Automatic music generation with mood attenuation

Rodrigo Moreno,[1] **Sergio Puertas Pérez**,[2] and **Pierre Iamurri**[3]

[1] *lrmoreno@student.uliege.be (s2304505)*
[2] *SPuertas@student.uliege.be (s2303863)*
[3] *pierre.iamurrig@student.uliege.be (s2309324)*

I. ABSTRACT

This project explores the generation of music samples aligned with specific mood descriptors using a diffusion model. Motivated by the need for adaptive music generation in role-playing games, the study aims to provide a tool for enhancing narrative immersion. Using a dataset of medieval-themed audio segments categorized by mood, the model generates Mel spectrograms to represent music samples. Through a UNet architecture with time and genre embeddings, the model learns to denoise and generate plausible music spectrograms. While denoising performance is promising, the model's ability to capture nuanced musical features remains a challenge. Further directions include expanding the dataset and fine-tuning hyperparameters to enhance music generation capabilities.

II. INTRODUCTION

A. Music as a narrative tool

For centuries, music has served as a narrative device accompanying storytelling in various contexts such as religious ceremonies, theater, opera, television, and cinema. It enhances emotional engagement and can drastically alter the interpretation of narrative elements. The right type of music can transform a scene of a child playing in the park from a happy memory using a merry tune, into a terrifying moment using dissonance or uncommon (for the cultural context) keys, or even to a nonsensical scenario using genres like ska, metal, or disco.

The use of music is relatively straightforward for media such as film or TV. As everything is pre-scripted and filmed, a musical motif can be added in post-production with a known duration and exact information about the events that follow. Media such as opera, theater, or religious ceremonies also benefit from this advantage, as those coordinating the music are separate from the performers involved in storytelling, whether through a full orchestra, a small ensemble of musicians, or a person playing pre-recorded music at specific cues.

While potentially expensive and time-consuming, the use of music is relatively simple in these contexts. However, this is not the case in all settings. Take, for instance, the large community engaged in role-playing games (RPGs). In these games, a person called the Game Master (GM) is responsible for narrating the story to a group of players who then take action to advance the narrative. In this context, music is highly valued as a storytelling device; entire online communities are dedicated to sharing background music for different settings, with hours upon hours of music files. However, this requires the GM to pause the narration to find and play the right track, thus breaking the story's immersion.

Our goal with this project was to build a deep-learning model to generate music samples that fit specific mood descriptors useful for RPGs. We aimed to provide a tool to help the storyteller seamlessly evolve the music's ambiance along with the story, allowing for greater improvisation and immersion for the listeners.

B. Mel spectrograms

Spectrograms serve as a visual representation of the change of sound through time. This is done by decomposing the wave using Fourier series, and plotting the weights of each frequency at each time window as a heatmap. However, when hearing a sound, humans do not perceive changes in pitch linearly. At higher pitches, the human ear perceives frequencies as being closer together than at lower ones, even if the frequency difference is the same. Thus, a change in the spectrogram needs to be done to more clearly reflect what is being perceived.

In 1937, Stevens *et al.* [1] introduced the Mel scale, a scale that is linear in terms of the perception of pitch change. The conversion from Hz to Mel is given by the following formula, and shown in fig. 1:

$$\mathrm{m} = 1127 \cdot \ln\left(1 + \frac{f}{700}\right)$$

In comparison with a regular spectrogram, a Mel spectrogram uses the decibel scale for amplitude and the Mel scale for frequency, providing a more realistic representation of human sound perception. The spectrogram is displayed as a graph with time on the x-axis, Mel frequencies on the y-axis, and amplitude in dB represented using a color scale. To generate a spectrogram, we apply a Short-Term Fourier Transform (STFT) to short samples of the audio signal to compute each pixel column of the output image.

This creates an image to which we can easily add noise to by modifying the value on the color scale of each pixel, allowing us to process the sound through a diffusion model. In order to generate sound, we can use the image of the spectrogram instead of the sound itself.

