JOHANNES KEPLER
UNIVERSITY LINZ

Submitted by
Matthias Plasser

Submitted at
Institute of Computational Perception

Supervisor
Gerhard Widmer

Co-Supervisor
Silvan Peter

February 2023

Symbolic Music Generation using Discrete Diffusion Probabilistic Models

Master Thesis
to obtain the academic degree of
Master of Science
in the Master's Program
Artificial Intelligence

**JOHANNES KEPLER
UNIVERSITY LINZ**
Altenbergerstraße 69
4040 Linz, Austria
www.jku.at
DVR 0093696

Abstract

The idea of automatic, algorithmic symbolic music generation dates back to a time where the most powerful computers operated mechanically. Diffusion Probabilistic Models (DPMs) on the other hand are a latest class of deep generative models that were proposed in 2015. DPMs have already been applied to symbolic music generation, but only in an indirect way. Mittal et al. [2021] used a second generative model to translate the discrete task of symbolic music generation into a continuous task, which is then processed by the DPM. This diploma-thesis presents the direct application of Discrete DPMs to the inherently discrete problem of symbolic music generation. This direct approach does not only outperform state-of-the-art methods with just a fifth of trainable parameters of comparable models, but also offers additional flexibility. While the approach of Mittal et al. [2021] only supports infilling at a resolution of two bars (64 notes), the model proposed in this work (SCHmUBERT) is able to fill in single notes. Moreover, the generative capabilities of SCHmUBERT can be used to accompany given melodies. In addition to conditioning on existing musical pieces, the possibility of conditioning on abstract features such as note-density (classifier-guidance) is explored in this work.

Code and supplementary material for this work are available at `https://github.com/plassma/symbolic-music-discrete-diffusion`.

Zusammenfassung

Die Idee automatischer, algorithmischer Generierung symbolischer Musik stammt aus einer Zeit, in der die leistungsstärksen Computer noch mechanisch waren. Im Gegensatz dazu sind Diffusion Probabilistic Models (DPMs) eine Klasse neuester Deep Generative Models, die seit 2015 erforscht werden. DPMs wurden zwar schon auf symbolische Musikgenerierung angewendet, allerings nur indirekt. Mittal et al. [2021] nutzen ein zweites Deep Generative Model, um das diskrete Problem der symbolischen Musikgenerierung in ein kontinuierliches Problem umzuformen. Erst dieses kontinuierliche Problem wird in der Arbeit von Mittal et al. [2021] von einem DPM bearbeitet. Diese Diplomarbeit präsentiert die direkte Anwendung diskreter DPMs auf das von sich aus diskrete Problem der symbolischen Musikgenerierung. Dieser direkte Ansatz erzielt nicht nur state-of-the-art Ergebnisse mit nur einem fünftel trainierbarer Parameter vergleichbarer Modelle, sondern bietet auch zusätzliche Flexibilität. Während das Modell von Mittal et al. [2021] nur mit einer Auflösung von 2 Takten (32 Noten) ergänzen kann, kann das Modell aus dieser Arbeit (SCHmUBERT) einzelne Noten einfüllen. Außerdem können die generativen Qualitäten von SCHmUBERT genutzt werden, um Begleitungen für bestehende Melodien zu generieren. Außer der Konditionierung auf bestehende Musikstücke wurde auch die Konditionierung mit abstrakteren Merkmalen wie Notendichte untersucht (Classifier Guidance).

Der Code und Ergänzungsmaterial für Experimente dieser Arbeit sind hier vefügbar: `https://github.com/plassma/symbolic-music-discrete-diffusion`.

Contents

List of Figures

1 Introduction

Generative neural networks demonstrate impressive progress across various domains, like image, text and audio generation (StableDiffusion, Riffusion, ChatGPT [Rombach et al., 2022; Forsgren and Martiros, 2022; OpenAI, 2022]). A commonality all previously mentioned latest generation generative models share is the formulation of their generative approach as Diffusion Probabilistic Model (DPM) [Sohl-Dickstein et al., 2015]. This formulation, in contrast to other deep generative models allows training generative neural networks with tens to hundreds of thousands of layers depth. The necessity of according computational power to train networks of such exceptional depth may be a reason DPMs only began to outperform previous state-of-the-art generative models 6 years after their invention in 2015 [Sohl-Dickstein et al., 2015; Dhariwal and Nichol, 2021].

A domain, in which deep generative models in my opinion still fail to produce as impressive results is Symbolic Music, although the idea of automatic algorithmic composition dates back almost two centuries [Augusta Ada King, 1843]. In Symbolic Music, pieces consist of a series of symbols - notes that encode information like instrument, pitch, duration or onset. The information a note encodes typically is of discrete nature - although technically possible, pitches and timing information are not encoded as continuous values, but categorical. Symbolic music can be understood as instructions on how to synthesize audio music given an instrument. This symbolic representation of music is very compact compared to audio signals, but still encodes all compositionally relevant information.

DPMs have been applied to symbolic music generation by Mittal et al. [2021] in an indirect way: they first transform musical pieces in the form of a series of discrete values into a continuous domain, and only use DPMs to model these continuous representations. For the transformation from discrete to continuous and back again, they use an autoencoder (MusicVAE, [Roberts et al., n.d.]), and even discard most of its output. Thus, the DPM does not perceive the discrete nature of its task.

DPMs were formulated for both discrete and continuous domains [Sohl-Dickstein et al., 2015], and Bond-Taylor et al. [2022] show that discrete DPMs are very capable of modelling discrete sequences.

In this work, I aim to explore, whether a direct application of DPMs to Symbolic Music generation entails advantages over existing generative models for this domain.

1.1 Structure of Work

The remainder of this work is divided in five chapters. **Symbolic Music Generation** (Chapter 2) introduces the concept of symbolic music and three generative approaches related to my work. In **Deep Generative Models** (Chapter 3), the theoretical background of the three generative models related to my work is explained. **Discrete Diffusion Model Implementation and Experimental Setup** (Chapter 4) describes the methods and structures of experiments that produced the results of this work. **Results** (Chapter 5) are given in the form of training and evaluation statistics, and also include randomly selected generated samples and a discussion of them. Finally, **Conclusion and Future work** (Chapter 6) completes the work.

2 Symbolic Music Generation

This chapter introduces the concept of symbolic music, and three symbolic music generation approaches related to my work.

Symbolic Music is music represented as a collection of music events (notes) that usually contain information about timing (onset, offset, duration), pitch, volume and instrument, but not the sounds of the instruments themselves. Symbolic Music can be viewed as instructions on how to synthesize an actual audio signal given some instruments.

Synthesizing this audio signal is the act of modulating pitch, volume, but possibly also other parameters of an instrument according to the symbolic instructions, and can be performed by a human playing notes from a sheet, but also by a computer rendering a MIDI file.

A key difference between symbolic and audio representation is the number of datapoints per time: A typical piece of western music usually consists of clearly below 100 events per second, while the number of datapoints in a second of the same piece in audio representation is fixed to the sample rate, for which 44,100 datapoints per second is a very typical value. This value for the sample rate results from the maximal frequency humans can typically perceive (below 20kHz), and that in order to be able to reconstruct signals sampled with a constant sampling rate r, $r > 2 * f_{max}$ should hold [Shannon, 1949].

Although an audio datapoint consists of only one value – while a symbolic music datapoint typically consists of more than 5 values – symbolic representations are orders of magnitudes more memory-efficient than audio signals. Furthermore, symbolic representations explicitly express important characteristics of music like the pitch and duration of a note played with an instrument, which might be very beneficial for generative tasks. The advantages of symbolic representations over signals mentioned above lead to the assumption that symbolic music generation is a computationally easier problem than direct audio signal music generation.

2.1 Symbolic Music

This section lists common representation forms of symbolic music and their characteristics.

2.1.1 Music Sheet

This form of Symbolic Music is arguably the oldest and most commonly known representation listed in this section. Various forms of notation were developed historically, some date even back to antiquity. The most commonly known form of notation is the Modern Staff Notation, which allows encoding pitch and duration of a sound by positioning a note head with an optional stem on horizontal lines. Thereby, the horizontal position encodes the onset, while the type of note head and stem encode duration, which is measured in fractions of bars (full, half, quarter, ...). Pitch is indicated by the relative vertical position of the note head and staffs. Polyphony refers to the simultaneous occurrences of multiple notes or voices, and multiple voices can be distributed among several staves.

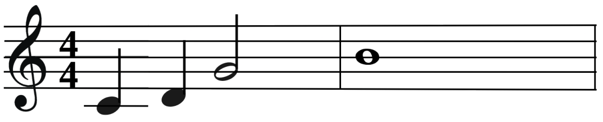


Figure 2.1: Two bars of staves with with horizontal lines and notes.
Durations from left to right: quarter, quarter, half, full.
Piches from left to right: C, D, A, C.

2.1.2 MIDI

The Musical Instrument Digital Interface protocol was introduced in August 1982, and designed for the communication between keyboards and synthesizers. In that mode, MIDI events consist of 3 values: on/off+channel-number, pitch, volume. Channels can be assigned to instruments at synthesizing time. If instead of directly synthesizing sound from a keyboard the information is to be stored in a file, information about timing is stored as delta-time to the next event.

2.2 MusicVAE

In 'A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music', Roberts et al. [n.d.] address the problem of generating symbolic music with a Hierarchical Recurrent Variational Autoencoder (MusicVAE). My work, as well as the work of Mittal et al. [2021] use the music encoding of Roberts et al. [n.d.], which is explained in section 4.2.

Figure 2.2 shows MusicVAE's architecture, which uses a bidirectional LSTM [Hochreiter and Schmidhuber, 1997] as encoder. Roberts et al. [n.d.] state that previous publications have shown that while short sequences can be compressed into a single latent, and decoded to the original sequence again using Recurrent Neural Networks, this approach performs worse with increasing sequence length. Roberts et al. [n.d.] overcome those issues by employing a hierarchical RNN as decoder: a conductor RNN decodes the latent z into a series of high level latents. These high level latents are then decoded by a bottom-level RNN autoregressively.

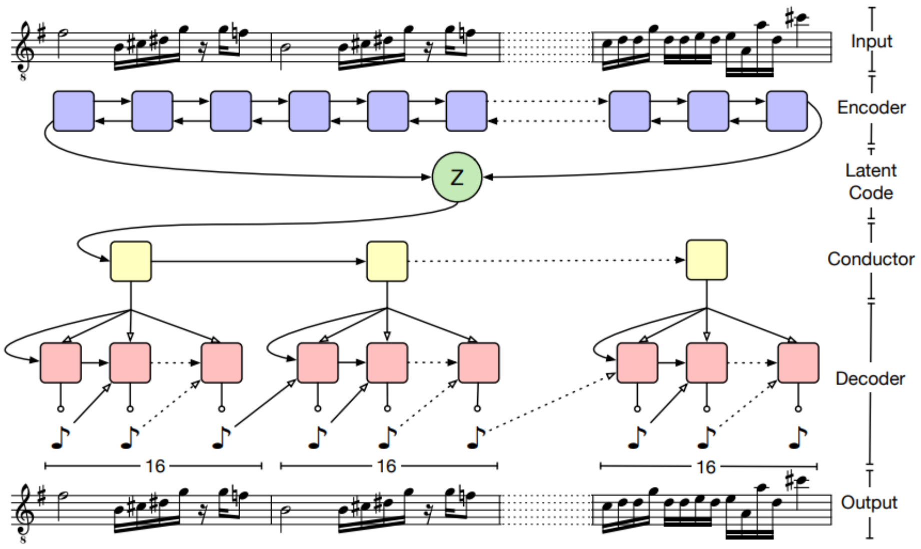


Figure 2.2: MusicVAE architecture [Roberts et al., n.d.]

2.3 Symbolic Music Diffusion

In this work, Mittal et al. [2021] use a DDPM (see section 3.3) to model a sequence of MusicVAE latent variables with high long-term coherence. While MusicVAE cannot reliably reconstruct sequences with lengths of 16 bars (256 tokens) anymore, the hybrid model of Mittal et al. [2021] successfully models sequences of 64 bars (1024 tokens).

In figure 2.3, the architecture of the hybrid model is illustrated: In the top part, the the DDPM, which iteratively refines Gaussian noise into 32 MusicVAE latents (labelled VAE Posterior) is depicted. Each of those latents is then separately decoded into a sequence of 2 bars (32 tokens) by MusicVAE (bottom part). Mittal et al. [2021] use a framewise self-similarity metric (see section 4.7) to evaluate long-term structure in their samples. In lack of an alternative, this metric is also used to assess sample quality in my work. In addition to being able to generate samples with high long time coherence, the proposed composite model allows for post-hoc conditioning (infilling, outpainting). The statement of Mittal et al. [2021] that "despite these exciting advances, DDPMs have not yet been applied to symbolic music generation because their iterative refinement sampling process is confined to continuous domains such as images and audio." inspired the idea of my work: As DDPMs can in fact be applied to discrete domains [Sohl-Dickstein et al., 2015; Austin et al., 2021; Bond-Taylor et al., 2022], they can also be directly applied to symbolic music.

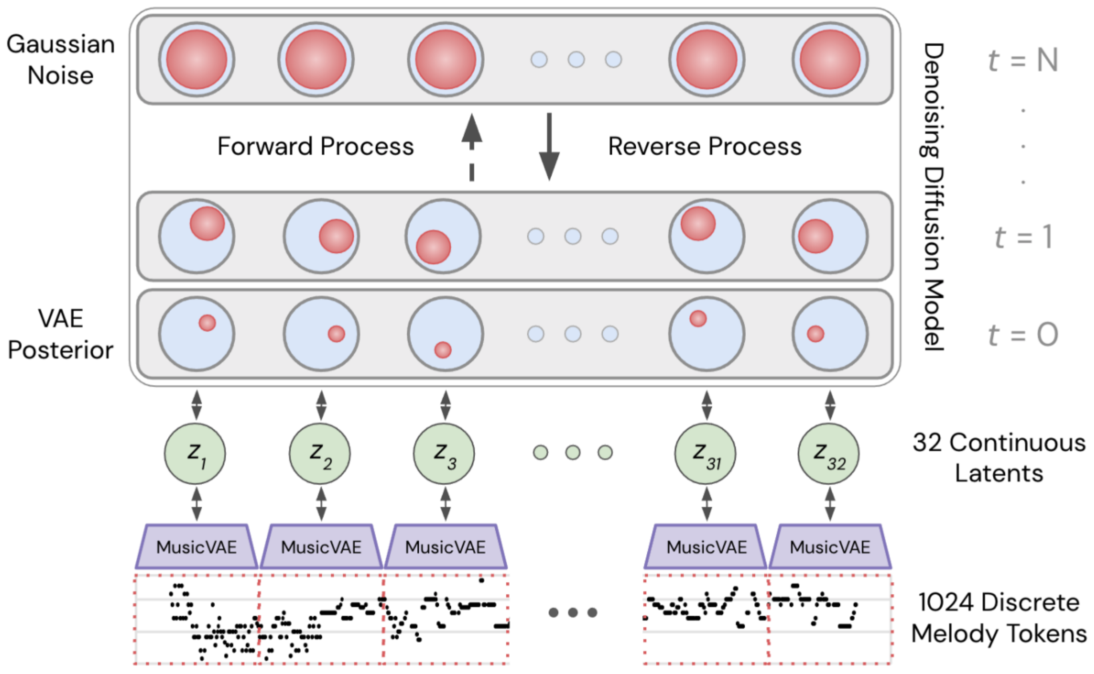


Figure 2.3: DDPM applied to a sequence of 32 MusicVAE latents [Mittal et al., 2021]

2.4 Unleashing Transformers

In their publication, Bond-Taylor et al. [2022] show the capabilities of Discrete Diffusion Probabilistic Models generating images and sequences. In a composite model, they use a VQ-VAE to compress images into a series of discrete tokens. In the second stage of their model, an Absorbing State Discrete Diffusion Probabilistic Model then generates this discrete sequence. Although the work of Bond-Taylor et al. [2022] is about image generation, their model could be adapted to symbolic music generation with very little effort. In my work, I use a very similar Discrete Diffusion Model (see section 4.3), and the basic Transformer Model architecture (see section 4.6.1) is inspired by the work of Bond-Taylor et al. [2022].

3 Deep Generative Models

Generative models are statistical models that approximate data from a desired distribution X. In contrast to classification, where a model predicts a label Y given data X from an observable distribution $P(Y \mid X)$, generative models approximate the distribution of X given a label Y: $P(X \mid Y)$, where the label Y may be an implicit parameter of the model and can be omitted if the target distribution X is not divided into classes.

This chapter lays out the three deep generative models relevant for this work and discusses their similarities and differences.

3.1 Generative Adversarial Networks

In 2014, Generative Adversarial Networks were first described by Goodfellow et al. [2014] and have since shown great success in modeling data across various domains such as natural images or audio signals [Goodfellow et al., 2014; Donahue, McAuley and Puckette, 2018]. The crucial novelty in GANs is their configuration which allows training using only backpropagation and sampling using only forward propagation. A generator network G maps random noise from a prior distribution $p_z(z)$ into data space $p_g(x)$. A discriminator D maps the generators output from the data space into a single scalar $s \in [0, 1]$ indicating whether the presented sample was produced by the generator or stems from the real distribution. The following formalisations of GAN loss function, training process and sampling process are based on [Goodfellow et al., 2014].

3.1.1 Loss function

The goal of the discriminator is to predict whether a sample x is real or generated by mapping its scalar output s to values close to 0 for generated data and close to 1 for real data. The optimal discriminator D^* maximizes $\mathbb{E}_{x \sim p_r}[\log(D(x))]$ while minimizing $\mathbb{E}_{x \sim p_g}[\log(D(x))]$ which is equivalent to maximizing $\mathbb{E}_{x \sim p_g}[\log(1 - D(x))]$:

$$D^* = \arg\max_D \mathbb{E}_{x \sim p_r}[\log(D(x))] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))] \tag{3.1}$$

The goal of the generator on the other hand is to fool the discriminator by generating samples which D cannot distinguish from real data. The optimal generator G^* generates samples which the discriminator recognizes as real, it maximizes $\mathbb{E}_{z \sim p_z}[\log(D(G(z)))]$ which is equivalent to minimizing $\mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))]$.

$$G^* = \arg\min_G \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))] \tag{3.2}$$

Thus, G and D play the following two-player min-max game in which they minimize respectively maximize their joint value function V:

$$\min_G \max_D V(G, D) = \mathbb{E}_{x \sim p_r}[\log(D(x))] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))] \tag{3.3}$$

3.1.2 Training

The better one player is, the more effective the optimization of the other player will be. Thus, in practice, D and G are not optimized simultaneously, but alternately. Moreover, D typically needs more update steps than G. The following algorithm illustrates GAN training:

Algorithm 1 GAN training

```
Input: batchsize $b$, real data samples $p_r$, learning rate $\alpha$, $D$ updates per step $k$
  for number of training iterations do
      for k steps do
          $\mathbf{z} \leftarrow \{z^{(i)}\}_{i=1}^{b}$; $z^{(i)} \sim p_z$
          $\mathbf{x} \leftarrow \{x^{(i)}\}_{i=1}^{b}$; $x^{(i)} \in p_r$
          $\mathbf{w}_D \leftarrow \mathbf{w}_D + \alpha * \nabla_{\mathbf{w}_D} \log(D(\mathbf{x})) + \log(1 - D(G(\mathbf{z})))$
      end for
      $\mathbf{z} \leftarrow \{z^{(i)}\}_{i=1}^{b}$; $z^{(i)} \sim p_z$
      $\mathbf{w}_G \leftarrow \mathbf{w}_G - \alpha * \nabla_{\mathbf{w}_G} \log(1 - D(G(\mathbf{z})))$
  end for
```

In algorithm 1, for each training iteration, the discriminator is trained for k steps (k typically lies between 1 and 10, as the discriminator needs more updates than the generator). A discriminator training iteration samples a batch of random noise $\mathbf{z}$ and a batch of real samples $\mathbf{x}$. The weight of the discriminator is then updated with a gradient ascend step to recognize real samples as real and generated samples as generated. After k discriminator updates, the generator is trained for a single step. In this single step, a batch of random noise z is sampled, and the weight of the generator is updated with a gradient descend step on $\log(1 - D(G(z)))$. This can be interpreted as using the discriminator as a learned loss function for the generator.

