Chord Llama

William Parker
Computer Engineering Department
San Jose State University
San Jose, USA
william.j.parker@sjsu.edu

Martin Alvarez Lopez
Computer Engineering Department
San Jose State University
San Jose, USA
martin.alvarezlopez@sjsu.edu

Motivation

Music generation is a well known field in the realm of machine learning. Despite this, until recently there have not been many notable advancements in the field relative to other areas in AI in the past few years. For this project, we wanted to find if we could apply the principles of fine tuning a language model on a new language for the task of music generation. Music has many similarities to language. It can be described in text though sheet music, while also being creative like how humans speak and write. This project focuses on fine-tuning Llama 3 [1], a large language model on sheet music, for the model to continue any given song.

Music continuation has several use cases for different groups. We see three key audiences that our model can be used in. Firstly, songwriters that are stuck midway through a song and don't know how to continue can use the model for inspiration and build off the model to create their musical piece. Listeners that enjoyed a short clip of music can arbitrarily extend the song with minimal knowledge of music theory while keeping the consistency of the original piece. Finally, fans of songs that want a remix can use our model to transform the song to create an alternative version.

Data

Our model was trained on our own custom dataset for this task. As our task specifically is to generate sheet music, we needed to look for formats that are popular enough to be used outside of the research setting and data sources of those formats. To do this, MusicXML [2] was chosen as a base format for our dataset. This is the standard way sheet music is represented for professionals in the music industry. While MIDI [3] is a much more popular format in the wider population, it defines how the music should be played at a byte level, and was not deemed suitable for this task. MusicXML has a very straightforward XML structure, primarily dealing with a series of measure elements each defining notes or other elements to display on the page. Our unedited MusicXML dataset contains approximately eleven thousand songs from public databases of Wikifonia [4] and MScoreLib [5]. These databases largely consist of classical music without copyright concerns.

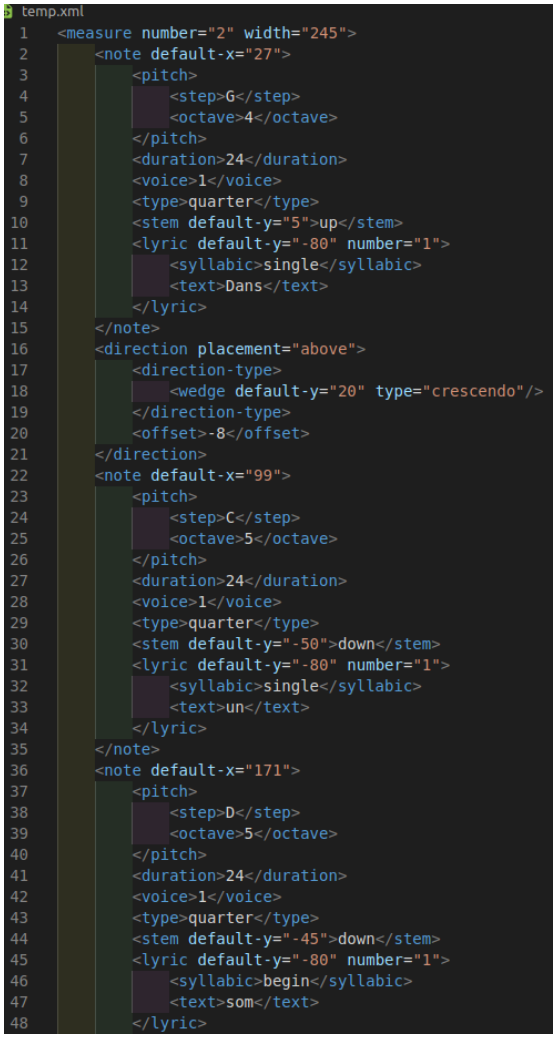
A large hurdle in our project was how to best prepare the data for the model. Many MusicXML components don't have any effect on the sound of the note, and deep research into the standard was necessary to understand what could be removed without impacting the meaning of the music. XML is also very expensive for tokenization in comparison to other formats [6], [7]. To make this project possible, we needed to alter the MusicXML format to fit within the context length of Llama 3.