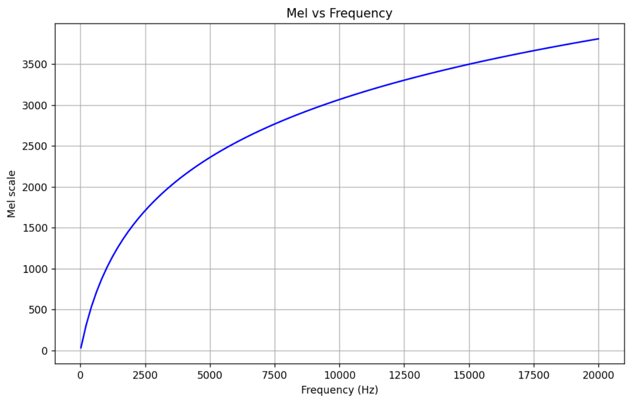


FIG. 1: Mel scale vs Hertz scale (plot by authors)

This image can be modified iteratively using a diffusion model, which can then be trained to recover the original image in the reverse process. This task is much simpler than working with the sound itself, and has been used extensively by other authors [2–4].

C. Diffusion models

A diffusion model is a probabilistic model that defines the process of adding noise to the data and learning to reverse this process to generate new samples.

1. Forward diffusion process

The first component of a diffusion model is the forward diffusion process, which consists of a Markov chain gradually adding Gaussian noise to a data sample over a series of T steps. With x_0 as the original sample, the forward process defines a sequence of latent variables $\mathrm{x}_1, \mathrm{x}_2, \ldots, \mathrm{x}_\mathrm{T}$ according to the distribution:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{\alpha_t}x_{t-1}, (1-\alpha_t)\mathbf{I})$$

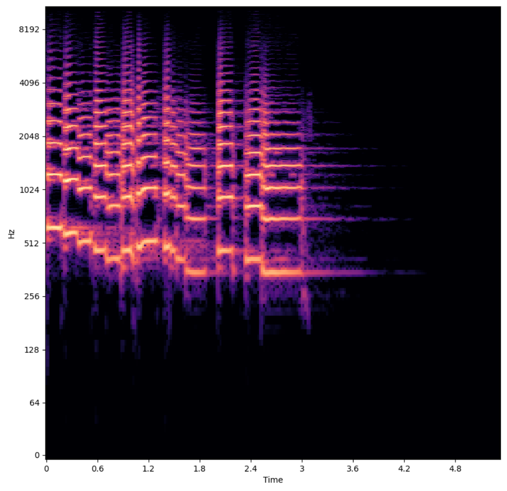
where α_t is a sequence of scaling factors that control the amount of noise added at each step in the forward diffusion process. By applying this process recursively, we can derive the distribution of any x_t given x_0:

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)\mathbf{I})$$

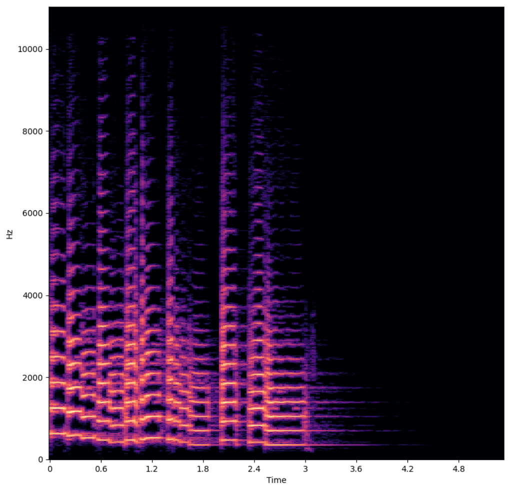
$$\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$$

2. Reverse diffusion process

The goal of the reverse diffusion process is to learn how to retrieve the original image from a completely noisy one, which can be achieved in a number of ways. During the development process many approaches were studied,



(a) Example of the Mel spectrogram of a trumpet sound (plot by authors)



(b) Regular spectrogram of the same sound (plot by authors)

FIG. 2: Same sound represented by (a) Mel Spectrogram and (b) Regular Spectrogram. The top portion of the regular spectrogram is squeezed in the Mel spectrogram, as there is no clear distinction between the higher frequencies in the human auditory system.

