Master thesis on Sound and Music Computing
Universitat Pompeu Fabra

# Disentangle and Deploy: Generative Rhythmic Tools for Musicians

Julian Lenz

**Supervisors:** Behzad Haki and Sergi Jorda

August 2023

Universitat Pompeu Fabra Barcelona

# Contents

## Acknowledgement

I wish to offer my heartfelt gratitude to four people, as this thesis simply would not exist if it were not for their continued support.

First, to my supervisors, Behzad and Sergi - thank you for your patience, your expertise, and your generosity in allowing me to take part in your research group. I am forever a better person for having the opportunity to learn from you.

Second, to my partner, Karuna - from the proof-reading, to the late-night Zelda sessions, to the words of encouragement, you were my rock. Every step of the journey, you have been by my side, and I will cherish these memories forever.

Finally, and most deeply, to my sister Maggie, and my mom, Lilli. I am the luckiest boy in the world. I love you both, so much.

## Abstract

In recent years a number of deep learning models have been developed to convert tapped rhythmic ideas into fully-voiced, dynamic drum performances. This master thesis extends the research by introducing a number of controllable features, namely *Density*, *Intensity* and *Genre*, allowing users to meaningfully augment the output whilst retaining the core rhythmic pattern identity. Our proposed models are comparatively small, enabling real-time usage on modern laptops. After trialing a number of methodologies and hyperparameters, we introduce our final model: **VAEDER** (Variational Autoencoder for Disentangled Expressive Rhythms). In addition to the model development, we introduce a number of open-source software packages that allow researchers to quickly deploy symbolic generation models into Digital Audio Workstations. We hope that this will enable a new level of participation and collaboration between researchers and musicians in the field of artificial intelligence for music generation.

Keywords: Drum generation, tap2drum, symbolic music generation, deep learning

# Chapter 1

# Introduction

This master thesis is dedicated to the creation and deployment of a rhythm-generation system that can empower musical creativity in both studio and live-performance environments. To achieve this, we build upon the works of GrooVAE[1] and GrooveTransformer[2] with the objective of improving musical *controllability* of the outputs. Specifically, our model allows users to specify the density, intensity and genre - enabling them to sculpt the rhythms to their bespoke artistic requirements. This research serves as an opportunity to advocate for *human-in-the-loop* development processes for generative AI. In addition to the system itself, we propose several methods to more easily deploy symbolic deep-learning models, with the hopes that our work will contribute to a more robust feedback cycle between researchers and musicians.

We invite you to watch a short video demonstrating VAEDER:

`https://youtu.be/v6VtPNv7cXI?feature=shared`

The model code, pre-trained checkpoints, and evaluation tools are available here:

`https://github.com/behzadhaki/GrooveTransformer`

A selection of MIDI and audio renderings can be accessed here:

`https://github.com/behzadhaki/GrooveTransformer/tree/dev/VAE_Control_Classifiers/demos/vaeder`

## 1.1 Motivation

We have a deep love and fascination with *rhythm*, as it is one of the most universal elements of music. Cultures around the world have developed incredible rhythmic systems, each with their own distinctive patterns and rules. Nearly every modern musical genre is built upon a rhythmic foundation, from the swing of jazz to the double kick drum in heavy metal.

Symbolic music generation systems, which are further defined in section 2.2, provide an exciting frontier to further explore the endless possibilities of rhythm. As they deal with representations of sounds, symbolic AI models are often lightweight, require less training data, and can be deployed directly in music-creation software environments.

We believe that, in order for such models to become a valuable part of a musician's toolkit, they should be *controllable.* Many models, such as the aforementioned GrooVAE[1] and GrooveTransformer[2] have a *one in, one out* system; if the output does not meet your needs, the only option is to re-generate, perhaps dozens of times, in the hopes of stumbling upon something more desirable. Simply put, one cannot just say, "do that again, but a bit quieter".

The central motivation of this thesis is to create a rhythm generation model that is meaningfully controllable in order to increase its practical application for music creation tasks. Creatively speaking, we hope that such a model is joyful to interact with, and can provide threads of inspiration which can be woven into a greater compositional tapestry. As opposed to replacing creativity, we wish to provide a new medium of exploration.

Throughout the research and development process, we discovered a consistent barrier: deployment. Interacting with existing symbolic models often required hours of troubleshooting old code notebooks, or downloading cherry-picked files. As we wanted to give musicians the opportunity to interact with our own model, we had to spend a significant portion of time developing a multi-threaded C++ application. We believe that the time and technical proficiency needed is preventing a number of existing models from being shared with creative communities.

With the above in mind, we developed a secondary motivation for this thesis: making deployment easier. We hope that, by developing and releasing a number of open-source tools, our work will provide beneficial to other researchers working on symbolic generative AI tasks.

We also just want something fun to jam with.

## 1.2 Objectives

We aim to develop a deep-learning model that can consistently provide appealing rhythmic ideas whilst allowing a user to fine-tune its outputs. We also wish to deploy this model in Digital Audio Workstations (DAW) and make it easier for future researchers to deploy their own models. Therefor the research conducted throughout the thesis has two concrete objectives:

**Disentanglement** When collaborating with musicians it is often helpful to 'nudge' their creations; picture a conductor who directs her string section to play "a little softer", or a jazz guitarist who asks the drummer to play "with more syncopation". These instructions presume that the next iteration will be similar to the previous performance, albeit with a minor modification to the intended metric. We aim to develop a musically-controllable rhythm generation system which mimics this feedback-loop by creating a *feature-invariant latent space*, a concept that is further described in 2.3.

**Deployment** We acknowledge that it is technically difficult to deploy deep-learning models in the environments that musicians are accustomed to, namely DAWs. With the recent explosion of generative AI systems there has been a growing acknowledgement that researchers need to actively incorporate the feedback from non-technical end-users throughout all stages of development. Among other things, this can help with the mitigation of serious issues such as bias, harmful content, and economic displacement. We therefor present several systems to more easily deploy symbolic models directly into modern DAWs, with the hope that they will lead to more dialogue between researchers and musicians.

# Chapter 2

# State of the Art

The work of this thesis builds upon a rich history of research and design in computational systems for symbolic music generation. The advent of deep learning in particular has allowed for better modeling of intricate temporal relationships, a capacity that is critical to the domain of music. While the goals, methods and implementation details are widely varied, there has been a consistent increase in complexity and output capability. In this state of the art, a brief overview of the most relevant architectures will be presented, followed by a review of the most recent advances of score and performance generation systems, and finally, a deep-dive into proposed methods of controllability for deep learning models.

## 2.1 Architectures

As the goals of deep learning research have grown in complexity and size, a number of architectures have been proposed to address specific challenges. While this section should not be viewed as a comprehensive review of all existing architectures, we hope that it will provide a beneficial overview overview of the most commonly utilised methods for constructing deep learning models for symbolic music generation tasks in recent years, with particular focus on those that provide the building blocks of the final model.

### Feed Forward Networks

A neural network, the fundamental building block of deep learning models, utilises linear regression to model the relationship between a scalar variable $y$ and one or more input variables, which can be represented as a vector $x$. In multivariate regression, the modeled output $h(x)$ can be represented as:

$$h(x) = b + Wx$$

Here, $b$ is is the bias vector, and $W$ is the weight matrix. The configuration of these weight and bias variables will determine the output of the model. Through a process called *backpropagation*, we calculate the gradients of these variables with regard to the *loss* (the difference between ground truths and predictions) and adjust the variables accordingly. This process is run iteratively, gradually updating the weights and biases with each calculation in an attempt to reach the *global minimum* distance between $y$ and $h(x)$.

In order to formulate a typical feed forward network (FFN), an *activation function*, such as Sigmoid, Tanh, or Rectified Linear Unit (ReLU) is applied to the output of each node to introduce nonlinearity. It is then possible to chain together several iterations of these calculations, as demonstrated in figure 1, with the intermediary layers between the input and output nodes referred to as *hidden layers*.

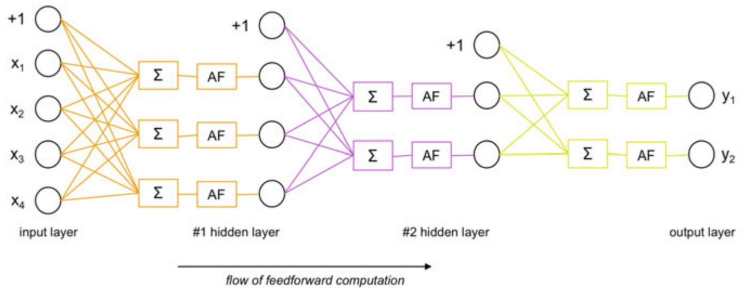


**Figure 1:** An example of a feed forward network with two hidden layers. Reproduced from [3].

## Recurrent Neural Networks

Whilst a feed forward network can predict static models, its design prevents it from modelling the dynamics of a system that has outputs that vary over time. This is vital in contexts such as natural language processing (NLP) and music, where the prediction of a single token should influence subsequent predictions. To this end, the Recurrent Neural Network (RNN) allows us to compute temporal information due to their autoregressive structure. In its most basic format, an RNN is an FFN that continually re-feeds its previous hidden state as an input for the following calculation. This means that, for time $t$, the models input will be $x_t$ as well as the previous hidden state of $h_{t-1}$ in order to predict $o_t$. This formulation allows it to take the context of prior outcomes (as well as new inputs) into consideration of its current calculation.