A few large edits were made to the original data sources. First, all of the elements in the tree that did not have any meaning to how the song was played were removed. MusicXML has several unnecessary features for our use case. For example, it allows for notes to have custom placement on the line regardless of their actual value, corrections to the song have a special notation to state it is not part of the original song, notes can have custom colors, stem directions, and much more that can all be removed. Then, we removed all lyrics from the dataset and songs that changed their "attributes" tag midway through the song, as this was not in the scope of this project. All files were standardized to use the "score-partwise" version of the MusicXML format. Lastly, the format of the file was changed from XML to YAML as the format uses less tokens in the Llama 3 tokenizer. After these changes were made, we then needed to split the data into instruction, input and output. The instruction would be the single "attributes" tag in the song, input would be the first half of the song, and the output would be the second half. If the song was longer than 3,500 tokens, the song was split into multiple elements in the dataset to be below this limit.

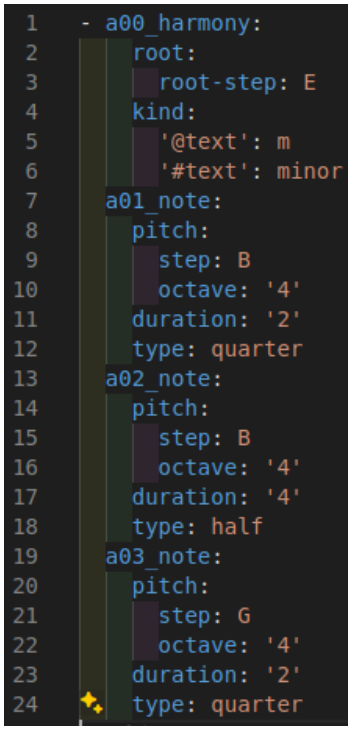
In Fig 1 you can see an example measure in MusicXML, and a measure in the YAML version we used for the model where we removed all unnecessary data. In MusicXML, each note may be over 100 tokens, while in YAML it is reduced down to about 30. By condensing the data, we can fit more notes into the context length. This allows us to input more notes into the model, increasing useability.

State of the Art Models

Much research has been done in the domain of music generation. Additionally, there are many different sub-domains. While recent research has been developing custom solutions for this problem, most of the open source projects are hard to use, as they are technical demos for a research audience. Little is known about how fine tuning a general language model can be used as a form of music generation, specifically continuing an existing song.



(a) Original MusicXML



(b) Our YAML Version

Fig. 1: Music Data Formats

MuseFormer was released by Microsoft in 2022 that applies the transformer architecture to music [8]. In their work, each concept like tempo, beat, pitch and duration are each a separate token. The attention mechanism has two stages the fine grained attention only attends to the elements in the same bar, while the coarse grained attention only attends to the summary of other bars. This allows it to be aware of the entire song without saving all of it in the context length.

We used Llama 3 which we fine tuned for our music generation task. We chose this as it is the state of the art in the field of large language models. The tokenization of the model was not touched. The model has three values, instruction, input and output. We set the instruction to the simplified version of the "attributes" tag. This contains data like the tempo, and key that is kept throughout the entire length of the song. The tokens in the instruction input are kept as important for the attention mechanism. Then, the existing portion of the song is put into the input, and the model then generates future tokens in the output.

To fine tune our Llama 3 model we used Unsloth [9], a state of the art open source tool for fine tuning LLMs. The Llama 3 family of models created by Meta has an 8 billion parameter version can be run on most consumer hardware, so apps made with the model can be used by anyone locally. Additionally, Llama 3 has a large ecosystem with a context length of 8 thousand tokens which is long enough for a large portion of a song.

RESEARCH

For our solution, we looked into 2 topics to utilize in our solution which are briefly explained in this section.

A. *Quantization*

Quantization converts the weights and activations into smaller data types, reducing the size and complexity of the model [10]. Most models are trained in 16 bit floating point numbers, and while that is useful when in initial training, it is unnecessary for fine tuning and inference. We reduced

the model to four bit quantization so that it would fit on our computer's GPU so we could train locally.

B. LoRA

Low-Rank Adaptation of Large Language Models (LoRA) [11] is an efficient way to fine tune a pretrained model. It achieves this by adding small, adjustable components to each layer of the model, leaving the original model unchanged. Instead of using a single delta matrix, LoRA employs two matrices to reduce the dimensionality. Because there are less calculations and memory needed, the fine-tuning process is much faster. Additionally, this method also typically produces higher quality results than normal fine-tuning. This allows us to train our model for anything with as little as 1k samples.

Architecture

Our LLama 3 model has 32 layers with the same amount of attention heads. When training the model, each step was composed of 8 samples in our data.