but the method finally implemented consists on iteratively predicting $\mathbf{x_{t-1}}$ from $\mathbf{x_t}$ using the sampling rule:

$$p(x_{t-1}|x_t) = \mathcal{N}\left(x_{t-1}; \mu_\theta(x_t, x_0, t), \sigma(t)\mathbf{I}\right) \tag{1}$$

where

$$\mu_\theta(x_t, x_0, t) = \frac{1}{\sqrt{\alpha_t}}\mathbf{x_t} - \frac{1-\alpha_t}{\sqrt{(1-\bar{\alpha}_t)\alpha_t}}\epsilon \tag{2}$$

$$\sigma(t) = \frac{(1-\alpha_t)(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} \tag{3}$$

All the necessary information to calculate (3) is available. However, we cannot calculate (2) as we do not know ϵ.

The goal then is to train a network that predicts ϵ from $\mathbf{x_t}$ and t, and use it to generate $\mathbf{x_{t-1}}$ using (1). For the present work, we chose $\alpha_t = 1 - 0.9t$, as done elsewhere.

3. *Training*

To train these models, we derive the loss function from the Evidence Lower Bound (ELBO):

$$\mathcal{L} = \mathbb{E}_q \left[\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)} \right]$$

We can simplify this by reducing it to a Mean Squared Error (MSE) between the predicted noise $\epsilon_\theta(x_t, t)$ and the actual noise ϵ added in the forward process:

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{t,x_0,\epsilon} \left[\|\epsilon - \epsilon_\theta(x_t, t)\|^2 \right]$$

In our case, we will train our model to generate a plausible Mel spectrogram from one representing Gaussian noise using the reverse diffusion process. We will then apply reverse STFT to generate an audio sample from the image.

III. RELATED WORKS

Music generation with artificial neural networks has been the subject of extensive experimentation. Various methods have been used to achieve it. To generate a sequence of notes, Recurrent Neural Networks (RNNs) seem particularly suited for the task; therefore, many previous works have used them. For example, Chu *et al.* (2016) [5] used Long Short-Term Memory (LSTM) networks to generate melodies.

Generative Adversarial Networks (GANs) have also been a popular option. For instance, Yang *et al.* (2017) [6] used GANs to generate audio tracks with multiple MIDI tracks (multiple melodies/instruments in parallel).

With the increasing popularity of transformer-based models, transformers have also been used to generate music samples. For example, Choi *et al.* (2021)[7] used transformers to generate melodies given specific chord progressions.

In recent years, diffusion models have emerged as a promising approach for music generation tasks. These probabilistic models are adept at capturing complex data distributions and have shown potential in generating high-fidelity audio samples.

Kong *et al.* (2021) [8] proposed DiffWave, a diffusion model-based architecture capable of generating high-quality waveform samples. By modeling the sequential diffusion process, the model learns to generate audio waveforms that exhibit realistic characteristics, thereby bypassing the need for intermediate representations such as spectrograms.

Goel *et al.* (2022) [9] extended the application of diffusion models to music generation, demonstrating the generation of diverse musical samples from learned diffusion processes. Their work highlights the adaptability of diffusion models in capturing the complex temporal structures inherent in music data, paving the way for novel approaches in music synthesis.

The field of music generation has witnessed diverse approaches, ranging from RNNs and GANs to transformer-based models and diffusion models. While each approach offers unique advantages and challenges, diffusion models have shown promise in capturing the intricate patterns and structures present in music data. In our work, we focus on the diffusion model approach as it has been the most promising in recent years.

IV. MODEL ARCHITECTURE

A. Global Architecture

For our model, we use the UNet architecture [10] (fig. 3) with embeddings for time and genre. We use a total of 8 layers: 1 `fatten` layer to increase the number of channels on the input, 3 down-sampling blocks that decrease the size of the input and increase the number of channels, and 3 up-sampling blocks that return the input to the original size of the image. We include a skip connection between the down and up layers of the same size to transmit higher-level features through the network, and to retain information features from the original data. The final output of the up layers goes through a `thin` layer that reduces the number of channels back to 1.

