly contributing to the highly repetitive and limited-range melodies observed in Appendix J.

### **5.5 Limitations and Future Work**

Several limitations are present in this study that could be further improved for future research.

First, our evaluation primarily focuses on automated metrics, such as perplexity, syntactic validity, MIDI conversion, and brief qualitative assessments by one individual. To extend the exploration of this research, a rigorous study involving multiple sources of human evaluation, providing feedback on the quality of musicality, stylistic features, and overall satisfaction, may provide more meaningful measures of each model’s quality of generation.

Second, the tendency for the models to add excessive rests indicates preprocessing deficiencies. Future work should normalize the beginnings of the music by trimming leading rests and ensuring consistent formatting across the corpus.

Finally, extending this research to more multimodal analysis may offer more promising results and research opportunities. Further work could explore representations beyond ABC notation and also incorporate machine listening to take into account additional factors such as the volume of music (e.g. if the music is getting louder (crescendo) or softer (decrescendo)), or even analyze real audio recordings of music rather than just MIDI files. By exploring more hierarchical architectures, a more optimal model may be able to have improved long-range coherence and output more human-like representations of music.

## 6. Conclusion

This research explored neural scaling laws for symbolic music generation with the help of ABC notation, comparing the performance of transformer and LSTM architectures across model sizes ranging from 1M to 85M parameters. Our experiments present evidence that both architectures exhibit clear power-law scaling behavior of these musical sequences, with their validation loss decreasing accordingly as model size increases.

The key insight is that it is seemingly feasible to apply neural language modeling techniques meaningfully to symbolic music, as is the case with scaling laws. While both the transformer and LSTM model successfully learn the structure of ABC notation, the rapid convergence to low loss values and shallow scaling exponents suggest that ABC notation's nature of constrained vocabularies and repetitive patterns is simply easier to model than previous natural language applications. This suggests that progress in symbolic music generation may depend less on the scale of the models used and more on the quality of data provided, tokenization, and architectural decisions tailored to hierarchical musical structures.

## 7. Appendices

### (A) Token Length Distribution (Before Filtering)

Percentile	Token Count
0th (min)	120
1st	232
5th	791
25th	7426
50th (median)	19,363
75th	31,295
95th	54,596
99th	87,370
100th (max)	549,109,888
<b>Mean</b>	<b>78,833</b>
<b>Std Dev</b>	<b>3,336,546</b>

## (B) Data Pipeline Statistics

Stage	Num Files
Original MIDI Files	178,561
After ABC Conversion, Cleanup & Validation	175,961
Removing Short Outliers (<200 token threshold)	-1083
Removing Long Outliers (>100K token threshold)	-1226
<b>Final Cleaned Dataset</b>	<b>173,652 (Consisting of 3,737,314,329 tokens)</b>

## (C) Train/Validation/Test Split

Split	Token	Percentage
Training	3,662,568,043	98.0%
Validation	37,373,143	1.0%
Test	37,373,143	1.0%
<b>Total</b>	<b>3,737,314,329</b>	<b>100%</b>

