Diffusion Based Audio Generation

Ruben Nithyaganesh

***Abstract*—Generating data from complex data distributions has been a long-standing problem in the field of artificial intelligence, with generative models offering many opportunities in rapid content creation, increasing efficiency, and many other use cases. Diffusion models are one class of generative models that have seen great success in recent years. In this work, we look to leverage current state of the art diffusion methods to generate musical audio. By estimating the gradient of an unknown target distribution, diffusion models have the capacity to generate new data samples from complex data distributions. Recent work has seen improvements to diffusion methods, particularly in training and sampling procedures that have allowed for improvements in sampling quality and cost. We present the usage of contemporary diffusion techniques for the purpose of musical audio generation and discuss the effectiveness of diffusion models in this setting. Our work comprises of converting a dataset of classical piano pieces into a set of spectrogram images that our used within a diffusion-based setup to generate novel spectrogram images. We convert generated spectrogram images back to raw audio, resulting in audio sequences that resemble audio from our training set. Finally, we discuss opportunities for future work for diffusion methods, particularly regarding our current model and improvements that could be made to it.**

I. Introduction

Generative modelling with artificial intelligence has become a major area of research in the field of deep learning. The common goal of generative models is as follows: given a set of samples, generate new samples that resemble that of the original set. Many variants of generative models exist such as GANs, VAEs, and Normalising Flow models. In recent years diffusion models have become among some of the best performing generative methods, outperforming existing methods. Due to the nature of their design, diffusion models bypass specific limitations that have constrained previous systems. In our work we look to use existing diffusion methods to facilitate the generation of musical audio. We use the MAESTRO dataset, a collection of classical piano performances as the training data to our model. Our objective is to then be able to generate audio segments that resemble the piano music in our training set. We are able to convert the raw audio into spectrogram images using the Short Time Fourier Transform (STFT) that we use as input into our model. Our model is then trained to generate novel spectrogram images, which can be converted back to raw audio via the Griffin-Lim algorithm. In addition to using diffusion methods, we define a method of generating arbitrary length audio sequences by generating blocks of audio that are generated conditioned on the block of audio that came before it.

II. Related Work

In this section we review work relating to our project. Primarily we describe the formulation of diffusion methods, and the audio processing concepts used in our work. We also include discussion about any relations this project may have to sustainability concerns as described in [13].

This project was supervised by Bastiaan Kleijn

A. Diffusion Models with Stochastic Differential Equations

We begin by describing diffusion models through a framework of stochastic differential equations. Diffusion models work by a process of corrupting data points taken from a target data distribution in a well-defined iterative fashion, such that at the end of the process the samples are transformed into approximately pure Gaussian noise. The task of a diffusion model is then to learn a reversal process that takes the corrupted data laying on the Gaussian distribution back to the target distribution. With this learned reversal process new data samples can be synthesised by sampling data from Gaussian noise and running this data through the reversal process. Thus, diffusion models define a method of taking a previously complex and intractable data distribution, and formulating a process to sample new data points from it.

The process of corrupting the target data distribution is commonly referred to as the forward process, and is defined (in the 'variance preserving' case) as follows:

$$q(x_t|x_{t-1}) := \mathcal{N}(x_t : \sqrt{1-\beta_t}x_{t-1}, \beta_t I) \tag{1}$$

Here we obtain data point x_t by applying a small amount of Gaussian noise to the previous data point x_{t-1}. Thus, the forward process is a Markov process where the next data point is determined purely from the previous data point in the sequence. The process begins by sampling a data point from our target distribution $x_0 \sim q(x)$. With the forward process we then obtain a series of increasingly noisy samples $x_1, x_2, ..., x_T$. β_t defines a variance schedule that dictates the amount of noise applied at each time step.

We can represent the forward process in closed form by:

$$x_t = \sqrt{1-\beta_t}x_{t-1} + \sqrt{\beta_t}\mathcal{N}(0, I) \tag{2}$$

Song et al [12] show that this forward process can be written in the form of a Stochastic Differential Equation (SDE), in which the change in the data point is described in the infinitesimal limit:

$$x_t = \sqrt{1-\beta(t)\Delta t}x_{t-1} + \sqrt{\beta(t)\Delta t}\mathcal{N}(0, I)_{(\beta_t := \beta(t)\Delta t)} \tag{3}$$

$$\approx x_{t-1} - \frac{\beta(t)\Delta t}{2}x_{t-1} + \sqrt{\beta(t)\Delta t}\mathcal{N}(0, I) \tag{4}$$

$$dx_t = -\frac{1}{2}\beta(t)x_t dt + \sqrt{\beta(t)}\, dw_t \tag{5}$$

Where dw_t is the standard Wiener process (represents infintesimal noise). The equation shown in (5) conforms to the form of a general SDE, which is given by

$$dx = \underbrace{f(x,t)}_{\text{drift coefficient}} dt + \underbrace{g(t)}_{\text{diffusion coefficient}} d\omega_t \tag{6}$$

In equation (5) $f(x,t) = -\frac{1}{2}\beta(t)x_t$ and $g(t) = \sqrt{\beta(t)}$. The SDE corresponding to these specific settings of the drift and diffusion coefficients is referred to as the 'Variance Preserving' SDE. The key insight into formulating the forward noising process into an SDE, is the fact that an equivalent 'reverse SDE' can be derived that reverses the noising process.

$$dx = [f(x,t) - g^2(t)\nabla_x \log\, q_t(x_t)]dt + g(t)d\omega \tag{7}$$

Which in the Variance Preserving SDE case presented in (5), corresponds to a reverse SDE given by

$$dx = [-\frac{1}{2}\beta(t)x_t - \beta(t)\nabla_x \log\, q_t(x_t)]dt + \sqrt{\beta(t)}d\omega \tag{8}$$

The reverse SDE consists of the same drift and diffusion terms, but introduces $\nabla_x \log\, q_t(x_t)$, which is known as the score function, a vector field indicating the direction of increasing probability density. The score function is not directly accessible due to the intractability of $q_t(x)$ however. Diffusion methods propose that we can approximate the score function through a parameterized model $s_\theta(x_t, t)$. Naively, we could learn this model by optimising the objective

$$\mathbb{E}_{t\sim U(0,T)}\mathbb{E}_{x_t\sim q_t(x_t)}||s_\theta(x_t,t) - \nabla_x \log\, q_t(x_t)||_2^2 \tag{9}$$

However, we don't have access to the true score function $\nabla_x \log\, q_t(x_t)$ as we don't have access to the true underlying data distribution $q_t(x_t)$. Song et al. [11] proposed *denoising score matching* as a solution to this problem, where we take advantage of the fact that we can diffuse a data sample x_0 into a noised x_t using the forward process.