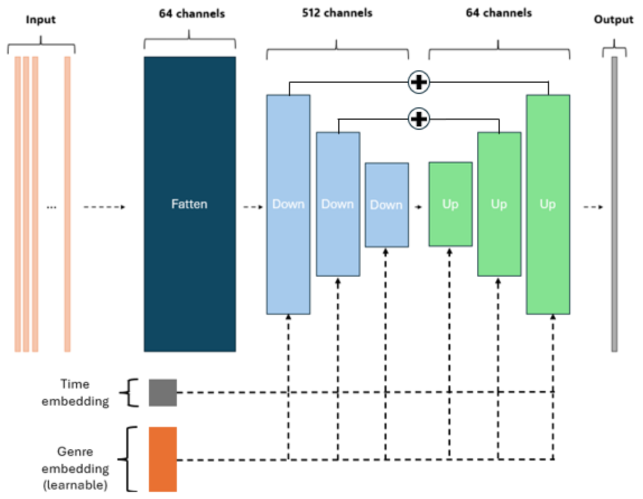


FIG. 3: Model Architecture.

For time embedding, we create an $\mathbb{R}^C$ vector where C is the number of channels by creating an array of $C/2$ frequencies. We then calculate 2 sets of embeddings, p_1 and p_2, by taking the sine and cosine of the product of the frequencies and the time, respectively. We obtain an $\mathbb{R}^C$ vector that we can add to the input of the layer. This embedding is not learnable.

For genre embedding, we use the Embedding class to create a learnable embedding $\mathbb{R}^C$ where that maps each genre, represented as an independent class, into an appropriate vector. We can then add the column corresponding to the genre to the input of the layer. Making this embedding learnable allows us to have an optimal encoding of the desired genre.

B. Architecture of the UNet Block

The UNet is composed of two types of blocks: the Down block, which reduces the size of the image and increases the number of channels, and the Up block, which increases the size of the image and decreases the number of channels.

1. *Residual Layer*

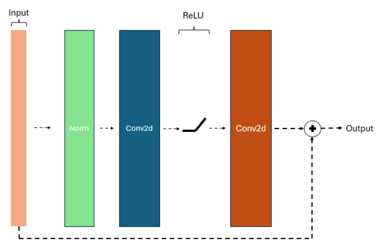
The main component of a UNet block is the residual layer, which is composed of a normalization layer, a convolutional layer, a Rectified Linear Unit (ReLU) activation, and another convolutional layer (fig. 4a). We add a skip connection from each layer to the next so that the model can learn the difference (residual) between the desired output and the input, rather than learning the entire output from scratch. This simplifies the optimization process and helps prevent degradation in performance as the network depth increases.

2. *Down Block*

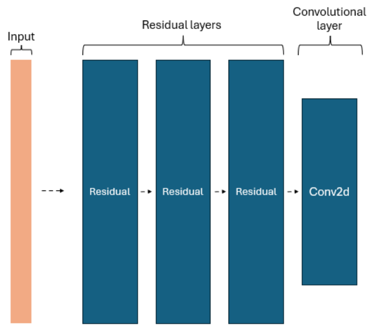
The down block is composed of 3 residual layers with the same size for input and output followed by a downsampling convolutional layer. The objective of using such blocks is to reduce the spatial dimensions of the feature maps while increasing the channel depth. This allows the model to capture high-level abstract features while discarding fine-grained details, which are typically preserved through the residual connections.

3. *Up Block*

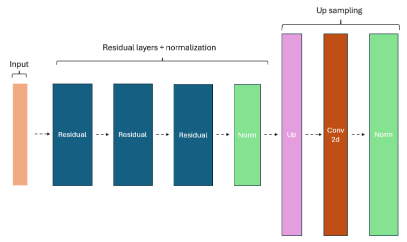
The second part of the UNet consists of Up blocks, each comprising 3 residual layers followed by an upsampling operation that duplicates pixels to create a larger image. This is followed by a convolutional layer and normalization. The function of this block is to enhance the resolution of the image and generate finer details and textures in the generated images, thereby improving the visual quality of the output samples.