## (D) Model Configurations

### (D.1) Transformer Configurations

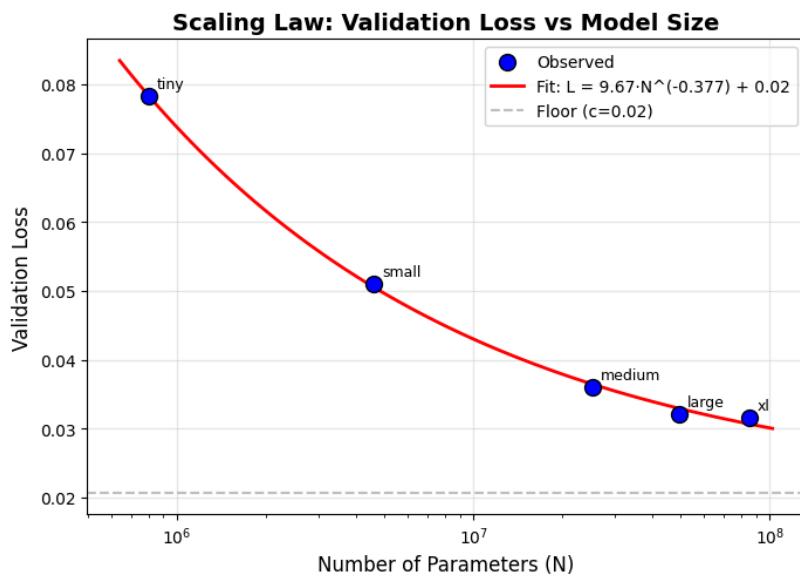
Model	Layers	Heads	d_model	d_head	Parameters
Tiny	4	4	128	32	800K
Small	6	6	252	42	4.6M
Medium	8	8	512	64	25.2M
Large	10	10	640	64	49.2M
XL	12	12	768	64	85.0M

## (D.2) LSTM Configurations

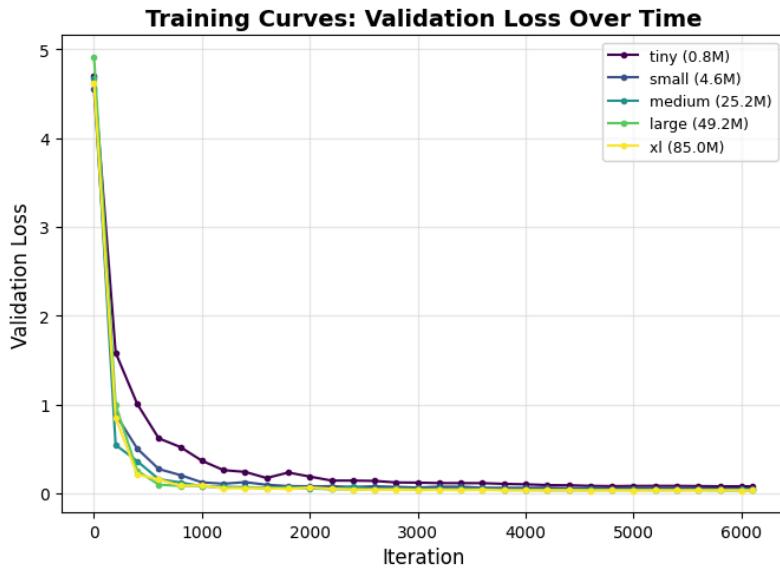
Model	Layers	d_embed	d_hidden	Parameters
Small	2	256	256	1.1M
Medium	2	384	512	4.0M
Large	3	512	768	13.5M
XL	4	768	1024	32.7M

## (E) Transformer Model Analysis

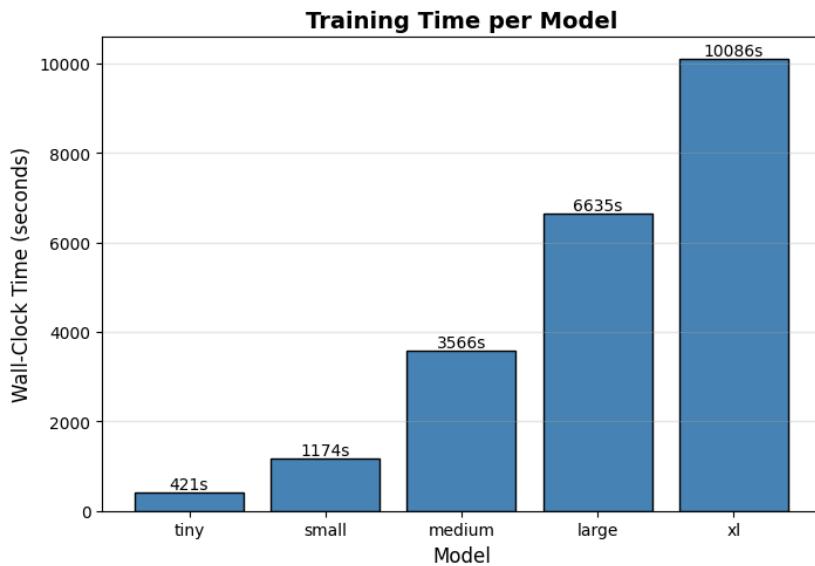
### (E.1) Transformer: Scaling Law Analysis