In the case of the forward SDE defined in (5), we can parameterize the process into one that takes an initial data point x_0 and returns a noised data point x_t at a given time step t:

$$q_t(x_t|x_0) = \mathcal{N}(x_t; \gamma_t x_0, \sigma_t^2 I) \tag{10}$$
$$\gamma_t = e^{-\frac{1}{2}\int_0^t \beta(s)ds}$$
$$\sigma_t^2 = 1 - e^{-\int_0^t \beta(s)ds}$$

A noised sample x_t can be obtained in closed form by:

$$x_t = \gamma_t x_0 + \sigma_t \epsilon \quad \epsilon \sim \mathcal{N}(0, I) \tag{11}$$

With $q(x_t|x_0)$ being tractable, we can now define the objective as:

$$E_{t\sim U(0,T)}E_{x_0\sim q_0(x_0)}E_{x_t\sim q_t(x_t|x_0)}||s_\theta(x_t,t) - \nabla_x \log\, q_t(x_t|x_0)||_2^2 \tag{12}$$

Optimising this objective will mean $s_\theta(x_t,t) \approx \nabla_x \log\, q_t(x_t|x_0)$. The score function can be written explicitly as:

$$\nabla_x \log\, q_t(x_t|x_0) = -\nabla_x \frac{(x_t - \gamma_t x_0)^2}{2\sigma_t^2} = -\frac{x_t - \gamma_t x_0}{\sigma_t^2} \tag{13}$$

Substituting our closed form representation of x_t in terms of x_0 we get:

$$-\frac{\gamma_t x_0 + \sigma_t \epsilon - \gamma_t x_0}{\sigma_t^2} = -\frac{\epsilon}{\sigma_t} \tag{14}$$

Interestingly, we see the value of $\nabla_x \log\, q_t(x_t|x_0)$ is the noise ϵ added to our original data point x_0 when sampling x_t, scaled by the inverse of σ_t. With this in mind, we can parameterize our network in the following fashion:

$$s_\theta(x_t, t) := -\frac{\epsilon_\theta(x_t, t)}{\sigma_t} \tag{15}$$

$s_\theta(x_t, t)$ can be optimised by minimising the following objective

$$\mathbb{E}_{t\sim U(0,T)}\mathbb{E}_{x_0\sim q_0(x_0)}\mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}\frac{1}{\sigma_t^2}||\epsilon - \epsilon_\theta(x_t,t)||_2^2 \tag{16}$$

Given a noised sample x_t and time step t, our parameterised model is tasked with predicting the noise values that were added to x_t. Once we obtain an optimised model $s_\theta(x_t,t) \approx \nabla_x \log\, q_t(x_t|x_0)$ we can substitute it in the Reverse SDE from (8) to generate new data samples.

$$dx_t = -\frac{1}{2}\beta(t)[x_t + 2s_\theta(x_t,t)]dt + \sqrt{\beta(t)}d\bar{w}_t \tag{17}$$

We can sample new data points by solving the Reverse SDE, one such way to do this is through the Euler-Maruyama method:

$$x_{t-1} = x_t + -\frac{1}{2}\beta(t)[x_t + 2s_\theta(x_t,t)]\Delta t + \sqrt{\beta(t)\Delta t}\mathcal{N}(0,I) \tag{18}$$

We sample $x_T \sim \mathcal{N}(0,I)$, and use the Reverse SDE to obtain the next sample x_{t-1}. Once we reach x_0, our data sample should resemble a data point that exists within the original complex data distribution we are interested in sampling from, $x_0 \sim q(x)$.

Song et. al further define a 'probability flow' ODE, by way of the Fokker-Planck equation, that describes the same evolution of a data point x over time, but without the stochastic component present in the SDE. The general form of the probability flow ODE is defined as

$$dx = [f(x,t) - \frac{1}{2}g^2(t)\nabla_x \log\, q_t(x_t)]dt \tag{19}$$

The probability flow ODE can be reversed in time by simply adding a minus sign to the term.

B. A unified diffusion framework

For our work we utilise a framework for diffusion models proposed by Karras et. al [5]. In this work the probability flow ODE in (19) is defined as the following:

$$dx = \frac{\dot{s(t)}}{s(t)}x - s(t)^2\dot{\sigma}(t)\sigma(t)\nabla_x \log p\left(\frac{x}{s(t)}; \sigma(t)\right)dt \tag{20}$$

Here $f(x,t)$ is reformulated to $\frac{\dot{s(t)}}{s(t)}$ and $g(t)$ is reformulated to $s(t)\sqrt{2\dot{\sigma(t)}\sigma(t)}$, and $p(x;\sigma)$ represents the data distribution of the training data perturbed with $\mathcal{N}(0, \sigma^2 I)$. The ODE in (20) is defined such that the respective $s(t)$ and $\sigma(t)$ schedules

correspond directly to the scaling of x at time t and the standard deviation of noise added to the data at time t. This framework offers an arguably more coherent view of the ODE as the scaling and noise schedules define more explicitly the evolution of x. Karras et. al show that various settings of the schedules $s(t)$ and $\sigma(t)$ correspond to existing diffusion methods. Karras et. al further derive and equivalent SDE in this framework as

$$dx_{\pm} = \underbrace{-\dot{\sigma}(t)\sigma(t)\nabla_x \log p\left(x;\sigma(t)\right) dt}_{\text{ODE from (1) with } s(t)=1} \tag{21}$$

$$\pm \underbrace{\beta(t)\sigma(t)^2\nabla_x \log p\left(x;\sigma(t)\right)}_{\text{noise decay}} + \underbrace{\sqrt{2\beta(t)}\sigma(t)d\omega_t}_{\text{noise injection term}} \tag{22}$$

Here dx_+ moves forward in time, while dx_- moves backward. In this framework $\beta(t)$ describes the relative rate at which existing noise is replaced with new noise. To make this SDE equivalent to the SDEs of Song et al. $\beta(t) = \frac{\dot{\sigma(t)}}{\sigma(t)}$. Intuitively this setting means the score function vanishes from the forward SDE making it consistent with the SDE defined by Song et. al.

Within this framework the settings of $s(t) = 1$ and $\sigma(t) = t$ were found to be effective choices of schedules for defining the ODE and SDE. While previous diffusion methods solve the SDE and ODE through Euler methods akin to that in (18), Karras et. al present an improved sampling algorithm that was found able to achieve improved sampling performance. With the improvements presented by this work, we choose to utilise this framework alongside the improved sampling algorithm presented within it.