3.1.3 Sampling

Once G and D are trained, G can be used to map samples z from the prior distribution p_z to the data distribution p_g which should be very close to p_r after training.

3.2 Autoencoders

Autoencoders (AEs) are a class of generative model that learn an identity function, with a low-dimensional intermediate representation as bottleneck (latent representation).

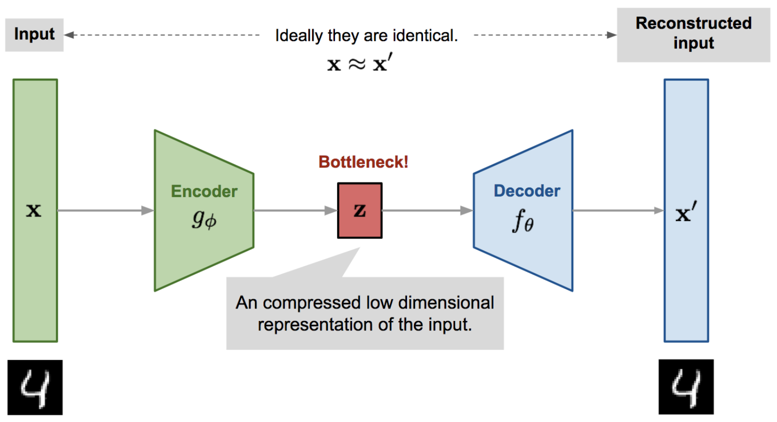

Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
$\mathbf{x}$
Encoder
g_ϕ
Bottleneck!
$\mathbf{z}$
Decoder
f_θ
$\mathbf{x}'$
An compressed low dimensional representation of the input.

Figure 3.1: Autoencoder architecture [Weng, 2018]

This identity function is composed of an encoder that maps real data x into the latent representation z, and a decoder that maps this z back to x', which should be as close to x as possible. AE's origins date back to the 1980s [Weng, 2018]. Initially, AE's primary property of interest was their ability to compress data into their lower dimensional latent space [Hinton and Salakhutdinov, 2006]. With the proposal of Variational Autoencoders (VAEs), Kingma and Welling [2014] drew attention on the generative capabilities of AEs. In VAEs, the bottleneck representation (latent representation) is restricted to follow a known distribution, which enables sampling new latents from the known, random distribution, and mapping them into new, real samples.

Figure 3.1 shows the architecture of an Autoencoder: the encoder g_ϕ compresses samples $\mathbf{x}$ into the latent space $\mathbf{z}$. The decoder f_θ decompresses samples from $\mathbf{z}$ back into x'. The following derivation of the VAE loss function is based on [Weng, 2018].

3.2.1 VAE Loss Function

As previously mentioned, the latent space of VAEs is restricted to follow a known distribution $p_{\boldsymbol{\theta}(\mathbf{z})}$. Probabilistically, the relation between real samples $\mathbf{x}$ and $\mathbf{z}$ can be defined by:

- Prior $p_{\boldsymbol{\theta}}(\mathbf{z})$ (independent of $\mathbf{x}$)
- Likelihood $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ (Probability $\mathbf{x}$ is obtained from the decoder given $\mathbf{z}$)
- Posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ (Probability $\mathbf{z}$ is obtained from the encoder given $\mathbf{x}$)

In inference, new samples are generated by drawing a latent vector from the prior $p_{\boldsymbol{\theta}}(\mathbf{z})$, and then mapping it to the data space using the generative distribution $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$. The optimal parameter $\boldsymbol{\theta}^*$ of the generative distribution maximizes the probability to generate samples from the real distribution:

$$\theta^* = \arg\max_\theta \sum_{i=1}^{n} \log p_\theta(\mathbf{x}^{(i)}) \tag{3.4}$$

$$= \arg\max_\theta \sum_{i=1}^{n} \log \int p_\theta(\mathbf{x}^{(i)}|\mathbf{z}) p_{\boldsymbol{\theta}}(\mathbf{z}) d\mathbf{z} \tag{3.5}$$

As the distribution of $\mathbf{z}$ poses a very large search space (practically not infinite due to limited decimal precision), it is computationally intractable to calculate the integral in equation 3.5. Therefore, the density of the prior $p_{\boldsymbol{\theta}}(\mathbf{z})$ given a sample $\mathbf{x}$ is estimated using the distribution $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$. To enable generation of realistic samples, the estimate should be very close to the real posterior, which amounts to minimizing their Kullback-Leibler divergence $D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x}))$.

$$D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) \tag{3.6}$$

$$= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} d\mathbf{z} \tag{3.7}$$

$$= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x}) p_\theta(\mathbf{x})}{p_\theta(\mathbf{z},\mathbf{x})} d\mathbf{z} \qquad \text{; Because } p(z|x){=}p(z,x)/p(x) \tag{3.8}$$

$$= \int q_\phi(\mathbf{z}|\mathbf{x}) \big(\log p_\theta(\mathbf{x}) + \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z},\mathbf{x})} \big) d\mathbf{z} \tag{3.9}$$

$$= \log p_\theta(\mathbf{x}) + \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z},\mathbf{x})} d\mathbf{z} \qquad \text{; Because } \textstyle\int q(z|x)dz{=}1 \tag{3.10}$$

$$= \log p_\theta(\mathbf{x}) + \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})} d\mathbf{z} \qquad \text{; Because } p(z,x){=}p(x|z)p(z) \tag{3.11}$$

$$= \log p_\theta(\mathbf{x}) + \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z})} - \log p_\theta(\mathbf{x}|\mathbf{z})] \tag{3.12}$$

$$= \log p_\theta(\mathbf{x}) + D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) - \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) \tag{3.13}$$

Maximizing the likelihood of real data while minimizing the divergence between the posterior and its estimate results in minimizing the following equation:

$$\sum_{i=1}^{n} - \log p_\theta(\mathbf{x}^{(i)}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) \| p_\theta(\mathbf{z}|\mathbf{x}^{(i)})) \tag{3.14}$$

$$= \sum_{i=1}^{n} - \log p_\theta(\mathbf{x}^{(i)}) + \log p_\theta(\mathbf{x}^{(i)}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) \| p_\theta(\mathbf{z})) - \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x}^{(i)})} \log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}) \tag{3.15}$$

$$= \sum_{i=1}^{n} -\mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x}^{(i)})} \log p_\theta(\mathbf{x}^{(i)}|\mathbf{z}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) \| p_\theta(\mathbf{z})) \tag{3.16}$$

As all terms in equation 3.16 are tractable, it can directly be used as VAE loss, the optimal encoder parameters θ^*, ϕ^* can be found as follows:

$$L_{\mathrm{VAE}}(\theta, \phi) = - \log p_\theta(\mathbf{x}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x})) \tag{3.17}$$

$$= -\mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z})) \tag{3.18}$$

$$\theta^*, \phi^* = \arg\min_{\theta,\phi} L_{\mathrm{VAE}} \tag{3.19}$$

Note that as a KL-divergence is non-negative, the VAE loss is an evidence lower bound (ELBO):

$$-L_{\mathrm{VAE}} = \log p_\theta(\mathbf{x}) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x})) \leq \log p_\theta(\mathbf{x}) \tag{3.20}$$

The evidence is the likelihood a (generative) model assigns to (real) data. Maximizing the ELBO means maximizing the evidence. The ELBO of VAEs is used to draw a connection between DPMs and VAEs in section 3.4.

3.3 Diffusion Probabilistic Models

Diffusion Probabilistic Models (DDPMs) are a class of generative models that was proposed in 2015 by [Sohl-Dickstein et al., 2015]. They have since matched or raised the state-of-the-art performance in generative problems across various domains [Ho, Jain and Abbeel, 2020; Mittal et al., 2021; Bond-Taylor et al., 2022; Li et al., 2022]. A large part of the success of DDPMs may be owed to their configuration, which allows the training of models counting thousands of layers (timesteps) without suffering from the vanishing gradient problem [Sohl-Dickstein et al., 2015; Bengio, Simard and Frasconi, 1994a]. In addition to their strong generative performance, DDPMs can easily be conditioned on existing data, and be used to complete or extend existing data (in- and outpainting). Their functioning is inspired by nonequilibrium thermodynamics [Sohl-Dickstein et al., 2015; Langevin, 1908].

3.3.1 Theoretical Background

In a forward diffusion process, data from an arbitrary, complex target distribution $q(\mathbf{x}^{(0)})$ is converted into data from a tractable, simple distribution $\pi(\mathbf{y})$. Thereby, $\pi(\mathbf{y})$ is the stationary distribution at the equilibrium of the forward diffusion process. The inversion of this process can be learned, and enables the conversion of data from the tractable, simple distribution $\pi(\mathbf{y})$ into $q(\mathbf{x}^{(0)})$. In practice, $\pi(\mathbf{y})$ is an easily sampleable random distribution like the Gaussian or the binomial distribution. To generate new data from the target distribution $q(\mathbf{x}^{(0)})$, the reverse diffusion process is applied to randomly drawn samples from $\pi(\mathbf{y})$. The remainder of this subsection Theoretical Background is based on [Sohl-Dickstein et al., 2015].

Forward diffusion process

The forward diffusion process repeatedly applies a Markov diffusion kernel $K_\pi(\mathbf{y}|\mathbf{y'};\beta)$, where β, the diffusion rate is a hyperparameter to the process, which controls the size of one diffusion step.

$$\pi(\mathbf{y}) = \int d\mathbf{y'} K_\pi(\mathbf{y}|\mathbf{y'};\beta)\pi(\mathbf{y'}) \tag{3.21}$$

$$q(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)}) = K_\pi(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)};\beta_t) \tag{3.22}$$

$$q(\mathbf{x}^{(0\dots T)}) = q(\mathbf{x}^{(0)}) \prod_{t=1}^{T} q(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}) \tag{3.23}$$

Equation 3.21 illustrates that $\pi(\mathbf{y})$ can be constructed by repeated application of the Markov diffusion kernel, for which an example is given in equation 3.24. Equation 3.22 states that one step in the forward diffusion process corresponds to one application of the Markovian diffusion kernel. Equation 3.23 denotes the density of q after T diffusion steps. The most commonly used type of diffusion kernel in DDPMs is the Gaussian diffusion kernel:

$$K_\pi(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)};\beta_t) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \tag{3.24}$$

Reverse diffusion process

To reverse the diffusion process, a generative distribution $p(\mathbf{x})$ is trained. Once learned, $p(\mathbf{x})$ can be used to convert samples from $\pi(\mathbf{y})$ to samples from $q(\mathbf{x}^{(0)})$.

$$p(\mathbf{x}^{(T)}) = \pi(\mathbf{x}^{(T)}) \tag{3.25}$$

$$p(\mathbf{x}^{(0\dots T)}) = p(\mathbf{x}^{(T)}) \prod_{t=1}^{T} p(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}) \tag{3.26}$$

At diffusion step T, the generative distribution $p(\mathbf{x}^{(T)})$ and $\pi(\mathbf{x}^{(T)})$ are assumed to be congruent (Equation 3.25, provided T and β are chosen sufficiently large).

Given a sufficiently small diffusion rate β, the functional form of the reverse diffusion process is identical to the functional form of the forward diffusion process [Feller, 1949]. This is desirable, as it enables modelling the reverse diffusion process by estimating the parameters of the Markovian diffusion kernel that was used in the forward diffusion process. The larger the number of steps T, the smaller β may be chosen, to ensure convergence of $q(\mathbf{x}^{(0)})$ into $\pi(\mathbf{y})$ after T applications of K.

Model Probability

The probability of data $\mathbf{x}$ under a generative distribution $p(\mathbf{x})$ can be calculated as follows:

$$p(\mathbf{x}^{(0)}) = \int d\mathbf{x}^{(1\dots T)} p(\mathbf{x}^{(0\dots T)}) \tag{3.27}$$

Given only real data $\mathbf{x}^{(0)}$, the integral in equation 3.27 is intractable. Nonetheless, an estimate using the forward diffusion process can be calculated:

$$\begin{aligned}
p(\mathbf{x}^{(0)}) &= \int d\mathbf{x}^{(1\dots T)} p(\mathbf{x}^{(0\dots T)}) \frac{q(\mathbf{x}^{(1\dots T)}|\mathbf{x}^{(0)})}{q(\mathbf{x}^{(1\dots T)}|\mathbf{x}^{(0)})} && (3.28)\\
&= \int d\mathbf{x}^{(1\dots T)} q(\mathbf{x}^{(1\dots T)}|\mathbf{x}^{(0)}) \frac{p(\mathbf{x}^{(0\dots T)})}{q(\mathbf{x}^{(1\dots T)}|\mathbf{x}^{(0)})} && (3.29)\\
&= \int d\mathbf{x}^{(1\dots T)} q(\mathbf{x}^{(1\dots T)}|\mathbf{x}^{(0)}) p(\mathbf{x}^{(T)}) \prod_{t=1}^{T} \frac{p(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})}{q(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)})} && (3.30)
\end{aligned}$$

Note that in equation 3.30, $p(\mathbf{x}^{(T)}|\mathbf{x}^{(T-1)}) = p(\mathbf{x}^{(T)}) = \pi(\mathbf{x}^{(T)})$ (equation 3.25) is assumed.

Training

Training the generative distribution means maximizing the model (log) likelihood L:

$$\begin{aligned}
L &= \int d\mathbf{x}^{(0)} q(\mathbf{x}^{(0)}) \log p(\mathbf{x}^{(0)}) && (3.31)\\
&= \int d\mathbf{x}^{(0)} q(\mathbf{x}^{(0)}) \log \left[\int \mathbf{x}^{(1\dots T)} q(\mathbf{x}^{(1\dots T)}|\mathbf{x}^{(0)}) p(\mathbf{x}^{(T)}) \prod_{t=1}^{T} \frac{p(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})}{q(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)})} \right] && (3.32)\\
&\geq \int d\mathbf{x}^{(0\dots T)} q(\mathbf{x}^{(0\dots T)}) \log \left[p(\mathbf{x}^{(T)}) \prod_{t=1}^{T} \frac{p(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})}{q(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)})} \right] = K && (3.33)
\end{aligned}$$

In equation 3.33, Jensen's inequality is used to derive the lower bound K of the log-likelihood. Given identical functional form of the forward and reverse trajectory (sufficiently small β), the lower bound K becomes tight.

Thus, the optimal generative distribution $\hat{p}(\mathbf{x})$ is the one that maximizes K:

$$\hat{p}(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}) = \underset{\hat{p}(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)})}{\arg\max} \; K \tag{3.34}$$

K can further be decomposed in a tractable sum of KL-Divergences:

$$K = \mathbb{E}_q[\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \parallel p_\theta(\mathbf{x}_T))}_{L_T} + \sum_{t=2}^{T} \underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{- \log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0}] \tag{3.35}$$

3.3.2 Denoising Diffusion Probabilistic Models

If not stated otherwise, equations in this section are based on the blogpost of [Weng, 2021]. In 2020, Ho, Jain and Abbeel [2020] demonstrated Diffusion Probabilistic Model's capability to generate high quality samples. They used a Gaussian Diffusion Probabilistic Model, for which the forward diffusion process is defined as:

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}) \tag{3.36}$$

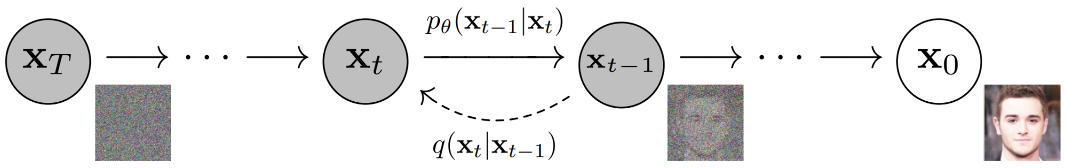

$\mathbf{x}_T$
$\mathbf{x}_t$
$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$
$\mathbf{x}_{t-1}$
$q(\mathbf{x}_t|\mathbf{x}_{t-1})$
$\mathbf{x}_0$

Figure 3.2: The Gaussian Diffusion process [Ho, Jain and Abbeel, 2020]

In the forward diffusion process (figure 3.2), images gradually gets noisier, until they are isotropic Gaussian Noise. For efficient training, the forward diffusion process allows sampling at arbitrary time steps t:

$$\alpha_t := 1 - \beta_t \tag{3.37}$$

$$\bar{\alpha}_t := \prod_{s=1}^{t} a_s \tag{3.38}$$

$$\begin{aligned}
\mathbf{x}_t &= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_{t-1} && \text{;where } \boldsymbol{\epsilon}_{t-1}, \boldsymbol{\epsilon}_{t-2}, \cdots \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) && (3.39)\\
&= \sqrt{\alpha_t \alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\bar{\boldsymbol{\epsilon}}_{t-2} && \text{;where } \bar{\boldsymbol{\epsilon}}_{t-2} \text{ sums two Gaussians} && (3.40)\\
&= \dots && && (3.41)\\
&= \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon} && && (3.42)\\
q(\mathbf{x}_t|\mathbf{x}_0) &= \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}) && && (3.43)
\end{aligned}$$

For the reverse diffusion process, the reverse conditional probability is needed, which is tractable when conditioned on $\mathbf{x}_0$. A detailed derivation can be found in the Appendix in section 7.1 Gaussian reverse transition probabilities. Ultimately, the only unknown value in the reverse transition probabilities is $\boldsymbol{\epsilon}_\theta$, which is the Gaussian noise that was used to distort the original data in the forward diffusion process. This $\boldsymbol{\epsilon}_\theta$ is in practice approximated using a neural network in the reverse diffusion process.

$$\mathbf{x}_{t-1} = \mathcal{N}(\mathbf{x}_{t-1}; \frac{1}{\sqrt{\alpha_t}}\Big(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\Big), \tilde{\beta}_t) \tag{3.44}$$

Training Loss

As during training, $\mathbf{x}_0$ is known, the MSE loss between samples from $\tilde{\boldsymbol{\mu}}(\mathbf{x}_t, \mathbf{x}_0)$ and $\tilde{\boldsymbol{\mu}}(\mathbf{x}_t)$ can be inferred.

$$\begin{aligned}
L_t &= \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}}\Big[\frac{1}{2\|\tilde{\beta}_t\|_2^2}\|\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) - \boldsymbol{\mu}_\theta(\mathbf{x}_t, t)\|^2\Big] && (3.45)\\
&= \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}}\Big[\frac{1}{2\|\tilde{\beta}_t\|_2^2}\|\frac{1}{\sqrt{\alpha_t}}\Big(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_t\Big) - \frac{1}{\sqrt{\alpha_t}}\Big(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\Big)\|^2\Big] && (3.46)\\
&= \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}}\Big[\frac{(1-\alpha_t)^2}{2\alpha_t(1-\bar{\alpha}_t)\|\tilde{\beta}_t\|_2^2}\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2\Big] && (3.47)\\
&= \mathbb{E}_{\mathbf{x}_0, \boldsymbol{\epsilon}}\Big[\frac{(1-\alpha_t)^2}{2\alpha_t(1-\bar{\alpha}_t)\|\tilde{\beta}_t\|_2^2}\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t, t)\|^2\Big] && (3.48)
\end{aligned}$$

Intuitively, this can be interpreted as distorting an image with random noise $\boldsymbol{\epsilon}_t$, and letting the neural network predict this $\boldsymbol{\epsilon}_t$ (weighed by a factor). Ho, Jain and Abbeel [2020] empirically found that a simplified loss ignoring the weighing factor yields even better results:

$$L_t^{\text{simple}} = \mathbb{E}_{t\sim[1,T],\mathbf{x}_0,\boldsymbol{\epsilon}_t}\Big[\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2\Big] \tag{3.49}$$

$$= \mathbb{E}_{t\sim[1,T],\mathbf{x}_0,\boldsymbol{\epsilon}_t}\Big[\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t, t)\|^2\Big] \tag{3.50}$$

Training and Sampling Algorithms

Ho, Jain and Abbeel [2020] formulate the following training and sampling algorithms:

Algorithm 2 Diffusion model training

```
Input: $q(x_0), T$ steps, noise schedule $\beta_1, ..., \beta_N$
  repeat
    $x_0 \sim q(x_0)$
    $t \sim \mathbb{U}(\{1, ..., T\})$
    $\sqrt{\bar{\alpha}} \sim \mathbb{U}(\sqrt{\bar{\alpha}_{t-1}}, \sqrt{\bar{\alpha}_t})$
    $\epsilon \sim \mathcal{N}(0, I)$
    Take gradient descent step on
    $\nabla_\theta \|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, \sqrt{\bar{\alpha}})\|^2$
  until converged
```

Algorithm 2 illustrates Diffusion Model training: random terms L_t from the variational bound K are sampled, and optimized using L_t^{simple} (equation 3.50).