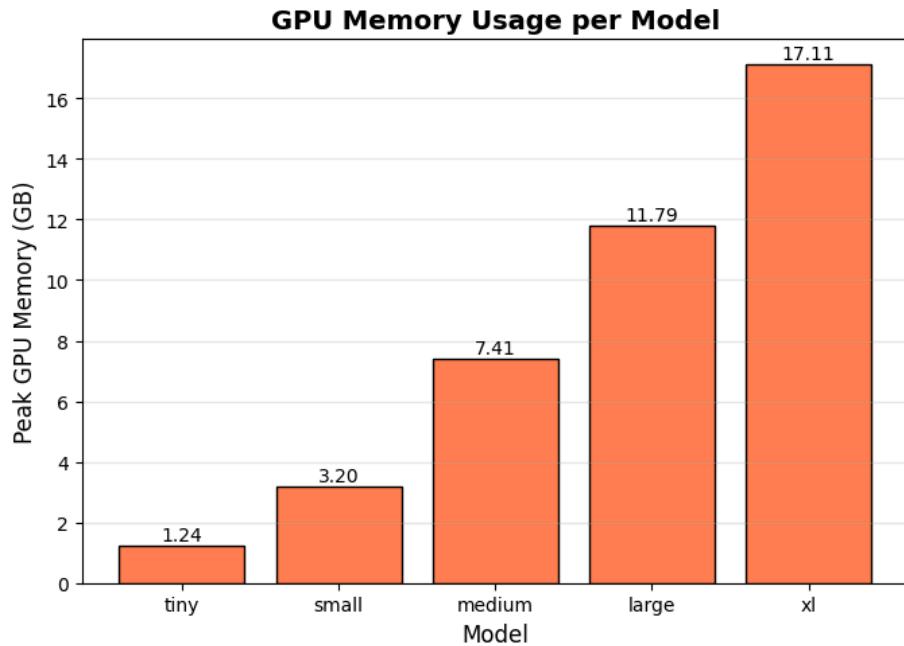
## (E.2) Transformer: Training Curve Comparisons



