Melodic Machines: A Dual-Model Approach to Artist-Conditioned Music Generation

Github Link

Who:
Qien Lin (qlin23)
Alzahra Fayie (afayie)
Man-Fang Liang (mliang23)
Zetao Wu (zwu119)

Introduction:

Music creation is both an art and a science: crafting lyrics that feel "authentic" to a particular artist and producing audio that captures their signature sound can require extensive domain expertise and manual effort. Recent advances in deep generative models—transformers for text and diffusion models for images—open the door to automating these creative tasks.

In this project, Melodic Machines, we explore a two-stage framework for artist-conditioned music generation. First, a transformer-based lyric generator takes as input an artist's name and genre and produces novel lyrics in their stylistic voice. Second, a stable-diffusion model operates in the spectrogram domain to synthesize new audio textures that mimic the artist's sonic palette. By coupling these modules, our system can output both the words and the sound of a "new" song in a target artist's style, streamlining creative workflows and offering insights into the interplay between lyrical content and audio style.

Related Work:

When we first came up with the idea, we were unaware of whether or not it had been implemented before. As a whole, we couldn't find if our dual-model approach with a transformer model lyric generator and a stable diffusion model spectrogram generator has been done before, but we have found some implementations of each model separately with some variations.

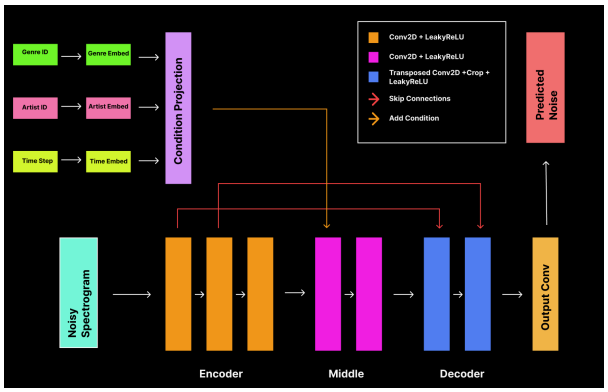
We found a website called Riffusion that claims to use spectrogram generation with stable diffusion, but we have no access to its source code. However, it takes in music descriptions/genres/etc. and produces full-fledged songs with clear lyrics instead of our model, which takes in an artist as input and produces an audio sample in their "style". Riffusion can be found here and a brief explanation of its architecture can be found here. Lyric generation using transformers doesn't seem like a completely novel idea either as we have found implementations of it online, but they use GPT2 pretrained models. A medium article showing lyric generation with GPT2 can be found here. A hugging face fine-tuning of GPT2 for lyric generation can be found here. Neither of these resources was used in the construction of our models, but they are worth mentioning as they serve similar purposes to our models.

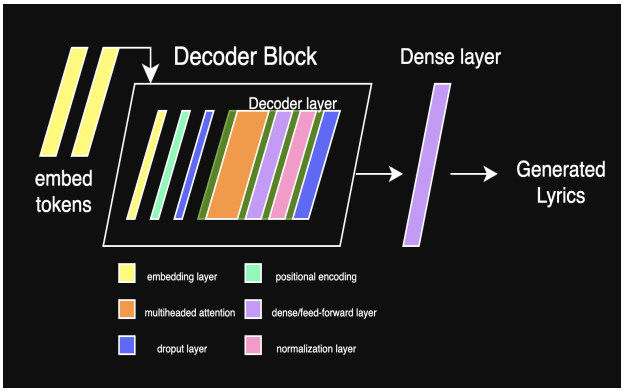
Methodology:

Since we did not re-implement a paper, we had to search for appropriate datasets ourselves for both generation tasks. However, since a dataset with the exact data we needed to train our models on didn't exist, we had to do significant preprocessing to have our data ready for training. Our initial plan was to train both our models on the common artists between the audio dataset FreeMusicArchive(FMA) Dataset and MillionSongDataset (MSD) lyric dataset. We ran into issues during preprocessing the data for that as the artist intersection turned out to be very small (about 900 artists), and we also ran into some incompatibility issues with the MSD dataset (further detailed under the challenges section).