For LoRA [12], the attention dimension of the low-rank matrices was 8. This decomposes a large matrix into 2 smaller matrices in the attention layers. We tested the model with higher ranks but no significant benefits were seen. The alpha parameter for LoRA scaling was set to 16. This changes how the low-rank matrices are introduced into the model during the LoRA adaptation process. If the alpha parameter is set higher it means the LoRA layer acts stronger on the base model. Additionally, we targeted all linear layers for the fine-tuning process [13].

Training

Ultimately we trained the model with 1.6K samples. We selected this because originally we trained a model with 10K samples and noticed the model was not improving after 1k samples. For our model, we used Adaptive Moment Estimation with Decoupled Weight Decay (AdamW) with 8 bit precision.

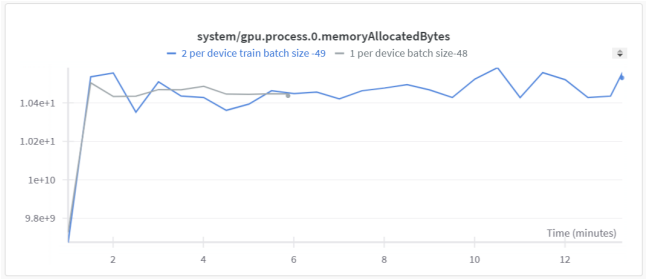
While experimenting with the learning rate scheduler, our model worked better with cosine when considering time and hardware resources. Fig 2 shows the significant difference between cosine, linear, and polynomial learning rate schedulers. Cosine performed equally as good as the other schedulers however it was able to complete 18% - 35% more samples per second. Additionally, we determined the optimal learning rate and weight decay is 2e-4 and .01 respectively.

Our training batch size per device was set to 1 for memory efficiency. As you can see in Fig 3, using a higher batch size, going from 1 to 2 doubled the training time with no significant changes in the models performance. Increasing the gradient accumulation steps resulted in smoother loss, but it also extended our training time by around 50% with each increment. Due to memory efficiency and hardware utilization reasons, we determined setting gradient accumulation steps to 3 was optimal for training because with our data the model reached the same loss quicker.

Lastly, due to the nature of our model's task and because we have a large amount of good data while using LoRA to fine tune a LLM, having an epoch of 1 was enough for our models purpose.

diff only	Plynomial-62	Linear-50	Cosine-48
META			
runtime	17m 4s	11m 34s	7m 59s
CONFIG (6 collapsed)			
SUMMARY			
_runtime	478.066	416.43	352.336
_timestamp	1715282918.368	1715222446.363	1715220913.779
_wandb (1 collapsed)			
total_flos	13573164507365376	14421995287068672	14421995287068672
train (4 collapsed)			
train_loss	0.1513	0.1521	0.1517
train_runtime	479.671	418.039	353.881
train_samples_per_second	0.25	0.287	0.339
train_steps_per_second	0.083	0.096	0.113

Fig. 2: Schedulers comparison.



(a)

(b)

Fig. 3: (a) System Utilization (b) Train Loss

Evaluation

We struggled thinking of an evaluation process for our model's generated music. This is a difficult problem to address because it's not clear what the merits of a good solution would look like. For example, initially we noticed our model tended to repeat music. However, music is repetitive by nature. Therefore, it's not clear if our model was working well or overfitting. We considered the output should be different than the input but sound good still. Furthermore, there is no metric that we can use to evaluate music especially for our models use case. Other research uses human preference for evaluation rather than numerical metrics [14].Ultimately, we decided to go the same route and use human preference for our evaluation.

Optimization

After fine tuning a model, it still must be converted to a format that is optimal for inference. Once fully fine tuned, it will be in a .safetensor format. This format can be used for further development by researchers, so we uploaded the model to HuggingFace, but the model needed further work to be used for inference locally. The optimized format for LLMs is GGUF [15], which is a standard .bin file that is optimized for quick loading and saving, and is standard for locally hosted inference applications. To convert to GGUF, we used llama.cpp, a standard tool for preforming optimizations in LLMs [16]. We continued to use the 4 bit quantization we used in training, as it lowers RAM use for users and has a faster inference time. We could then share the model on Ollama, which could then be ran on almost any computer at a speed at or faster than what is commonly acceptable for the field depending on the computer's specifications.