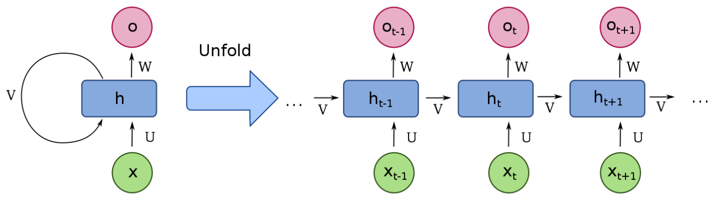


**Figure 2:** A typical RNN model, courtesy of Wikipedia.

### Gated Recurrent Unit

Although the vanilla RNN is still a widely popular model, it often suffers from the "vanishing gradient" problem. When sequence lengths are long (which is often the case in musical contexts), the gradients that are back-propagated can often vanish (become close to zero) or explode (become very large). This results in weight updates that have negligible or harmful impacts. Thus, as sequence lengths grow, RNNs lose the capability to learn meaningful representations of the data.

The Gated Recurrent Unit (GRU), which is visualized in figure 3, was proposed as one solution to this issue. The GRU is an RNN with the addition of an **Update Gate** $z$, which determines how much past information should be passed along to future calculations, and a **Reset Gate** $r$ which decides how much past information to "forget".

Each gate is an FFN, with the calculations as follows:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t] + b_z$$

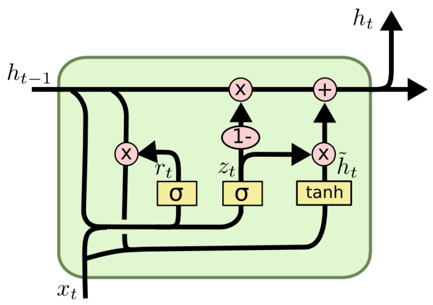


**Figure 3:** Standard GRU cell architecture. Reproduced from [4].

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r$$

The previous hidden states $h_{t-1}$ are subsequently multiplied by these matrices in separate operations, determining how much information is utilised in the current $O_t$ calculation, as well as for future operations. With these learnable gating operations, the model is able to selectively discard information of low importance, thus helping it retain quantitative focus on the most important elements in longer sequences.

### Long Short-Term Memory

Similar to the GRU architecture above, Long Short-Term Memory (LSTM) networks are designed to improve on long-term dependency modelling. To accomplish this, a continuous hidden state $C_t$ is introduced, which can be understood as the hidden state $h_t$ with further gating mechanisms. A typical LSTM cell architecture is displayed in figure 4.

With each iteration, the cell takes inputs $x_t$, $h_{t-1}$ and $C_{t-1}$ and modifies them with the following three gates:

- **Forget Gate ($f$)**: Determines how much of the previous cell state $C_{t-1}$ should be retained for current and future calculations.
- **Input Gate ($i$)**: Determines how much the new candidate cell state $\tilde{C}_t$, which is a function of the current input $x_t$ and previous hidden state $h_{t-1}$ should be added to the previous hidden state $C_{t-1}$, thus transforming it into $C_t$.
- **Output Gate ($o$)**: Determines which parts of the new cell state $C_t$ should be output as the hidden state $h_t$, using $x_t$ and $h_{t-1}$ to make this decision.

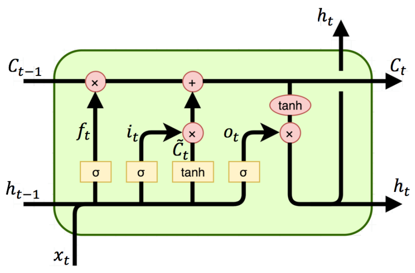


**Figure 4:** Standard LSTM cell architecture. Reproduced from [5].

The cell state $C$ can pass through long sequences with relatively minor modifications at each step, allowing it to retain a *memory* of the important elements. Whilst

LSTMs typically offer the superior performance of the three architectures, they also have the highest number of learnable parameters, thus requiring increased computing power, more data and longer training runs.

### 2.1.1 Transformers

In the seminal work *Attention is All You Need*[6], the authors proposed a new network, called the Transformer, which was initially designed for machine translation tasks. The paper demonstrated state of the art performance in several translation benchmarks by leveraging a new method called Self-Attention, in conjunction with positional embeddings, to altogether remove the need for sequential processing. By processing entire sentences at once the model benefits from parallelization and eliminates the long-term context issues detailed in the previous section.

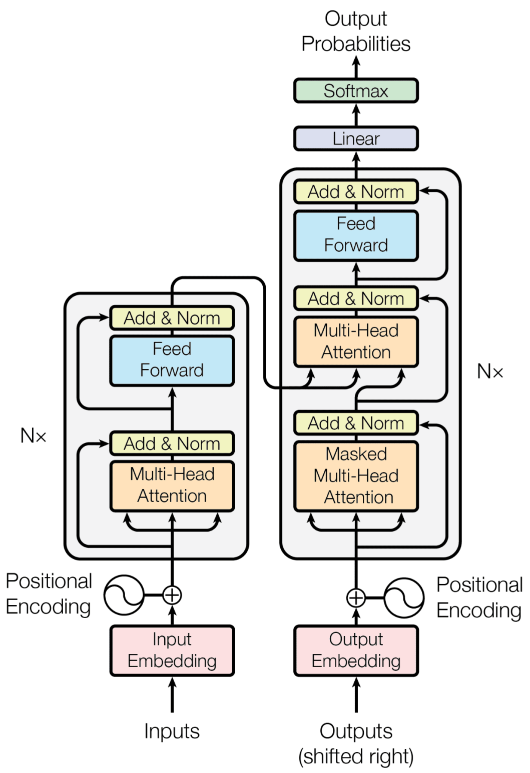


**Figure 5:** The full transformer network with encoder and decoder networks. Figure from [6].

#### Positional Encoding

The non-sequential architecture of Transformers means they possess no inherent knowledge as to the position of a given element within a sequence. "The quick brown fox" could equally be processed as "Brown quick fox the". To solve this, researchers add positional encoding values to each element at the beginning of the process. These values are calculated with sine and cosine functions for even- and odd-values, respectively.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

Intuitively, this results in the model adding increasingly larger numbers to each element, which is enough information for the model to infer the positions of these elements.

It should be noted that efforts have been made to improve upon the standard Positional Encoding function, such as in [7], where the authors create relative positional representations and a modified *Relation-aware Self-Attention* to better model the relative distance between each element. This technique was further improved upon in Music Transformer [8], which proposed that efficiently modelling relative positional distances is key to performance-generation systems. Whilst outside the scope of this thesis, we see the application of alternative positional encoding functions in the context of rhythm-generation tools as an exciting area for future research.

**Self-Attention**

The self-attention mechanism, also known as *scaled dot-product attention*, is the key ingredient of Transformers. This mechanism operates by generating three distinct matrices for a given set of input vectors: the query matrix $Q$, the key matrix $K$, and the value matrix $V$. Each of these matrices is created by multiplying the input vectors by their respective, learnable weight matrices. To determine the relevance or "attention" between different elements in the sequence, the dot-product of $Q$ and $K$ is calculated. This attention matrix is then divided by the square root of the dimension of the key vectors $d_k$ (to prevent large values and unstable gradients) and scaled with softmax between 0 through 1. The result is then multiplied with the value matrix, thus creating an effective representation of the relevance of each other element to a single element in the sequence. Formally, the self-attention mechanism for a sequence is calculated as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

### Multi-head Attention

Multi-Head attention is another key component which allows the model to selectively focus different attention computations on various patterns in the data in parallel. This is achieved by projecting the embeddings into $N$ embedding spaces, each forming an *attention head*. For each attention head, the $Q$, $K$ and $V$ matrices are computed independently, and the attention mechanism is calculated. After the Self-Attention is calculated (as per the equation detailed in the previous section) for each head, each attention value is concatenated and linearly transformed to result in the final output of the multi-head attention layer. Similar to how Convolutional Neural Networks (CNNs) can learn to focus on different spatial details in an image, Multi-Head Attention allows the model to focus on different features of the pattern, such as the note selection, timing or velocities.

### Encoder

Pictured on the left-hand side of Fig. 5, the encoder of a Transformer is a powerful architecture within its own right. After embedding the sequence with positional encodings, it is propagated through $N$ layers. Each layer consists of two primary elements: Multi-Head Attention and a standard Feed Forward Network. Residual connections are added between each calculation stage, allowing some information to pass through relatively unmodified, which solves the issue of vanishing gradients and enables deeper networks. The process is repeated through $N$ layers. The outputs can provide a number of task-dependent functionalities: the $Q$ and $K$ vectors for a decoder (as in the original paper), a fully-fledged output (as in [9]), or a set of $\sigma$ and $\mu$ matrices for sampling. The encoder is non-auto-regressive, which makes it particularly applicable for real-time applications due to its faster inference speeds.