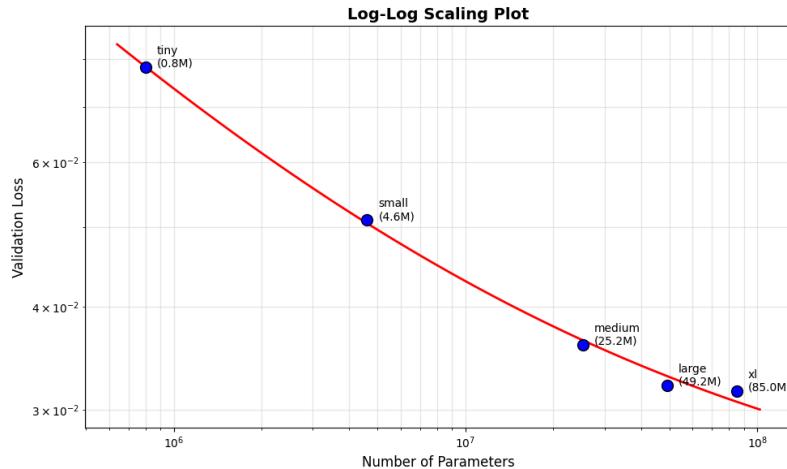
## (E.3) Transformer: Wall-Clock Time



## (E.4) Transformer: GPU Memory Usage



## (E.5) Transformer: Log-Log Scaling

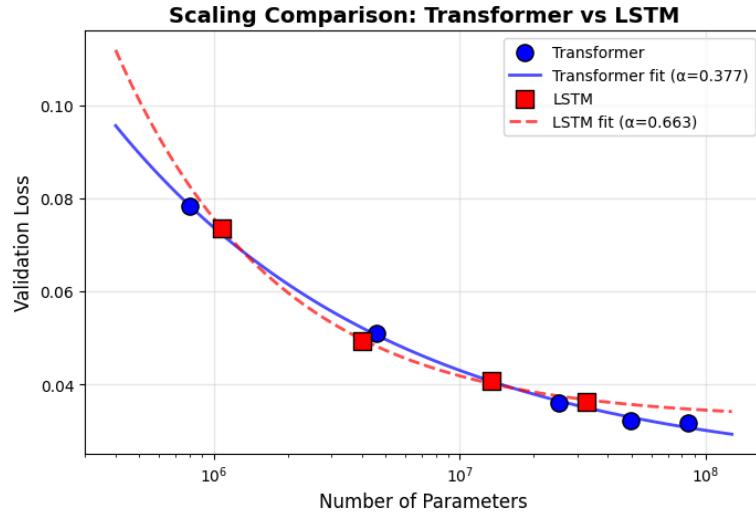


## (E.6) Transformer: Model Configuration and Training Results

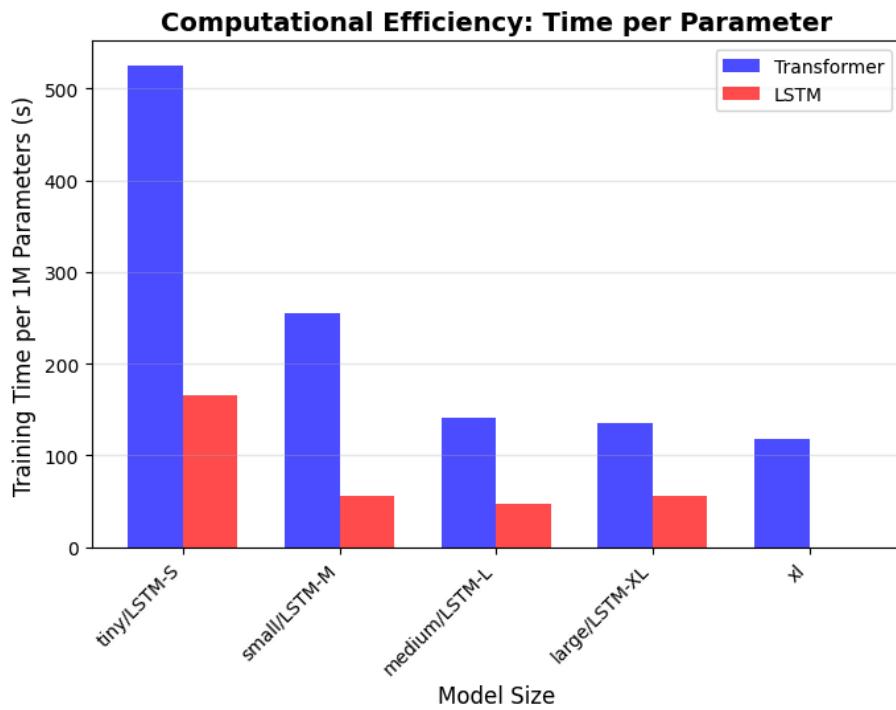
Model	Params	Val Loss	Train Loss	Time (s)	GPU (GB)	Tok/s
tiny	800,384	0.0783	0.2790	420.8	1.24	237,621
small	4,600,764	0.0510	0.2064	1174.1	3.20	85,166
medium	25,225,728	0.0360	0.1595	3566.3	7.41	28,038
large	49,229,440	0.0321	0.1593	6634.6	11.79	15,071
xl	85,030,656	0.0316	0.1595	10086.2	17.11	9,914