Application

To show off the results of the model, we created a simple web application. For this, we used Flask for the backend and React for the front end. The front end allows for users to upload a MusicXML file they want to extend, or the attribute and measure YAML files separately. When the "Run Inference" button is clicked, it will send the attribute and measure files to the backend for Chord Llama to be run using Ollama. The results are then streamed back to the output text block. The final application can be found on Github: https://github.com/Chord-Llama/Chord-Llama

Comparison Results

We fully trained 2 models to compare performance. One model was the full Llama 3 model and the other was a smaller 22 hidden layer version called TinyLLama. The purpose of this comparison was to view how a lighter model would perform with our intended task. TinyLlama was a lot more efficient at training. However, LLama 3 generated better results overall and used it for our Chord Llama project. A detailed comparison can be accessed at: https://api.wandb.ai/links/martin-chivo/24rsptsc

We tested Chord Llama on excerpts of MusicXML to manually determine the quality of music that the model could generate. The overtrained model would simply repeat the input with little to no modifications. Training on a limited number of samples led to better variety of music generation.

The output of Chord Llama can be reverted back into MusicXML by undoing the steps in data cleaning. We can then view the sheet music in a MusicXML reader like MuseScore 4. See Appendix A for the output in sheet music form.

The final project of Chord Llama has a file upload for a MusicXML file, or you can upload the attributes and measures separately. You can then press "Run Inference" to send the inputs to the backend. This will send the data to Ollama running Chord Llama, generating the output that is then sent back to the user. The model's response can then be converted back into a MusicXML file through the code block in the data cleaning notebook. See Appendix B for the exmample user interface.

Task Distribution

Martin

Model selection and Fine tuning: Worked on fine tuning several different versions of our Llama 3 model on a subset of our data and comparing and evaluating the performance differences. Then after identifying what is optimal, fully trained 2 models the full Lllama 3 model and a smaller 22 hidden layer version called TinyLLama for comparing.

This was all completed in a jupyter notebook within the file named "Selection_Training.ipynb". This file can be accessed at: https://huggingface.co/Chord-Llama/Llama-3-chord-llama-fullModel/blob/main/Selection_Training.ipynb. This work was connected to wandb for tracking and documenting each model's performance. Notably, the notebook is broken down into different sections or subsections, where each section is a configuration or version of the model that was used for training and comparing performances. The notebook was set up this way to make things modular and quickly run all cells within a specific section which helped with each iteration. All of this is followed by a section called "Full Model Training" with the optimal configuration used to train the full model. Then there is a section for saving the model locally and pushing it to Huggingface. Lastly, it is not part of the submission requirements but there are two final sections used for inferencing where we load a model or checkpoint and perform a test to see how the model is working. One of the sections uses cuda and the other section is to experiment with Hugginface's pipeline transformer to see a different method to set up the model to generate music text.

William

Data: Defined the data to be used for the model. Found the data, cleaned the data to remove all unnecessary components, and prepared it for fine-tuning.

Data Cleaning was done in a Python notebook in the "music_xml_converter.ipynb" file available on HuggingFace at: https://huggingface.co/datasets/Chord-Llama/chord_llama_dataset/blob/main/music_xml_converter.ipynb. This code implements the process in the data section. In this step, we go from downloading the data, cleaning it, converting it to YAML, and converting the data into JSON Lines format so it can be uploaded to HuggingFace as a dataset to be used for fine-tuning.

Optimization and Application: Once the model was published to HuggingFace, it needed to be optimized for inference, and an application needed to be made to use it.

Optimization was done by converting from a .safetensor file to GGUF using llama.cpp. Once done, it was transformed into an Ollama model to be used by applications. A simple web application was then made using Flask and React. The app has two main functionallities. First, it allows a user to convert a MusicXML file into the YAML format identical to the format

used to train the model. Then, using that output, it can then send the data for inference to Chord Llama to predict new sheet music. The final project can be found on GitHub at: https://github.com/Chord-Llama/Chord-Llama

TABLE I: Task Distribution

Tasks	Description	William	Martin
Presentation	Project Presentation	X	X
Document	Project Document	X	X
Research	All Research	X	X
Code	Data	X	
	Model Selection		X
	Training/Fine Tuning		X
	Optimization	X	
	Application	X	

REFERENCES

[1] AI@Meta, "Llama 3 model card," 2024. [Online]. Available: https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

[2] Michael Good, *MusicXML*, version 4.0, Jun. 1, 2021. [Online]. Available: https://www.w3.org/2021/06/musicxml40/ (visited on 04/09/2024).