### Decoder

The transformer's decoder has the same basic structure as the encoder, but with an additional sub-layer that performs multi-head attention over the encoder's output. The first multi-head attention layer, which is identical in design to that of the encoder, is processed with a triangular mask that prevents the attention at element $i$ from attending to the $Q$, $K$ and $V$ values beyond $i+1$. This mimics auto-regressive functionality, thus teaching the model to predict the next token with only knowledge of the current and previous tokens. The output of the Masked Multi-Head Attention sub-layer is understood as the $V$ matrix, and combined with the $Q$ and $K$ outputs from the encoder, fed through an additional sub-layer of Multi-Head Attention.

Whilst the masking allows for parallelization during training, the inference process of a decoder requires true auto-regression, in which each output is fed back as the input for the next token. For rhythm-generation, it was hypothesized by [9] that a transformer could make reasonable predictions with knowledge of all elements in the sequence. Based upon these results, this thesis continues to utilise only the encoder module, thus allowing for faster inference times and enabling real-time interaction.

### 2.1.2 Autoencoders

Autoencoders have become increasingly popular in generation tasks due to their ability to learn latent representations of complex data. In its purest form, an autoencoder consists of an encoding function $f(x)$ and a decoding function $g(f(x))$. This middle layer, often referred to as the *latent space*, is created by ensuring that the output dimensionality of $f(x)$ is identical to that of the input to $g(f(x))$. The goal of the autoencoder is to accurately reconstruct the original input $x$ through unsupervised training. One notable variation, the **sparse autoencoder**, is illustrated in figure 6. This design forces the model to learn a compressed representation of the data by ensuring that the latent layer has a lower dimensionality than the surrounding encoder/decoder layers.

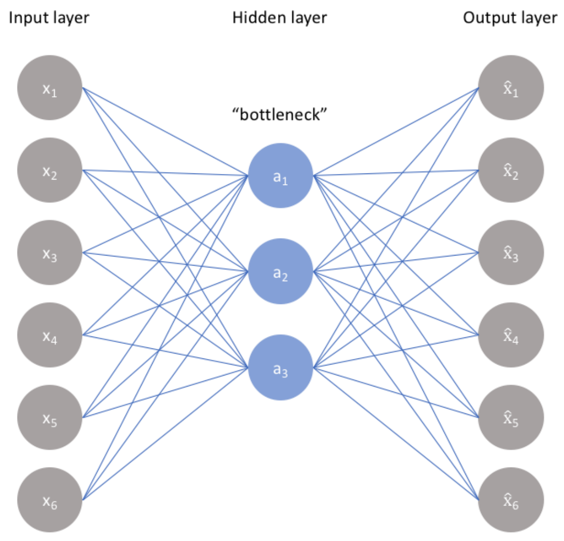


**Figure 6:** A sparse autoencoder, reproduced from [10].

#### Variational Autoencoders

Variational Autoencoders (VAE) differ from vanilla Autoencoders by creating a variational distribution $q_\phi(z|x)$ from which to sample. Concretely, instead of producing a single $z$ representation, the encoder, parametrized by $\phi$, learns $\mu$ (mean) and $\sigma$ (standard deviation) vectors to represent a multivariate distribution. This introduces an issue, as directly sampling a random variable prevents gradient calculation. This is ameliorated with the *reparameterization trick*:

$$z = \mu + \sigma \odot \epsilon$$

In which $\epsilon$ is randomly sampled from a normal distribution, thus removing the stochastic element and making the operation differentiable. To encourage a more functional latent space, a new loss function is introduced: the Kullback-Leibler (KL) divergence between $q_\phi(z|x)$ and the prior $p(z)$, which is typically a standard multivariate normal distribution, $N(0, I)$. This KL divergence term serves as a regularizer that encourages the learned distribution $q_\phi(z|x)$ to be close to the prior $p(z)$. The KL divergence is combined with a reconstruction loss from the approximate posterior distribution to produce the final loss function:

$$\mathcal{L}(\theta, \phi; x) = -\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] + \mathrm{KL}(q_\phi(z|x)||p(z)) \tag{2.1}$$

where $\theta$ represents the parameters of the decoder, $\phi$ represents the parameters of the encoder, $x$ is the input data, and $z$ is the reparametrized latent variable. Intuitively, the first half of the equation encourages the model to accurately reconstruct the input, whereas the KL term in the second half is enforcing a regularized latent space that provides more meaningful interpolations.

## 2.2 Deep Music Generation

### 2.2.1 Symbolic Generation Tasks

Broadly speaking, the goal of a symbolic music generation system is to create a *representation* of sound, based on conceptually predefined features, which can later be converted into audio through a separate process. Within this field, there are a large variety of tasks, each posing its own particular set of challenges, data requirements, and goals. A classification system by *Ji et al.*[11], detailed in figure 7, proposes four types of generation categories: score, performance, audio and fusion.As the model of this thesis is attempting to compose novel rhythms with natural characteristics, this section will briefly detail the definitions and relevant research in both score and performance generation task domains.

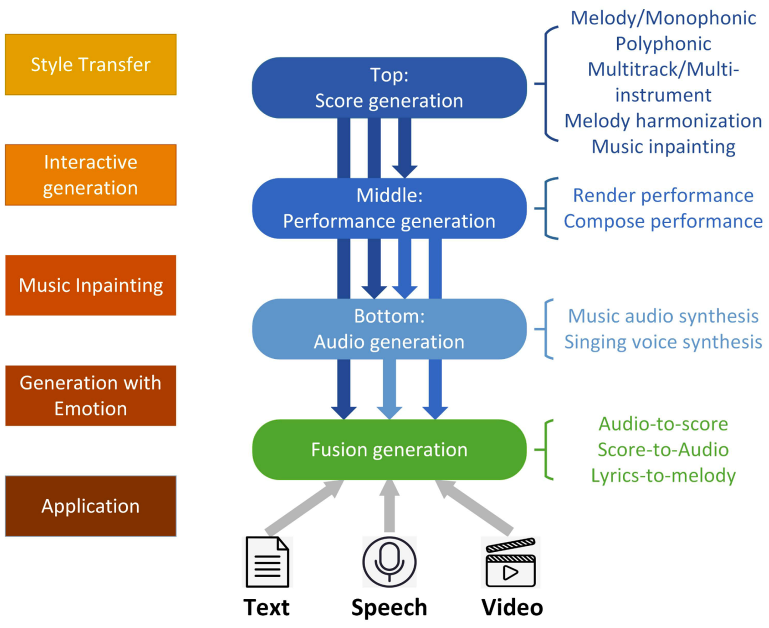


**Figure 7:** Categorization of common symbolic music generation tasks, with examples of each domain. Reproduced from [11].

## Score Generation

In a score generation model the objective is generally to create or augment musical information that is similar to that which is read and performed by musicians. Common tasks include *generation* ([12], [13], [14]) in which the model produces a novel composition; *in-painting* ([15], [16]) where it inserts missing information; and *augmentation* ([17], [18], [19]), in which the model transforms a pre-existing musical idea. For the sake of brevity we simply wish to highlight a brief set of examples here to contextualize the concept of score generation - for a more comprehensive overview of recent models and techniques, we encourage the reader to refer to the excellent work of [20].

Notably, score generation models do not provide detailed information on velocity changes (dynamics) or micro-timing. As a result the outputs will often sound *robotic* and subjectively less appealing to most listeners. It was highlighted in [21] that state of the art deep learning models, such as Magenta's MusicTransformer [8], do not show tangible improvements over older generative techniques such as Markov-based algorithms in subjective listening tests. We posit that in order for symbolic deep learning techniques to become truly beneficial to the music-making community, it is necessary to model the intricate temporal and dynamic attributes that arise from a human performance. Imagine an image-generator that could only create 6 colors or a large language model that only uses nouns; by focusing symbolic research on quantized musical information, we are placing limits on the creativity that can be achieved.

### Performance Generation

In contrast to the above, a performance model is designed to create more dynamic symbolic representations by including detailed information on velocity and micro-timing, components which can reasonably mimic the expressive nature of a human performance. Effectively encapsulating these minute details is an area of active research, and could provide a number of useful applications in the context of music creation and live performance. [11] further subcategorizes these models into those which *render* existing compositions into dynamic performances, such as CVRNN [22] and Conditional Transformer Autoencoders [23], and those which compose an entirely new musical idea. Significant advancements in this task-domain were made in 2018 with the release of both Music Transformer [8], which addressed the quadratic memory limitations of transformers on long sequences by introducing a new relative self-attention mechanism, and Transformer-NADE [24], which utilised a novel *Note-Tuple* data representation to reduce the number of tokens needed to model long, expressive performances.

Although further examples are provided in 2.2.2, it is noted by Ji et al. [11] that there is not a significant quantity of research in this domain, and there exist many exciting opportunities for future work. We further hypothesize that research in this subdomain could provide some of the most musically-utilitarian models, which is a guiding principal for the work of this thesis. To put it succinctly, our model is designed to both compose and perform rhythmic ideas in order to provide appealing accompaniments to musicians.