(a) Residual Layer Architecture.



(b) Down Block Architecture.



(c) Up Block Architecture.

FIG. 4: UNet Block Architectures: (a) Residual Layer, (b) Down Block, and (c) Up Block.

V. DATASET AND DATA PREPARATION

To develop a model capable of generating music fitting specific mood descriptors while maintaining a medieval theme, we required a dataset that aligned with our thematic and emotional requirements. We sought out hour-long audio files, each representing a fixed mood, specifically curated for use by RPG game masters. These files provided the consistency needed for each mood descriptor, ensuring that the generated music would remain cohesive within the medieval context.

A. Data Collection

We gathered a total of 16 hours of music, divided equally into 8 different moods: epic, festival, fight, mysterious, romance, sad, tavern, and town. Each mood had 2 files of an hour of continuous music in the form of 50MB .mp3 files. The choice of these moods was motivated by their frequent use in RPG settings, where epic music might underscore a dramatic battle or quest, and festival music could enhance scenes of celebration or social gatherings.

B. Preprocessing

The raw audio data in .mp3 format needed to be transformed into a form suitable for training our model. We chose Mel spectrograms as the representation of audio data because they effectively capture the essential features of the audio signal in a format conducive to neural network processing.

1. Mel Spectrogram Generation

To convert the audio files into Mel spectrograms, we used the MelSpectrogram class from the transforms module in torchaudio. The Mel spectrogram provides a time-frequency representation of the audio signal, which is particularly useful for tasks involving audio pattern recognition. We followed the conditions of spectrogram generation used in [8].

2. Audio Segmentation

To create a manageable dataset, we subdivided the hour-long audio files into 10-second samples with no overlap between samples. This segmentation was chosen to balance the need for sufficiently long samples to capture musical phrases with the practical constraints of memory and processing power during training. This process resulted in approximately 720 Mel spectrograms for each mood descriptor, providing a substantial dataset for training the model.

C. Post-Processing and Reconstruction

Once the model has been trained and generates Mel spectrograms as output, we need to convert these back into audio signals. For this, we use the librosa library to perform the inverse transformation. In order to do this, we first need to change the obtained Mel Spectrogram into a linear spectrogram, and then use the result of that conversion to obtain the audio using the Short Term Fourier Transform (STFT).

VI. TRAINING

To train our model, we defined methods for both the forward diffusion process, which computes the noisy image at a given time step, and the reverse diffusion process, which denoises the image to try to generate the original and, therefore, the brand-new spectrogram. The loss used was the accumulated Mean Squared Error between the estimated $\tilde{\mathbf{x}}_\mathbf{0}$ and the real image $\mathbf{x_0}$ at each timestep.

In a first approach we calculated the loss as the Mean Squared Error (MSE) between the true noise added during the forward diffusion process and the expected noise to remove in the reverse diffusion process calculated by the model (”Noise prediction” interpretation of diffusion models).

During training, we experimented with different setups for the parameters of the optimizer:

- **Weight initialisation:** We used a He initialisation to ensure a more efficient training.
- **Learning rate:** We experimented with multiple settings for the learning rate. What gave us the best results was adding an adapting learning rate that begins at a given value and is multiplied by 0.1 every 5 epochs.
- **Weight Decay:** To mitigate overfitting, we incorporated weight decay into our training process. Weight decay acts as a regularization technique by adding a penalty to the loss function based on the magnitude of the weights, effectively discouraging the model from learning overly complex patterns that may not generalize well to unseen data.
- **Batch Size:** We utilized a batch size of 16 images. This batch size was chosen to strike a balance between the efficient use of computational resources and the stability of the training process.
- **Epochs:** The model was trained for 10 epochs. This number of epochs was determined through preliminary experiments, ensuring that the model had sufficient time to learn from the data without overfitting.