Algorithm 3 Sampling

```
Input: $T$ steps, noise schedule $\beta_1, ..., \beta_N$
  $x_T \sim \mathcal{N}(0, I)$

  for t = N, ..., 1 do
    $\boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$ if $t > 1$ else $\boldsymbol{\epsilon} = 0$
    $x_{t-1} = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(x_t, \sqrt{\bar{\alpha}_t})) + \beta_t\boldsymbol{\epsilon}$
  end for
  return $x_0$
```

Algorithm 6 illustrates Diffusion Model sampling: $\mathbf{x}_T$ is sampled from a random normal distribution (equation 3.25). Then the generative distribution $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is used to iteratively infer $\mathbf{x}_{t-1}$ from $\mathbf{x}_t$ using equation 3.44.

3.3.3 Discrete Diffusion Probabilistic Models

Sohl-Dickstein et al. [2015] already formulated a version of Discrete Diffusion Probabilistic Models which supports binomial distributions. Hoogeboom et al. [2021] extended this definition to categorical distributions. Austin et al. [2021] later proposed a more general framework with four different flavors of Discrete Diffusion Probabilistic Models, which contained all previous definitions of D3PMs as special cases and is described in this section.

Like in the Gaussian DDPM version, a forward diffusion process $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ is required. For scalar discrete random variables $x_t \in \{1, ..., K\}$, the forward transition probabilities can be expressed using matrices: $q(x_t = j|x_{t-1} = i) = [\mathbf{Q}_t]_{ij}$. Given x_t in one-hot notation, the forward diffusion process can be written as:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \text{Cat}(\mathbf{x}_t; \mathbf{p} = \mathbf{x}_{t-1}\mathbf{Q}_t) \tag{3.51}$$

Where $\text{Cat}(\mathbf{x}, \mathbf{p})$ is a categorical distribution, over the one-hot vector $\mathbf{x}$, and probabilities are given by the vector $\mathbf{p}$. $\mathbf{x}_{t-1}\mathbf{Q}_t$ denotes the (one-hot) vector-matrix product, and can be viewed as selecting a specific column from the transition matrix.

Like in Gaussian Diffusion, x_t can be sampled from x_0 in closed form in a single step for efficient training:

$$q(\mathbf{x}_t|\mathbf{x}_0) = \text{Cat}(\mathbf{x}_t; \mathbf{p} = \mathbf{x}_0\bar{\mathbf{Q}}_t) \qquad ,\text{where } \bar{\mathbf{Q}}_t = \prod_{i=1}^{t} \mathbf{Q}_i \tag{3.52}$$

Furthermore, the reverse conditional probability can as well be expressed using the forward diffusion distribution and Bayes' theorem:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0)q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)} = \text{Cat}\Big(\mathbf{x}_{t-1}; \mathbf{p} = \frac{\mathbf{x}_t\mathbf{Q}_t^\top \circ \mathbf{x}_0\bar{\mathbf{Q}}_{t-1}}{\mathbf{x}_0\bar{\mathbf{Q}}_t\mathbf{x}_t^\top}\Big) \tag{3.53}$$

As the forward diffusion process is defined as a Markov chain, $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = q(\mathbf{x}_{t-1}|\mathbf{x}_t)$. In contrast to the continuous case, where only Gaussian forward diffusion was considered in recent works, in the discrete case the corruption process can be controlled by the choice of transition matrices $\mathbf{Q}_t$. The only restrictions on $\mathbf{Q}_t$ are that its rows must sum to 1 to preserve probability mass, and that the rows of $\bar{\mathbf{Q}}_t$ must converge to a known stationary distribution (otherwise equation 3.25 could not be applied). Domain specific knowledge can be embedded in $\mathbf{Q}_t$, resulting in different flavors of Discrete Diffusion:

- **Uniform**: Each step t, each random variable either stays in its state with probability $1 - \beta_t$, or transitions to any other state with probability β_t/K ($\beta_t \in [0, 1]$).

$$\mathbf{Q}_t = \begin{bmatrix} (1-\beta_t) & ... & \beta_t/K \\ ... & ... & ... \\ \beta_t/K... & (1-\beta_t) & ...\beta_t/K \\ ... & ... & ... \\ \beta_t/K & ... & (1-\beta_t) \end{bmatrix} \tag{3.54}$$

  Since this matrix is doubly stochastic (all entries positive, real valued, rows and columns sum to 1), $\bar{\mathbf{Q}}_T$ converges toward a uniform distribution, given enough steps T and sufficiently large β_t.

- **Absorbing state** Motivated by the success of BERT [Devlin et al., 2019] and Conditional Masked Language Models, Austin et al. [2021] introduced the Absorbing state Diffusion: an additional state that does not occur in the domain, but indicates corruption of data is introduced (similar to the [MASK] token in BERT). Each step t, each random variable either stays in its state with probability $1 - \beta_t$, or transitions to the Absorbing state with probability β_t.

  $$\mathbf{Q}_t = \begin{bmatrix} (1-\beta_t) & ... & \beta_t \\ ... & ... & ... \\ 0... & (1-\beta_t) & ...\beta_t \\ ... & ... & ... \\ 0 & ... & 1 \end{bmatrix} \tag{3.55}$$

  In this case, $\bar{\mathbf{Q}}_T$ has all the mass on the Absorbing state, given enough steps and sufficiently large β_t, $\mathbf{x}_T$ converges towards a distribution consisting of only [MASK] tokens.

  I use this flavor of Discrete Diffusion in my work, as it allows for flexible inpainting. Of a given set of variables $\mathbf{x}_t$, any can be transitioned to the Absorbing state, and unmasked by the generative distribution.

- **Discretized Gaussian**: Transition probabilities to other states are sampled from a Gaussian PDF. The farther apart the numbers assigned to two states are, the lower the probability a random variable will transition between them. This Diffusion flavor is suitable for ordinal states, where closer states are more similar to each other.

- **Token embedding distance**: This type of transition matrix enables the embedding of semantic relationships into the transition matrix. In text generation problems for example, vowels may be closer to each other than vowels to consonants. In symbolic music generation, the same notes of different octaves might be closer to each other than different notes of different octaves, and notes in the same octave might also be closer in the transition matrix than notes in different octaves.

  Nonetheless, I did not choose this type of transition matrix, because it does not permit as flexible generation as the Absorbing state model.

Loss function

Different parameterizations for the training loss of D3PMs exist. In this section, the formulation of Bond-Taylor et al. [2022], which I also use in my experiments, is described.

In continuous DDPMs, parameterizations which only require an approximation of $\boldsymbol{\epsilon}$ exist (see section 3.3.2). For the discrete case, similar parameterizations are not known [Bond-Taylor et al., 2022]. However, Austin et al. [2021] propose an auxilary loss using the $\mathbf{x}_0$ parameterization:

$$L_\lambda = L_{vb} + \lambda \mathbb{E}_{q(\mathbf{x}_0)} \mathbb{E}_{q(\mathbf{x}_t|\mathbf{x}_0)}[-\log p_{\boldsymbol{\theta}}(\mathbf{x}_0|\mathbf{x}_t)] \tag{3.56}$$

The minimum of this additional term coincides with the minimum of L_{vb}, both are minimized if $p_{\boldsymbol{\theta}}(\mathbf{x}_0|\mathbf{x}_t)$ places all its mass on $\mathbf{x}_0$.

Bond-Taylor et al. [2022] found that directly predicting $\mathbf{x}_0$ instead of $\mathbf{x}_{t-1}$ is very suitable for the Absorbing state Diffusion model. This seems plausible, as once a random variable is unmasked, it stays in its state in the reverse diffusion process. In a forward step of their ASD3PM, tokens are masked with probability $\frac{t}{T}$ The reparameterization for predicting $\mathbf{x}_0$ reduces the variational bound to:

$$L_{VB} = \mathbb{E}_{q(\boldsymbol{x}_0)}\left[\sum_{t=1}^{T} \frac{1}{t} \mathbb{E}_{q(\boldsymbol{x}_t|\boldsymbol{x}_0)}\left[\sum_{[\boldsymbol{x}_t]_i = m} \log p_\theta\left([\boldsymbol{x}_0]_i \mid \boldsymbol{x}_t\right)\right]\right] \tag{3.57}$$

,which coincides with the cross entropy between $q(\mathbf{x}_0)$ and $p(q(\mathbf{x}_{0:T}))$. The inner sum in equation 3.58 forms the joint log-probability of all random variables in $\mathbf{x}$, while the weighing with $\frac{1}{t}$ compensates the number of masks in a specific $\mathbf{x}_t$ (number of masks is proportional to t). Bond-Taylor et al. [2022] further state that the $\boldsymbol{\epsilon}$ parameterization in the continuous case allows to easily minimize the loss at steps close to T. To mimic this, Bond-Taylor et al. [2022] reweigh the loss ELBO loss as follows:

$$L_{VB} = \mathbb{E}_{q(\boldsymbol{x}_0)}\left[\sum_{t=1}^{T} \frac{T-t-1}{T} \mathbb{E}_{q(\boldsymbol{x}_t|\boldsymbol{x}_0)}\left[\sum_{[\boldsymbol{x}_t]_i = m} \log p_\theta\left([\boldsymbol{x}_0]_i \mid \boldsymbol{x}_t\right)\right]\right] \tag{3.58}$$

3.4 Connections between DPMs and other Generative Models

The previously listed generative models share one central commonality: to generate, they map randomly sampled data (seed) into the space of realistic samples. Using deterministic algorithms, generative models need a form of a random seed to be able to generate different samples. The random seeds of both VAEs (latent representation) and DDPMs (x_T) are constrained to follow a certain distribution (Gaussian is the most popular for both cases). Considering all T forward diffusion steps of a DDPM as one joint encoding step and all reverse diffusion steps as a joint decoding step, DPMs can be seen as Autoencoders. The latent dimension of DPMs is equal to their data dimension, and all their trainable parameters lie in their decoder. The x_0-parameterized Absorbing state DPM-ELBO can even be derived using the VAE-ELBO:

$$\begin{aligned}
\text{ELBO}_{\text{VAE}} &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) - \overbrace{D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}))}^{=0} && (3.59)\\
&= \mathbb{E}_{q(\boldsymbol{x}_0)}\mathbb{E}_{\mathbf{x}_T\sim q(\mathbf{x}_T|\mathbf{x}_0)} \log p_\theta(\mathbf{x}_0|\mathbf{x}_T) && (3.60)\\
&= \mathbb{E}_{q(\boldsymbol{x}_0)}\mathbb{E}_{\mathbf{x}_T\sim q(\mathbf{x}_T|\mathbf{x}_0)} \sum_{t=T}^{1} \log p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) && (3.61)\\
&= \mathbb{E}_{q(\boldsymbol{x}_0)}\mathbb{E}_{\mathbf{x}_T\sim q(\mathbf{x}_T|\mathbf{x}_0)} \sum_{t=T}^{1} \frac{1}{t} \log p_\theta(\mathbf{x}_0|\mathbf{x}_t) && (3.62)\\
&= \mathbb{E}_{q(\boldsymbol{x}_0)} \sum_{t=1}^{T} \mathbb{E}_{\mathbf{x}_t\sim q(\mathbf{x}_t|\mathbf{x}_0)} \frac{1}{t} \log p_\theta(\mathbf{x}_0|\mathbf{x}_t) \qquad = \text{ELBO}_{\text{ASD3PM}} && (3.63)
\end{aligned}$$

In equation 3.59, the D_{KL} between the latent distribution and the prior is assumed to be 0, as this has to be asserted by the choice of diffusion rate β_t and number of diffusion steps T (and q_ϕ has no trainable parameters). In equation 3.62, the joint probability between all possible x_{t-1} instead only one particular is taken (see section 3.3.3). In equation 3.63, x_t from the forward diffusion process are assumed to be equally distributed as x_t from the reverse diffusion process. This assumption is fundamental in DPMs, and enables the independent training of different reverse diffusion steps.

Although they are not listed as generative models in this chapter, I point out similarities between autoregressive models and Absorbing state diffusion probabilistic models: Austin et al. [2021] showed that fixing the unmasking order of an ASD3PM to a linear one yields exactly the cross entropy loss of autoregressive formulations. The key difference that is not mentioned in [Austin et al., 2021] is that autoregressive models allow for variable sequence lengths. DPMs strength lies in their joint modelling of the whole output sequence with the drawback of requiring a fixed sequence length.

Austin et al. [2021] state that BERT Devlin et al. [2019] is a one-step diffusion model: from a sequence of tokens, a fixed portion is masked out (forward diffusion) and restored (reverse diffusion) in one step. This equivalence might not hold very strictly, as the partially masked sequence does not follow a distribution, from which sampling without real data is possible.

GANs could be viewed as Diffusion Models respectively Autoencoders with an encoder ignoring the input, and a decoder maximizing the likelihood that its output matches any real sample, but this is not the reason it is listed in this section. Dhariwal and Nichol [2021] showed that DDPMs' sample quality can be improved using a classifier to guide their sampling process.

Employing the idea of the adversarial loss and domain specific knowledge, a classifier that distinguishes real from generated samples could be trained. This classifier could be used to guide the sampling process of DPMs towards more realistic samples, or even provide an auxiliary loss during DPM training.

4 Discrete Diffusion Model Implementation and Experimental Setup

This chapter describes the methods and the structure of the experiments performed in this work. The key difference to [Mittal et al., 2021] is that the diffusion model is directly applied to the sequence of discrete tokens, instead of encoding those tokens into the continuous domain using an autoencoder first, and then modelling the Autoencoders' latents using the DDPM. The proposed generative model is trained on melody only and trio pieces consisting of a melody, a bass and a drum track.

4.1 Dataset

I used the Lakh MIDI dataset [Raffel, 2016], as to my best knowledge it is still the largest publicly available dataset of symbolic music. The dataset consists of about 175,000 unique midi files, of which most are polyphonic pop pieces.

4.2 Preprocessing & Encoding

The preprocessing of the dataset is very similar to the preprocessing of [Mittal et al., 2021; Roberts et al., n.d.]: Onsets and durations are quantized to 16^{ths}. The proposed model only supports fixed size data during training. As Mittal et al. [2021] provide evaluations for pieces with a length of 64 bars, I assume the same length of pieces for comparability. This results in a total length of $16 * 64 = 1024$ events per track of training data. Shorter pieces are discarded, while longer pieces can be cropped at different positions, which enables the extraction of multiple training tracks. In the melody only setting, tracks with suitable instruments (MIDI program numbers 0-32) are extracted from all pieces. As the models' in- and output only supports one event per time, polyphonic tracks are monophonized by considering only the highest pitch if multiple notes are played at the same time in the unprocessed MIDI files. A preprocessed melody consists of 1024 discrete events, of which each has one of 90 types:

- **0**: continue note, can be pause if preceded by note-off
- **1**: note off
- **2-89**: note on with MIDI pitch 21-108

For the trio setting, a bass and a drum track are added, resulting in a tensor of $[1024, 3]$ events. The bass track is handled like the melody track, but only tracks with suitable bass instruments (MIDI program numbers 32-40) are considered. The drum pattern track is treated differently: MIDI events, in addition to pitch also have an optional *is drum* property. If this property is set, the program number is ignored, and the pitch encodes the type of drum. To reduce the number of drum types several drum MIDI pitches were condensed to of of 9 drum types:

Instrument	MIDI pitches
kick drum	36
snare drum	37, 38, 40
closed hi-hat	22, 42, 44
open hi-hat	26, 46
low tom	43, 58
mid tom	45, 47
high tom	48, 50
crash cymbal	49, 52, 55, 57
ride cymbal	51, 53, 59

A preprocessed drum track event supports beats of all of the encoded drums simultaneously, resulting in $2^9 = 512$ event types per time. For trio extraction, pieces with at least one suitable track for each melody, bass and drums with sufficient length are required. Pieces with multiple suitable tracks for either melody, bass or drums enable the extraction of multiple trios (cross product).

4.3 Absorbing State Discrete Denoising Diffusion Probabilistic Model

In my work, I use an Absorbing State Discrete Denoising Diffusion Probabilistic Model (ASD3PM), which is very similar to the model Bond-Taylor et al. [2022] use in their work. The model introduces an additional event type for each track: the Absorbing State. This additional event does not translate to a musical event, but indicates that data is missing at the position of the event, similar to the [MASK] token in BERT models [Devlin et al., 2019].

4.3.1 Sampling

The ASD3PM uses an unmasking function (denoising function in the continuous case) modelled by a neural network. This unmasking function takes a (partially) masked sequence of musical events, and predicts probabilities of each event at each position in the unmasked sequence. To sample unconditionally, the ASD3PM takes a fully masked sequence (tensor consisting only of mask events) as input. Although a whole unmasked sequence is predicted by the unmasking function in one single unmasking step, sampling is performed in T steps. In each sampling step t, each mask token is unveiled with probability $\frac{t}{T}$. This way, each sampling step is conditioned on a larger number of unmasked notes than the previous one. Algorithm 4 illustrates this sampling process:

Algorithm 4 ASD3PM Sampling

```
Input: number of sample steps $T$, (partially) masked tensor $x_t$
  for t in (T..1) do
      $x_0 \leftarrow$ unmask_fn($x_t$)
      for event position $p$ in $x_t$ do
          if $x_t[p]$ is mask and uniform_random(0, 1) < $\frac{1}{t}$ then
              $x_t[p] \leftarrow x_0[p]$
          end if
      end for
  end for
```

The ASD3PM allows for flexible conditioning on existing pieces: known events can be combined with mask events arbitrarily. Some of the possible generation scenarios are:

- **unconditional**: all input events are set to mask initially
- **outpainting**: only the start (or the end) of a piece is provided, the rest is filled with masks
- **inpainting**: a musical piece where a subsequence is set to mask
- **interpolation**: two different musical pieces, a subsequence between them is set to mask
- **accompaniment**: one or more tracks are provided, the rest is set to mask

4.4 Training

During training, the forward diffusion process samples $x_t|x_0$. This is achieved by setting each event in the original piece to mask with probability t/T. The masking probability is closely related to the diffusion rate β_t in the continuous case. All trainable parameters of the reverse diffusion process are in the unmasking function. As already mentioned in the sampling section (figure 4.3.1), the unmasking function takes a (partially) masked sequence of musical events, and predicts probabilities of each event at each position in the unmasked sequence. This enables the use of the cross entropy loss for each event that was masked in the forward diffusion process (see section 3.3.3). Bond-Taylor et al. [2022] empirically find that a reweighting of this cross entropy loss with $1 - \frac{t}{T}$ improves results. Intuitively this reweighting corresponds to assigning a higher importance to reconstructions that have less masks in their input. Algorithm 5 illustrates a training epoch of the ASD3PM.