[3] Hubert Howe. "Music 726.1: Electronic music studio 1," MIDI: The Musical Instrument Digital Interface. (), [Online]. Available: https://qcpages.qc.cuny.edu/~howe/music726.1/MIDI.html.

[4] "Download for wikifonia all 6,675 lead sheets! — general arranger keyboard forum — synth zone forums." (), [Online]. Available: http://www.synthzone.com/forum/ubbthreads.php/topics/384909 (visited on 04/09/2024).

[5] "MScoreLib - music library in MusicXML." (), [Online]. Available: http://mscorelib.com/ (visited on 04/09/2024).

[6] E. Livshitz. "YAML vs. JSON: Which is more efficient for language models?" Medium. (Jul. 25, 2023), [Online]. Available: https://betterprogramming.pub/yaml-vs-json-which-is-more-efficient-for-language-models-5bc11dd0f6df (visited on 05/21/2024).

[7] Matt Rickard. "A token efficient language for LLMs," Matt Rickard. (), [Online]. Available: https://matt-rickard.com/a-token-efficient-language-for-llms (visited on 04/09/2024).

[8] B. Yu, P. Lu, R. Wang, *et al.*, *Museformer: Transformer with fine- and coarse-grained attention for music generation*, Oct. 30, 2022. arXiv: 2210.10349[cs,eess]. [Online]. Available: http://arxiv.org/abs/2210.10349 (visited on 05/21/2024).

[9] D. Han and Michael Han, *Unsloth*, version Llama-3 Support, Apr. 18, 2024. [Online]. Available: https://github.com/unslothai/unsloth.

[10] B. Jacob, S. Kligys, B. Chen, *et al.*, *Quantization and training of neural networks for efficient integer-arithmetic-only inference*, Dec. 15, 2017. arXiv: 1712.05877[cs,stat]. [Online]. Available: http://arxiv.org/abs/1712.05877 (visited on 05/21/2024).

[11] E. J. Hu, Y. Shen, P. Wallis, *et al.*, *LoRA: Low-rank adaptation of large language models*, Oct. 16, 2021. arXiv: 2106.09685[cs]. [Online]. Available: http://arxiv.org/abs/2106.09685 (visited on 05/21/2024).

[12] HuggingFace. "LoRA." (), [Online]. Available: https://huggingface.co/docs/peft/main/en/conceptual_guides/lora (visited on 05/21/2024).

[13] Meta and Unsloth, *Unsloth/llama-3-8b-bnb-4bit*, version 8b-bnb-4bit, Apr. 18, 2024. [Online]. Available: https://huggingface.co/unsloth/llama-3-8b-bnb-4bit.

[14] C. Hernandez-Olivan, J. Hernandez-Olivan, and J. R. Beltran, *A survey on artificial intelligence for music generation: Agents, domains and perspectives*, Nov. 3, 2022. arXiv: 2210.13944[cs,eess]. [Online]. Available: http://arxiv.org/abs/2210.13944 (visited on 05/21/2024).

[15] Georgi Gerganov, *Ggml*. [Online]. Available: https://github.com/ggerganov/ggml/tree/master.

[16] G. Gerganov, *llama.cpp*. ggml.ai, May 2024. [Online]. Available: https://github.com/ggerganov/llama.cpp.

APPENDIX A
SONG VISUALIZATION

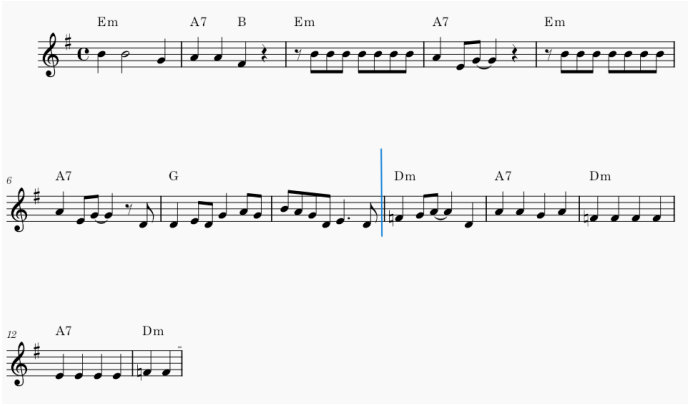


Fig. 4: Sheet music representation of the prompt plus output. Everything after the blue line is generated by model.

APPENDIX B
USER INTERFACE