### 2.2.2 Rhythm Generation

Generating rhythms that can expressively accompany a track in a variety of styles is an active area of research. As drums are often interacting with other instruments in a piece of music, there are a number of proposals on how to accurately model the complex relationships between drums and other inputs. Significant contributions include [25], which utilises a convoluted gated Autoencoder to model a rhythmic *mapping code* between two input signals, as well as [26] who propose a *novelty function* to calculate whether a drum pattern should be *repeating* or *improvising* when conditioned on a melodic accompaniment. In [27] it was demonstrated that by combining a Bidirectional LSTM encoder with a Transformer-based decoder, researchers could generate drums conditioned on several accompanying tracks, as well as contextual information such as time signature and tempo.

In 2019, the seminal model GrooVAE [1] was released, which utilised several Seq2Seq LSTM architectures to create expressive drum performances. In addition to *Humanization* and *Infilling*, Gillick et al. proposed a novel *Tap2Drum* task, in which the model can convert a monophonic *tapped* rhythm into a fully-voiced drum performance. It was later demonstrated by [2] and [9] that a transformer architecture achieves similarly dynamic results in *Infilling* and *Tap2Drum* tasks - and created the foundations for the work of this thesis. Similarly, [28] used a TransformerXL in conjunction with a novel tokenization method in order to model more expressive, less-quantized drum patterns. We note that the models detailed thus far lack the capability to alter their outputs subject to meaningful creative control mechanisms, a key feature in bringing this technology into the modern music studio.

## 2.3 Controllability

The central question of this thesis revolves around how to reliably generate novel rhythmic patterns that are subjectively aligned with a given set of user-augmented control parameters. Within the context of developing machine learning applications that are designed to empower the creative process (as opposed to replacing it), controllability is a fundamentally necessary component; without it, musicians cannot shape the outputs to meet their bespoke requirements. Within the broader domain of symbolic music generation, controllability has become an increasing focus of research, with a plethora of recent studies demonstrating promising results in a variety of use-cases. For a broad perspective, we provide a general taxonomy of controllable generation methods from [29] in figure 8.

A particularly influential work in this domain was Fader Networks [30], wherein the authors proposed an Autoencoder architecture with Convolutional Neural Networks (CNNs) to generate realistic images whilst providing users with *faders* (like a mixing console) to control certain attributes. To achieve this, it is necessary for the encoder to generate an accurate latent representation that is *invariant* to the chosen parameters. The decoder must be able to then process this latent code, embedded with the specified parameters, into a realistic image. To achieve this they introduce a *discriminator* tasked with predicting the attribute from the latent space. Its loss function is therefore deployed in an adversarial fashion - the encoder is penalized in

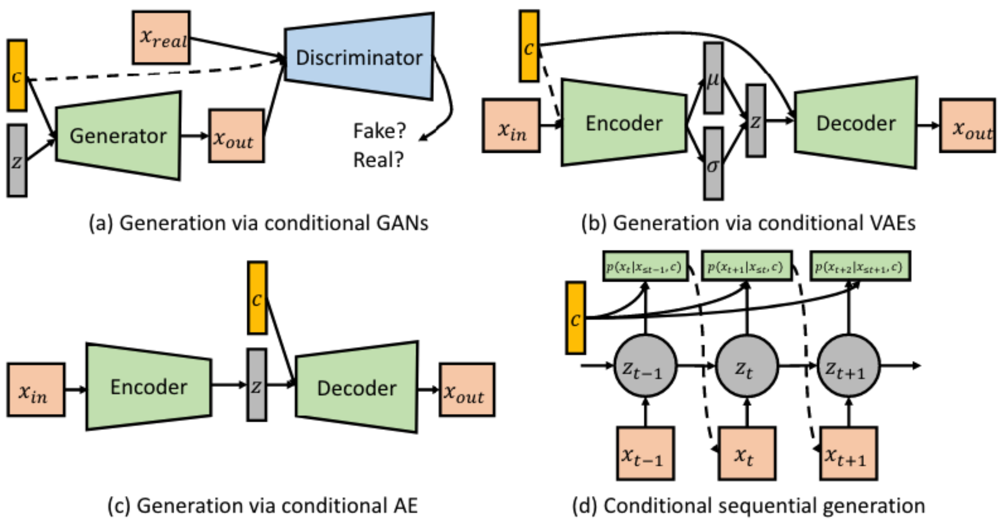


**Figure 8:** A taxonomy of controllable generation methods courtesy of [29]. Most methods for symbolic music generation fall into categories **b** and **c**; Autoencoders that in which the control parameters $c$ are injected into the Decoder stage .

proportion to the accuracy of the discriminator's capability to predict the attribute. With this technique they were able to successfully generate attribute-invariant latent representations that contained enough information for meaningful decoding, as visualized in figure 9.

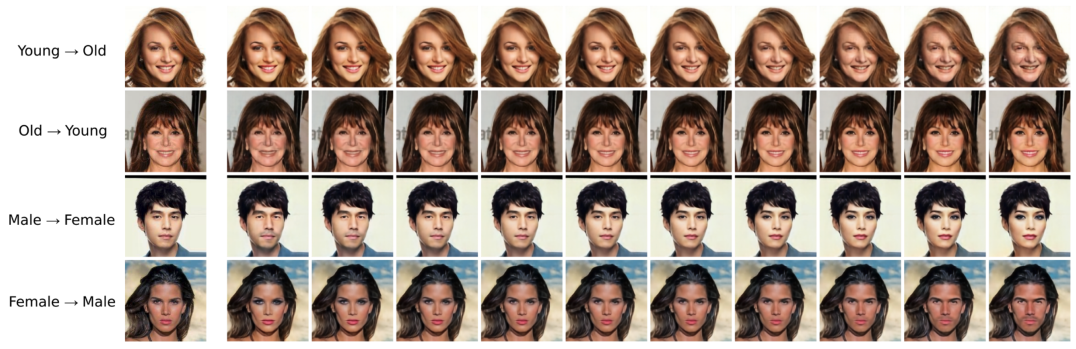


**Figure 9:** Fader Networks: By creating latent representations that are attribute-invariant, the authors of [30] can construct images that retain a core identity while responding to detailed control inputs.

In the musical realm, in 2017 it was demonstrated by [31] that it is possible to create identity-preserving variations of *pitch-class* and *note density* on an original, unconditional theme by constraining the latent space of a pre-trained LSTM VAE. This was expanded upon by [32] which adds an attribute-specific regularization loss function to the training objective, thus enabling manual modification of the regularized dimension during inference. Whilst these early results were promising, in both cases the conditioning was performed on quantized, monophonic melodies without subjective evaluations - thus making it difficult to determine their suitability for artistic applications.

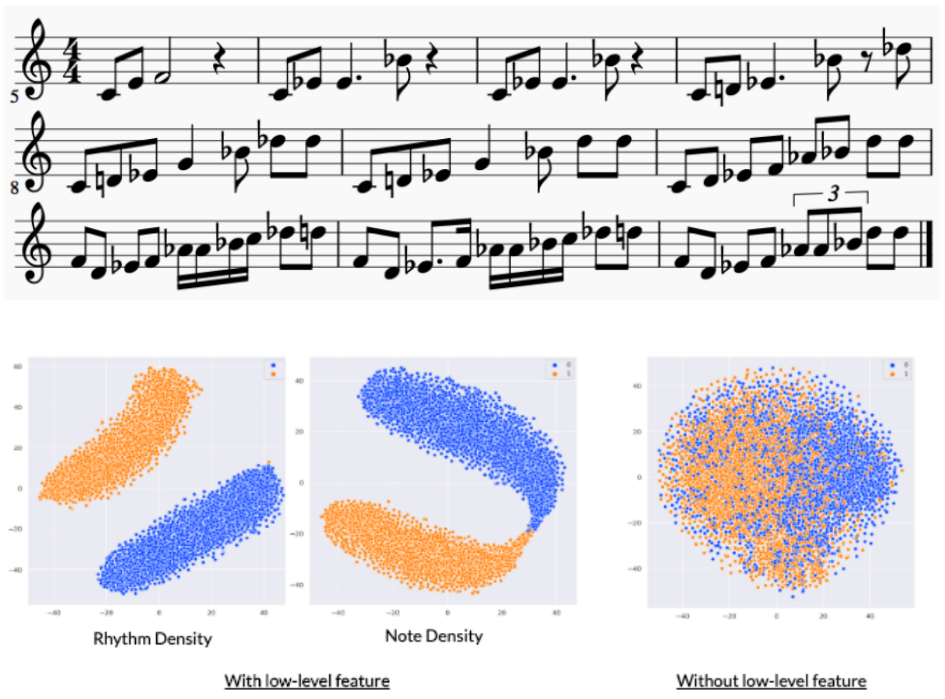


**Figure 10:** How a regularized latent space can translate to controllable music features. **(Top)** The regularized dimension of *rhythm complexity* is gradually increased in [32]; **(Bottom)** Disentanglement of rhythm and note density in [18].

Further techniques were proposed in [18], which utilised a Gaussian Mixture VAE to perform disentanglement representation learning in conjunction with regularized low-level symbolic music features. In addition to the decoder, the latent space vectors are separately passed to a discriminator model (to ensure the low-level features are present in the latent space) and a *cluster inference* model (to predict higher-level features). By specifying and enforcing a number of musicologically-grounded low-level features - *rhythm density* and *note density* - researchers could then apply semi-supervised learning to attach these to a controllable high-level attribute - *arousal*. This provided several novel advancements in the field, enabling models to map the relationship between several low-level features to higher-level descriptors, as well as reducing the necessary amount of labelled training data.