Another finding from the work in [5] that we employ is their parameterization and preconditioning of the neural network model. For this parameterisation, we consider the model $D_\theta(x;\sigma)$ that attempts to be the optimal denoising function that minimizes the equation $\mathbf{E}_{y\sim p_{data}}\mathbf{E}_{n\sim\mathcal{N}(0,\sigma^2 I)}||D_\theta(y+n;\sigma) - y||_2^2$, where the score function can be obtained by

$$\nabla_x \log p(x;\sigma(t)) = \frac{(D(x;\sigma) - x)}{\sigma^2} \tag{23}$$

This parameterisation differs from that shown in (15) as the network now attempts to output the denoised image, rather than the noise values added to the image. The preconditioning of the network is defined as follows

$$D_\theta(x;\sigma) = c_{\text{skip}}(\sigma)x + c_{\text{out}}(\sigma)F_\theta(c_{\text{in}}(\sigma)x; c_{\text{noise}}(\sigma)) \tag{24}$$

This parameterization of the denoiser network allows the raw neural network to estimate either the original signal, the noise added to the signal, or something between the two. The loss function using this parameterisation is expressed with respect to the raw neural network layers F_θ as

$$\mathcal{L} = \mathbf{E}_{\sigma,y,n}[\lambda(\sigma)c_{\text{out}}(\sigma)^2||F_\theta(c_{\text{in}}(\sigma)\cdot(y+n); c_{\text{noise}}(\sigma)) \tag{25}$$

$$- \frac{1}{c_{\text{out}}}(y - c_{\text{skip}}(\sigma)\cdot(y+n))||] \tag{26}$$

Where $\lambda(\sigma)$ is an additional loss weighting function. We define the settings for these preconditioning parameters in the Design section.

C. Benefits of Score Based Modelling

In the diffusion model described above, we work closely with the score function $\nabla_x \log q(x)$. Compared to other generative modelling methods, working with the score function offers some distinct advantages [10]. One challenge when modelling a distribution $q(x)$ is that the distribution must be normalised (the probability values assigned by $q(x)$ to each possible x must sum to one), and this requires calculating a normalising constant for the model distribution that is typically intractable at high dimensions. Previous generative models have attempted to address this issue in various ways. Flow based models [8] attempt to approximate a normalising constant, which leads to inaccurate probability evaluation. Other approaches such as Variational Auto-Encoders (VAEs) [7] restrict the family of models that can be used to satisfy the normalised constraint. While approaches such as Generative Adverserial Networks (GANs) [2] look to only model the generative process, meaning these models cannot be used to evaluate the probabilities of data points, and GAN training has been known to be unstable. Working with the score function bypasses the need to calculate the intractable normalising constant, but simultaneously does not restrict the family of models that can be used and allows for probability evaluation. In recent work, diffusion models have been able to beat GANs in the domain of image synthesis[1].

D. Representation of Audio

One important consideration is the representation of audio data. Audio is typically represented through waveforms and spectrograms.

1) Waveforms: Waveforms describe an audio signal in terms of audio pressure with respect to time. A waveform can be represented by a tensor of shape $[C, T]$ where C is the number of audio channels and T is the number of samples (our data is mono, so $C = 1$). T is determined by the sampling rate and duration of the audio clip. A typical sampling rate is 48kHz, so a tensor containing 1 second of audio would contain $T = 48000$ samples (for mono audio).

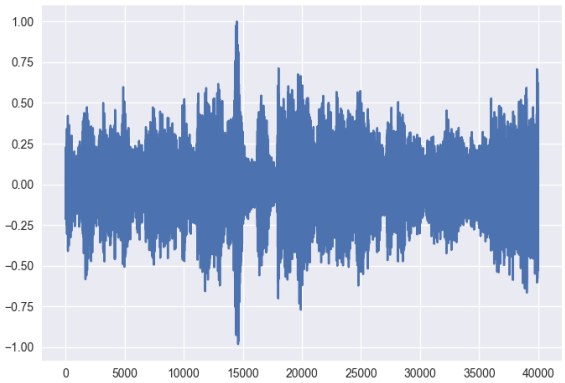

Fig. 1. Waveform representation of audio.

2) Spectrograms: Spectrograms can be computed from audio waveforms using the short-time-Fourier-Transform (STFT). The STFT is used to determine the frequency and phase of the sinusoidal components of a signal at different windows in time and is performed (discretely) by splitting a signal into equal sized frames (typically overlapping to reduce artifacts), and performing a Fourier Transform on each frame. The result of the STFT is a complex-valued matrix, carrying with it the magnitude and phase information of the component frequencies of a signal at different points in time. The STFT in the discrete case can be expressed as

$$X(m,w) = \sum_{n=-\infty}^{\infty} x[n]w[n-m]e^{-i\omega n}$$

with $x[n]$ being the signal to be transformed, and $w[n]$ being a window function that isolates the signal to some local time frame. Key parameters to the STFT include the windowing function, the window size, and the hop length between windows. The STFT produces a complex valued matrix as output, encoding both magnitude and phase information. It is typical to discard the phase information as it is largely random and contains little structure to be learnt by a deep learning model, and utilise only the magnitude information. The difficulty here is that to reconstruct a waveform the phase is an important component to recovering the original signal. An example of a magnitude spectrogram and its corresponding phase spectrogram are depicted in Figure 2 and 3 respectively.

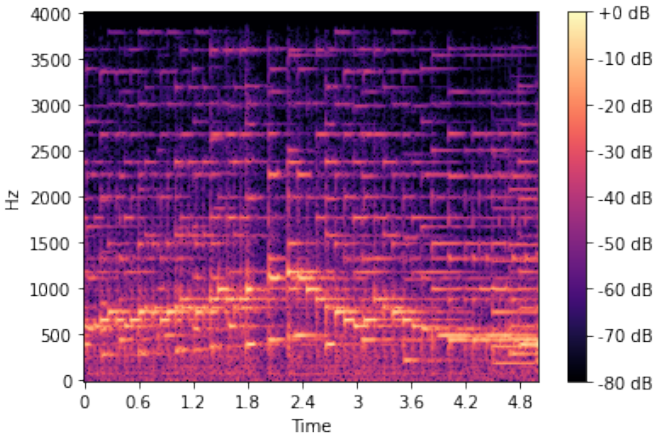

Fig. 2. Visualisation of magnitude components of spectrogram. Present are clear structures that identify patterns in the audio waveform.

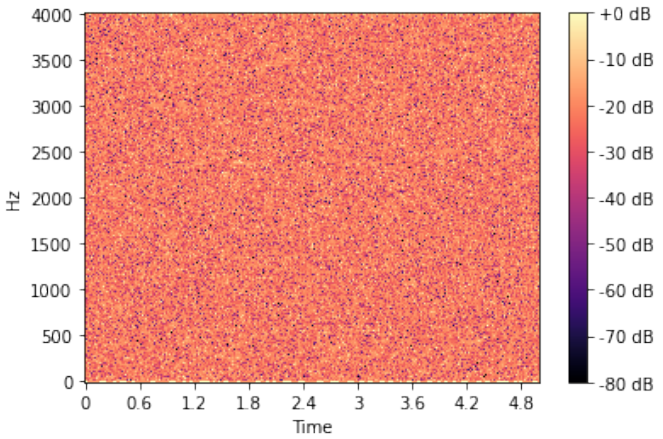

Fig. 3. Visualisation of phase components of spectrogram. Compared to the magnitude spectrogram in Figure 2, there is little to no structure to the phase data.

3) Griffin-Lim algorithm: In this work we are interested in the generation of audio waveforms, and thus using a magnitude spectrogram representation of audio necessitates reconstructing waveforms from their magnitude spectrogram. This can be achieved with the Griffin-Lim algorithm, which is an iterative algorithm that attempts to recover phase information given a magnitude spectrogram. The Griffin-Lim algorithm offers only an approximation of the original waveform and can be prone to producing artefacts. In our work we observe that increasing the resolution of spectrograms can aid in reducing the number of artefacts produced.