Algorithm 5 ASD3PM Training Epoch

The q_sample function takes a batch of $\mathbf{x}_0$ and a timestep t for each x_0 and masks out each event in each x_0 with probability $\frac{t}{T}$

The masked_cross_entropy masks out all positions where $\mathbf{x}_t$ is not masked

```
Input: number of diffusion steps $T$, batches of training data $X$
  **for** batch $\mathbf{x}_0 \in X$ **do**
      $\mathbf{t} \leftarrow \{t^{(i)}\}_{i=0}^{|\mathbf{x}_0|}; t^{(i)} \in [1, T]$
      $\mathbf{x}_t \leftarrow$ q_sample$(\mathbf{x}_0, \mathbf{t})$
      $\hat{\mathbf{x}_0} \leftarrow$ unmasking_fn$_\theta(\mathbf{x}_t)$
      **loss** $\leftarrow \frac{\mathbf{t}}{T} \cdot$ masked_cross_entropy$(\hat{\mathbf{x}_0}, \mathbf{x}_0, \mathbf{x}_t)$
      gradient descent step on $\nabla_\theta$**loss**
  **end for**
```

4.5 Classifier Guided Sampling

Dhariwal and Nichol [2021] show that the sampling process of continuous DDPMs can be guided towards samples of a desired class using gradients inferred from a classifier. In their approach, the classifier needs to be trained on noisy intermediate samples to enable a provision of reliable gradients. The $\mathbf{x}_0$ parametrization circumvents this necessity, and allows using an off-the-shelf classifier to guide the sampling process:

Algorithm 6 Guided Absorbing State D3PM Sampling

```
**Input:** $T$ steps, unmasking function $f_\theta(x_t)$
differentiable loss function $l_\phi(x_0)$, guidance scale $s$
  $x_T \leftarrow$ [MASK]
  **for** t = T, ..., 1 **do**
      $x_{0\text{logits}} = f_\theta(x_T)$
      $x_{0\text{probs}} = \text{softmax}(x_{0\text{logits}})$
      $loss = l_\phi(x_{0\text{probs}})$
      $x_{0\text{probs}} -= \nabla_{x0\text{probs}} loss \cdot s$
      $x_{0\text{cond}} = \text{Cat}(x_{0\text{probs}})$.sample()
      with probability $\frac{1}{t}$replace masks in $x_T$with $x_{0\text{cond}}$
  **end for**
  **return** $x_T$
```

Algorithm 6 is a modified version of the ASD3PM sampling algorithm (figure 4). The key difference is that after probabilities for x_0 are predicted by the neural network, they are modified with a gradient descend step on the provided loss function. Intuitively, this can be understood as letting the ASD3PM predict probabilites of $\mathbf{x}_0$ each step and modifying them so the loss function is minimized. The loss function in algorithm 6 could for example be a classifier on note density combined with the cross entropy loss for the desired note density class.

4.6 Model Architectures

Different model architectures were explored as unmasking function in the ASD3PM. The architectures were inspired by the neural network Bond-Taylor et al. [2022] use in their work.

4.6.1 Transformer

This model architecture serves as baseline, and is almost identical to the model Bond-Taylor et al. [2022] use in their work, and only differs in the number of different embeddings supported. A 24 layer Transformer similar to GPT-2 medium [Radford et al., 2019] operates on embeddings of musical events.

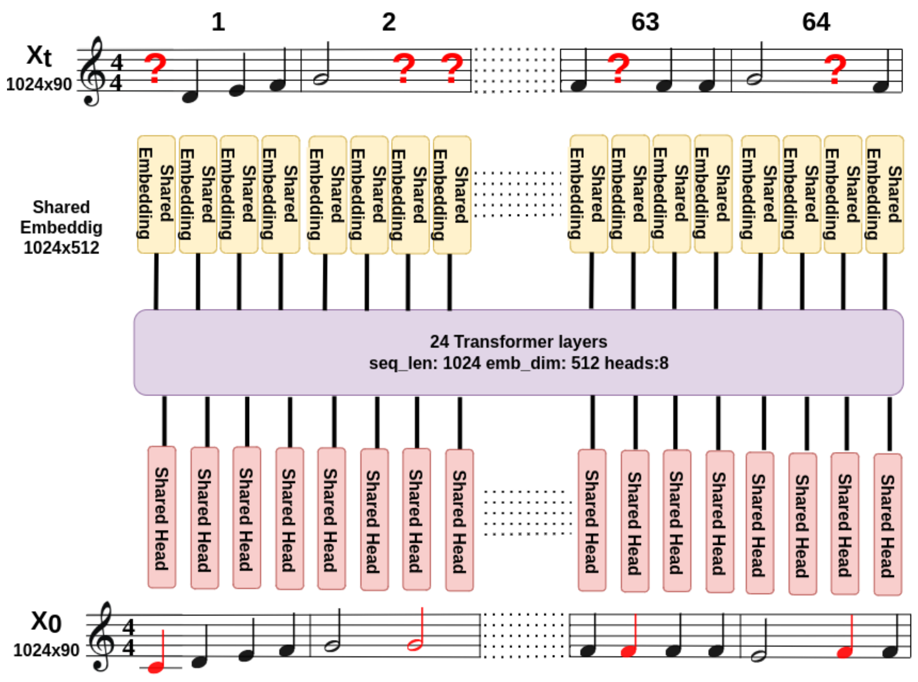


Figure 4.1: Transformer architecture

4.6.2 Convolutional Transformer (SCHmUBERT)

I named this model architecture SCHmUBERT (Symbolic Creative Hierarchical music Unmasking Bidirectional Encoder Representation Transformer). As depicted in figure 4.2, each musical event is embedded into a vector of size 128 separately. In the case of melody and bass track, this means an index in the range $[0, 90]$ (88 pitches + note off + continue + mask) is mapped to a vector of 128 continuous values, in the drum pattern track, vectors of the same dimensionality are indexed with values in the range $[0, 512]$ ($2^9 = 512$ patterns + mask). Following the intuition that most notes occuring in the dataset are in fact longer than a 16^{th}, after embedding, the embeddings of 4 consecutive

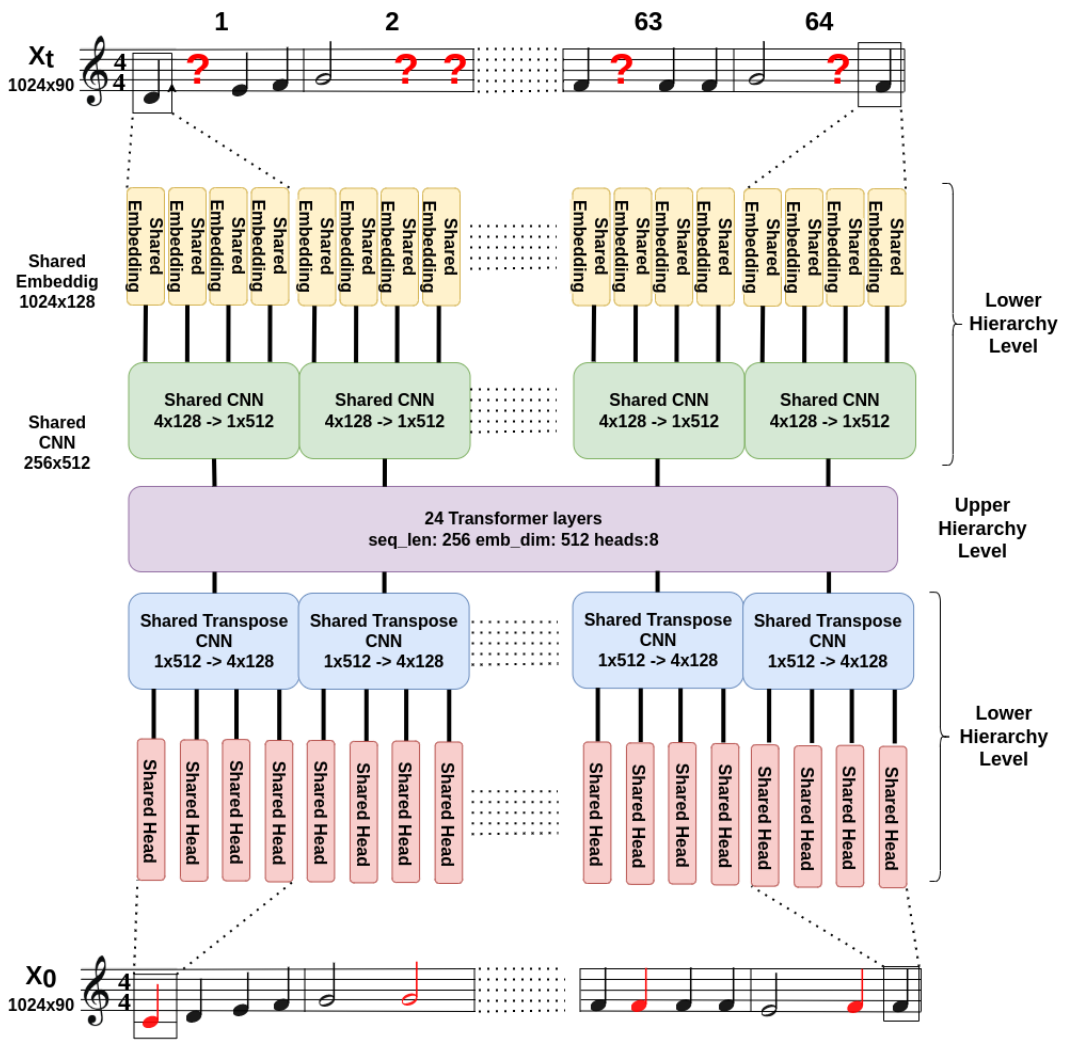


Figure 4.2: SCHmUBERT Melody Architecture
The staves in the upper half represent the model's partially masked input, question marks symbolize [MASK] tokens. Actually, all depicted notes are formed by a concatenation of 16th notes. The unmasking function reconstructs all notes, but in the sampling process, only some of the uncovered notes are transferred to X_{t-1}

events are summarized into a single embedding by a shared 1-D convolution. As the dimension of the combined embeddings is 4 times as large (= 512) as the single embeddings (128), a lossless compression should be possible. Although the summarization using the CNN does not reduce the amount of data the transformer stack receives, load is redistributed from the sequence length to the embedding dimension. This boosts training and sampling performance, as the transformer's asymptotic runtime grows with $O(n^2d+d^2n)$ it is desireable to keep d and n close together (self-attention has runtime complexity $O(n^2d)$ [Bengio, Simard and Frasconi, 1994b], n feedforward operations $O(d^2n)$). In the experiments conducted, I found that this summarizing does not have a negative impact on the model's fitting capacity.

4.6.3 Hierarchical Transformer

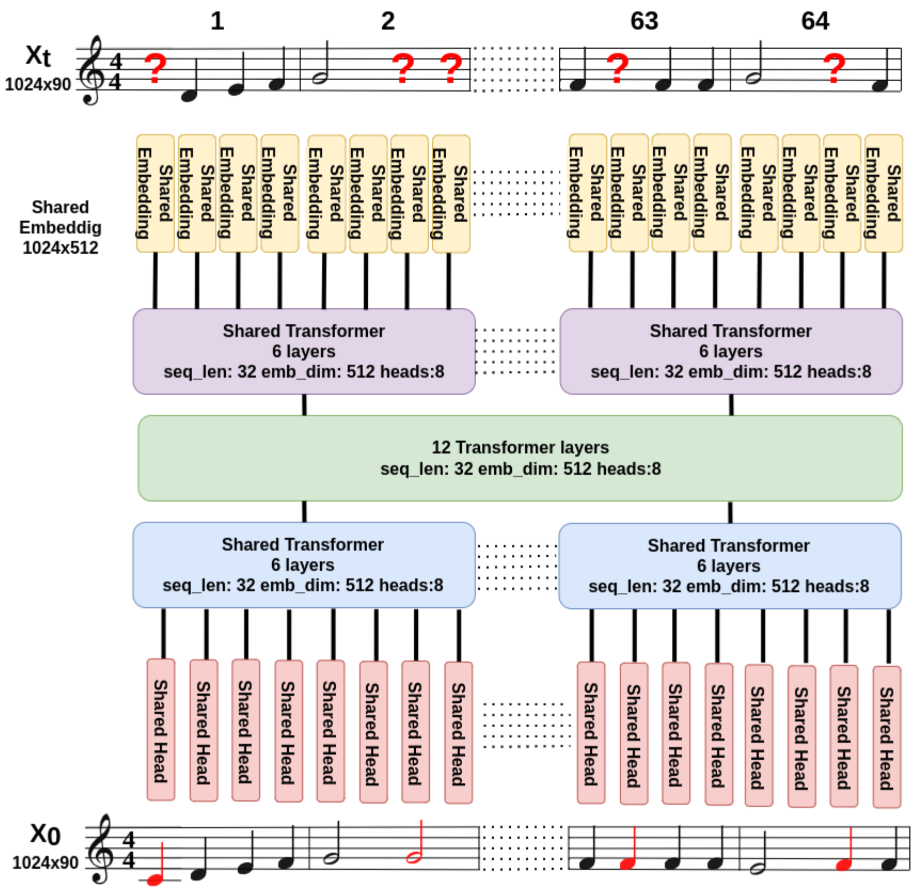


Figure 4.3: Hierarchical Transformer architecture

Figure 4.3 shows the hierarchical transformer model architecture. Like in the previous models, each musical event is embedded into a vector separately. The resulting sequence of embeddings is then divided into 32 non-overlapping subsequences with length 32. A shared lower hierarchy level transformer (purple) then summarizes each subsequence into a single embedding. This summarization results in a sequence of 32 higher hierarchy level embeddings, which is processed by the higher hierarchy level transformer (green). Afterwards, a shared lower hierarchy level transformer (blue) splits each of the 32 higher hierarchy level vectors into 32 vectors, resulting in a sequence of embeddings with the original length (1024). Again, a shared head maps each embedding to output probabilities.

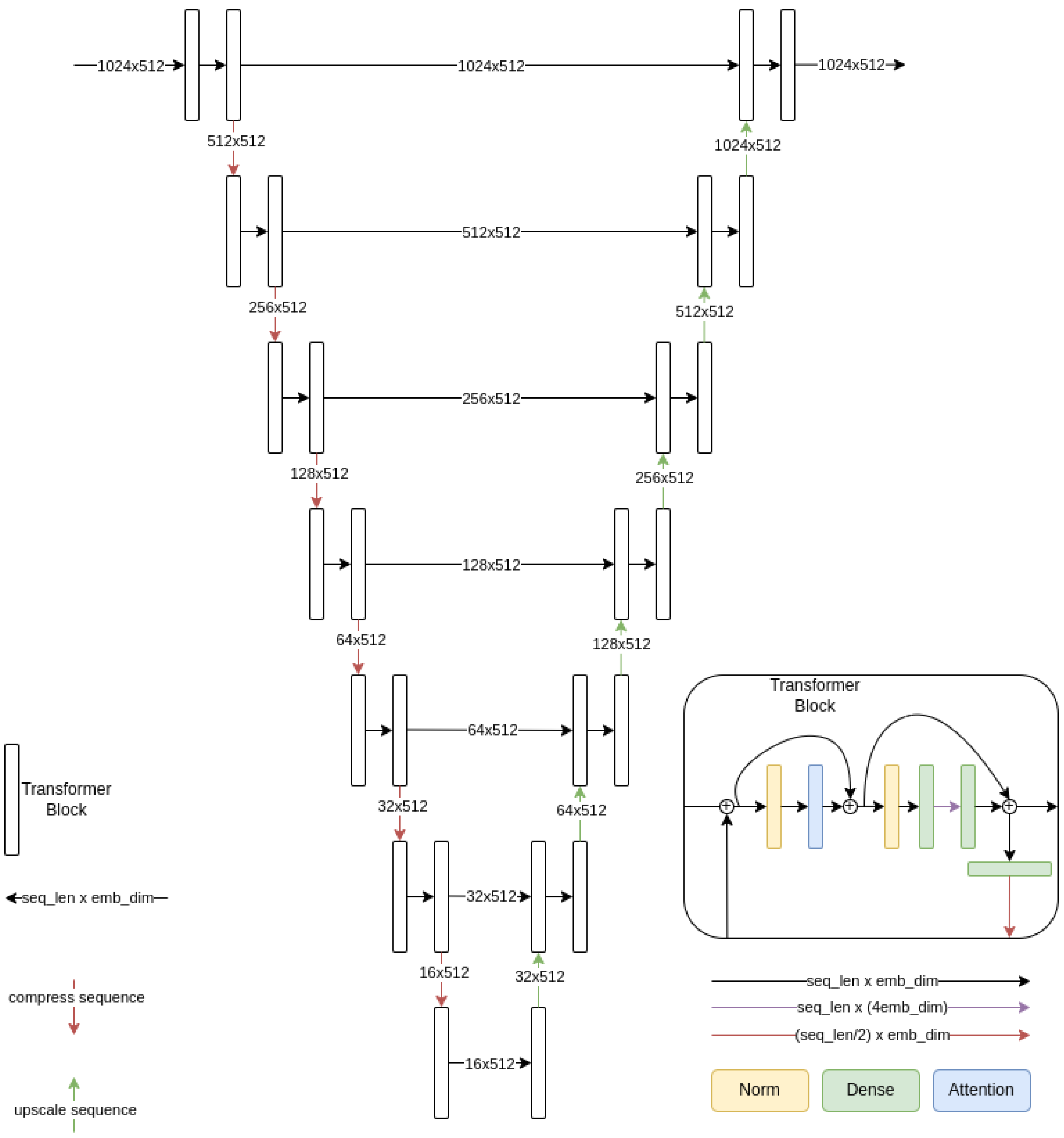


Figure 4.4: UTransformer architecture

4.6.4 U Transformer

Figure 4.4 shows the U Transformer model architecture. Like in the previous models, the sequence of discrete values is first embedded into a vector. To preserve space for the rest of the model, this part is left out in figure 4.4. The downscale operations (left half of the U) in the following are carried out recursively, and aim to process the sequence of embeddings with increasing levels of abstraction. After the previous block, the sequence passes through 2 consecutive transformer blocks. The resulting sequence of embeddings is reduced to half its length (red arrows) using a perceptron (Dense) layer, it is processed by the next level. After the original sequence of length 1024 is reduced to only 16, each embedding vector represents one bar in the musical piece. Starting from the bottom of the U, the sequence is scaled up to its original length and processed by two consecutive transformer blocks in each level recursively. Again, the length of the sequence is scaled up using a perceptron layer. In addition to the scaled up embeddings of the previous level, the transformer blocks in the right half of the U also receive the output sequence of the transformer blocks from the left half of the U using skip connections. The architecture of the U Transformer is inspired by the UNet of Ronneberger, Fischer and Brox [2015], where CNNs instead of transformers are used as blocks to form the U.

4.7 Evaluation

Generated pieces should be similar, but not identical to training pieces. A possibility of assessing the similarity between generated and real pieces is a comparison of abstracted statistical attributes. For evaluation, the framewise self-similarity metric from Mittal et al. [2021] is used in this work. This metric captures local self-similarity patterns. Pieces are divided into 32 windows, and the similarity between those windows is used as a statistical indicator. The similarity between adjacent windows is based on Overlapping Areas of Gaussian density functions. These Gaussian density functions describe the distributions of pitch and duration in 4-bar windows. If two adjacent windows contain the exact same notes, the mean and variance of their pitches (durations) are equal, thus the Overlap Area between them is 1. With a hop size of 2 bars, the mean and variance of pitch and duration of notes in each 4-bar window is computed. Thus, a Gaussian for pitch and variance can be defined for each window w: $\mathcal{N}(\mu_{P_w}, \sigma^2_{P_w}), \mathcal{N}(\mu_{D_w}, \sigma^2_{D_w})$. The Overlapping Area for pitch respectively duration of adjacent frames can then be calculated as follows:

$$OA(w, w+1) = 1 - \frac{1}{2}\mathrm{erf}\Big(\frac{c - \mu_w}{\sqrt{2}\sigma^2_w}\Big) + \frac{1}{2}\mathrm{erf}\Big(\frac{c - \mu_{w+1}}{\sqrt{2}\sigma^2_{w+1}}\Big) \tag{4.1}$$

erf denotes the Gaussian error function, c the point of intersection between the two Gaussian PDFs with $\mu_w < \mu_{w+1}$. The OA metric is symmetric, therefore its arguments can be sorted with ascending μ. Aggregated over all adjacent frames of a set of musical pieces, μ_{OA} and σ^2_{OA} can be inferred for pitch and duration. This μ_{OA} and σ^2_{OA} of generated pieces is then compared to mean and variance of Overlap Areas of real pieces (μ_{GT} and σ^2_{GT}) from the dataset. Based on these values, the relative normalized similarities *Consistency* and *Variance* are calculated as follows:

$$Consistency = \max(0, 1 - \frac{|\mu_{OA} - \mu_{GT}|}{\mu_{GT}}) \tag{4.2}$$

$$Variance = \max(0, 1 - \frac{|\sigma^2_{OA} - \sigma^2_{GT}|}{\sigma^2_{GT}}) \tag{4.3}$$

An intuitive interpretation for consistency is, how similar adjacent windows of pieces are on average. Variance denotes the variance of this consistency, and can be interpreted as a rating of how different pieces are to each other (based on their consistency). Like in Mittal et al. [2021], the self similarity metric is evaluated for sets of 1000 musical pieces in this work. Note that while the self-similarity metric is suitable for measuring statistical similarity, it does not capture any musical aspects.