The approach of disentangling control attributes in a VAE has already been demonstrated in the context of rhythmic-generation tasks. In [33], the authors propose a *rhythm complexity* parameter measurement, and subsequently construct a $\beta$-VAE to disentangle the latent space. Detailed in figure 11, the researchers trained the

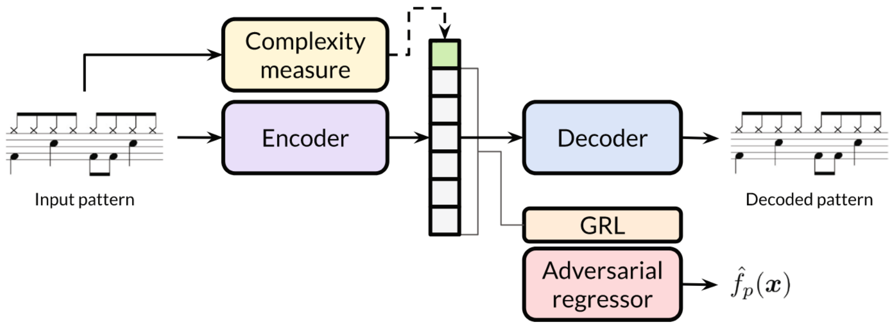


**Figure 11:** The modified $\beta$-VAE model proposed by [33] for rhythm generation utilises several additional loss functions to disentangle the control attribute. Illustration from the original paper.

model to calculate and encode this measurement into a single element of the latent space, $z_i$ whilst simultaneously disentangling it from the remainder of the vector $z_*$. This is accomplished by adding an auxiliary loss function:

$$\mathcal{L}_{reg} = \text{MSE}(f_p(\mathbf{g}), z_i)$$

Where $f_p(\mathbf{g})$ is the the calculated complexity measure, and $z_i$ is the $i$th element of the latent code $\mathbf{z}$. Subsequently, an adversarial regressor is trained to estimate the rhythm complexity on the elements of $z_*$ (those which should be disentangled), and connect its loss function to the encoder with a gradient reversal layer (GRL). This simultaneously encourages the encoder to put a high correlation between $z_i$ and *rhythm complexity* whilst removing this information from the other elements. Although this was demonstrated in objective evaluations to effectively regularize and disentangle the control function, the model was exclusively trained on drum patterns in a 4/4 meter, with 16$^{\text{th}}$-note quantization, no micro-timing, and all velocity information removed.

In a similar approach, the authors of [34] rely on a discriminator mechanism to manually disentangle a number of calculated control-values from the latent space. After encoding a monophonic melody, the model's latent space is separately passed to a multivariate discriminator, whose training function is designed to identify pre-calculated characteristics such as *amount of arpeggiation*, *pitch kurtosis*, and *rhythmic value variability* (among many others). The parameter vectors are then concatenated with the hidden state, thus enabling the decoder to draw upon this information for accurate reconstruction of the target input. In contrast to [33], the Encoder is not designed to make any prediction of the target attribute(s). Furthermore, by

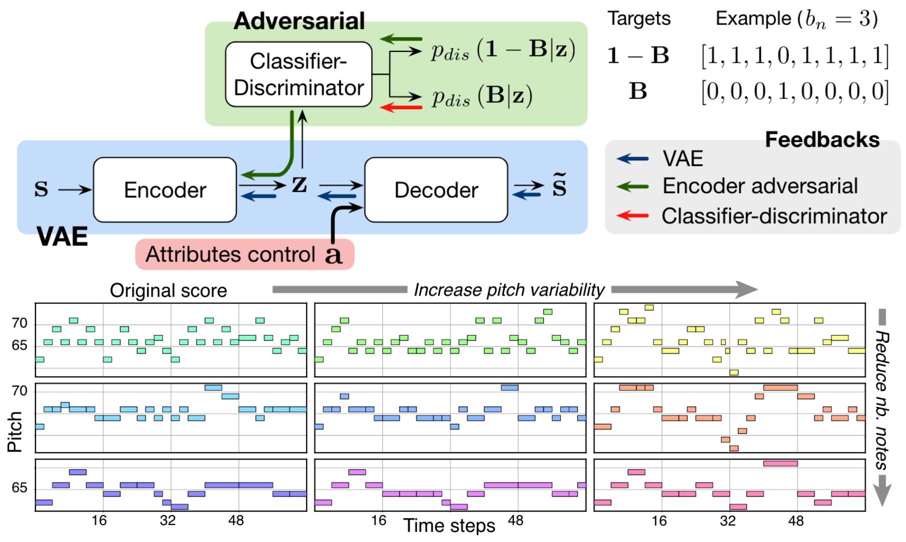


**Figure 12:** By utilising a multivariate discriminator in the latent space, [34] is able to influence the output with several conditions simultaneously. Illustrations from the original paper.

converting each attribute's continuous value to a one-hot encoding and framing the discriminators as a classification task, they are able to introduce multiple parameter controls into a single model similar to [30].

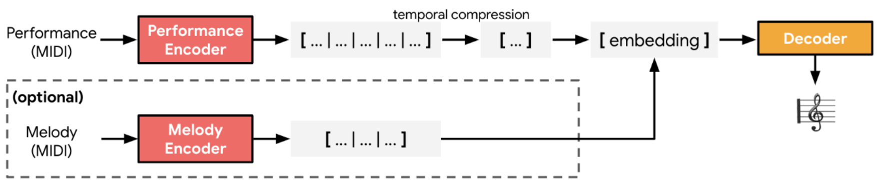


**Figure 13:** The architecture of [23] combines the hidden states of two Music Transformer[8] encoders prior to decoding in order to alter the style of an input melody performance. Illustration from the original paper.

Recent work has been conducted to understand the potential benefits of combining the self-attention capabilities of transformer modules with the information bottleneck produced by VAEs. For example, [23] uses a modified transformer network in order to perform *style transfer* to piano performances; that is, re-processing a melody in the style of another performer or composer. As a foundation, they utilise an encoder and decoder from Music Transformer[8] (thus benefiting from *relative* attention) to recreate an input melody, with an information bottleneck to ensure the decoder is not simply copying the input. They separately train another (structurally-identical) encoder to temporally identify the performance characteristics, the output of which is then combined with the output of the first encoder (with either summation, concatenation, or tiling), and then fed to the decoder.

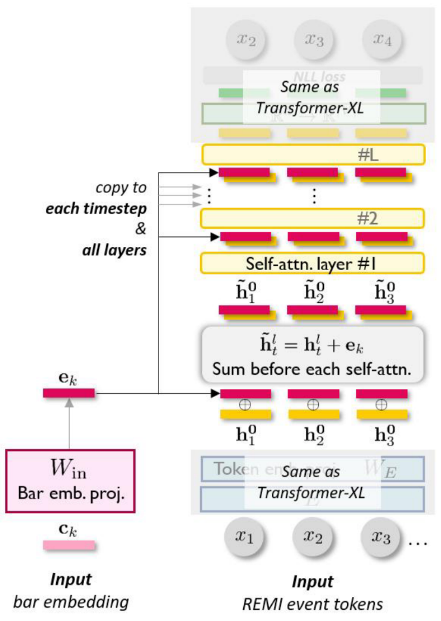


**Figure 14:** The *in-attention conditioning* utilised by Musemorphose to achieve bar-level attribute control. Illustration from [35].

While we have highlighted a number of methods to disentangle latent spaces, it is also important to investigate techniques to then inject the desired parameters into the decoder. In this regard, the authors of Musemorphose [35] experimented with several methods of injecting bar-level conditioning mechanisms into a Transformer-XL [36] decoder, demonstrating particularly strong results with a novel *in-attention* conditioning technique. Detailed in figure 14, this was achieved by repeatedly *reminding* the decoder of the desired control signals by summing the embeddings with the hidden states prior to each self-attention layer (except for the last one). This decoder (which is intended to process the entire generation) is combined with a number of parallel bar-level transformer encoders, whose hidden states are concatenated with two control-mechanism embeddings - *rhythmic intensity* and *polyphony* - and then fed to the decoder via the aforementioned *in-attention conditioning.* This technique allowed the authors to create a style-transfer model with bar-level attribute control, which enables musicians to sculpt dynamic control curves over the course of a sequence generation.

In this section we have provided an overview of symbolic music generation systems, with a specific emphasis on rhythm and controllability. It has been highlighted that many of these models do not contain micro-timing or velocity information, limiting their musical practicality. For the models that do render expressive performances, they lack the capability for meaningful control inputs. We therefore aim to fill a specific yet important gap in the current research: a rhythmic generation model that is both expressive and controllable.

# Chapter 3

# Methodology

In this thesis, we train a Transformer VAE on a repository of symbolic representations of drum performances. The dataset, architecture, training methods and evaluation metrics will be detailed in the following sections. The code repository is fully open source and can be accessed here.