The model was trained in two different settings for the optimizer parameters, varying learning rate and weight decay. The first setting had a learning rate of 1×10^{-4} and a weight decay of 1×10^{-5} (fig. 5). The second setting had the adaptable learning rate starting at 1×10^{-3} and a weight decay of 1×10^{-4} (fig. 6). The conditions for the second training resulted in an overall better performance of the model, as the model converged more quickly. The higher learning rate facilitated faster learning, and the higher weight decay helped to maintain generalization by preventing overfitting.

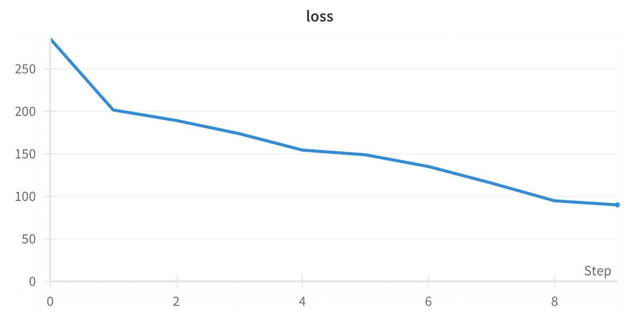


FIG. 5: Training loss: $lr = 1 \times 10^{-4}$, $wd = 1 \times 10^{-5}$.

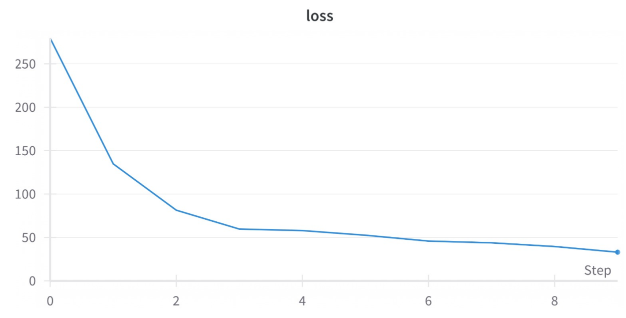


FIG. 6: Training loss: $lr = 1 \times 10^{-3}$, $wd = 1 \times 10^{-4}$.

We used the mean squared error loss on the predicted noise to compute the loss of the model. Adapting the parameters of the optimizer allowed us to significantly reduce the training loss score at the end of the 10 epochs.

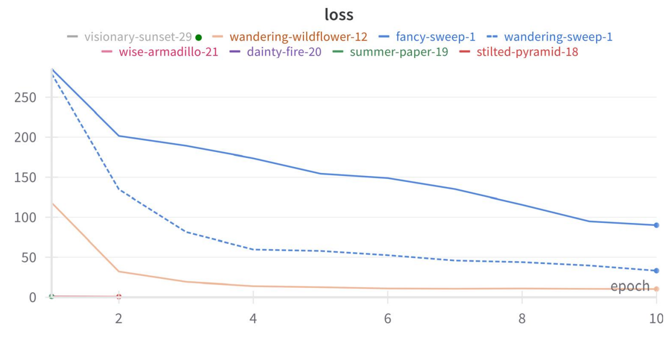
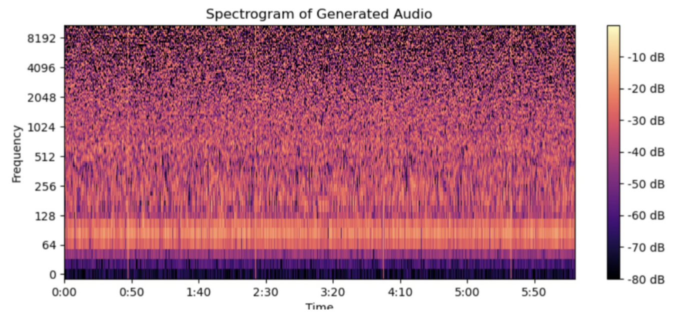