Instead, our final datasets were the FMA dataset, along with a Kaggle lyric dataset of song lyrics of the top 20 artists. To train our stable diffusion model, we used the FMA dataset which provided .mp3 audio files along with their associated metadata (i.e. artist name, genre, release date, etc.). Since this data format is incompatible with the purpose of our model, we had to extract the audio files along with their respective metadata, which we later used as our conditional embeddings (i.e. artist name/id, genre) and converted the .mp3 files into spectrogram images using the librosa Python library. Our initial plan was to train our model on a csv file with each artist name-genre pair associated with a path to its respective spectrogram; however, we decided to then turn them into pickle files with artist name-genre pairs that are instead associated with matrix representations of the spectrogram images. The switch from training on images to matrix representations significantly decreased training times for our stable diffusion model.

As for the transformer model's data, once we decided against web-scraping and the MSD dataset, we had to do a mix of manual and automated data refining. Most of the data refining wasn't in changing the dataset's format, but was rather focused on identifying invalid entries, removing invisible characters, and splitting the data into training and testing files. The csv files from Kaggle originally had every artist and their song lyrics in a separate csv file, which contained metadata that we removed as we prepared our dataset. After removing the extra metadata, we merged the artist-specific csv files into a single file, shrunk the file into a manageable size, and then manually went over the entries to ensure they were valid. While inefficient, the reason behind this manual refining is due to the fact that this dataset was manually inputted, so entries of unreleased songs were written in so many different ways by the creator of the dataset. For example, an unreleased song could have "lyrics unreleased come back later", "snippet [insert snippet here] i don't know the rest", or just empty song lyrics as entries. This made automating this task quite difficult, so we shrunk the dataset of about 6000 entries into 400 and refined it manually.





As for our model architectures, the figures above (both created by us using draw.io and Figma, respectively) show a brief glimpse into our models' architecture. The first of which is the architecture for our stable diffusion model which has genre, artist ID, and time as its conditional embedding. The architecture we followed is considered a very simplified version of stable diffusion as we were limited by computational resources. The second figure shows the architecture of our decoder-only transformer which has genre and artist id as its embed tokens.

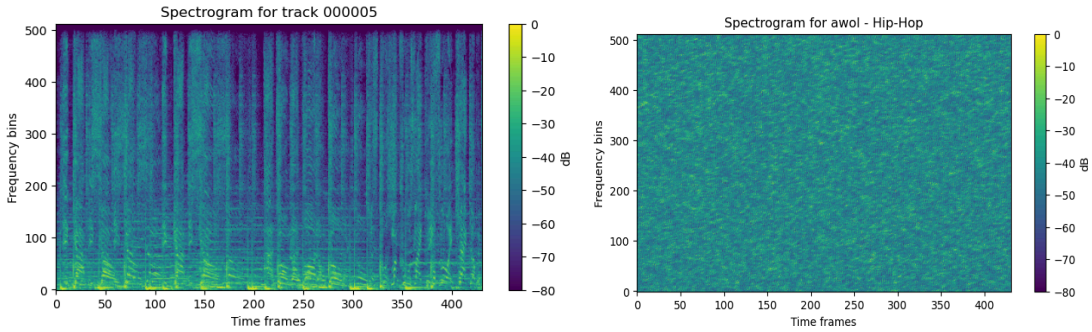
The Stable Diffusion model synthesizes audio spectrograms conditioned on artist, genre, and diffusion time step using a U-Net architecture. To reduce computational demands, we simplified the model to consist of three main blocks: encoder, decoder, and middle layer, each built from Conv2D and transported Conv2D with LeakyReLU as activation functions. The input is a noisy spectrogram, which is progressively denoised over inference steps. Time, artist, and genre embeddings are projected through a shared dense layer and injected into the U-Net via additive conditioning. This is to inform the model of the desired target style and help guide the denoising toward that style domain. Here is a simplified process of how our stable diffusion operates in a U-Net structure. A noisy spectrogram is passed to the first encoder layer, where it is progressively downsampled to extract multi-scale hierarchical features. These features are then processed by a middle block that captures global context and begins the denoising process. This is where the conditioning projections come in, which are derived from the artist, genre, and timestep. The conditional projection is injected into the middle layer to inform the model of the projected target style and guide it towards denoising. Finally, the decoder samples the latent representations back to the original spectrogram resolution. One important step is the skip connections, which bridge corresponding encoder and decoder layers, allowing the decoder to recover spectral details that might have been lost during downsampling. The final output is a prediction of the residual noise, which is a progressively denoised spectrogram that mimics the desired artistic and genre-specific style.