## (F) LSTM Model Analysis (Compared Against Transformer Models)

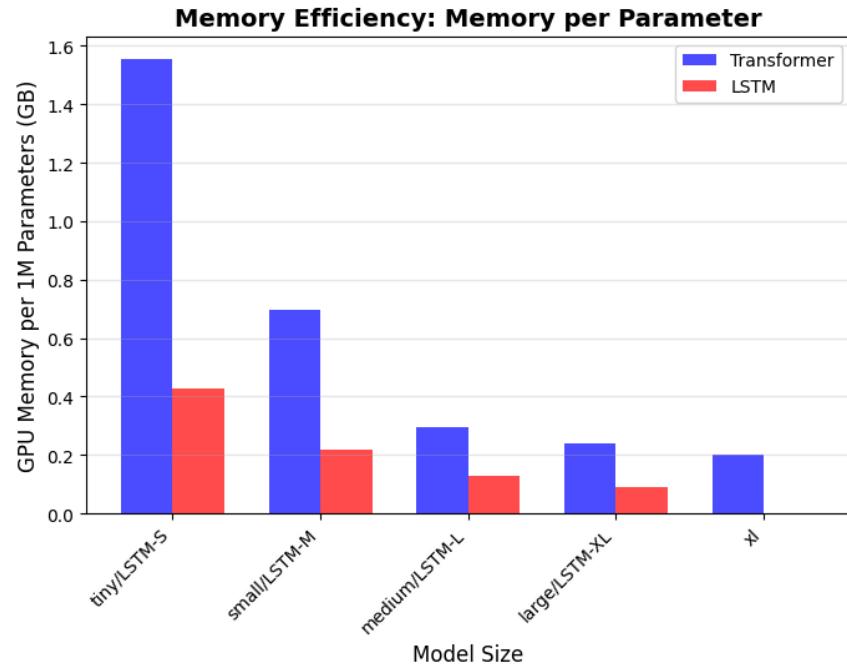
### (F.1) LSTM: Scaling Law Analysis



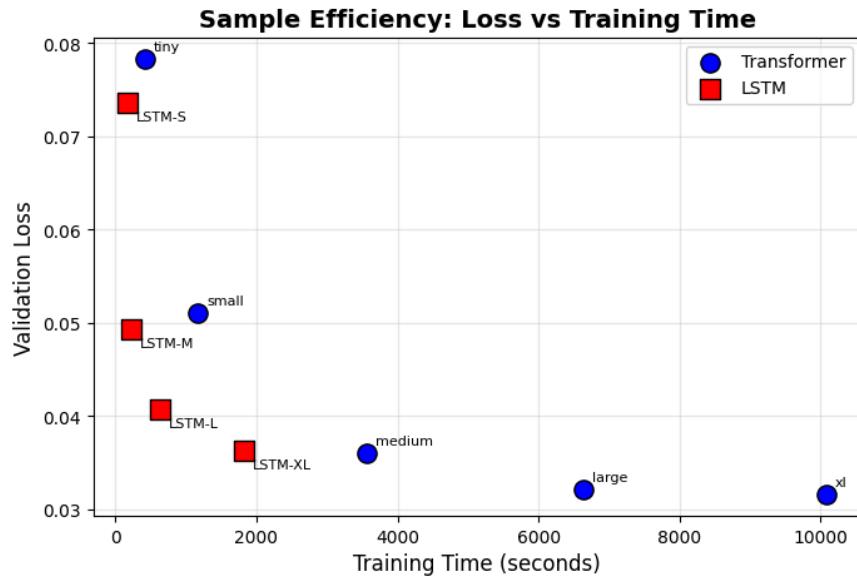
### (F.2) LSTM: Wall-Clock Time (Per 1M Parameters)