A simple algorithm could produce pieces that score very high in this metric:

Algorithm 7 Perfect self-similarity sampling

```
Input: target overlap area mean $\mu$, target overlap area variance $\sigma^2$
Bars per piece $L$, window width $W$, hop size $H$
  for $i = 0..1000$ do
      $target\_OA \leftarrow \mathcal{N}(\mu_{OA}, \sigma_{OA})$
      init first window of piece randomly
      for $w = 1..(L/H)$ do
          $next\_win \leftarrow$ reversed($prev\_window$)
          while overlap($prev\_win, next\_win$) $> target\_OA$ do
              increment a random element in second half of $next\_win$
          end while
          while overlap($prev\_win, next\_win$) $< target\_OA$ do
              select 2 random elements in the second half of $next\_win$, increment the
  larger, decrement the smaller
          end while
          add new window to piece
      end for
      add piece to result
  end for
```

Overlap Areas for pitch and duration can be optimized independently, thus in the following μ_{OA} and σ_{OA} do not take into account whether they refer to duration or pitch. Algorithm 7 creates a set of 1000 musical pieces with Overlap Areas distributed according to a given target mean μ_{OA} and variance σ_{OA}. It does so by sampling a target mean Overlap Area for each of the 1000 pieces according to μ_{OA} and σ_{OA}. Each of the 1000 pieces is then constructed window-wise so they exhibit a target mean Overlap Area between adjacent windows. To construct a piece appropriately, the first window of the piece is initialized randomly. Each of the following windows is then initialized with a reversed copy of the previous window. The Overlap Areas between (reversed) copies is 1 trivially (same mean and variance). To reduce the Overlap Area, the mean of the new window is moved away from the mean of the previous window by incrementing (or decrementing) a random element from the window. In this section, incrementing and decrementing means adding or subtracting 1, the smallest possible change in the discrete domain. Random elements are incremented or decremented until the Overlap Area between the windows is smaller than the target Overlap Area. To increase the Overlap Area again, the variance of the newly constructed window is increase without modifying the mean by selecting two random elements and incrementing the larger and decrementing the smaller. This way, a set of 1000 'musical pieces' with Overlap Areas distributed according to μ_{OA} and σ_{OA} can be constructed. As Consistency and Variance (Equation 4.2 and 4.3) only take into account those highly abstracted values, they evaluate to a high value (close to 1) for pieces generated with algorithm 7.

5 Results

In this section, the most relevant results of this work are presented. Multiple architectures of unmasking functions were trained in different settings (melody/trio). In all settings, 5% of the training data was left out as validation set. In addition to validation losses during training, the framewise self-similarity metric (section 4.7) was considered to evaluate the model's performance.

5.1 Training

In the following subsections, the results of the training runs that are conducted to assess whether the ASD3PM is suitable for symbolic music generation, are described. The experiments' results further should enable comparing the performance of different unmasking function architectures. During training, train- and validation-loss and ELBO are logged. Additionally, *Consistency* and *Variance* according to the metric in section 4.7 are evaluated. The training loss used in all of the following experiments is obtained by reweighing the ELBO (see section 4.4).

5.1.1 Convolutional Transformer (SCHmUBERT) - Melody

To allow a direct comparison to the work of Mittal et al. [2021], no augmentation was applied in this setting. Figure 5.1 shows that the training loss and ELBO quickly converge torwards a plateau. After approximately two full iterations over the training set (step 29000), the training loss begins to decrease faster again for about 5000 training steps, before it reaches its final plateau. The validation loss (figure 5.2) shows that the model does not overfit after 190k steps, but could possibly even be improved further with additional training. Figure 5.1 also visualizes that the reweighed ELBO (Loss) is smoother than the ELBO, which supports the claim of Bond-Taylor et al. [2022] that optimizing the reweighed ELBO yields better results than directly optimizing the ELBO.

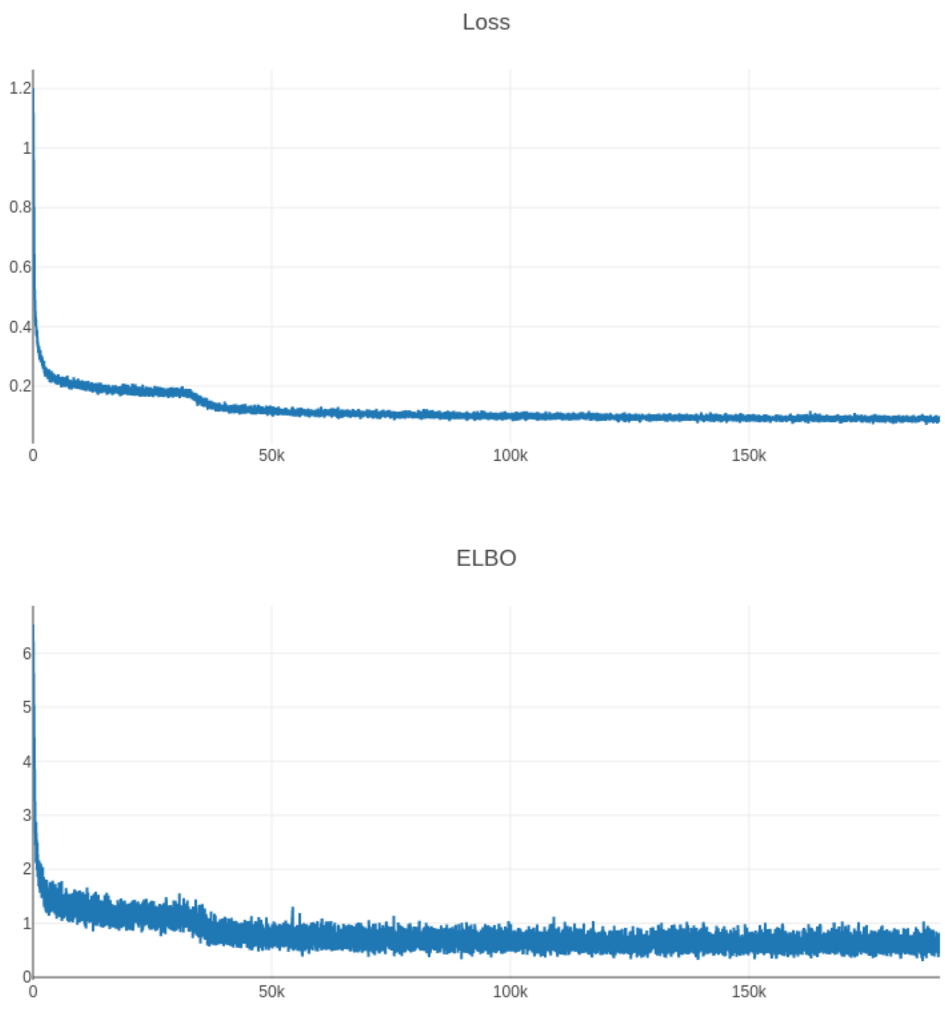


Figure 5.1: Convolutional Transformer training loss and ELBO over 10 epochs

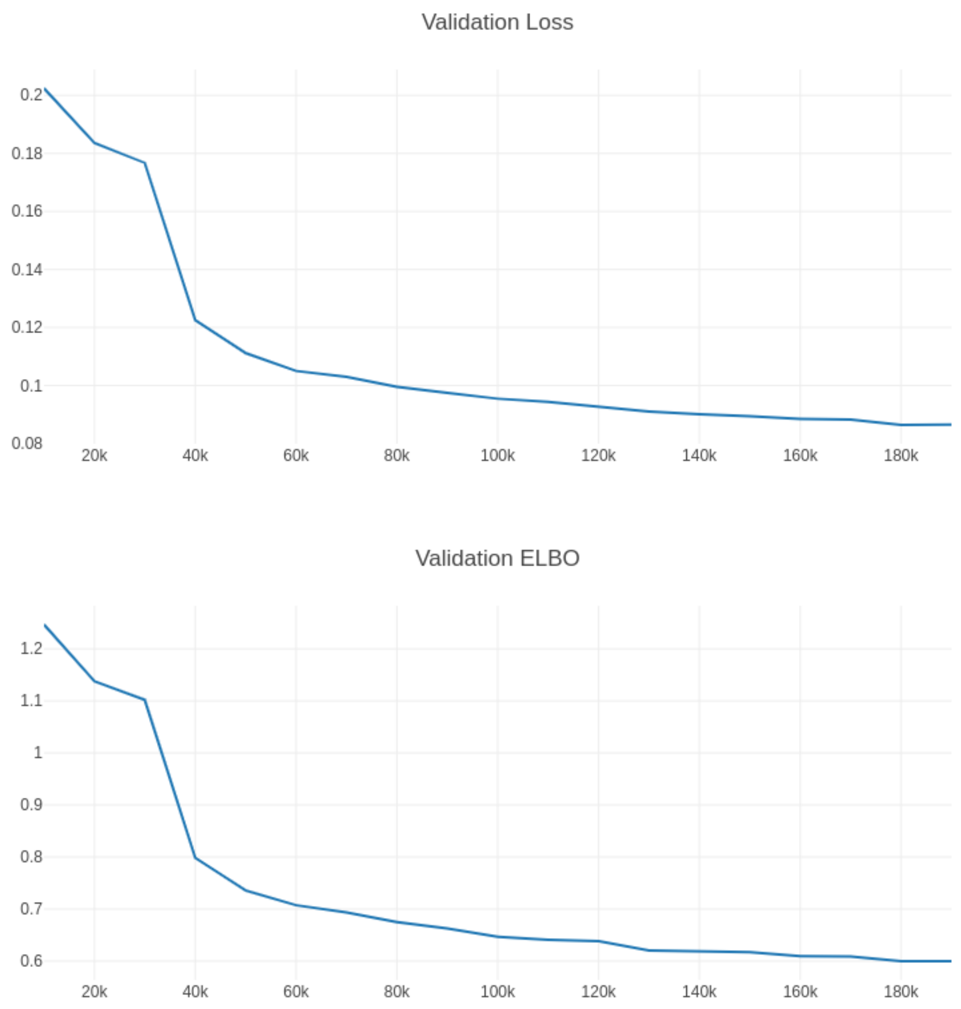


Figure 5.2: Convolutional Transformer validation loss and ELBO over 10 epochs

Name	Value
Tracks	Melody
Melody pitches	90
Augmentation	yes
Piece shape	(1024, 1)
Diffusion Model timesteps	1024
Optimizer	Adam @ lr= $5 * 10^{-4}$
Transformer layers	24
Transformer embedding size	512
Transformer attention heads	8
Total parameters	76601088
Model capacity	$\sim$ 300MB
Train steps	190,000
Validation split	0.05
Validation loss	0.087
Dataset size	19000 Batches
Dataset size	$\sim$ 2GB
Batch size	50
GPU	4x NVIDIA 2080 Ti
Duration	$\sim$ 24h

Table 5.1: Parameters of training run

5.1.2 Hierarchical Transformer - Melody

Figure 5.3 shows that the training loss and ELBO drop much quicker than the loss and ELBO of the Convolutional Transformer. Despite the initial quick learning, the Convolutional Transformer outperforms the Hierarchical transformer after less than 40k steps, according to the validation loss (figure 5.4). The bottleneck between the higher and lower level hierarchies might be a reason for this behavior: In the bottleneck, 32 embeddings are summarized into one single embedding. To avoid this bottleneck, I developed the U-Transformer model architecture.

Name	Value
Tracks	Melody
Melody pitches	90
Augmentation	yes
Piece shape	(1024, 1)
Diffusion Model timesteps	1024
Optimizer	Adam @ lr= $5 * 10^{-4}$
Hierarchy Levels	2
lower level sequence length	32
lower level Transformer embedding size	256
upper level sequence length	32
upper level Transformer embedding size	512
Transformer attention heads	8
Total parameters	56465664
Model capacity	$\sim$ 220MB
Train steps	190,000
Validation split	0.05
Validation loss	0.0701
Dataset size	190000 Batches
Epochs	10
Dataset size	$\sim$ 2GB
Batch size	50
GPU	4x NVIDIA 2080 Ti
Duration	$\sim$ 24h

Table 5.2: Parameters of hierarchical transformer training run

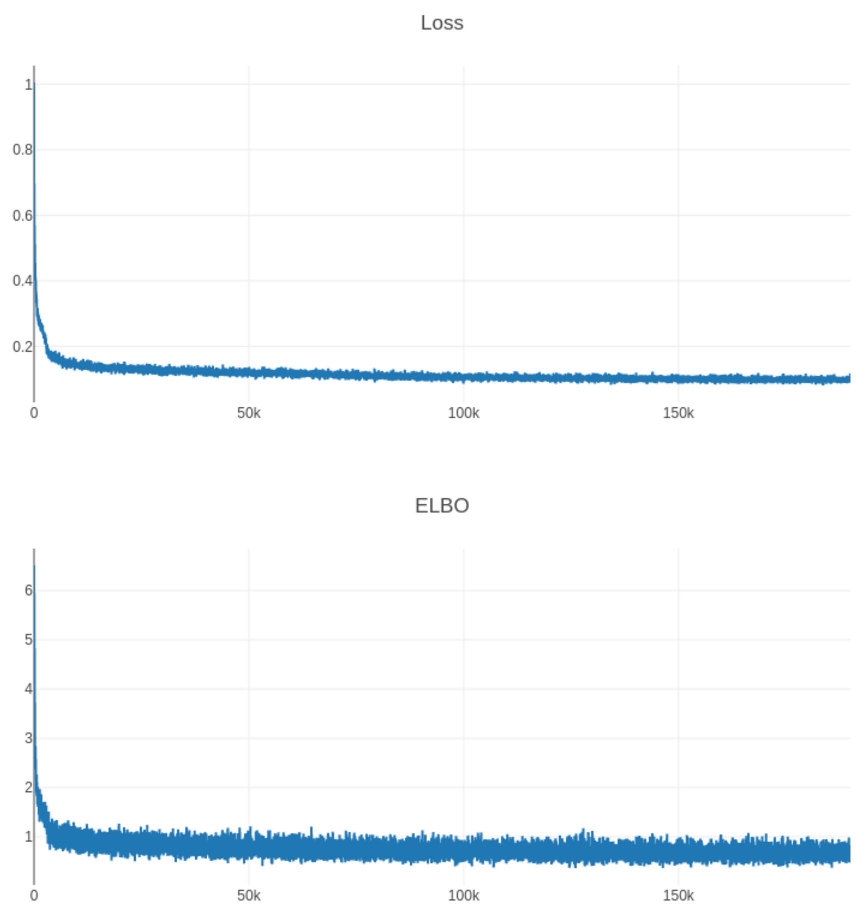


Figure 5.3: Training Loss and ELBO over 1 iteration of the dataset

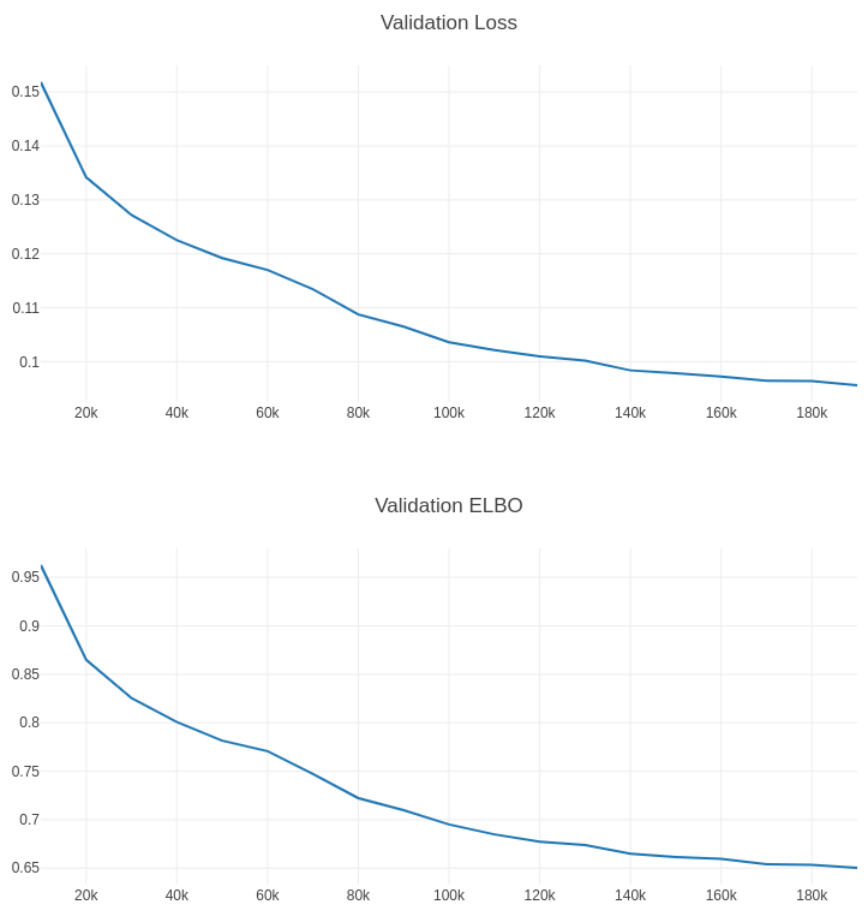


Figure 5.4: Validation Loss and ELBO over 1 iteration of the dataset

5.1.3 UTransformer - Melody

Name	Value
Tracks	Melody
Melody pitches	90
Augmentation	yes
Piece shape	(1024, 1)
Diffusion Model timesteps	1024
Optimizer	Adam @ lr= $5 * 10^{-4}$
Hierarchy levels	6
Transformer layers	4*6
Transformer embedding size	512
Transformer attention heads	8
Total parameters	80478208
Model capacity	$\sim$ 300MB
Train steps	190,000
Dataset size	19000 Batches
Validation split	0.05
Validation loss	0.092
Dataset size	$\sim$ 2GB
Batch size	50
GPU	4x NVIDIA 2080 Ti
Duration	$\sim$ 24h

Table 5.3: Parameters of training run

The UTransformer initially learns much more quickly than the Convolutional Transformer (compare figures 5.1 and 5.5). After less than 2000 steps, the loss sinks below 0.15, which takes about 30000 steps in the Convolutional architecture. Despite its quick initial learning, the UTransformer is (marginally) outperformed by the Convolutional Transformer after about 70000 steps.