E. Diffusion for Audio Generation

There have been various implementations of diffusion methods for audio generation to date. Some notable contributions are ArchiSound [9], and Noise2Music [4]. Both approaches utilise similar diffusion methods to take randomly sampled Gaussian noise and iteratively denoise it into recognisable audio. As is common among diffusion models they employ a UNet architecture for the denoising neural network. One common characteristic of these models is that they are not autoregressive. This means that generated audio is of fixed size, and there is no notion of probabilistic generation based on data that has already occurred. The music generating models in [9] and [4] are capable of generating a variety of audio representing different genres and instruments. For our model we attempt to work purely with piano audio. This is to limit the amount of data needed to train and create a functioning model that works with piano music. With the scope of the project and the time available to us, we thought this was a sensible decision (for reference, [4]'s training data amounts to approx. 340k hours of audio).

1) ArchiSound: The work in [9] outlines the creation of several models. Both (text) conditional and unconditional audio generating models, a model for upsampling audio, a vocoder model that converts mel-spectrograms into audio, and an autoencoder that can compress audio into a compressed latent representation.

2) Noise2Music: The work in [4] outlines 4 parameterised models. A waveform generator that generates text-conditioned low fidelity audio and a corresponding cascader model that generates higher fidelity audio conditioned on the generated low fidelity audio and the same text conditioning. There is a similar setup that works with spectrograms. A generator model that produces a spectrogram conditioned on text, and a vocoder model that produces 16kHz audio conditioned on the generated spectrogram.
Both ArchiSound and Noise2Music employ a variety of models to facilitate audio generation. Our proposed method is a lot simpler only consisting of one trained model, however we look to define a method of generating arbitrary length audio segments rather than fixed block sizes. More detail is given in Section III.

F. Sustainability Concerns

We believe this project has minimal effect on many of the stated goals in [13]. We do however recognise that deep learning models in general require large amounts of computation, which can lead to large electricity consumption at large scales. We maintain awareness of this, however we believe that the scale of our model training and inference is well within reasonable in this regard. We also express concern regarding generative models and their potential implications on the future job market, particularly for creative professions. As generative algorithms improve, their wider use in place of human artists may well become more common. Generative technology is a rapidly developing field and the ethics behind the usage of AI algorithms in place of human workers is a contentious issue, particularly when generative algorithms that create artistic pieces depend on existing pieces created by humans to use as training data. For this project we do not state any intention to replace human creativity, nor do we make the claim that it would be ethical to do so.

III. Design

In this section we describe the key design choices surrounding our solution. We cover dataset construction and data processing, model architecture, diffusion settings, and our sampling process.

A. High-level design

The objective of our work is to create a generative model for piano music using diffusion methods. Our approach to this problem is to create a diffusion model that generates spectrogram images that can be converted to waveforms via the Griffin-Lim algorithm. Should our generative system be capable of generating spectrograms that resemble that of true piano audio, then the produced waveforms should in principle sound like piano music. In light of previous diffusion based audio generative models, our approach differs in that we look to create a system that can generate arbitrary length sequences of audio, rather than generating sequences of fixed size. Where previous methods look to condition generation on provided text, we look to condition generation based on the audio that has been generated previously, such that long sequences of audio can be generated iteratively by conditioning the generation of the current audio sequence with the audio sequence that came before it.

B. Dataset

For our task we must first define a dataset for our model to train on. We chose to work with the MAESTRO dataset, a set of recorded classical piano performances, amounting to around 200 hours of audio.

1) Waveform processing: The MAESTRO dataset comes in the form of raw .wav files of varying length, in the order of 2 to 10s of minutes. The audio is in stereo (2 channels), sampled at 44.1kHz. To limit the dimensionality and overall size of this data, we opted to work with mono audio (1 channel) and down sample all the audio to 8kHz. While this is a significant reduction in sampling rate, we thought the savings in space and dimensionality were justified. Nyquist's theorem states that we need a sampling rate at least double the frequency of any signal we want to sample accurately. At a sampling rate of 8kHz the theoretical highest frequency signal we could accurately sample would be 4kHz. There is only one note on a standard piano (C8 at 4186Hz) that exceeds this limit, thus we considered down sampling to 8kHz viable as we would only lose the ability to sample the fundamental frequency of one note from the original audio samples. Down sampling involves applying a low-pass filter to remove frequencies larger than half the desired sampling rate (preventing aliasing), and then removing every Nth sample where N is chosen to achieve the desired sampling rate. Another processing step that was performed was to split every 8kHz mono audio file into segments of approx. 2 seconds. This was done so that every sample was of consistent length and could be used in a model working with fixed size input.

2) Spectrogram generation: Given our waveform processing has been completed, we obtain a set of approx. 2 second 8kHz .wav audio files. To use as input to a neural network, we would like to generate spectrograms whose dimensions are a power of 2. Our neural network employs downsampling and upsampling layers that halve the spatial dimensions at each layer. Thus an input with dimensions of a power of 2 is convenient as up and downsampling will retain consistent dimension sizes at each layer. This is important as the neural network model employed involves skip connections that need the dimensions of the downsampling and upsampling layers to be consistent across every layer. We must set the STFT parameters such that they produce spectrograms with dimensions in accordance with this requirement.

Dimensions	Window	WinSize	HL
128x128	Hann	256	128
256x256	Hann	512	64

TABLE I
SETTINGS OF STFT

Table I displays the STFT settings we choose for two dimension sizes of 128x128 and 256x256. Window refers

to the windowing function that isolates the current window from the entire audio signal, WinSize refers to the number of samples captured in the sliding window, while HL indicates the number of samples the window slides across each time. We work with two dimension sizes 128x128 and 256x256, as we used the smaller resolution spectrogram images to train and test a smaller initial model, and then upscaled to higher resolution spectrogram images for a larger model. The resulting spectrgrams given by our settings will actually produce images of dimensions 129x128 and 257x256 respectively. For input to our neural network we simply discard the top row of these images. The width of the produced images is determined by the window size, hop length, and the length of the input signal. Keeping window size and hop length constant, the input signal length of 2.096s is required to produce our desired width in both cases. This is the reason for the choice of splitting our waveforms into approximately 2 second segments.