### (F.3) LSTM: GPU Memory Usage (Per 1M Parameters)



### (F.4) LSTM: Training Efficiency



### (F.5) LSTM: Training Results

Model	Params	Val Loss	Time (s)	GPU (GB)
LSTM-S	1,078,372	0.0736	178.9	0.46
LSTM-M	4,030,052	0.0493	222.5	0.88
LSTM-L	13,515,876	0.0407	628.1	1.72
LSTM-XL	32,717,924	0.0363	1829.3	3.00

LSTM Power Law Fit:  $L = 410.80 \cdot N^{-0.6629} + 0.03$   
 $\alpha = 0.6629 \pm 0.0713$   
 $R^2 = 0.9993$

## (G) Transformer vs LSTM Summary Table

Metric	Transformer	LSTM
Scaling exponent ( $\alpha$ )	0.3769	0.6629
Loss floor ( $c$ )	0.0208	0.0324
Best validation loss	0.0316	0.0363
Avg throughput (tok/s)	75,162	305,551
Total training time (s)	21882.0	2858.8
Peak GPU memory (GB)	17.11	3.00

## (H) Quantitative Evaluation of Generated Samples (XL Transformer Model)

Metric	Value
Total samples generated	20
Syntactically valid samples	13/20 (65.0%)
Successfully converted to MIDI	20/20 (100.0%)
Test loss	0.1883
Test perplexity	1.21

## (I) Unconditional Sample Header Prompt

```
X: 1
T: Generated Tune
M: 4/4
L: 1/8
K:C
```

## (J) Generated Samples (XL Transformer Model, ABC Notation)

To play the samples, please copy the content of each sample and paste it into the converter below:

<https://notabc.app/abc-converter/>

Sample	Unconditional
1	X: 1 T: Generated Tune M: 4/4 L: 1/8 K:C % 0 sharps V:1 %%clef treble %%MIDI program 34 %%MIDI program 0 z8  \ %%MIDI program 33 G,,z E,z G,,z  \ =E,,2 D,,2 =E,z  \ E,,2 D,,2 F,,2  \ =B,,,2 E,z G,,z  \ =E,2 D,,2 =E,z  \ =B,,,2 D,,2 F,,2  \ B,,,2 F,,2 A,,z  \ E,,2 D,,2 F,,2  \ D,,3D,, A,,z  \ G,,z F,z F,z  \ E,z G,z F,z  \ B,,,3-B,,,/2z/2 E,,2 F,,2  \ B,,,4 G,,2  \ z4 G,,z  \ =E,,2 C,,2 C,,2  \ z4 G,,3z  \ B,,,2 D,,2 F,,2  \ =B,,,3A,,, A,,,2  \ A,,,2 D,,2 F,,2  \ D,,2 F,,2 A,,,2  \ G,,2 F,,2 F,,2  \ B,,,2
2	X: 1 T: Generated Tune