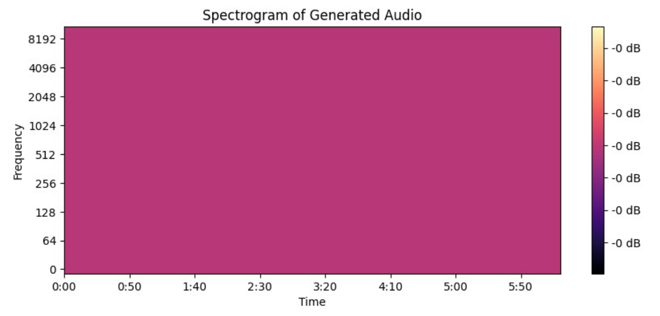
FIG. 7: Comparison of training logs (full blue : fixed learning rate and weight decay, dotted blue : adapting learning rate and fixed weight decay, yellow : adapting learning rate, fixed weight decay and He initialization)

VII. RESULTS

The result we obtained with this version of the model were inconsistent. We were able to generate some effectively denoised audio. The result lacked in actual musical coherence but did correspond to actual sound and not noise (fig. 8a). However this version of the model also gave very bad results, with instances in which the values blew up to infinity due to an infinitesimally increase in the computed variances on the diffusion process, resulting in blank spectrograms (fig. 8b).



(a) Example of a generated spectrogram



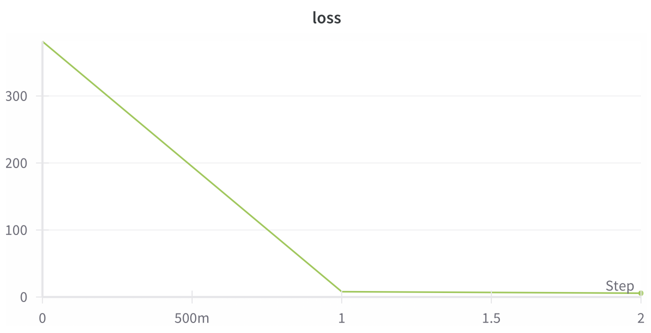
(b) Blank spectrogram generated by the model

FIG. 8: Generated Spectrograms: (a) Correct Spectrogram, (b) Blank Spectrogram

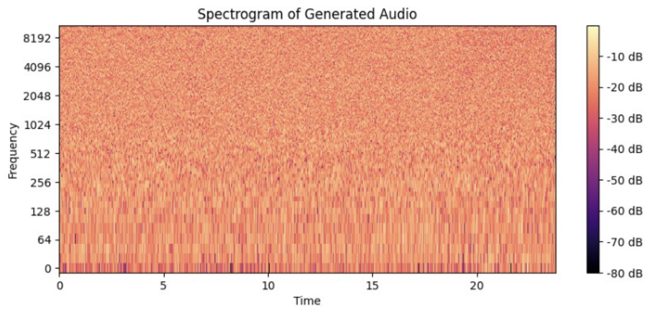
After spotting that issue, we tried to correct it but similar results were generated. That's why we decided to make a integral change in our model and implement the "Denoising" interpretation (predicting $\mathbf{x_0}$ from the current $\mathbf{x_t}$ step) instead of the noise added at each time step in the forward process.

As far as training hyperparameters are concerned, since the Unet model did not vary from one approach to another, we decided to keep some of the parameters who had given good performance before. Although, the learning rate was changed into 5×10^{-3} since the adaptative learning rate ensured smoother jumps at the final epochs even if the original rate was a bit higher.

This new interpretation involved several more computations than the previous due to its intrinsic theoretical complexity. Therefore, due to the fact that some hyperparameter tuning and testing needed to be done and the lack of powerful resources, we decided to reduce the noising/denoising steps to a feasible amount which allowed us to see some results. Moreover, the training epochs also needed to be highly reduced to 3 so the results were not expected to be clean (although the error plot did show a consistent and fast learning of our model).



(a) Loss plot of the training of the final model



(b) Example of a spectrogram generated

FIG. 9: Generated plots: (a) Training Loss, (b) Generated Spectrogram

VIII. DISCUSSION

Although the performance of the model is not quite up to the expectations we could have for the specified use case, it did show proof of the possibility of generating music samples using such architecture. There are several points that limited the effectiveness of our approach.