## 3.1 Dataset

All training was conducted with the Groove MIDI Dataset (GMD)[1] by Google Magenta. It contains roughly 13.6 hours of human performances, within 1,150 MIDI files representing 22,000 measures. The ten drummers were recorded on a Roland TD-11[1] electronic drum kit. While there are a number of large repositories with rhythmic information, such as Lakh-MIDI[37], the majority of these provide only *score* information with quantized timing and no velocity information, thus limiting their suitability for training performance generative systems. The dataset contains annotations specifying the drummer, genre and a "beat" or "fill" classification. We further filter the GMD to train specifically on samples of 4/4.

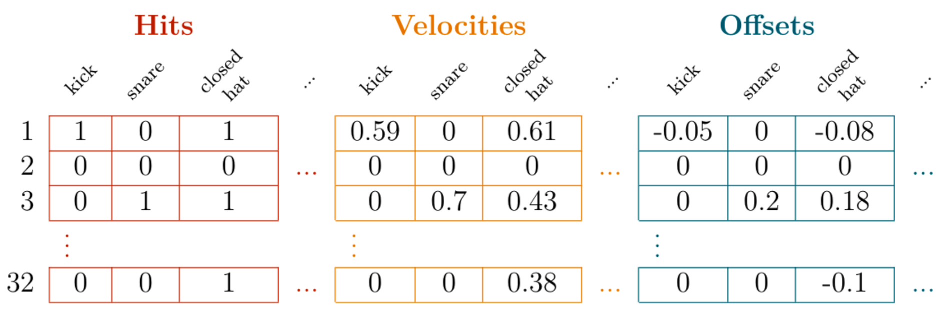
### 3.1.1 Data Representation

In line with [1] and [2] we represent each drum loop with the "Hits, Velocities, Offsets" (HVO) system. The HVO representation, which is visualized in figure 15, divides time into discrete indices, with each drum voice assigned three elements:

[1]https://www.roland.com/global/products/td-11/

- **Hit**: (0 or 1) Whether a note is played at this time-step
- **Velocity**: (0.0 - 1.0) Velocity of the note, normalized
- **Offset**: (-0.5 - 0.5) Displacement of the timing from a perfectly quantized center of 0.0

In our case, time is divided into four indices per quarter note (16th notes). The HVO method allows for a careful trade-off between the granularity necessary for performance systems and the quadratic memory complexity of Transformers in relation to sequence length.



| | Hits | | | | Velocities | | | | Offsets | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | kick | snare | closed hat | ... | kick | snare | closed hat | ... | kick | snare | closed hat | ... |
| 1 | 1 | 0 | 1 | | 0.59 | 0 | 0.61 | | -0.05 | 0 | -0.08 | |
| 2 | 0 | 0 | 0 | ... | 0 | 0 | 0 | ... | 0 | 0 | 0 | ... |
| 3 | 0 | 1 | 1 | | 0 | 0.7 | 0.43 | | 0 | 0.2 | 0.18 | |
| | ⋮ | | | | ⋮ | | | | ⋮ | | | |
| 32 | 0 | 0 | 1 | ... | 0 | 0 | 0.38 | ... | 0 | 0 | -0.1 | ... |

**Figure 15:** An example of the HVO representation for 3 of the 9 drum voices, reproduced from [2].

### 3.1.2 Tapped Sequence

Our aim is to transform an unvoiced "tapped" sequence into a 9-voice drum kit performance, defined as the Tap2Drum task in [1]. To accomplish this, we *collapse* each training sequence into a single voice, as demonstrated in figure 16. In the case of multiple voices on a given time index, we select the offset and velocity values corresponding to the hit with the highest velocity. The model is then trained to predict a fully-voiced drum pattern based on this tapped sequence by using the reconstruction loss functions described in section 3.6.

This approach allows a trained model to take virtually any MIDI information as input. For example, a piano melody that contains dozens of notes can be reduced to a single tapped representation. This becomes an effective way to build the rhythm generation model as an accompaniment device, where it can produce predictions regardless of the input instrument type.

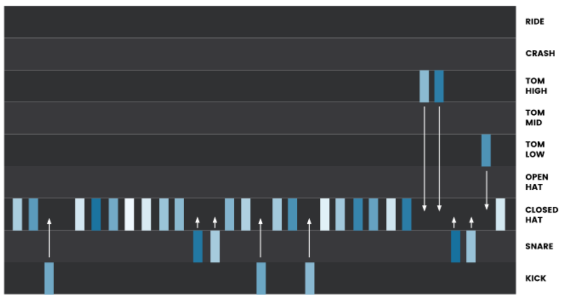


**Figure 16:** Visualisation of a drum performance being converted to a tapped sequence, reproduced from [2].

## 3.2 Control Parameters

We propose several metrics that are functionally beneficial from a musical perspective and can be quantitatively defined for unsupervised training.

### Density

Density can be understood as the overall quantity of notes present in the performance. An experienced drummer can alter the density of their performance while retaining the core rhythmic identity in relation to fixed metrics such as genre and accompanying parts. Modulating the density allows one to express certain emotional indicators, such as a reduction in intensity or an imminent section change. Let $H$ represent the number of active hits in the pattern, $T$ be the number of time-steps, and $N_v$ be the number of drum voices:

$$\text{density} = \frac{H}{T * N_v} \tag{3.1}$$

### Intensity

Intensity is calculated as the average of all velocity values within each rhythmic pattern. Considering that velocity values are initially normalized between 0.0 - 1.0, let $V$ represent the sum total velocities of a given pattern:

$$\text{intensity} = \frac{V}{H} \tag{3.2}$$

It has been noted that this may not correlate directly with a listener's perception of intensity; a single crash cymbal at medium velocity may induce a higher sense of

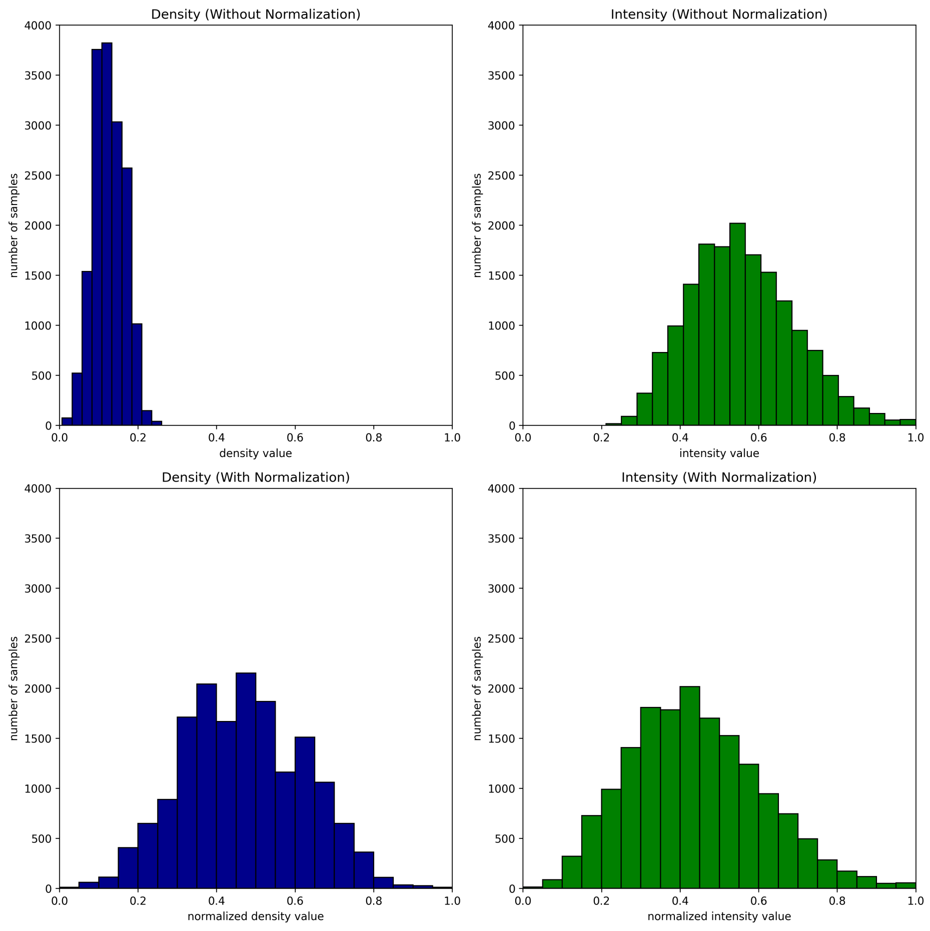


**Figure 17:** Distribution of measured *Density* and *Intensity* values, before and after normalization.

intensity than a kick or closed hi-hat at maximum velocity. As this thesis is more focused on the methodology of disentanglement, we leave this tantalizing question to the work of future research.

Drum patterns within the GMD have *Density* and *Intensity* values tightly clustered around a small range. We therefore implement normalization on both continuous values, to represent the lowest and highest values present as 0.0 and 1.0, respectively. We detail in figure 17 the distribution of density and intensity values before and after normalization.

### Genre

We rely on the genre labels provided with the GMD and detailed in section 3.1. The genre input is represented as a vector $g$ which is sized in proportion to the number of genres in the dataset. Each element of $g$ corresponds to a single genre, and we provide a mapping dictionary embedded within each model for easy reference.