	M: 4/4 L: 1/8 K:C K:G % 0 sharps V:1 %%clef treble %%MIDI program 48 z8  \ z8  \ z8  \ z8  \ z8  \ z8  \ z8  \ C,,2- [E,C,,]/2z/2z/2z/2 [GE]/2z3/2 [GE]/2z3/2  \ z8  \ C,,2- [G,C,,,-]/2C,,/2z/2C,/2 [GE]/2z3/2 [GE]/2z3/2  \ [FD]4 [FD]/2z3/2 [FD]/2z3/2  \ C,,2 [GE]/2z/2E/2z/2 [GE]/2z3/2 [GE]/2z3/2  \ z8  \ z8  \ C,,B,, A,,G,/2z/2 G,/2z/2G,/2z/2 G,/2z/2G,/2z/2  \ zG,/2z/2 G,/2z/2G,/2z/2 G,/2z/2G,/2z/2 G,/2z/2G,/2z/2  \ zG,/2z/2 G,/2z/2G,/2z/2 G,/2z/2G,/2z/2 G,/2z/2G,/2z/2  \ zG,/2z/2 G,/2z/2
3	X: 1 T: Generated Tune M: 4/4 L: 1/8 K:C % 0 sharps V:1 %%clef treble %%MIDI program 0 %%MIDI program 39 %%MIDI program 64 [A,-G,,,-][E-C-A,G,,,-]/2[E-C-G,,,-]/2 [G-E-C-G,,,-][G-ECG,,,-]/2[G-G,,,-]/2  \ [GEKG,,,-][G-E-C-G,,,-]/2[G-ECG,,]/2 [G-E-CE,,,-][G-E-C-E,,,-][G-ECE,,,-]/2[G-E-C-E,,,-]/2 [GCB,-E,,,-][E-C-B,E,,,-]/2[E-C-B,E,,,-]/2 [ECE,,]/2z/2[A,-F,-]/2[E-C-A,F,,,-]/2 [E-C-A,F,,][ECA,,]/2z/2 [G-E-C-E,E,,,-][GECE,,]/2z/2 [G-E-C-F,,,-][G-E-C-F,,,-]  \ [G-E-CF,,]/2[G-E-C-F,,,-]/3/2 [GECF,,]/2z/2[A,-F,-] [E-C-F,
	<b>Conditional</b>

4	<p>X: 1  T: Reel  M: 4/4  L: 1/8  K:G  V:1</p> <pre> [: GABC   \ zd Ac cB dA  \ cA cA BA AB  \ AB AA GA cA  cA cA cA AB  \ cA cA GA cA  \ cA cA dA dA  \ cA cA dA dA  cA cA dA cA  \ cA cA cA Ac  \ dA cA cA AA  \ cA cA cA AA  cA cA cA Ac  \ cA cA cA Ac  \ cA cA cA Az  \ cA cA cA Ac  cA cA dA cA  \ cA cA cA Ac  \ cA cA cA Ac  \ cA cA cA Ac  cA cA cA Ac  \ cA cA cc cA  \ cA cA cA Ac  cA </pre>
5	<p>X: 1  T: Slow Air  M: 3/4</p>

```

L: 1/4
K:Cmaj
%%clef treble
%%MIDI program 30
[cC]3/2z/2 [cC]3/2z/2 [cC]3/2[cC]/2| \
[cC]3/2z/2 [cC]3/2z/2 [cC]3/2z/2 [cC]3/2[cC]/2| \
[cC]3/2[cC]/2 [cC]3/2z/2 [cC]3/2z/2 [cC]3/2[dD]/2| \
[AA,]3/2z/2 [AA,]3/2z/2 [AA,]3/2[AA,]/2 [AA,]3/2z/2|
[cC]3/2z/2 [cC]3/2z/2 [cC]3/2[cC]/2| \
[cC]3/2[cC]/2 [cC]3/2z/2 [cC]3/2[cC]/2 [cC]3/2[cC]/2| \
[cC]3/2[cC]/2 [cC]3/2[cC]/2 [cC]3/2[cC]/2 [cC]3/2[cC]/2| \
[gg]3/2[gg]/2 [gg]3/2[gg]/2 [gg]3/2[gg]/2 [gg]3/2[gg]/2|
[gg]3/2[gg]/2

```

## 8. References

- Benini, D. (2025). *ABC notation, the basics - Nota ABC*. Notabc.app. <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 [Cs, Stat]*. <https://arxiv.org/abs/2001.08361>
- Karpathy, A. (2022). *GitHub - karpathy/nanoGPT: The simplest, fastest repository for training/finetuning medium-sized GPTs*. GitHub. <https://github.com/karpathy/nanoGPT/tree/master>
- Raffel, C. (n.d.). *The Lakh MIDI Dataset v0.1*. Colinraffel.com. <https://colinraffel.com/projects/lmd/>