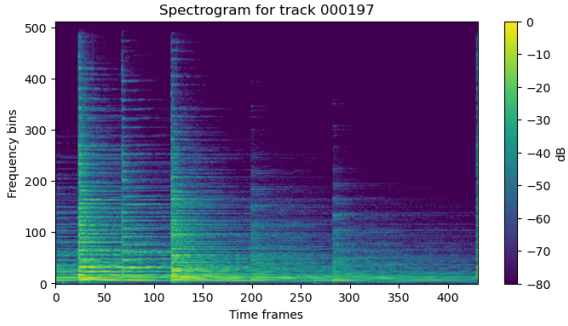
For the transformer, it adopts a decoder-only architecture that is tailored specifically for conditional text generation. It starts with embedding input lyric tokens into high-dimensional vectors, followed by positional encoding to retain word order information. The embedded sequence is then passed to a stack of decoder layers, each composed of masked multi-head self-attention, feed-forward network, dropout, and residual connections with layer normalization to stabilize training. Furthermore, the model incorporates artist and genre identity through

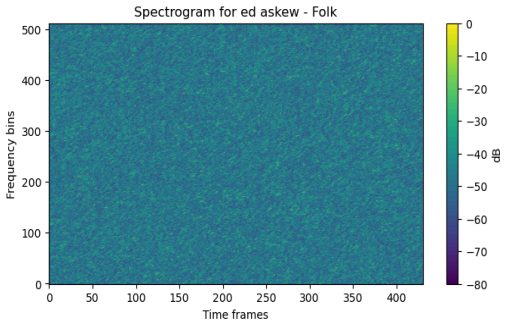
separate embedding layers whose outputs are combined and added to the decoder output for each timestep. These conditions influence the lyric generation on both musical style and the artist identity. The final layer is a dense projection that maps hidden states to vocabulary logits. This produces artist and genre-specific lyric predictions in an autoregressive manner.

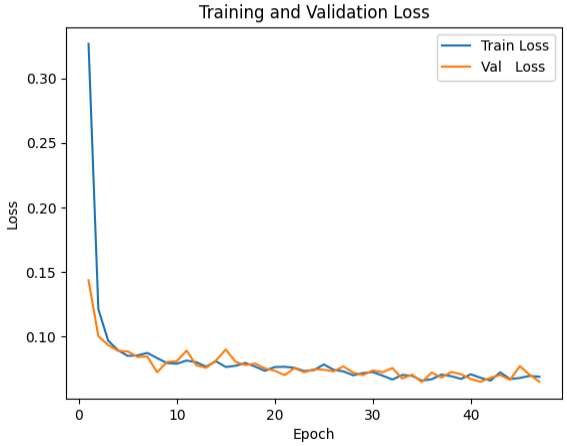
Results:

The figure below presents original spectrograms on the left and our model's generations on the right, each conditioned on the same artist and genre. In the top row, the original spectrogram appears lighter overall, and the generated output exhibits a similar light hue. In the bottom row, the original is predominantly dark, and the generated version is likewise uniformly deep blue. This demonstrates that, despite residual noise, the model successfully captures the overall brightness level of each artist–genre example. Moreover, the generated spectrograms do not exhibit any extremely high or low intensity values, remaining within a moderate intensity range without abrupt spikes. Additionally, since the audios were clipped to 10 seconds, there is not a lot of tonal variation, which could have helped us achieve more noticeable decibel distribution. Despite the noisy generative results, our model displayed quite promising training metrics (i.e. loss). Over 40 training epochs, the loss has reached less than 0.01, which is a result we're quite happy with. If we had the computational power and time, we would've increased the complexity and depth of our stable diffusion model and hoped for similar metrics as well.