Whilst training $g$ is encoded as a one-hot vector with the relevant element set to 1. During inference it is possible to have multiple genre elements set to a value between $0 - 1$, which opens exciting creative use-cases of generating hybrid rhythms.

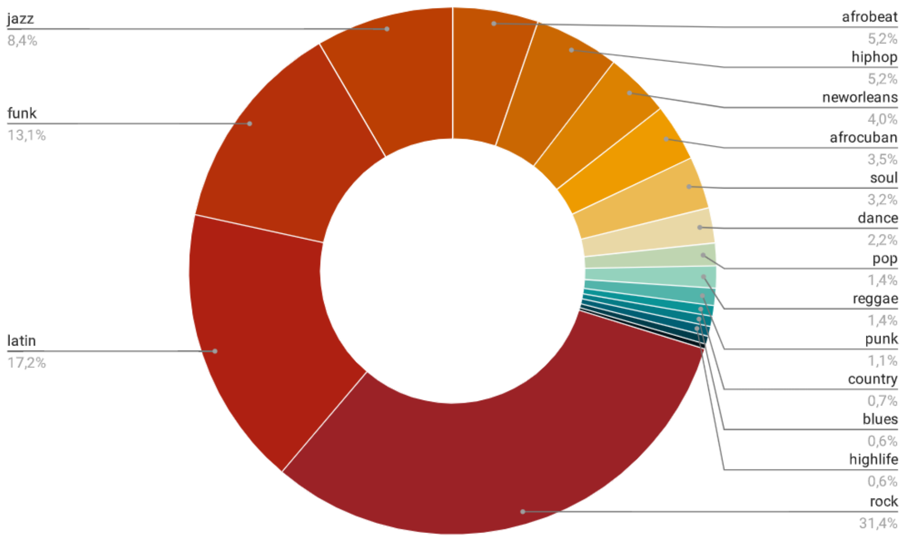


**Figure 18:** Distribution of genres in the Groove MIDI Dataset.

## 3.3 Model Architecture

The goal of **VAEDER** is to convert a tapped monophonic input rhythm into a fully-voiced drum performance whilst incorporating a number of user-specified control parameters. Our proposed architecture is a VAE, with Transformer encoders serving as the backbone of both the encoding and decoding process. We provide a general overview of the architecture in figure 19.

### 3.3.1 Encoder Layer

The encoder layer is tasked with taking the initial HVO input and creating a latent representation. First, in the **Encoder Input Layer** the HVO matrix is expanded through a learnable matrix $W_{input}$ to $d_{model}$, as well as a ReLU activation and summed with a positional encoding per [6]. Subsequently, this processed input is directed through the **Encoder** block, a composite of $n$ successive layers of Transformer encoders. The resulting output of these layers, maintaining a shape of $d_{model}$ is finally passed through the **Latent Layer**. In adherence to conventional VAE methodology, it is projected by two learnable matrices, $W_\mu$ and $W_\sigma$ to create the mean and standard vectors. We then utilise the reparameterization trick to sample from this multivariate space, generating the $z$ latent vector for the Decoder.

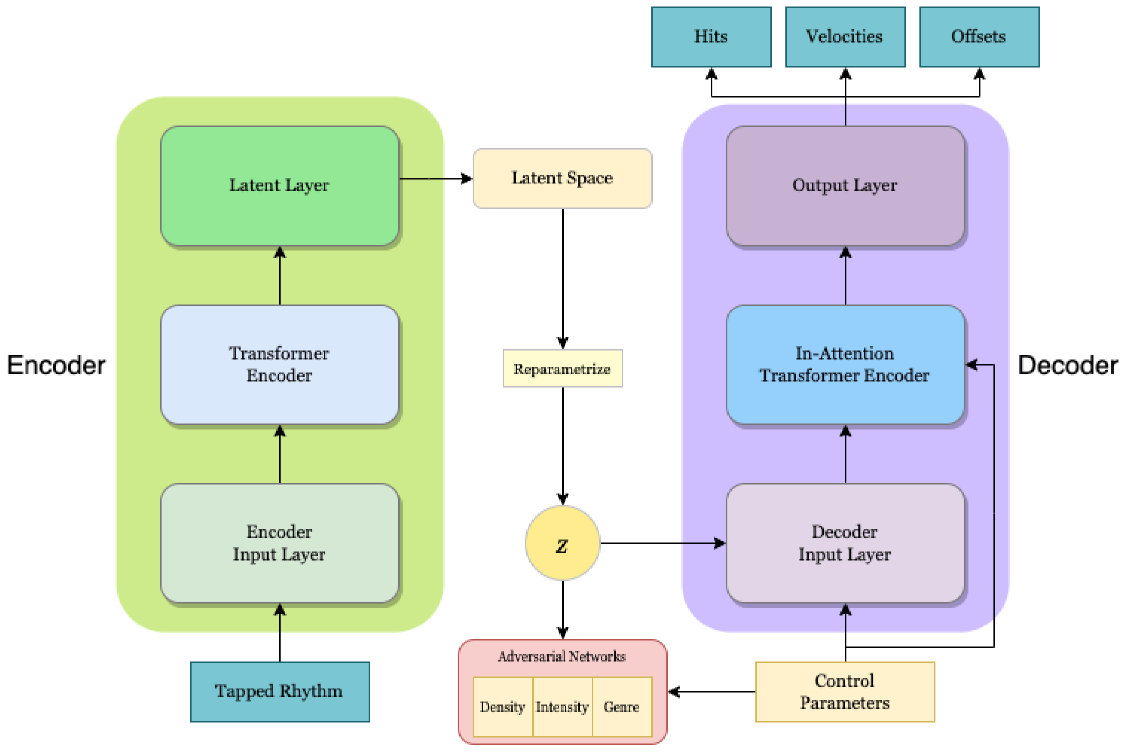


**Figure 19:** An overview of the VAEDER model architecture.

### 3.3.2 Decoder Layer

The goal of the decoder is to take the sampled $z$ vector, as well as the desired control parameters, and generate HVO logits that can be sampled into a MIDI drum performance. This process begins with the **Decoder Input Layer**, which combines $z$ with the *density*, *intensity* and *genre* vectors with the **Pre-Decoder** method described in section 3.5.1, resulting in a new latent matrix of $Z \in \mathbb{R}^{t,d}$ where $t$ is the number of discrete time indices (32 in our case) and $d$ is the dimensionality of the decoder model. We then process this through the **Decoder In-Attention** stage, in which $Z$ is summed with the parameter vector $p_{in}$ and fed through Transformer Encoder layers as detailed in section 3.5.2. The output of this process is then multiplied with output matrix $W_{out}$ to produce our output logits $HVO \in \mathbb{R}^{t,v\times3}$ in which $v$ represents the potential number of voices in our modelled drum performance. Thus, we have an element representing the *Hit*, *Velocity* and *Offset* for each voice.

In order to convert the logits into usable values, we split and process the H, V, and O matrices through sigmoid layers. The hits, which can only be represented as 0 or 1, are calculated as a binary result of a threshold, typically set to 0.5. The offsets are reduced by 0.5 so as to constrain their range between -0.5 to 0.5.

## 3.4 Adversarial Networks

In an ideal model, the encoder is able to create a latent representation of a drum pattern without any tangible information regarding the control parameters. This introduces a paradox: with the reconstruction loss alone, the encoder is incentivized to generate a descriptive $z$ vector which will be decoded into an accurate ground-truth recreation. Drawing upon the work of [30], [34] and [33] we introduce a system of adversarial networks to penalize the encoder when it includes this information in the latent space.

For each control parameter we create a corresponding adversarial network, which is tasked with predicting the specified parameter $p$ from $z$. All three parameters are treated as classification tasks; for the continuous parameters of *Density* and *Intensity* we quantize and convert the value to a one-hot encoding in a 10-element vector. While we experimented with a number of methods, this approach proved most reliable, as it ensured that the three networks would produce similar loss values.

Each model is composed of two hidden feed-forward layers that correspond in size to the *latent dimension* (which is a tunable hyperparameter) followed by Tanh activations. The output layer is an additional feed-forward layer that projects to $n$ *classes* with a sigmoid activation. The individual loss functions encourage these models to make accurate predictions of their respective parameter from the latent space, a task that is aided by an encoder that encodes significant information into $z$. In a process detailed in 3.6, we combine the adversarial losses in a Gradient Reversal Layer (GRL) and apply this to the encoder. Therefore, the encoder is now incentivized to remove parameter information from the latent vector.

## 3.5 Parameter Injection

Assuming a properly disentangled latent space, it becomes necessary to inject the parameter information into the Decoder. With little known research on disentanglement for performance rhythm generation systems, we experiment with two methods at separate stages of the decoding process, referred to as *Pre-Decoder* and *Decoder In-Attention*.

### 3.5.1 Pre-Decoder

We propose a novel method which can incorporate both continuous and categorical control parameters into a disentangled vector $z$ prior to decoding.