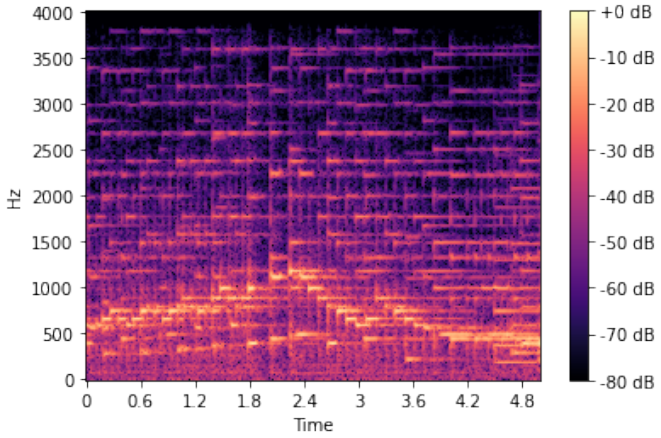

Fig. 4. Depiction of a 256x256 spectrogram. Compared to the equivalent 128128 spectrogram in Fig. 5, the higher resolution offers more detail

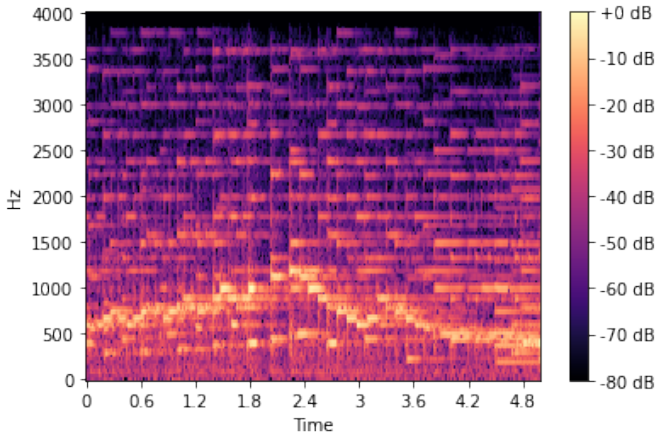

Fig. 5. 128x128 spectrogram.

While using 128x128 spectrograms constitutes less computational complexity and shorter training and inference times, they lead to worse waveform approximations with Griffin-Lim in comparison to using 256x256 spectrograms. Thus there is a trade-off here where faster sampling can be achieved using a lower resolution spectrogram at the cost of audio quality. We considered using spectrograms of even higher dimensionality, but the gain in audio quality in comparison to increase in dimensionality wasn't justified.

3) Normalisation: The final step in our dataset definition is applying normalisation to our produced spectrograms. As mentioned previously the first step was to discard the top row of produced spectrograms. In their raw form, our spectrograms are complex-valued. We take the absolute value of each complex number to produce a magnitude spectrogram. This discards phase information. We then convert from amplitude to dB scale, and finally normalise the data to lie between 0 and 1. With this completed, we define a pipeline for converting a set of audio waveforms into sets of spectrogram images ready to be used in a deep learning model.

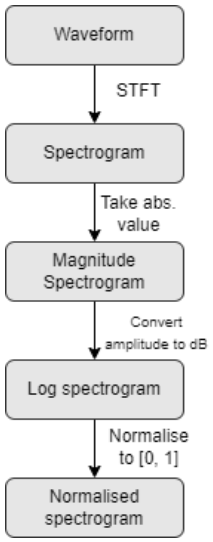

Fig. 6. Data processing pipeline for producing spectrogram images as input to neural network

C. Model Architecture

As described in Section II, we are interested in modelling the score function through a parameterised model $D_\theta(x, t)$. For this work we use the same architecture for the parameterised model as is seen in the work by Song et al [12], with slight modification and addition. From a high level the architecture of our model is that of a UNet, consisting of a series of down-sampling and then up-sampling convolutional blocks, such that the input and output dimensionality are the same. There are skip connections between corresponding up and down sampling blocks. A positional embedding that encodes the current time-step t is used at each layer of the UNet, and we also employ a new 'Spectrogram embedding' that is equivalently used at each layer. Our spectrogram embedding is a new addition to the architecture that facilitates conditioning generation on another spectrogram, effectively creating an image conditional model. It is implemented with a series of convolutional blocks, with a final fully connected layer that transforms the final convolutional feature map into a vector embedding that is passed to the main UNet layers. For

our model we employed 5 down-sampling and up-sampling layers. A high level overview of the employed architecture can be seen in Fig. 7, while a depiction of our added spectrogram embedding can be seen in Fig. 8.

1) UNet Architecture: We employ a UNet architecture as it is ubiquitous among diffusion implementations. A key constraint on the design of our model architecture is the fact that the input and output dimensionalities are the same. This alone encourages a UNet like structure. In addition to this UNets are able to learn lower level feature of the input image via the encoding layers, while more fine grained features are preserved by the networks skip connections.

2) Spectrogram Embedding: The design of our spectrogram embedding employs a similar structure to the encoder layer of the main UNet, where convolutional blocks facilitate the learning of useful features from an input image.

We considered an alternative to using a spectrogram embedding where we append the previous spectrogram as an additional channel in the noised image input. We thought this was a sub-par solution as we wanted the encoding and decoding layers to have access to all parts of of the conditioning image. This is particularly because we hypothesised that the end of the conditioning image should have a large influence on the beginning of the denoised image, as this would facilitate the generation of a spectrogram of which its beginning 'meshed' well with the end of the previous spectrogram. When we append these spectrograms together and produce a final waveform, this would hopefully result in a continuous audio sequence where the transition between each 'section' is natural and not noticeable.

Had the conditioning spectrogram been inputted as an extra channel, the nature of the convolutional layers in the UNet architecture would mean that information about the end of the spectrogram would only be locally accessible, and there would be no way for this information to influence the computation of the beginning of the generated image. In contrast, using an embedding means that this information is not only injected into each encoding and decoding layer, the embedding is capable of influencing any part of the denoised image.

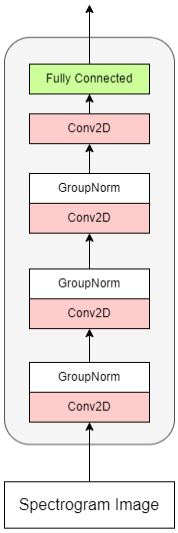

Fig. 8. Design of our implemented Spectrogram embedding as seen incorporated in the architecture displayed in Fig. 7. It receives a spectrogram image as input, down-samples in a similar fashion to the down-sampling layers of the main UNet, then applies a fully connected layer to transform the last feature map into a single vector embedding.

D. Diffusion setup

As mentioned, we use the diffusion framework defined by Karras et. al [5]. Under this framework, the main choices defining a diffusion method consist of the following: ODE solver, time step discretisation, scale schedule $s(t)$, noise schedule $\sigma(t)$, preconditioning functions $c_{\text{skip}}(\sigma)$, $c_{\text{out}}(\sigma)$, $c_{\text{in}}(\sigma)$, and $c_{\text{noise}}(\sigma)$, noise distribution during training, and loss weighting. For our setup we follow the same settings for these components as Karras et. al, which we define in the following section.

1) ODE Solver: We use the second order Heun sampler described in [5]. We employ their 'stochastic sampler', however opt for a parameter choice of $S_{\text{churn}} = 0$. This corresponds to no added noise during sampling steps as would be the case when solving an SDE, and is effectively solving the ODE defined in (20).