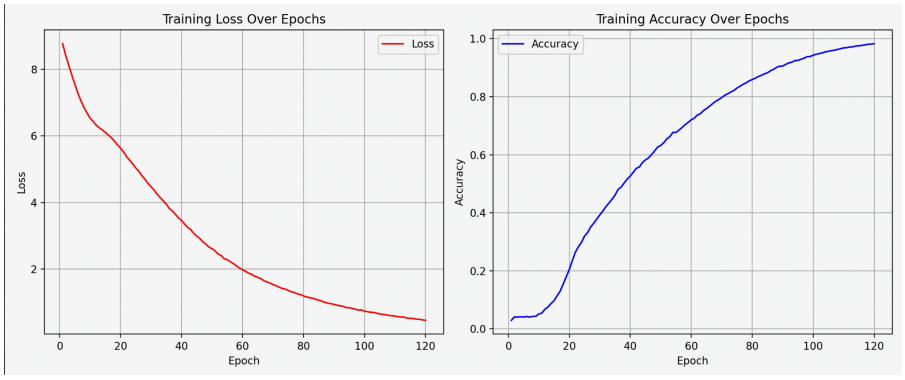
As for our transformer's results, the figure below showcases a sample output from our Transformer-based lyric generation model, which accepts artist and genre inputs to condition its predictions. In this example, the user inputs "Dua Lipa" and "pop", prompting the model to generate a stream of lyric-like tokens. The structure is able to remain syntactically plausible with appropriate spacing and line formatting. However, the outputs do reveal both pros and cons of the model. For one, the model was able to showcase and maintain a creative and diverse vocabulary, generating a mixture of noun phrases, verbs, and artist-relevant keywords. On the other hand, semantic coherence starts to degrade over time, with increasing randomness and disconnected phrasing. This reflects the challenges of stylistic consistency in text generation, which was mainly due to the limited access to lyric datasets. Nevertheless, the model succeeds in producing lexically varied, high-entropy outputs that reflect the punchy tone often found in pop lyrics. In addition, the training metrics (i.e. loss and accuracy) were quite significant. We just

hope we have a larger dataset that would help our model learn better, producing more coherent generative results.

```
Enter artist name (e.g., 'dua lipa'): dua lipa
Enter genre (e.g., 'pop'): pop
Enter optional starting text (or press Enter to skip):
Enter temperature (0.5–1.5, higher = more random): 1

Generating lyrics in the style of dua lipa (pop)...

==================================================
ipod galaxy daybed qualities static lalala rob figures size
lead center crystal lionel accusations public subway ever six
bit control prize pick choose advance twodoor tickin' seasoning
'96 waitress actin' miss well settling barefoot high incline
package belonged badder stripper saturn toptop slacks fools again
hardest travelled here's brrt hear world dodged poolside scratch
bake kirkpatrick bored invitations flippity moment ooooh stereo plant
must've mormons touring daylight lack 's heheadshot change ironin'
guts jingle necessarily toma nowi'm twirl comments distorted pick
these rapper follie's not've nyu shits pigeons kinda split
satisfactory rosaleem nonlyrical gettin' my shock kush spot origami
xo massoccur woah minime filthy lies ohohohohohohoh collins turnin'
economic mercy hahahaha masterpièce curl outlaw dogg headtop swallow
revolutionaries sleazy lakim detractors marvin leaving genes hattie cure
halleluhallehallelujah jobs christ wanted heyhey act here himeros bullets
room's monsters twitter accounts cannot youknowwho elevators donjae wet
sign powder applause spoken fixed buy
==================================================
```



Challenges:

We faced several interconnected challenges throughout the project. First, MSD's lyrics were only available in a bag-of-words format, which lacked any sequential structure, so we had to pivot to a much smaller Kaggle lyrics dataset. The Kaggle dataset only featured Top 20 artists (i.e. Dua Lipa, Ariana Grande, etc.) and was manually inputted by a Kaggle user, so our models were then trained on two completely different sets of artists. Given their manual input, the dataset also contained many human errors (i.e. typos, incorrect artist-song lyric pairs, incomplete pairs, etc.), so we had to shrink its size considerably to only include verifiably ground-truth artist-song lyric pairs for training.

While the models worked as expected, we no longer had a seamless artist-conditioned framework anymore. In other words, while our stable diffusion model generated results for independent artists such as Awol, our transformer model could only produce generated lyrics for famous artists like Taylor Swift. At the same time, FMA does not provide a public API, forcing us to rely on the pre-packaged fma_small subset. This limited both the number of tracks per artist and the stylistic diversity we could train on.

On the modeling side, our GPU memory constraints dictated a very simple U-Net diffusion architecture with small hidden dimensions, batch sizes, and epochs. While this setup captured coarse brightness levels in the spectrograms, it could not fully denoise or learn fine spectral details. Finally, using magnitude-only spectrograms discarded phase information, resulting in some audible distortion upon waveform reconstruction—an issue we could not fully address given our computational and time limitations.