A. Complexity of the model construction

Diffusion models are the state-of-the-art in Deep Learning architectures and require a powerful resources and, overall, deep knowledge. After hours of research we managed to get a gist of how they worked but it was not until we had a lecture on that topic that we finally began to work on the right track. Unfortunately, even though we believe we built a strong and well-performing architecture, this occurred late and the time we had to re-build and test our model was limited.

B. Limitation in the Number of Parameters

The relatively low number of parameters plays a significant role in the performance of our model. Our model is indeed quite shallow with only 3 down and up blocks in the global architecture. This surely had an impact on the model's ability to learn "musical" features from the provided spectrograms.

C. Limitation in Computing Resources

In the beginning of the project we did not think that would be a constraint in order to have a good performance so we did not take into account cloud computing for training. However, we were not considering to use Diffusion models, which are considerably more costly and time-consuming. Nevertheless, the previous limitations we identified both originate in our lack of computing resources. With our relatively small dataset and model, the training process already takes several hours. With more powerful machines, we could handle a larger dataset through a deeper model, which could actually learn to generate coherent music.

D. Hyperparameters Optimization

The difference in performance we observed between our 2 settings for learning rate and weight decay showed us that the fine-tuning of these parameters can be crucial in the final performance of our model. Additional experiments with different learning rates, batch sizes, and regularization techniques might yield better results but we think this trial processes have yielded satisfactory results in this area.

E. Future Developments

At this stage, we began training another version of our model which gives promising results in comparison to what we obtained previously. However this version takes a long time to train (several hours per epochs) and we have not yet finished the complete training. Although we are confident that it would be able to consistently generate a readable audio file that contains clear sound and from which we could distinguish the mood descriptor, given enough training time.

Additionally, different ways of adding noise can be used. For example, a non-linear diffusion kernel is used in [11], in which the change in mean and variance changes throughout the diffusion process. This kernel changes the diffusion process and allows to retain information about the inherent structure of the data throughout the diffusion process. Evaluating how this affects the learning of the model would be important to explore in the future.

IX. CONCLUSION

Generating music is a difficult task for a deep learning model. In this project, we managed to prove that using a diffusion model, we could generate original Mel spectrograms that correspond to credible sounds. However, we were not able to generate music that we can actually use in the scope of our initial use case. Further developments should focus on increasing the depth of the model so that it may have more success in learning musical features from the data, as well as expanding the dataset to provide more complexity. More attention can also be given to the fine-tuning of hyperparameters in the optimization process, which can have a great impact on final performance.

[1] S. S. Stevens, J. Volkmann, and E. B. Newman. A Scale for the Measurement of the Psychological Magnitude Pitch. *The Journal of the Acoustical Society of America*, 8(3):185–190, 01 1937.

[2] Ryan Prenger, Rafael Valle, and Bryan Catanzaro. Waveglow: A flow-based generative network for speech synthesis, 2018.

[3] Cyran Aouameur, Philippe Esling, and Gaëtan Hadjeres. Neural drum machine : An interactive system for real-time synthesis of drum sounds, 2019.

[4] Sean Vasquez and Mike Lewis. Melnet: A generative model for audio in the frequency domain, 2019.

[5] Hang Chu, Raquel Urtasun, and Sanja Fidler. Song from PI: A musically plausible network for pop music generation. *CoRR*, abs/1611.03477, 2016.

[6] Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang. Midinet: A convolutional generative adversarial network for symbolic-domain music generation using 1d and 2d conditions. *CoRR*, abs/1703.10847, 2017.

[7] Kyoyun Choi, Jonggwon Park, Wan Heo, Sungwook Jeon, and Jonghun Park. Chord conditioned melody generation with transformer based decoders. *IEEE Access*, 9:42071–42080, 2021.

[8] Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. Diffwave: A versatile diffusion model for audio synthesis, 2021.

[9] Karan Goel, Albert Gu, Chris Donahue, and Christopher Re. It's raw! Audio generation with state-space models. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 7616–7633. PMLR, 17–23 Jul 2022.

[10] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[11] François Rozet and Gilles Louppe. Score-based data assimilation, 2023.