2) Time step discretization: When solving the ODE or SDE, we must define a set of discrete time steps to numerically approximate a solution. These timesteps are defined as

$$\sigma_{i<N} = (\sigma_{\text{max}}^{\frac{1}{\rho}} + \frac{i}{N-1}(\sigma_{\text{min}}^{\frac{1}{\rho}} - \sigma_{\text{max}}^{\frac{1}{\rho}}))^{\rho} \text{ and } \sigma_N = 0 \quad (27)$$

Where N is the total number of timesteps, and i is the ith timestep. Formulating time steps in this manner means larger timesteps are made over larger noise levels, while smaller timesteps are made at smaller noise levels. we found that a total of 40 times steps reliably yielded good samples from our model.

3) Scale and noise schedules: Scale and noise schedules are set to $s(t) = 1$ and $\sigma(t) = t$ respectively.

4) Preconditioning values: Here we must set the preconditioning parameters for the parameterization of D_θ as described in (24).

$$c_{\text{skip}}(\sigma) = \frac{\sigma_{\text{data}}^2}{(\sigma_{\text{data}}^2 + \sigma^2)} \quad (28)$$

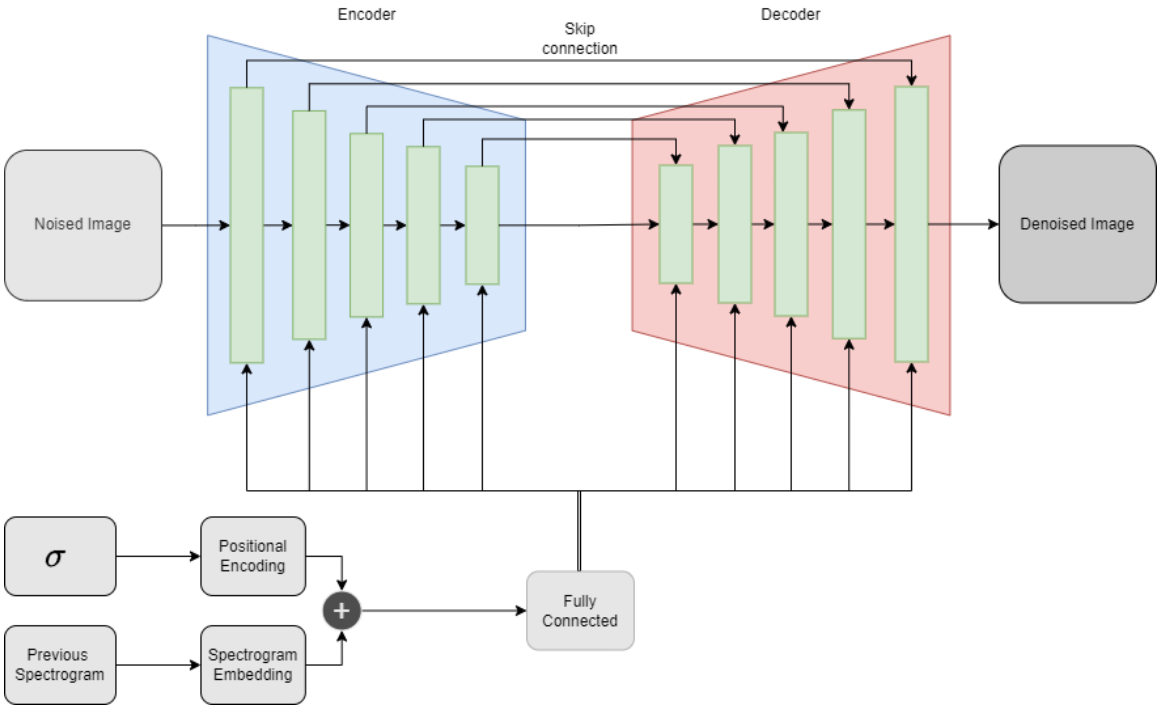

Fig. 7. High level design of the neural network trained to be the optimal denoiser. Network receives as input a noised image, noise level σ and a previous spectrogram. The noised image is passed through a series of up and down sampling convolutional layers. Meanwhile, the supplied σ and previous spectrogram are transformed into respective embeddings, added together and passed to each of the encoding and decoding layers.

$$c_{\text{out}}(\sigma) = \frac{\sigma \cdot \sigma_{\text{data}}}{\sqrt{\sigma_{\text{data}}^2 + \sigma^2}} \tag{29}$$

$$c_{\text{in}}(\sigma) = \frac{1}{\sqrt{\sigma_{\text{data}}^2 + \sigma^2}} \tag{30}$$

$$c_{\text{noise}}(\sigma) = \frac{1}{4}\ln(\sigma) \tag{31}$$

c_{out} and c_{in} are chosen to ensure network inputs and training targets have unit variance. c_{skip} attempts to prevent the amplification of errors in F_θ, particularly for high σ. c_{noise} is said to have been chosen empirically.

5) Noise distribution: During training, we sample noise levels that are used to noise data and train the denoising model. Where older methods simply sampled noise level from a uniform distribution, Karras et. al define a log-normal distribution in the hopes to target noise levels where more effective learning can take place.

$$\ln(\sigma) \sim \mathcal{N}(P_{\text{mean}}, P_{\text{std}}^2) \tag{32}$$

6) Loss weighting:

$$\lambda(\sigma) = \frac{(\sigma^2 + \sigma_{\text{data}}^2)}{(\sigma \cdot \sigma_{\text{data}})^2} \tag{33}$$

7) Parameters: These are the settings of the parameters for the components that have been described.

$$\sigma_{\text{min}} = 0.002, \sigma_{\text{max}} = 80 \tag{34}$$

$$\sigma_{\text{data}} = 0.5, \rho = 7 \tag{35}$$

$$P_{\text{mean}} = -1.2, P_{\text{std}} = 1.2 \tag{36}$$

σ_{data} represents the standard deviation of the original, unperturbed data distribution of our training data.

We choose to use these parameter settings as Karras et. al report SOTA performance on CIFAR-10 and ImageNet-64 datasets, as well as improved sampling speed compared to previous methods with these settings. As such, we reason that these parameter settings constitute good choices, reinforced by the fact that we observed good results with these settings also.

8) Training: For our training we employ the loss function defined in (25), except that the neural network F_θ receives an additional input representing the 'previous spectrogram' of the one being input into the neural network. For training, when sampling a spectrogram image from our dataset, we also retrieve the spectrogram image for the audio immediately preceding the sampled spectrogram. The currently sampled spectrogram is noised and passed to the network, while the 'previous spectrogram' is passed as an unperturbed signal to be used to create the conditioning spectrogram embedding.

