New York University Tandon School of Engineering
Computer Science and Engineering

CS-GY 6923: Optional Project.
Due Monday, Dec. 15th, 2025, 11:59pm.

Discussion with other students and AI models is allowed for this project, but solutions must be written-up individually.

Overview

In this project, you will explore scaling laws for language models trained on symbolic music data. Unlike natural language, music notation provides a structured, non-linguistic domain with interesting hierarchical patterns, making it ideal for studying how model capacity affects learning.

Deliverables:

- PDF report (5 –10 pages recommended)
- Code repository with all scripts and a README

Grading Policy: This is an optional extra credit project. Partial credit is available for incomplete but well-executed work. You will not be penalized for not doing this project: we will assign initial grades based on the labs, homeworks, midterms and finals, ignoring the project; we will only update the grades for the students who submit the project, and it can only improve your grade. Only submit your report if you actually do substantial work on the project, we will not give you any points for a half-baked report you produce in a couple hours. Presenting fake results in your report (i.e. just making up experimental results) is equivalent to cheating and will have consequences.

Project Goals

By the end of this project, you should:

1. Build a complete data preprocessing pipeline for symbolic music
2. Empirically derive scaling laws for transformer-based language models
3. Compare transformer vs. RNN scaling behavior on the same task
4. Analyze what musical structures emerge at different model scales
5. Generate and evaluate music samples from your best model

Part 1: Data Collection and Preprocessing (15%)

Recommended Dataset: Lakh MIDI → ABC Notation

Primary Resource: Lakh MIDI Dataset (`https://colinraffel.com/projects/lmd/`)

- 176,581 unique MIDI files covering diverse genres
- Free and openly available under permissive licensing
- Well-documented and widely used in music information retrieval research

Conversion to ABC Notation:

- Use Python's `music21` library or the `midi2abc` command-line tool
- ABC notation is human-readable text (e.g., `C D E F | G4 |` for a simple melody)
- Total corpus should easily exceed 1 billion tokens

Alternative Datasets (if you want to explore):

- **DadaGP**: 26K guitar tablature files with tokenization (https://github.com/dada-bots/dadaGP)
- **The Session**: ∼40K high-quality ABC folk tunes (http://abcnotation.com)
- Any large MIDI collection converted to a text-based music format

Your Tasks

1. **Download and convert data** to a text-based music notation format
2. **Design a tokenization scheme**
   - Options: character-level, note-level, or custom music-aware tokens
   - Justify your choice in the report
   - Document vocabulary size
3. **Create train/validation/test splits**
   - Recommended: 98% / 1% / 1% or similar
   - Ensure at least 100M tokens in training set for each model
4. **Data cleaning**
   - Handle invalid/corrupted files
   - Decide how to handle very short or very long sequences
   - Document any filtering criteria

Deliverable: Clear description of your pipeline with statistics including vocabulary size, total tokens, sequence length distribution, conversion success rate, and any quality filters applied.

Part 2: Transformer Scaling Study (40%)

Train a family of decoder-only transformer language models of varying sizes. Measure the validation loss after **1 epoch** of training for each model size.

Model Sizes to Test

Train at least **5 different model sizes**. Suggested configurations (you can adjust based on computational resources).:

- **Tiny**: ∼1M parameters
- **Small**: ∼5M parameters
- **Medium**: ∼20M parameters
- **Large**: ∼50M parameters
- **XL**: ∼100M+ parameters (or largest you can realistically train)

You should adjust the actual model sizes you use based on the amount of computational resources that you have access to.

Vary hyperparameters like: `n_layers`, `n_heads`, `d_model`, `d_ff` to achieve different parameter counts.

Requirements

1. **Use consistent training setup** across all models:
   - Same tokenization scheme
   - Same learning rate schedule
   - Same batch size (measured in tokens)
   - Same training data (at least 100M tokens)
   - Train for exactly 1 epoch for the scaling plot comparison
2. **Create a scaling plot:**
   - X-axis: Model size (number of parameters, log scale)
   - Y-axis: Validation loss after 1 epoch
   - Fit a power law of the form: $L = a \cdot N^{-\alpha} + c$ where N is the parameter count
   - Report the fitted scaling exponent α and discuss its implications
3. **Track additional metrics:**
   - Training loss curves over time for each model
   - Wall-clock time per epoch
   - GPU memory usage

Reference Code

You may use **nanoGPT** (https://github.com/karpathy/nanoGPT) as a starting point and adapt it to your needs. Clearly document in your report what you borrowed versus what you modified or implemented yourself.

Deliverable: Scaling plot with power law fit, training curves for all models, table of model architectures and training statistics, and analysis of how loss decreases with model size.

Part 3: RNN Scaling Study and Comparison (20%)

Repeat the scaling study using RNN-based models (use LSTM).