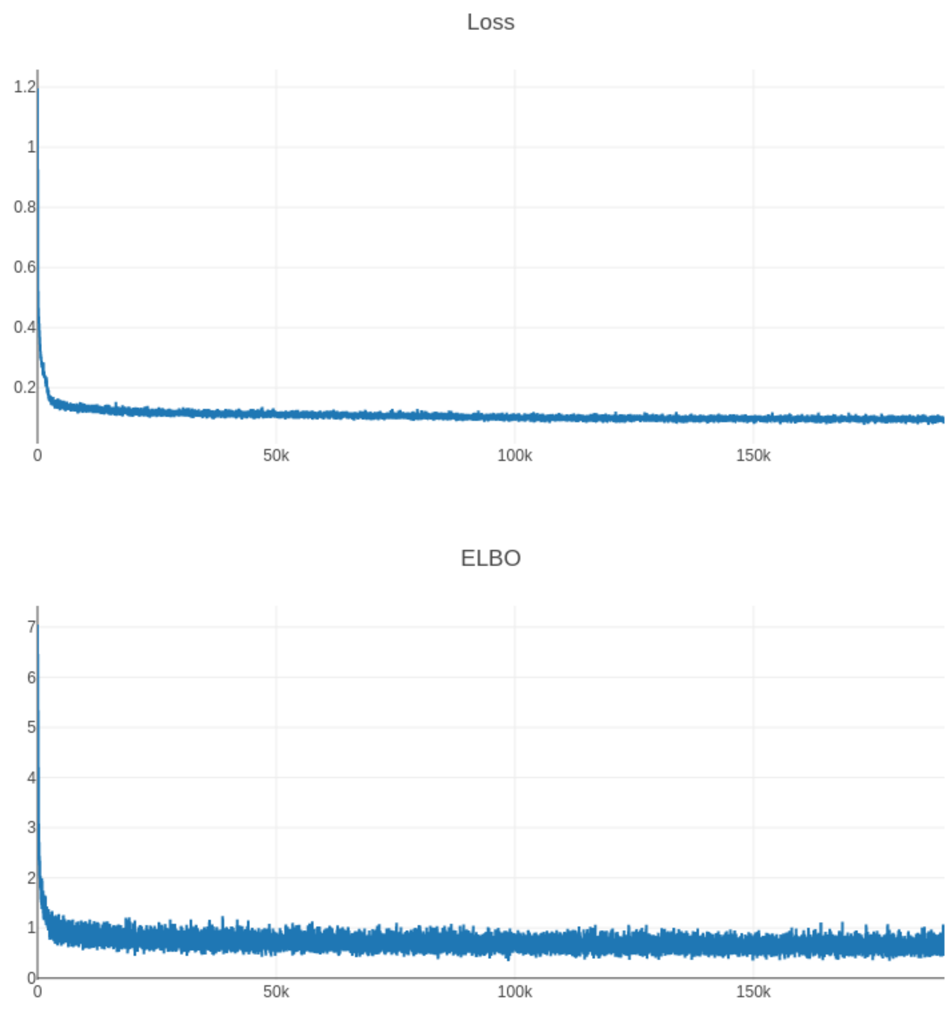
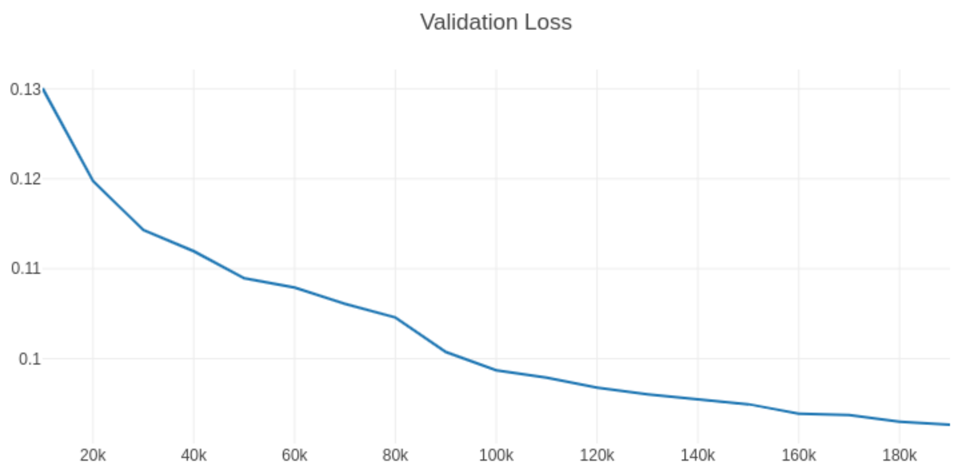


Figure 5.5: U Transformer training loss and ELBO over 10 epochs



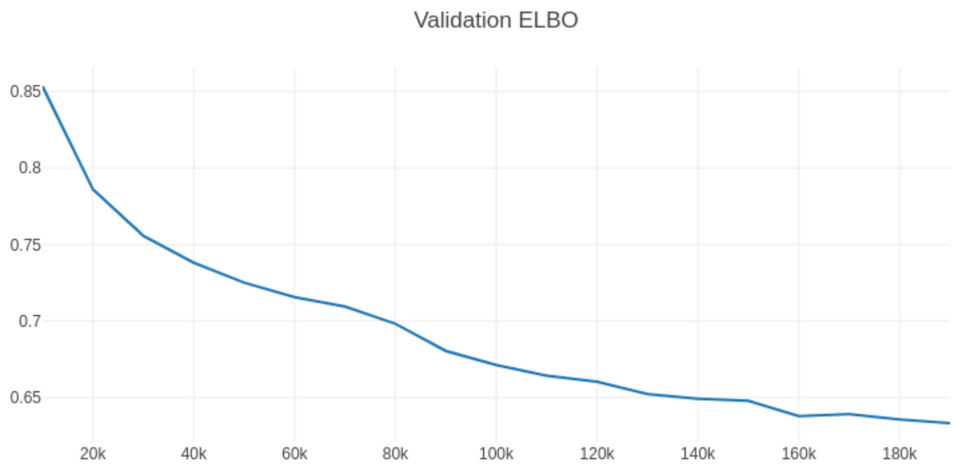


Figure 5.6: U Transformer validation loss and ELBO over 10 epochs

5.1.4 Convolutional Transformer - Trio

Name	Value
Tracks	Trio
Melody/Bass pitches	90
Augmentation	yes
Drum Patterns	$2^9 = 512$
Piece shape	(1024, 3)
Diffusion Model timesteps	1024
Optimizer	Adam @ lr= $5 * 10^{-4}$
Transformer layers	24
Transformer embedding size	512
Transformer attention heads	16
Total parameters	78,303,488
Model capacity	$\sim$ 300MB
Train steps	600,000
Validation split	0.05
Validation loss	0.046
Dataset size	37649 Batches
Epochs	16
Dataset size	$\sim$ 20GB
Batch size	50
GPU	4x NVIDIA 2080 Ti
Duration	$\sim$ 80h

Table 5.4: Parameters of training run

In this scenario, the melody and bass track were augmented by random shifting (equal offset for both tracks). Figure 5.7 shows that the training loss decreases the fastest after the first complete iteration over the training set (at train step 37000). Figure 5.8 shows that the validation loss and ELBO do not exhibit signs of overfitting, even after iterating over the training set 15 times (550k steps). This is plausible, particularly when recalling that the forward diffusion process masks out random positions, and thus creates a different unmasking problem each iteration. Furthermore, the model capacity of $\sim$ 300MB compared to the $\sim$ 20GB dataset size may prevent overfitting.

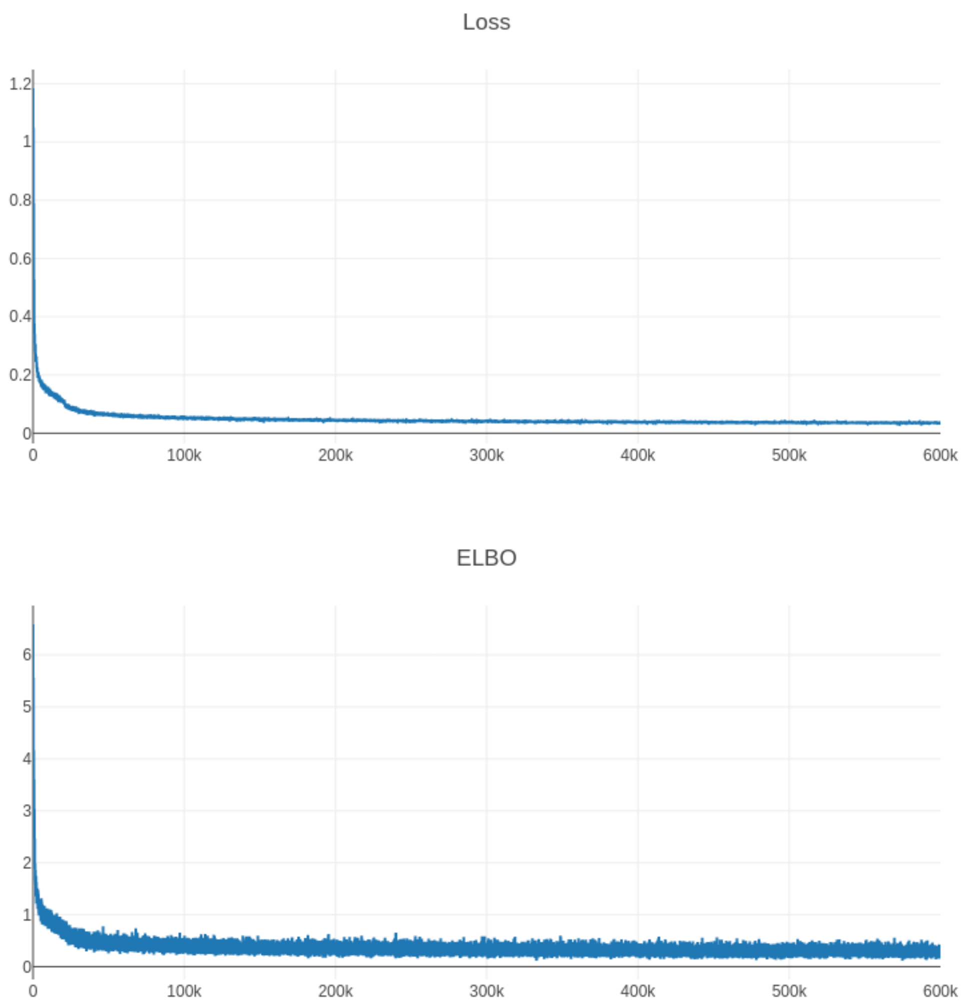


Figure 5.7: Trio Convolutional Transformer training loss and ELBO over 16 epochs

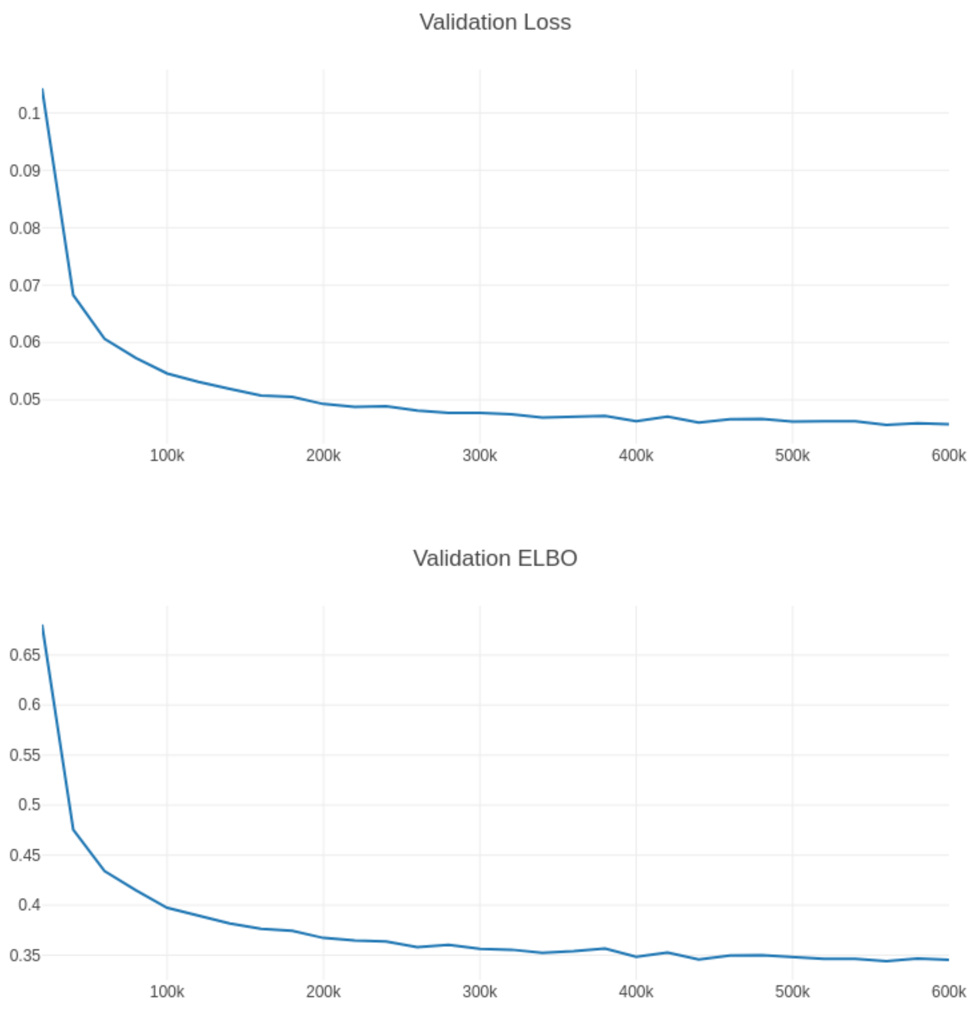


Figure 5.8: Trio Convolutional Transformer validation loss and ELBO over 16 epochs

5.2 Evaluation

In addition to the validation loss, the trained models are evaluated using the framewise self-similarity metric (section 4.7). This metric is evaluated on batches of 1000 generated, and 1000 random draws from the dataset. As the metric scores suffer from a large standard deviation, but evaluating the metrics is computationally costly (generating 1000 pieces per evaluation), values in table 5.5 denote the mean of 20 evaluations. The Convolutional Transformer consistently outperforms Magenta's Diffusion on MuiscVAE latents (MDMVAE). Especially the results of Algorithm 7 call into question, whether a higher score in this framewise self-similarity metric translates to higher musical quality.

Setting	Unconditional				Infilling			
Quantity	Pitch		Duration		Pitch		Duration	
Metric	C	Var	C	Var	C	Var	C	Var
Train Data	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Validation Data	1.00	0.96	1.00	0.96	1.00	0.96	1.00	0.91
Melody MDMVAE 64 bar	0.99	0.90	0.96	0.92	0.97	0.87	0.97	0.80
Melody 64 bar								
Convolutional Transformer	**0.992**	**0.920**	**0.993**	**0.937**	**0.997**	**0.970**	**0.996**	**0.970**
U Transformer	**0.952**	**0.930**	**0.977**	**0.932**	**0.950**	**0.943**	**0.984**	**0.972**
Hierarchical Transformer	**0.995**	**0.890**	**0.973**	**0.900**	**0.992**	**0.921**	**0.977**	**0.913**
Conv. Trans. Trio 64 bar	**0.996**	**0.893**	**0.990**	**0.896**	**0.997**	**0.964**	**0.995**	**0.924**
Classifier guided sampling	**0.96**	**0.93**	**031**	**0.22**	-	-	-	-
Algorithm 7	**0.99**	**0.98**	**0.93**	**0.86**	-	-	-	-

Table 5.5: Self similarity scores, values in **bold** represent ASD3PM models proposed in this work
The columns with header C contain the consistency scores, the columns with header Var contain the variance score of pitch respectively duration. The topmost table header distinguishes the experimental setting (Unconditional/Infilling) of the scores of its subcolumns.
The white lines from top to bottom describe the scores of the training and validation data themselves, followed by Magenta's Melody MDMVAE.
The line 'Melody 64 bar' is a subheader indicating that the following three models with different architectures are trained and evaluated on melody data.
Finally, the last three lines contain the scores of the Convolutional Transformer trained on trios, the results of a classifier guided sampling process of the Convolutional Transformer trained on trios and the results of algorithm 7. Section 5.2.1 discusses the results of the Convolutional Transformer (best-performing ASD3PM architecture) in comparison to Magenta's MDMVAE.

5.2.1 Comparison to Magenta's Symbolic Music Diffusion

In a comparison on a 1 MIO melodies subset of the Lakh MIDI dataset [Raffel, 2016], SCHmUBERT (Conv. Transfomer) achieves slightly superior performance (compare table 5.5 Melody MDMVAE 64 bar and Melody Convolutional Transformer 64 bar)

Name	Number of trainable parameters
MusicVAE melody 2bar big	403,713,296
MDMVAE	25,579,946
MusicVAE + MDMVAE	429,293,242
SCHmUBERT melody	76,603,392

Table 5.6: Comparison number of trainable model parameters

than Magenta's Symbolic Music Diffusion. Especially in the infilling setting, the model proposed in this work outperforms Magenta's Diffusion on MusicVAE latents in the framewise self-similarity metric (see equation 4.7). This is particularly remarkable, when comparing the total number of parameters of both models (table 5.6): SCHmUBERT has less than a fifth of trainable parameters of MDMVAE.

One possible reason for the slightly worse performance of MDMVAE might be the hybrid model architecture (see section 2.3): the discrete autoencoder was optimized without any information of the diffusion model that is used to model its latents. The continuous diffusion model on the other hand does not perceive the discrete nature of its actual task while modelling the latents of MusicVAE. Thus MusicVAE operates in its own local optimum (without knowledge about the diffusion model), and the diffusion model is optimized based on the local optimum of MusicVAE, although, possibly a better joint optimum may exist.

Secondly, the latent-trimming (see section 2.3) Mittal et al. [2021] perform before training the diffusion model may allow the second stage to operate more efficiently, but it might also limit the diffusion model's perception of the musical pieces it operates on. This means that the joint model might be able to express a large subset of the musical pieces consistently, but there are some pieces which are encoded using latent variables that are lost in the trimming process. The fact that MDMVAE performs almost as good as SCHmUBERT in unconditional generation, but not in infilling, also supports the hypothesis of trimmed latent variables limiting the expression space of the joint model.

Some of the discrepancy in trainable model parameters might be explained by the dataset sizes used for training: the autoencoder part (MusicVAE melody 2bar big) was trained on a set of 28 MIO melodies, while the version of SCHmUBERT used for comparison was only trained on 1 MIO melodies. Based on the previous findings, I conclude that a comparison of self-similartiy performance (see equation 4.7) is fair nonetheless, as the second stage of the hybrid model was also trained on a 1 MIO melodies subset of the LMD, and this subset was included in the 28 MIO melodies used to train MusicVAE.

5.2.2 Classifier Guidance

In a proof-of-concept experiment, I explored whether generation results could be guided using a classifier. Therefore, I trained a barwise classifier on note-density on synthetic data. Enforcing a constant note density of 8 across 1000 pieces resulted in 40% of all bars in those pieces to have a note density of 8 notes per bar. The rest of bars had note densities close to 8. In unconditional generation, 5% of all generated bars had a note density of 8. The mean squared difference to 8 of note-density was 0.1 for guided sampling and 16 for unguided. Consistency and Variance of the generated pieces suffered from this guidance, but that is not surprising considering that constant note-density was enforced across whole pieces. Note that the classifier used for this experiment was not adversarial robust. Dhariwal and Nichol [2021] state that adversarially robust classifiers provide more stable gradients, and thus could be even more suitable for sampling guidance.

5.2.3 Visual Examples

It is difficult to assess the musical quality of generated samples scientifically, as both validation loss and self-similarity metric do not allow direct conclusions about it. This section provides visualizations of different inference scenarios. All of the following results are inferred from the Convolutional Transformer trained on trios for 600k steps (see section 5.1.4), with 1024 sample steps. In the following plots, notes of the melody track are filled red, those of the bass track blue, and those of the drum track black.

Infilling Training Sample

In the melody infilling sample (figure 5.9), the area within the blue box was masked out from an original training sample (top), and filled in by SCHmUBERT (bottom). Although the original sample was not restored by the unmasking process, the result appears very coherent in my personal perception.

Figure 5.10 shows the original trio version of the previous sample (top). In the center, all three tracks were masked and restored by SCHmUBERT. Again, the model did not recover the original notes, but generated a substitution with a pause in the melody track, which sounds coherent and plausible in my perception. The bottom plot shows an accompaniment scenario: the bass and drums track were masked out and restored by the model.