E. Sampling

Our diffusion model defines a generative system where we can generate spectrogram images corresponding to piano music from sampled noise, conditioned on a spectrogram image representing the audio preceding the current generation. To create a new audio sequence, we generate an initial spectrogram generation unconditioned on any previous spectrogram. We can then generate subsequent spectrograms by conditioning the next generation on the previously generated spectrogram. This process can be repeated an arbitrary number

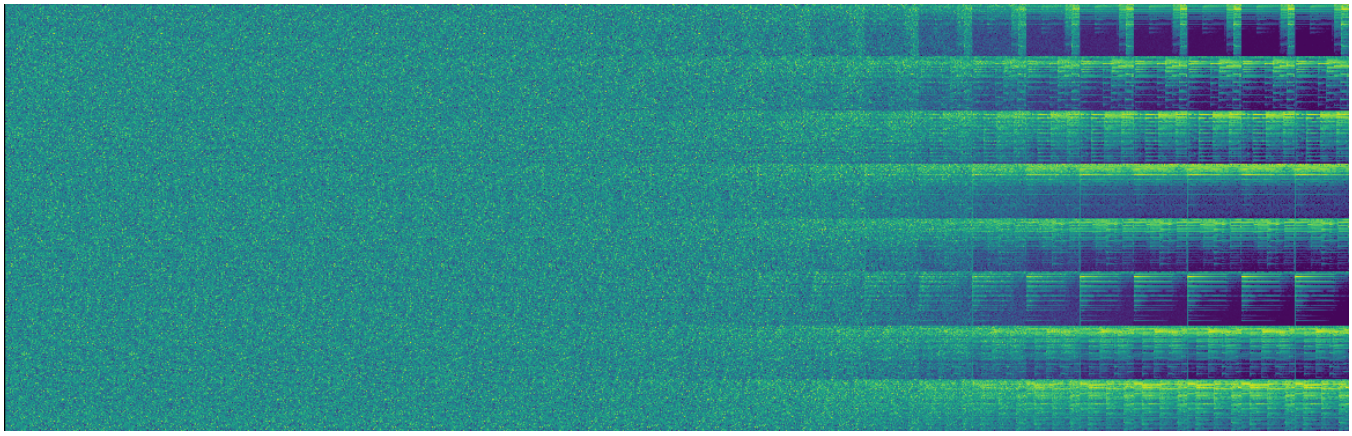

Fig. 9. Sampling 8 128x128 spectrograms with 24 sampling steps. Each spectrogram generation corresponds to approximately 2 seconds of audio. Each generation is begun by sampling Gaussian noise (left of figure), and denoising by solving the ODE in the reverse direction.

of times to generate audio sequences in roughly 2 second blocks. Once the chosen number of spectrograms have been generated, they can be concatenated into a single spectrogram and transformed into a waveform with the Griffin-Lim algorithm. As the input to our model is normalised between [0, 1] our generative model produces data in the same range. Thus prior to input into the Griffin-Lim algorithm we normalise the data to the range [-80, 0], and convert from dB to amplitude scale. As mentioned previously we discard the top row of our produced spectrograms so that they are square, here we also add a top row by padding with zero before performing Griffin-Lim.

IV. Implementation

In this section we describe the concrete components used to implement our chosen design. Our final implementation follows very closely to the design defined previously.

1) Programming Language: For development, we utilise the Python programming langauge. Python is a very flexible language that allows for rapid development. There exist various libraries readily available in Python that enable our development, such as PyTorch.

2) Dataset Storage: We store our dataset as raw .wav files using an Amazon S3 bucket. Having our data accessible in the cloud is vital as it allows for access to our training data from multiple machines without having to manually transfer data between each machine, which is important as we conduct training on the ECS GPU servers.

3) Model implementation: Our model is implemented in PyTorch. We use PyTorch as it has arguably become a de facto standard for deep learning research. Many existing works are implemented in PyTorch and thus comparison between implementations can more easily be performed if our model is implemented in the same framework. PyTorch offers extremely powerful tools for defining and training deep learning models, and enables GPU hardware acceleration that is vital for making training our model feasible in the time available.

4) Audio processing: For our audio processing needs we use the Librosa library in Python. Librosa is a widely used audio processing library that gives us access to all the audio processing functionality we require such as down-sampling, applying STFT, and the Griffin-Lim algorithm.

5) Training: Training was performed on the ECS GPU servers.

V. Evaluation

Compared to the more traditional machine learning tasks of classification or regression, generative modelling can be challenging to evaluate. The main difficulty stems from the fact that because generative models aim to generate novel data points, there aren't any ground truth data that correspond to the generated data to compare against. For generative models, we only have the training data to assess against.

A. Frechet Audio Distance

Frechet Audio Distance (FAD) is an adaption of the commonly used Frechet Inception Distance used for image generative models [6]. FAD allows for evaluation of a set of generated audio segments by comparing embedding statistics produced by the generated data points compared to a training set (or 'background set') when inputted into a classification model, typically the VGGish model [3]. FAD is calculated by computing multivariate Gaussians on the two sets of embeddings, and then calculating the following

$$F(\mathcal{N}_b, \mathcal{N}_e) = ||\mu_b - \mu_e||^2 + tr(\Sigma_b + \Sigma_e - 2\sqrt{\sigma_b \sigma_e}) \quad (37)$$

B. Evaluation results

For our FAD evaluation we generated 17k 8kHz 2 second samples using our 256x256 model, and evaluated these generations with a background set of 19k 8kHz 2 second samples from our training set. With this we achieved an FAD score of 4.62. For reference, wavefroms generated from complete noise obtained an FAD score of 169.48, while the best performing model in [4] is reported to have achieved an FAD score of 2.134. It is important to note that the FAD evaluation in [4] is computed for a different background evaluation set, so direct comparison cannot be made here. Additionally, FID is typically calculated on 50k generated samples, but this was not