Requirements

1. Train at least **4 different RNN sizes** with similar parameter counts to your transformer models
2. Use the same data and training setup (1 epoch, same 100M tokens, same tokenization)
3. Create an RNN scaling plot (validation loss vs. parameters)
4. **Compare transformer vs. RNN:**
   - Plot both scaling curves on the same graph
   - Which architecture scales better (steeper/better exponent)?
   - Analyze computational efficiency (training time and memory per parameter)
   - Discuss why you observe the differences you see
   - Consider both sample efficiency and computational efficiency

Deliverable: RNN scaling plot, combined comparison plot with both architectures, and detailed comparative analysis discussing the scaling behavior differences.

Part 4: Best Model Training and Sample Generation (15%)

Train your best model as well as you can within the project timeline:

1. **Choose your best architecture** (likely your largest feasible transformer)
2. **Train for as many tokens as you can**
3. **Tune hyperparameters** if time permits (learning rate, dropout, weight decay, etc.)
4. **Generate samples** from your trained model:
   - Generate at least 10 diverse samples
   - Convert generated ABC/tablature back to MIDI or use online ABC players
   - Include both unconditional generation and conditional generation (e.g., prompted with a prefix)

Sample Evaluation

Provide both quantitative and qualitative analysis of generated samples:

Quantitative Metrics:

- Final perplexity on test set
- Percentage of syntactically valid outputs
- For ABC: percentage that successfully convert to MIDI

Qualitative Analysis:

- Do samples sound musically coherent?
- Do they respect musical structure (rhythm, key, harmony)?
- What musical patterns did the model learn?
- Include specific examples and discussion

Playable Outputs:

- Convert generated ABC to MIDI using `music21`
- Provide audio files or links to online ABC players (e.g., `https://abcjs.net`)
- Include at least 5 example outputs in your report appendix

Deliverable: Sample outputs with detailed analysis, audio files or playback links, discussion of emergent musical patterns, and reflection on what worked well vs. what didn't.

Part 5: Design Decisions and Analysis (10%)

Throughout your report, clearly document and justify all major decisions:

1. **Tokenization strategy** – Why did you choose your approach? What alternatives did you consider?
2. **Architecture choices** – Model sizes, layer configurations, context window length, attention mechanisms
3. **Training decisions** – Batch size, learning rate schedule, optimizer choice, sequence length, regularization

4. **Scaling insights** – What did you learn about compute-optimal training in this domain? How do your results compare to known scaling laws for language models?
5. **Music-specific patterns** – What musical concepts (if any) did models learn at different scales? Did you observe phase transitions where certain capabilities emerged?
6. **Challenges encountered** – What didn't work? What would you do differently with more time or resources?

Deliverables

Code Repository

Your code submission should include:

- All preprocessing scripts (data download, conversion, tokenization)
- Training code (can be based on nanoGPT or other reference implementations)
- Evaluation and sample generation scripts
- Configuration files for all model architectures tested
- `README.md` with clear setup and usage instructions
- `requirements.txt` or equivalent with all dependencies and versions

PDF Report

Your report should be 6–10 pages (excluding references and appendices) and include:

Introduction (0.5–1 page)

- Brief motivation for studying scaling laws in the music domain
- Overview of your approach and contributions

Data (1–1.5 pages)

- Description of dataset and why you chose it
- Preprocessing pipeline and design decisions
- Tokenization scheme with examples
- Dataset statistics and visualizations

Methods (1.5–2 pages)

- Model architectures (table of configurations)
- Training setup and hyperparameters
- Experimental design for scaling studies
- Evaluation metrics

Results (2–3 pages)

- Transformer scaling plot with power law fit
- RNN scaling plot and comparison

- Training curves and computational costs
- Generated samples with evaluation
- Tables and figures should be clear and well-captioned

Discussion (1–2 pages)

- Key insights from scaling experiments
- Interpretation of results in context of music modeling
- Design decisions and their impact
- Limitations and future work

Conclusion (0.5 page)

- Summary of main findings
- What you learned about scaling laws in this domain

Appendices (optional, not counted toward page limit)

- Additional generated music samples
- Extended experimental details
- Code snippets for key components

Additional Resources

Code and Libraries:

- nanoGPT: https://github.com/karpathy/nanoGPT
- music21 library: https://web.mit.edu/music21/
- PyTorch: https://pytorch.org

Data:

- Lakh MIDI Dataset: https://colinraffel.com/projects/lmd/
- ABC notation resources: https://abcnotation.com
- ABC notation standard: https://abcnotation.com/wiki/abc:standard

Background Reading:

- Kaplan et al. (2020): "Scaling Laws for Neural Language Models" https://arxiv.org/abs/2001.08361
- Gwern's GPT-2 Folk Music experiments: https://gwern.net/gpt-2-music