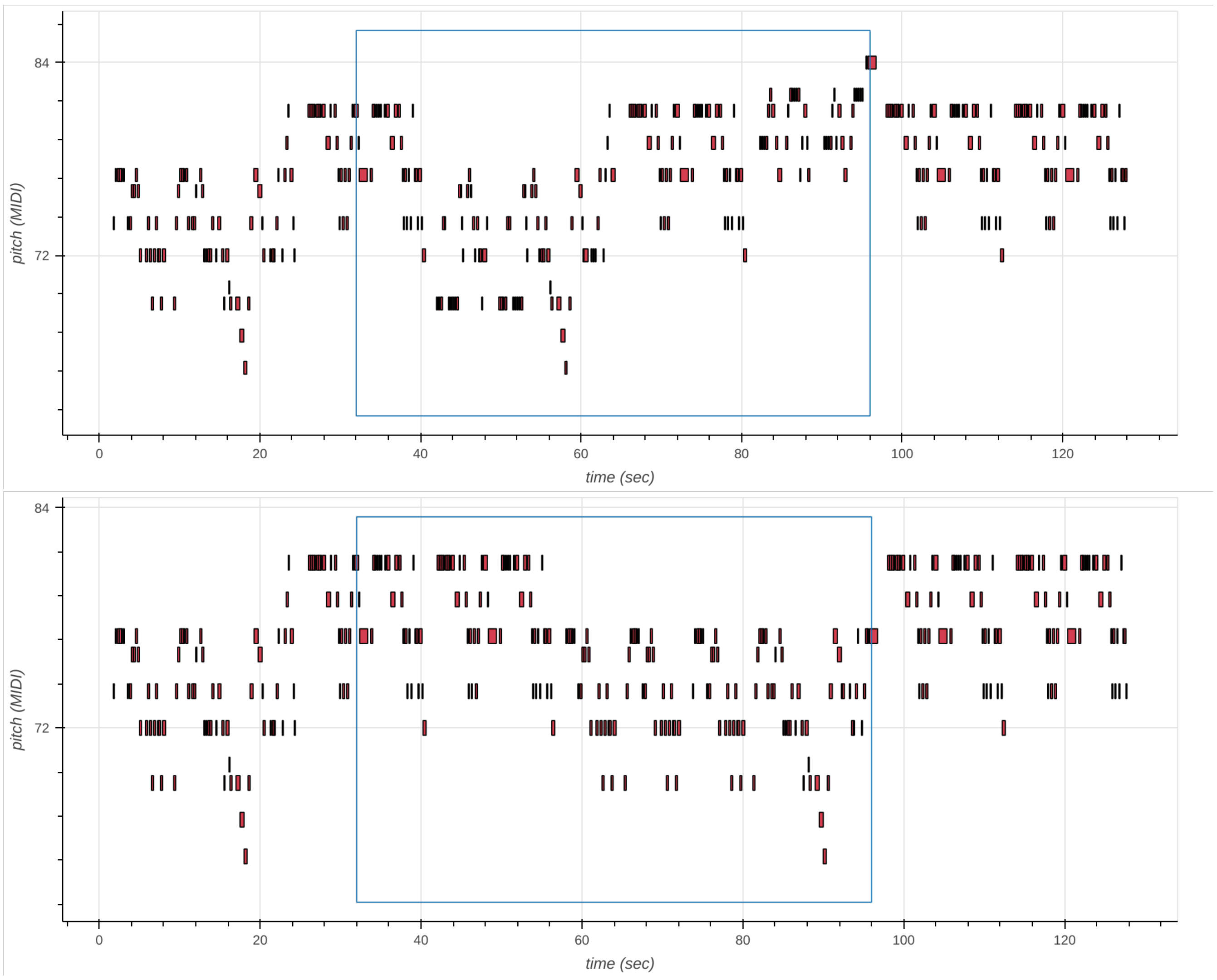


Figure 5.9: Infilling Training Sample - Melody only

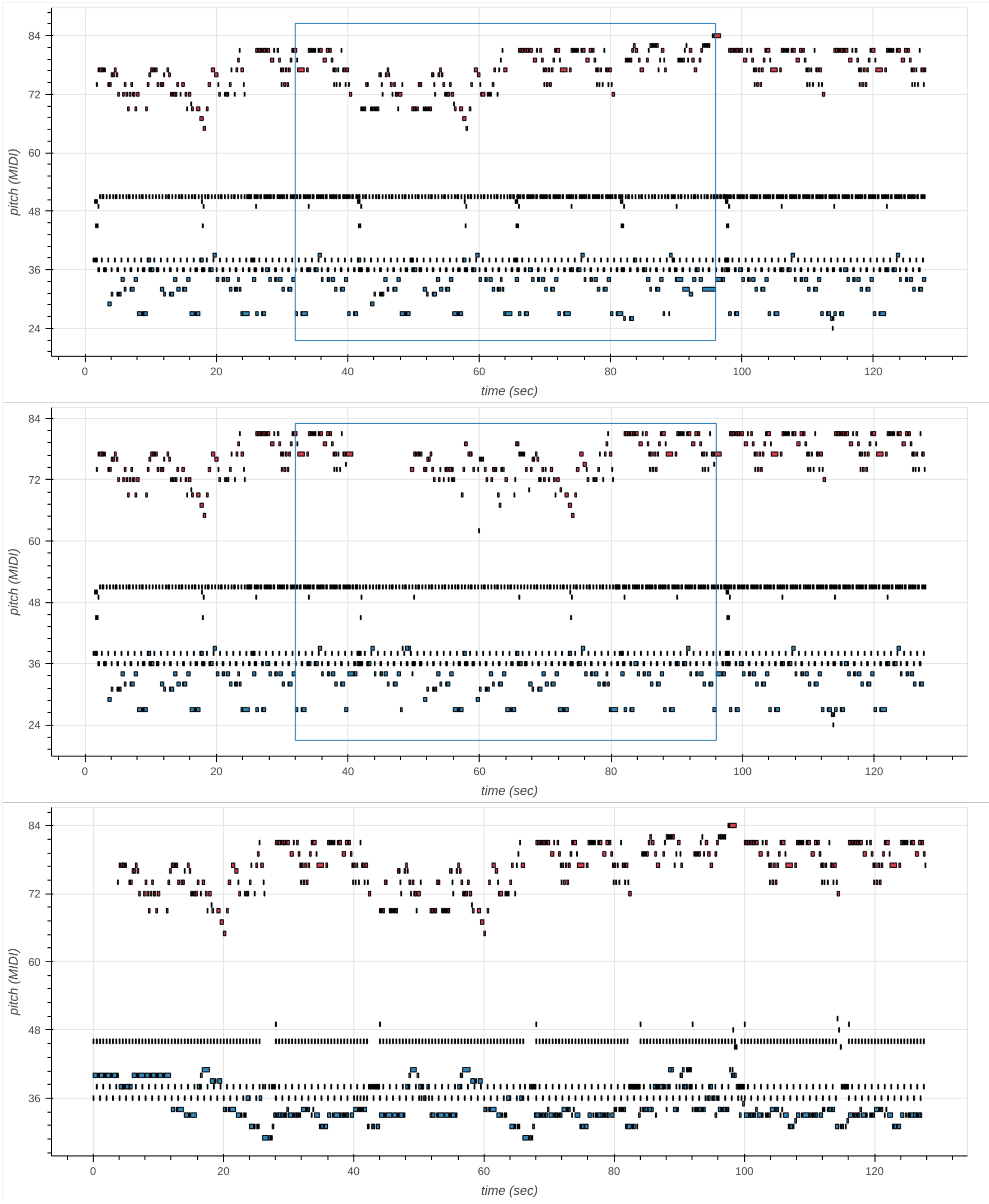


Figure 5.10: Infilling Training Sample - Trio

Infilling Validation Sample

Figure 5.11 shows melody infilling on an original sample the model was not trained on (top). Although the result (bottom) does not resemble the original sample, it looks and sounds coherent in my personal perception.

In the trio infilling sample (figure 5.12), the gap in the original sample (top) could be filled consistently. The accompaniment setting (bottom) shows that no drum track was restored for that sample (all drum notes were unmasked as note off). As all the pieces in the training data have notes in all tracks, the model does not work as expected in that case. A reason for this behavior might be that in the training process, notes are only masked with equal probabilities in all three tracks, and the setting where all notes of a track are masked out while another track is fully unmasked never occurs.

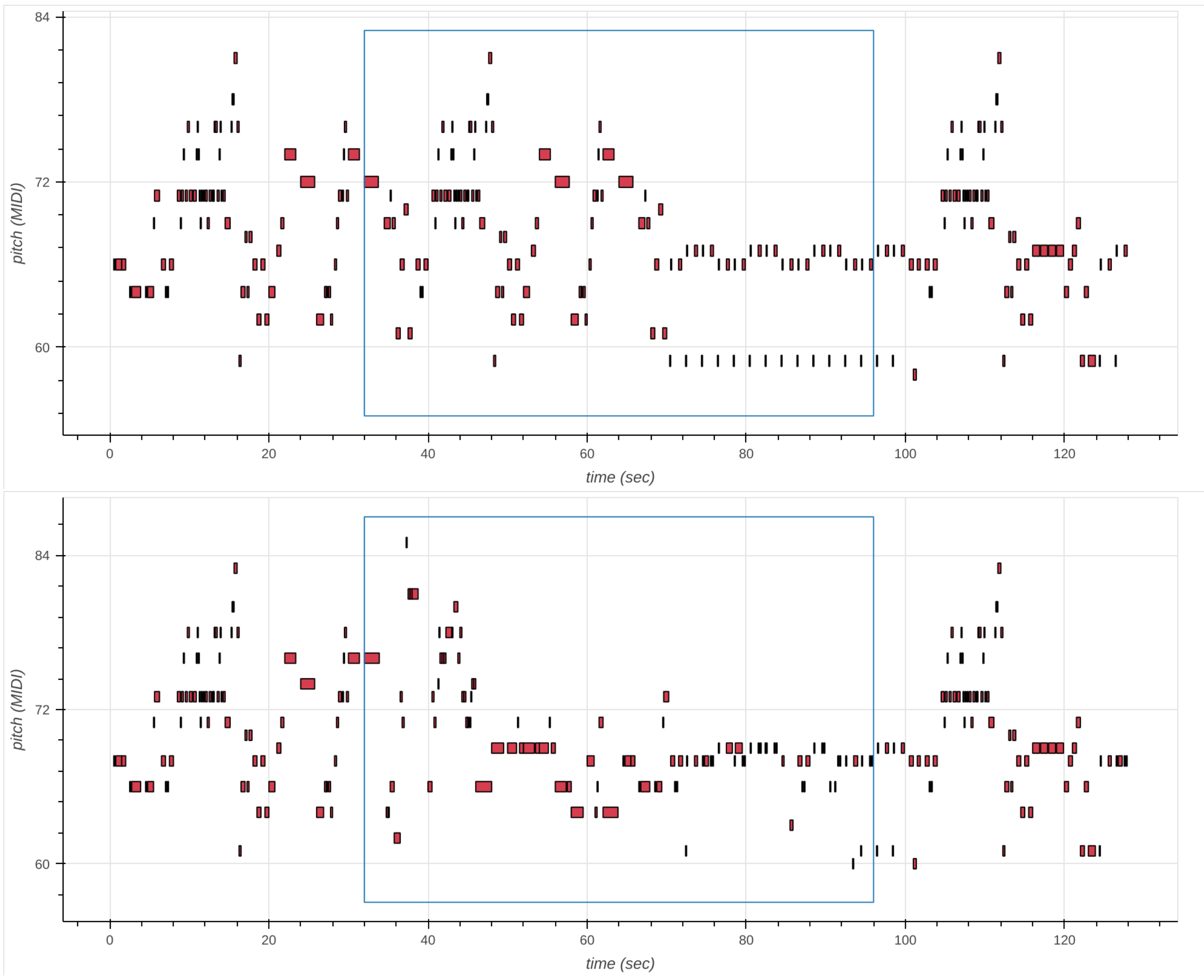


Figure 5.11: Infilling Validation Sample - Melody only

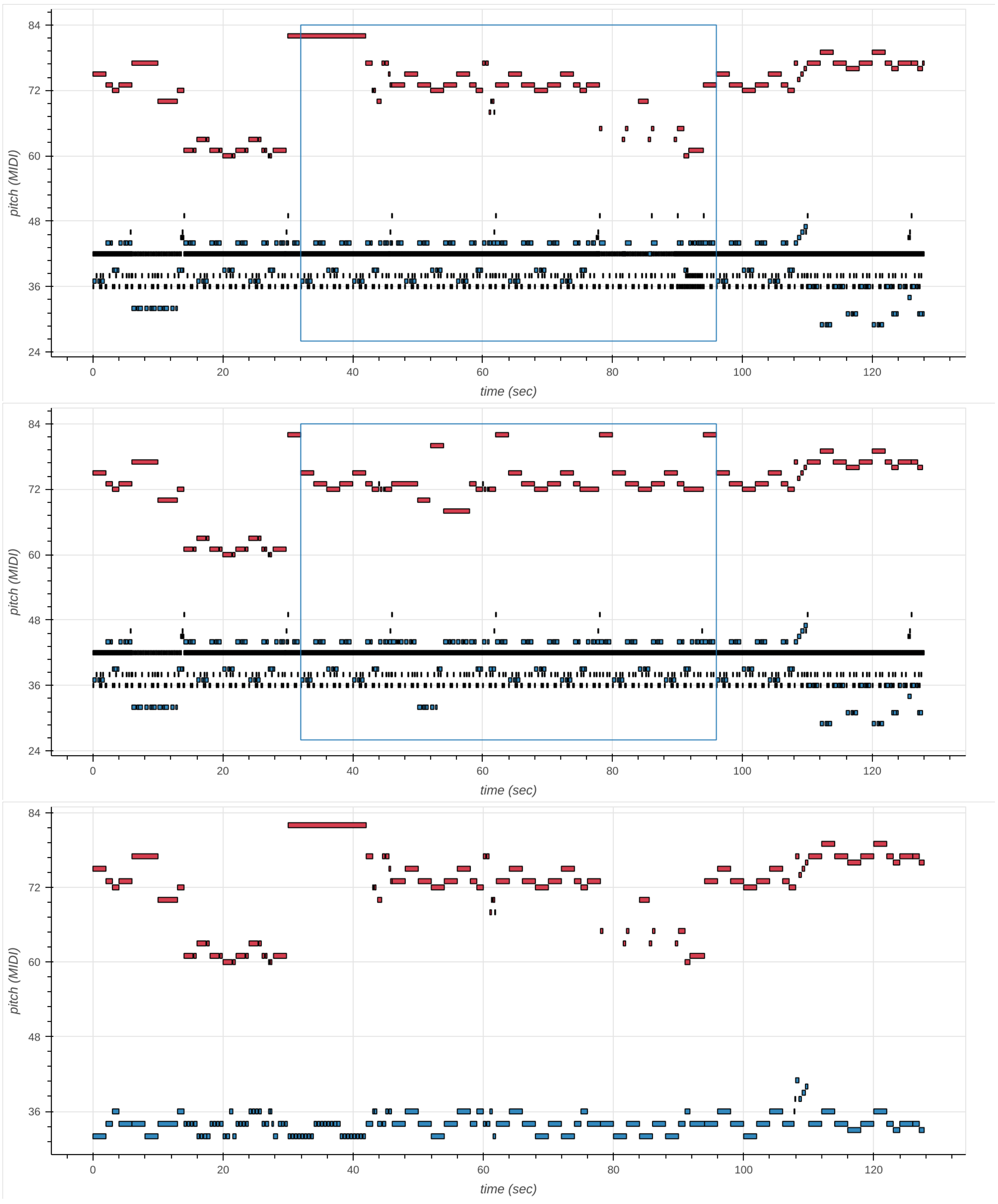


Figure 5.12: Infilling Validation Sample - Trio

Outpainting Validation Sample

Figure 5.13 shows a combination of the infilling and the accompaniment scenario: SCHmUBERT is provided with a melody that fills only 73/128 seconds of SCHmUBERTs input/output data structure. The the rest of the melody, and the whole bass and drum track are generated by SCHmUBERT. To me, the resulting melody track looks and sounds very coherent, and contains varied copies of phrases of the input melody. The bass and drum tracks are very monotonous and do not seem to be well conditioned on the melody part. Again, this behavior could probably be remedied by using different masking schedules during training.

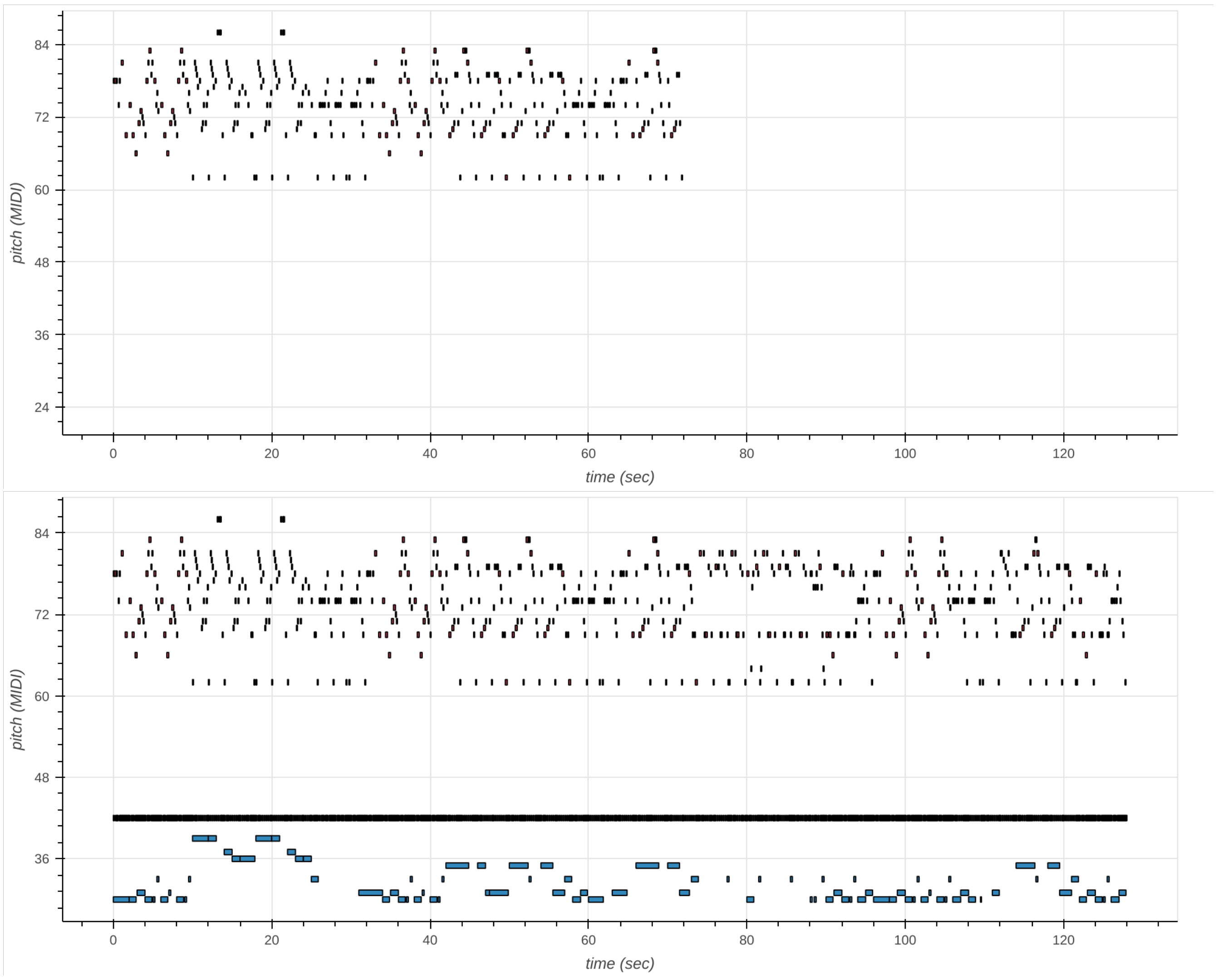


Figure 5.13: Outpainting Validation Sample - Trio

Outpainting Validation Sample - Adapted Masking Schedule

In an additional experiment, I trained the Convolutional Transformer architecture with a slightly altered forward diffusion process: notes are either masked in one, two or all three tracks. This improved the model's capability to generate accompaniment. In the sample shown in figure 5.14, all three tracks sound very coherent to me. It is also visually observable that the pitch of the bass track (blue, bottom) follows the pitch of the melody track (red, top).