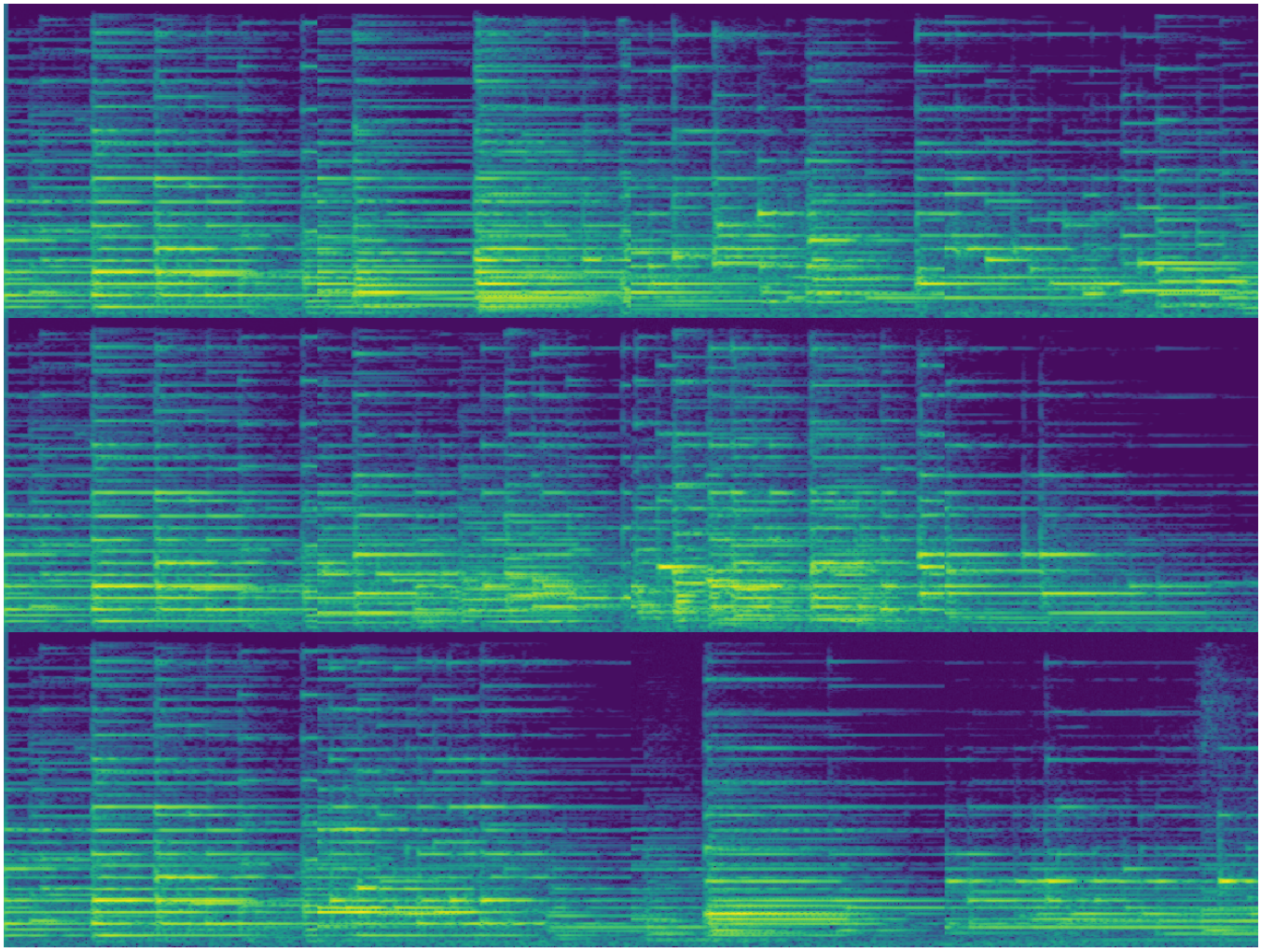

Fig. 10. Depicted above are three generations of approximately 8 seconds long (4 generated segments). Observe that the beginning of each sequence is identical. I.e, these generations were all started with the same initial generation. Further observe that the subsequent generations are all different, demonstrating that given the same initial starting point, due to the stochastic nature of the sampling process subsequent generations will vary. We argue that this indicates that the model is not simply reproducing audio it has seen during training, but learning meaningful relationships in the data that allow it to generate different sequences of feasible audio given the same starting conditions.

possible for us to generate in the time remaining for project completion. However we believe our FAD score indicates that our generations are reasonable and resemble that of our training set.

C. Novelty of Generations

For generative modelling, one concern is that the model may simply regurgitate samples from the training set, rather than truly synthesising new data points. Depicted in Fig. 10 we argue that our model is able to generate unique sequences of audio, even given the same starting sequence. This would indicate the model does not simply reproduce training examples, but learns and generates general structures that correspond to piano music.

VI. Conclusion and Future Work

To conclude, we believe we have been able to successfully define a diffusion model for the generation of piano audio. Compared to other diffusion based audio generative methods our method is somewhat simpler. Where previous methods train additional models for encoding, upsampling and vocoding functionality, we solely train a diffusion model. We also take a unique approach in working directly with audio spectrograms (not Mel-Spectrograms), and training an image conditional model that facilitates arbitrary sequence length generation. However we believe there to be many avenues for future work.

A. Vocoder model

Other generative audio models sometimes opt to transform (typically a mel-spectrogram) back to a waveform using a deep learning model, often referred to as a vocoder. Further work could be performed to use a vocoder in place of the Griffin-Lim algorithm in our design, and see if this produces better quality waveforms.

B. Higher resolution audio

At present our system only produces 8kHz audio. This is a very low sampling rate and further work could be performed into investigating ways to increase audio quality. This could potentially be achieved by employing upsampling networks, or introducing latent diffusion methods where training and sampling is performed on a lower dimensional latent representation of the audio data, allowing for the use of higher dimensional, higher definition audio.

C. Training on more data

Training on a larger dataset will likely improve the model. In particular training on a variety of musical genres and possibly allowing for the conditional generation of music from different genres or instruments. This extension will likely need to coincide with an increased audio quality output as the detail of particular instruments with high frequency components such as the high hat will largely be lost at our current sampling rate of 8kHz.

D. Improved sequential modelling

At present our system generates blocks of audio conditioned only on the previous audio generation, in a Markov-like fashion. Improved modelling of audio sequences would enable the model to consider longer sequences of audio when generating the next block. This could be achieved through a training and sampling procedure that incorporated an LSTM of some kind, where the output of the network at a given time encodes all the important features of the audio that has been generated previously.

REFERENCES

[1] Prafulla Dhariwal and Alexander Nichol. "Diffusion models beat gans on image synthesis". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 8780–8794.

[2] Ian Goodfellow et al. "Generative adversarial networks". In: *Communications of the ACM* 63.11 (2020), pp. 139–144.

[3] Shawn Hershey et al. "CNN Architectures for Large-Scale Audio Classification". In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. New Orleans, LA, USA: IEEE Press, 2017, pp. 131–135. DOI: 10.1109/ICASSP.2017.7952132. URL: https://doi.org/10.1109/ICASSP.2017.7952132.

[4] Qingqing Huang et al. *Noise2Music: Text-conditioned Music Generation with Diffusion Models*. 2023. arXiv: 2302.03917 [cs.SD].

[5] Tero Karras et al. "Elucidating the design space of diffusion-based generative models". In: *arXiv preprint arXiv:2206.00364* (2022).

[6] Kevin Kilgour et al. "Fréchet Audio Distance: A Metric for Evaluating Music Enhancement Algorithms". In: *arXiv preprint arXiv:1812.08466* (2018).

[7] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2022. arXiv: 1312.6114 [stat.ML].

[8] Yann LeCun et al. "A tutorial on energy-based learning". In: *Predicting structured data* 1.0 (2006).

[9] Flavio Schneider. *ArchiSound: Audio Generation with Diffusion*. 2023. arXiv: 2301.13267 [cs.SD].

[10] Yang Song. *Centre for Brains, Minds + Machines Diffusion and Score-Based Genrative Models*. 2022. URL: https://cbmm.mit.edu/video/diffusion-and-score-based-generative-models (visited on 05/30/2023).

[11] Yang Song and Stefano Ermon. "Generative modeling by estimating gradients of the data distribution". In: *Advances in neural information processing systems* 32 (2019).

[12] Yang Song et al. "Score-based generative modeling through stochastic differential equations". In: *arXiv preprint arXiv:2011.13456* (2020).

[13] *Transforming our world: the 2030 Agenda for Sustainable Development*. 2015. URL: https://sdgs.un.org/2030agenda (visited on 06/02/2023).