Reflection & Analysis:

Our team spent a significant amount of time on data collection and preprocessing. Initially, we attempted to link the MSD (lyrics) and FMA (audio) datasets by matching on artist names to obtain paired lyrics and audio. However, the MSD lyrics were provided in a bag-of-words format without any sequential information, so we pivoted to using a Kaggle dataset of famous artists—though it proved quite small. For audio, since FMA does not offer a public API key, we downloaded the pre-packaged fma_small dataset from GitHub, which nonetheless sufficed for our project's scale. We also found that conditioning purely on artist name was problematic—FMA's artists tend to be independent musicians with few tracks—so we added genre as a second embedding. Due to computational constraints, we built a very simple U-Net diffusion architecture with small hidden dimensions, batch sizes, and a limited number of epochs. While we saw promising learning patterns, the model was unable to fully denoise the spectrograms, and the FMA split emphasized breadth over depth in artist coverage, further limiting training quality.

If we had more time, we would begin data collection and preprocessing much earlier and on a larger scale: crawling a structured lyrics corpus (e.g., via web scraping) and moving from 10-second clips to 30- or 60-second segments—ideally using the FMA median or large splits. Since spectrograms discard phase information (leading to some distortion on reconstruction), we would explore conditioning the model on phase or jointly predicting magnitude and phase. On the model side, we'd increase capacity by deepening the U-Net (adding extra down- and up-sampling blocks), widening it (higher channel counts), and incorporating self- and cross-attention layers for stronger artist–genre conditioning. We'd also consider a latent-diffusion framework for faster, higher-resolution generation. A systematic hyperparameter sweep—larger batch sizes, more diffusion time steps, higher hidden-dimension widths, and longer training schedules—would follow, using tools like Optuna or Ray Tune. Finally, we'd benchmark alternative generative paradigms (flow matching, score-based models) and integrate a neural vocoder (WaveGlow, HiFi-GAN) to convert spectrograms back into high-fidelity audio. These steps would help us capture richer musical patterns and improve overall output quality.

If we had more time to improve the transformer-based lyric generation model, we would probably re-architect the conditioning mechanism. Rather than simply adding artist and genre embeddings to the decoder output, it would be better to dedicate a conditioning encoder or cross-attention layer that projects metadata into contextual space. Also, we might also consider transitioning from a decoder-only Transformer to a full encoder-decoder architecture, which would enable the model to better capture long-range dependencies between artist/genre style and lyric structure. To improve the output coherence, our sampling method could be improved to

a top-k sampling during the inference phase. Furthermore, validation loss tracking with early stopping during training could be introduced to prevent overfitting. Finally, the generated lyrics could be evaluated with specific metrics that might be more fitting for NLPs. We currently have a raw token accuracy, which might not fluently and stylistically reflect the consistency of the generated lyrics. These changes would enhance the quality and controllability of the generated lyrics from our Transformer model.

Despite all the previous challenges mentioned above, we are happy to say that we have managed to reach both our base and target goals for the project. Given that we had to build two models, significantly preprocess data for their training, and set aside time and GPU resources for training, our base goal was only completing our stable diffusion model, while our target goal was to complete both. Fortunately, we have completed both with very promising preliminary result,s even if they don't stylistically make a lot of sense at the moment. We are glad to have observed a notable learning pattern and great training metrics in both models. As detailed above, we would have been able to reach our stretch goals if we had more time and access to GPU resources, which were mostly about fine-tuning and beefing up our model to achieve more coherent generated predictions.

One of the biggest takeaways from this project was learning how to design and integrate two distinct deep learning architectures into an end-to-end generative pipeline for music content creation. This project's biggest challenge is multimodal generation, such as balancing stylistic fidelity with creativity, handling sparse and noisy datasets, and evaluating outputs that are inherently subjective. At first, we found that finding appropriate datasets was harder than anticipated. Due to limited artist overlap and unsuitable data formats, we need to web scrape and find alternative datasets after evaluating them. It also deepens our understanding of sequence modeling, progressive noise scheduling, and embedding design, while also emphasizing the importance of robust preprocessing and ethical considerations in a generative AI system. The process, from data mining to architecture design, training, and producing consistent outputs, was both challenging and rewarding. It forced us to troubleshoot unexpected behaviors, iterate rapidly, and apply theoretical concepts in a practical manner, ultimately reinforcing our confidence in building large-scale, multi-component AI systems from the ground up. At the same time, it also made us confront important questions about ethical questions such as creative ownership and proper attribution. We can now better appreciate the stakeholder implications when developing AI systems.