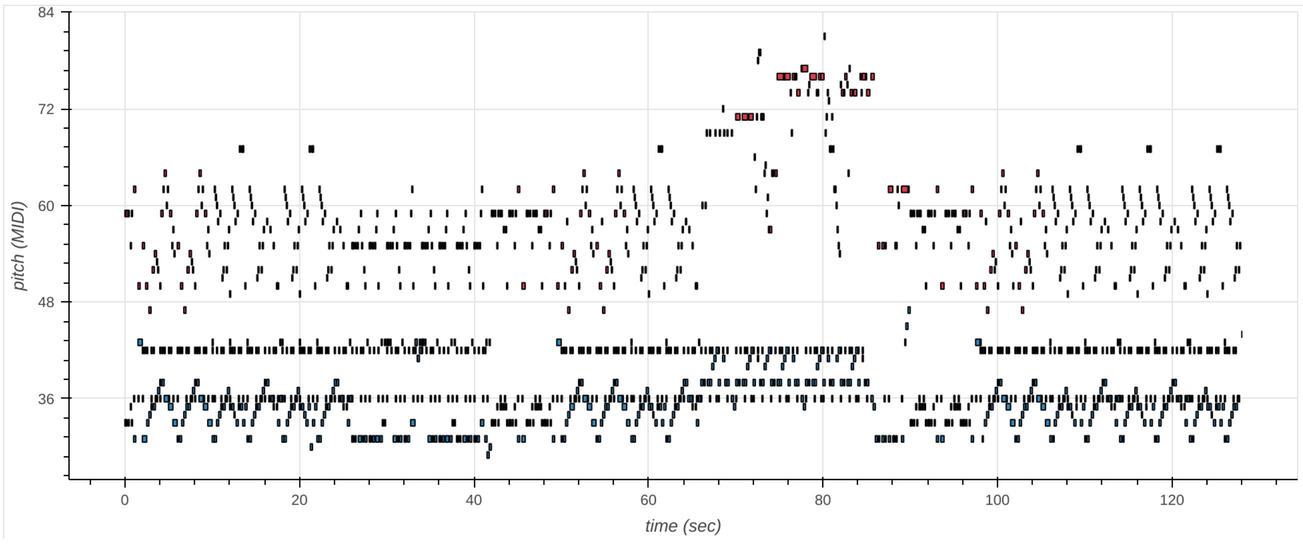


Figure 5.14: Consistent Accompaniment Validation Sample - Trio

Sample from Algorithm 7

Figure 5.15 shows an example that was constructed by algorithm 7. Although consistency and variance evaluated for pieces constructed with algorithm 7 rank very high, it is visually apparent, that such pieces exhibit no recognizable structure.

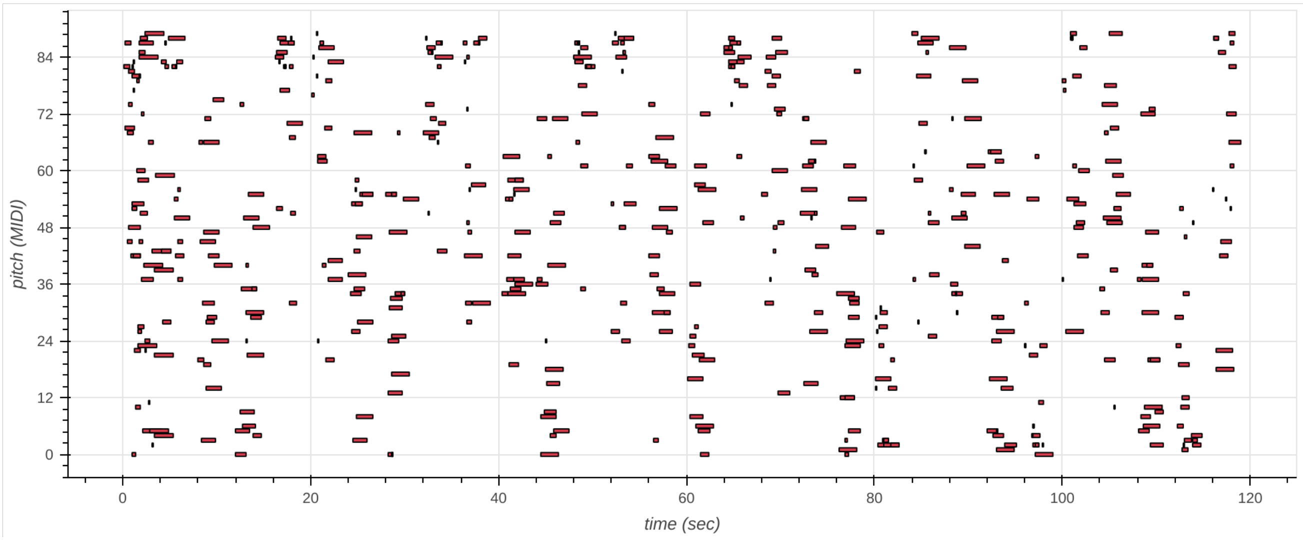


Figure 5.15: Sample constructed with a target Overlap Area

6 Conclusion and Future Work

6.1 Conclusion

The direct application of Discrete DPMs indeed offers advantages over over existing generative methods: the model proposed in this work is able to generate samples with state-of-the-art quality with only a fifth of parameters of the model from Mittal et al. [2021]. In conditional infilling, SCHmUBERT even clearly outperforms the model of Mittal et al. [2021] according to the framewise self-similarity metric. As this metric only captures statistical similarity, it is difficult to argue about the musical quality of generated pieces, but I assume that as long as this metric is not used as a direct target, but only optimized implicitly, higher scores still indicate higher musical quality. Regardless of sample quality, the direct symbolic music generation with Discrete DPMs offers additional advantages over the existing approach: Conditional infilling at note level and accompaniment generation. Furthermore, the proof-of-concept experiment shows that the sampling process of SCHmUBERT can be guided with off-the-shelf classifiers. These classifiers can be trained on high-level features such as note density or key signature.

6.2 Future Work

Classifiers on high-level features are a valuable starting point for future work: If human experts could still distinguish generated samples from training data, and pinpoint their distinction criteria into a differentiable classifier, this classifier could be used to guide the diffusion process towards more realistic samples. This guidance signal could possibly also be integrated into the DPM by finetuning (retraining) using the classifier as an auxiliary loss. This auxiliary loss should encourage realistic samples while not perturbing the original diffusion model loss. As a hypothetical real-generated classifier probably provides erroneous labels for many real samples, I suggest weighing the discriminator loss L_{adv} with the reconstruction error L_{CE}. This way, the adversarial loss is ignored, if the original sample is restored successfully, and only guides the generative process towards more realistic samples otherwise.

The resulting model could then be trained GAN-like, by alternately improving the generator and the discriminator (classifier). As mentioned in section 3.3, a fundamental strength of DPMs is their exceptional depth. The ASD3PM-formulation from this work is trained to predict $p(\mathbf{x}_0|\mathbf{x}_t)$. This is particularly difficult in timesteps close to T. Considering the first unmasking step, $p(\mathbf{x}_0|\mathbf{x}_T)$, it is not difficult to see that this task is impossible: there is exactly one $\mathbf{x}_T$ (only masks) for all $|dataset|$ possible x_0. This difficulty is reflected in the output probabilities of the first unmasking step, which are very flat and assign similar probabilities to all pitches. Although these flat $\mathbf{x}_0$ probabilities do permit obtaining realistic samples in one step, they are necessary for sample diversity of the entire reverse diffusion process. Thus, I suggest starting the exploration of a possible GAN+Diffusion hybrid model where the generative model is trained with a loss in the form of:

$$L_{hybrid} = \frac{1}{t} * L_{CE} * (1 + \lambda L_{adv}) \tag{6.1}$$

The flat output probabilities of the fist unmasking step could also motivate a different combination of DPM and GAN: the first (few) reverse diffusion steps could be completely replaced by a GAN. This way, the reverse diffusion process could start at more realistic samples. More realistic samples $\mathbf{x}_0$ do not harm the reverse diffusion process, as long as they are still diverse (in training, all x_t are produced from real samples). This diversity is enabled through the random seed of the GAN.

This random seed could possibly also be simulated for ASD3PMs. In addition to partially masked samples, the unmasking neural network would receive a seed that should determine characteristics of x_0. This seed should be constant for samples across all timesteps. Thus, unmasking steps with many masks could afford to produce more realistic samples, as diversity is achieved through the seed.
To elaborate further, this seed ideally contains some information about the original sample, while still being available without access to real samples. Such seeds could be generated using VAEs: The latent representation consistently captures features of pieces, and are still obtainable using a random generator. Providing ASD3PMs with VAE latents of unmasked samples could allow all reverse diffusion steps to predict more realistic $\mathbf{x}_0$, and thus combine the depth of DPMs with the single-step sample quality of GANs.

The area where I expect the most room for improvement of perceptual sample quality is the dataset. No (known) deep generative model can produce samples with a quality exceeding the quality of its training data. Thus, I think a dataset with higher quality samples or pieces from a desired genre (also difficult to assess quantitatively, a matter of taste after all) could greatly boost the quality of results. Also, more data is usually helpful in deep learning.

Finally, SCHmUBERT can easily be adapted to different numbers of tracks. In the current version, each of SCHmUBERT's tracks only supports one simultaneous pitch per track, although most tracks are polyphonic in the Lakh MIDI dataset. Probably the most promising approach to support polyphonic instrument tracks is stacking multiple monophonic tracks for a single instrument.

Acknowledgements

First, and foremost I would like to thank my supervisor Silvan Peter for the great support, guidance and valuable input throughout this entire work. I thank Gerhard Widmer for making this work possibile and giving very valuable feedback. Further, I thank the the Institute of Computational Perception for providing computational resources and hundreds of kilowatt-hours of electrical power.

Finally, I thank Ruth Höpler for support, feedback, and enduring listening to hours of shitty generation results and pretending it was actual music.

7 Appendix

7.1 Gaussian Reverse Transition Probabilities

The following derivation of the Gaussian reverse transition probabilities is based on the blogpost of Weng [2021]. The derivations show that the reverse diffusion process of a Gaussian forward diffusion process can also be expressed as a Gaussian distribution. For clarity, terms that belong to the mean μ of the resulting distribution are colored blue, while terms that form the standard deviation σ of the resulting distribution are colored red.

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0)\frac{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)} \tag{7.1}$$

$$\propto \exp\Big(-\frac{1}{2}(\frac{(\mathbf{x}_t - \sqrt{\alpha_t}\mathbf{x}_{t-1})^2}{\beta_t} + \frac{(\mathbf{x}_{t-1} - \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0)^2}{1-\bar{\alpha}_{t-1}} - \frac{(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0)^2}{1-\bar{\alpha}_t})\Big) \tag{7.2}$$

$$= \exp\Big(-\frac{1}{2}(\frac{\mathbf{x}_t^2 - 2\sqrt{\alpha_t}\mathbf{x}_t\mathbf{x}_{t-1} + \alpha_t\mathbf{x}_{t-1}^2}{\beta_t} + \frac{\mathbf{x}_{t-1}^2 - 2\sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0\mathbf{x}_{t-1} + \bar{\alpha}_{t-1}\mathbf{x}_0^2}{1-\bar{\alpha}_{t-1}} - \frac{(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0)^2}{1-\bar{\alpha}_t})\Big) \tag{7.3}$$

$$= \exp\Big(-\frac{1}{2}((\frac{\alpha_t}{\beta_t} + \frac{1}{1-\bar{\alpha}_{t-1}})\mathbf{x}_{t-1}^2 - (\frac{2\sqrt{\alpha_t}}{\beta_t}\mathbf{x}_t + \frac{2\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}}\mathbf{x}_0)\mathbf{x}_{t-1} + C(\mathbf{x}_t, \mathbf{x}_0))\Big) \tag{7.4}$$

$$= \exp\Big(-\frac{1}{2}(\frac{1}{\tilde{\beta}_t}\mathbf{x}_{t-1}^2 - \frac{1}{\tilde{\beta}_t}2\tilde{\mu}(\mathbf{x}_t, \mathbf{x}_0)\mathbf{x}_{t-1} + \frac{\tilde{\mu}(\mathbf{x}_t, \mathbf{x}_0)^2}{\tilde{\beta}_t})\Big) \tag{7.5}$$

$$= \exp\Big(-\frac{1}{2}(\frac{\mathbf{x}_{t-1} - \tilde{\mu}(\mathbf{x}_t, \mathbf{x}_0)}{\beta_t})^2\Big) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\mu}(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t\mathbf{I}) \tag{7.6}$$

Equation 7.1 to 7.6 show that the probability of the reverse diffusion process can be expressed as Gaussian distribution with mean $\tilde{\boldsymbol{\mu}}(\mathbf{x}_t, \mathbf{x}_0)$ and variance $\tilde{\beta}_t$ [1]. Recalling $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{T} \alpha_i$, mean and variance of the reverse diffusion process can be expressed as follows:

[1] Note that in equation 7.4, the rest of the term that does not depend on $\mathbf{x}_{t-1}$ was abbreviated with the function C

$$\tilde{\beta}_t = 1/(\frac{\alpha_t}{\beta_t} + \frac{1}{1-\bar{\alpha}_{t-1}}) = 1/(\frac{\alpha_t - \bar{\alpha}_t + \beta_t}{\beta_t(1-\bar{\alpha}_{t-1})}) = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \beta_t \tag{7.7}$$

$$\tilde{\boldsymbol{\mu}}(\mathbf{x}_t, \mathbf{x}_0) = (\frac{\sqrt{\alpha_t}}{\beta_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}}\mathbf{x}_0)/(\frac{\alpha_t}{\beta_t} + \frac{1}{1-\bar{\alpha}_{t-1}}) \tag{7.8}$$

$$= (\frac{\sqrt{\alpha_t}}{\beta_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}}\mathbf{x}_0)\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \beta_t \tag{7.9}$$

$$= \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 \tag{7.10}$$

$\tilde{\boldsymbol{\mu}}(\mathbf{x}_t, \mathbf{x}_0)$ is the only unknown term when sampling new data, and needs to be approximated using a neural network. Using equation 3.42, $\mathbf{x}_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t)$ can be derived. Plugging this closed form x_0 into equation 7.10, yields:

$$\tilde{\boldsymbol{\mu}}(\mathbf{x}_t) = \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\frac{1}{\sqrt{\bar{\alpha}_t}}(\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t) \tag{7.11}$$

$$= \frac{1}{\sqrt{\alpha_t}}\Big(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_t\Big) \tag{7.12}$$

Ultimately, the only term that needs to be approximated by a neural network is $\boldsymbol{\epsilon}_t$:

$$\mathbf{x}_{t-1} = \mathcal{N}(\mathbf{x}_{t-1}; \frac{1}{\sqrt{\alpha_t}}\Big(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\Big), \tilde{\beta}_t) \tag{7.13}$$

Bibliography

Augusta Ada King, Countess of Lovelace (1843). *Scientific Memoirs Selected from the Transactions of Foreign Academies of Science and Learned Societies*, pp. 693–695.

Austin, Jacob et al. (2021). 'Structured Denoising Diffusion Models in Discrete State-Spaces'. In: *NeurIPS 2021*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., pp. 17981–17993. URL: `https://proceedings.neurips.cc/paper/2021/file/958c530554f78bcd8e97125b70e6973d-Paper.pdf`.

Bengio, Y., P. Simard and P. Frasconi (1994a). *Learning long-term dependencies with gradient descent is difficult.* DOI: `10.1109/72.279181`.

– (1994b). 'Learning long-term dependencies with gradient descent is difficult'. In: *IEEE Transactions on Neural Networks* 5.2, pp. 157–166. DOI: `10.1109/72.279181`.

Bond-Taylor, Sam et al. (2022). 'Unleashing Transformers: Parallel Token Prediction with Discrete Absorbing Diffusion for Fast High-Resolution Image Generation from Vector-Quantized Codes'. In: *ECCV 2022*. Ed. by Shai Avidan et al. Cham: Springer Nature Switzerland, pp. 170–188. ISBN: 978-3-031-20050-2.

Devlin, Jacob et al. (June 2019). 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding'. In: *NAACL 2019*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 4171–4186. DOI: `10.18653/v1/N19-1423`. URL: `https://aclanthology.org/N19-1423`.

Dhariwal, Prafulla and Alexander Nichol (2021). 'Diffusion Models Beat GANs on Image Synthesis'. In: *NeurIPS 2021*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., pp. 8780–8794. URL: `https://proceedings.neurips.cc/paper/2021/file/49ad23d1ec9fa4bd8d77d02681df5cfa-Paper.pdf`.

Donahue, Chris, Julian McAuley and Miller Puckette (2018). 'Adversarial Audio Synthesis'. In: *ICLR 2018*.

Feller, William (1949). 'On the theory of stochastic processes, with particular reference to applications'. In: *Berkeley Symposium on Mathematical Statistics and Probability, 1949*, pp. 403–432. URL: `https://projecteuclid.org/ebooks/berkeley-symposium-on-mathematical-statistics-and-probability/On-the-Theory-of-Stochastic-Processes-with-Particular-Reference-to/chapter/On-the-Theory-of-Stochastic-Processes-with-Particular-Reference-to/bsmsp/1166219215`.

Forsgren, Seth* and Hayk* Martiros (2022). 'Riffusion - Stable diffusion for real-time music generation'. In: URL: `https://riffusion.com/about`.

Goodfellow, Ian J. et al. (2014). 'Generative Adversarial Networks'. In: NeurIPS 2014. URL: `https://arxiv.org/abs/1406.2661`.

Hinton, G E and R R Salakhutdinov (July 2006). 'Reducing the dimensionality of data with neural networks'. In: *Science* 313.5786, pp. 504–507. DOI: `10.1126/science.`

1127647. URL: http://www.ncbi.nlm.nih.gov/sites/entrez?db=pubmed&uid=16873662&cmd=showdetailview&indexed=google.

Ho, Jonathan, Ajay Jain and Pieter Abbeel (2020). 'Denoising Diffusion Probabilistic Models'. In: *NeurIPS 2020*. URL: https://proceedings.neurips.cc/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). 'Long Short-Term Memory'. In: *Neural Computation* 9.8, pp. 1735–1780.

Hoogeboom, Emiel et al. (2021). 'Argmax Flows and Multinomial Diffusion: Learning Categorical Distributions'. In: *NeurIPS 2021*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., pp. 12454–12465. URL: https://proceedings.neurips.cc/paper/2021/file/67d96d458abdef21792e6d8e590244e7-Paper.pdf.

Kingma, Diederik P. and Max Welling (2014). 'Auto-Encoding Variational Bayes.' In: *ICLR 2014*. Ed. by Yoshua Bengio and Yann LeCun. URL: http://dblp.uni-trier.de/db/conf/iclr/iclr2014.html#KingmaW13.

Langevin, Paul (1908). 'Sur la théorie du mouvement brownien'. In: *American Journal of Physics* 65.11, pp. 1079–1081. DOI: 10.1119/1.18725. eprint: https://doi.org/10.1119/1.18725. URL: https://doi.org/10.1119/1.18725.

Li, Xiang Lisa et al. (2022). 'Diffusion-LM Improves Controllable Text Generation'. In: NeurIPS 2022. DOI: 10.48550/ARXIV.2205.14217. URL: https://arxiv.org/abs/2205.14217.

Mittal, Gautam et al. (2021). 'Symbolic Music Generation with Diffusion Models'. In: *ISMIR 2021*. URL: https://archives.ismir.net/ismir2021/paper/000058.pdf.

OpenAI (2022). 'ChatGPT'. In: URL: https://openai.com/blog/chatgpt/.

Radford, Alec et al. (2019). 'Language Models are Unsupervised Multitask Learners'. In: URL: https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Raffel, Colin (2016). 'Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching'. PhD thesis. COLUMBIA UNIVERSITY. URL: https://colinraffel.com/projects/lmd/.

Roberts, Adam et al. (n.d.). 'A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music'. In: *ICML 2018*. URL: https://proceedings.mlr.press/v80/roberts18a.html.

Rombach, Robin et al. (2022). 'High-resolution image synthesis with latent diffusion models'. In: *CVPR 2022*, pp. 10684–10695.

Ronneberger, Olaf, Philipp Fischer and Thomas Brox (2015). 'U-Net: Convolutional Networks for Biomedical Image Segmentation'. In: *MICCAI 2015*. Ed. by Nassir Navab et al. Cham: Springer International Publishing, pp. 234–241. ISBN: 978-3-319-24574-4.

Shannon, C.E. (Jan. 1949). 'Communication in the Presence of Noise'. In: *Proceedings of the IRE* 37.1, pp. 10–21. DOI: 10.1109/jrproc.1949.232969. URL: https://doi.org/10.1109/jrproc.1949.232969.

Sohl-Dickstein, Jascha et al. (July 2015). 'Deep Unsupervised Learning using Nonequilibrium Thermodynamics'. In: *ICML2015*. Ed. by Francis Bach and David Blei. Vol. 37. PMLR 2015. Lille, France: PMLR, pp. 2256–2265. URL: https://proceedings.mlr.press/v37/sohl-dickstein15.html.

Weng, Lilian (2018). 'From Autoencoder to Beta-VAE'. In: *lilianweng.github.io*. URL: `https://lilianweng.github.io/posts/2018-08-12-vae/`.

– (July 2021). 'What are diffusion models?' In: *lilianweng.github.io*. URL: `https://lilianweng.github.io/posts/2021-07-11-diffusion-models/`.