We first multiply $z$ with a learnable matrix $W_{latent} \in \mathbb{R}^{t\times(d-n)}$ and transform it to obtain a new matrix $Z_* \in \mathbb{R}^{t,(d-n)}$ where $t$ denotes the number of time indices, $d$ is the decoder model dimensionality and $n$ is the number of parameters we wish to embed. The continuous values of *Density* and *Intensity* are repeated by $t$ steps, and the categorical vector corresponding to *Genre* is multiplied through matrix $W_{genre} \in \mathbb{R}^t$. Each of these vectors has an additional dimension added, thus producing $n$ parameter matrices $P \in \mathbb{R}^{t,1}$. We then concatenate our $Z_*$ matrix with the individual parameter $P$ matrices along the final dimension, producing the output matrix $Z \in \mathbb{R}^{t,d}$ for decoder processing. In the case of our model this is represented as:

$$Z = \text{concat}([Z_*; P_{density}; P_{intensity}; P_{genre}]) \tag{3.3}$$

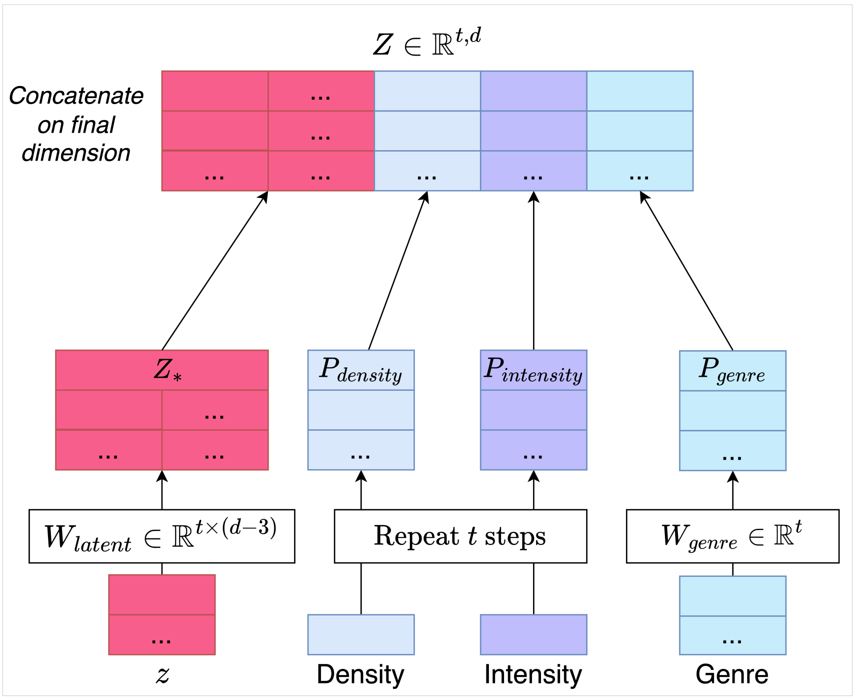


**Figure 20:** Pre-Decoder method visualized.

### 3.5.2 Decoder In-Attention

We adopt a modified version of the *in-attention* method utilised by Musemorphose[35] which is detailed in **figure 21**. We first concatenate the three parameters together,

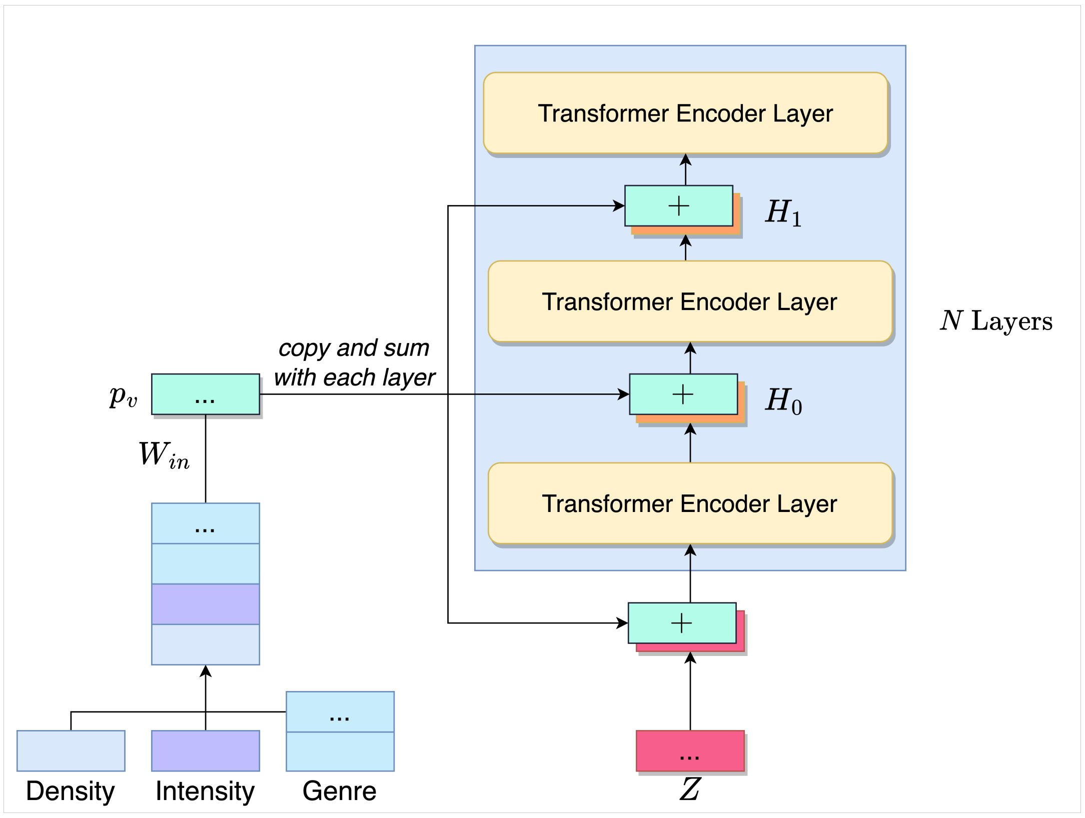

Transformer Encoder Layer
$H_1$
Transformer Encoder Layer
$N$ Layers
copy and sum with each layer
$p_v$
$H_0$
$W_{in}$
Transformer Encoder Layer
Density
Intensity
Genre
$Z$

**Figure 21:** The Decoder In-Attention mechanism visualised. .

forming a new parameter vector $p$. This is then projected with a matrix $W_{in} \in \mathbb{R}^{d \times p_v}$, where $p_v$ is the number of continuous parameters summed with the number of categorical parameter values. This results in our in-attention parameter vector $p_{in} \in \mathbb{R}^d$. We initially sum $p_{in}$ with the latent matrix $Z$ (from the **Pre-Decoder** stage), the output of which is then passed through a successive $n$ number of Transformer encoder layers; prior to each layer, the initial $p_{in}$ is summed again with the previous hidden state of $H_{n-1}$. Thus for each Transformer self-attention layer, the hidden state is calculated as:

$$H_n = \text{Transformer Self-Attention Layer}(H_{n-1} + p_{in}) \tag{3.4}$$

This can be understood as a type of residual network, where the attention heads are continuously *reminded* of the parameter information at every layer.

## 3.6 Loss Functions

The loss functions used throughout training fall into three categories: Reconstruction, Regularization, and Adversarial.

### Reconstruction

The primary goal of the model is to accurately convert the tapped sequence into the original 9-voice pattern. The method of obtaining a reconstruction loss is largely inspired by [2] by dividing the calculation into three components:

$$\mathcal{L}_{\text{recon}} = \mathcal{L}_{\text{hits}} + \mathcal{L}_{\text{velocities}} + \mathcal{L}_{\text{offsets}} \tag{3.5}$$

We utilise Mean Squared Error (MSE) for all three elements: *Hits*, *Velocities* and *Offsets*. We expand on this by calculating a binary (9x32) mask for each HVO matrix which corresponds to the *Hits* in the ground truth. This mask is multiplied with the $V$ and $O$ matrices, thus elminating loss accumulation on indices that do not corresponding with a hit. In other words, an incorrect velocity/offset prediction is not penalized if there is no hit. The inclusion of this mask is treated as a hyperparameter and listed in 3.7.

### Regularization

We utilise a traditional KL term as described in 2.1.2 to calculate the regularization loss $\mathcal{L}_{\text{reg}}$ as follows:

$$\mathcal{L}_{\text{reg}} = -\frac{1}{2}\sum_{j=1}^{J}(1 + log(\sigma_j^2) - \mu_j^2 - \sigma_j^2) \tag{3.6}$$

In which $j$ is the dimensionality of the latent space and $\mu$ and $\sigma$ are the mean and standard deviation matrices, respectively. We further introduce a $\beta$ scaling factor, a common technique for disentanglement for generative AI systems. In several experiments, we further modify the $\beta$ value with a cyclical annealing schedule as proposed in [38]. An example of this, in which you can visualize both the $\beta$ scaling factor and resultant $\mathcal{L}_{\text{reg}}$ is displayed in figure 22. The scaling factor and use of cyclical annealing are both defined as experimental hyperparameters.

### Adversarial

The adversarial networks are trained to accurately identify a target parameter from the latent $z$. To address the challenge of working with both continuous (*density*, *intensity*) and one-hot (*genre*) values, we treat all three as classification problems. Similar to the approach described in [34] we convert both predicted and ground-

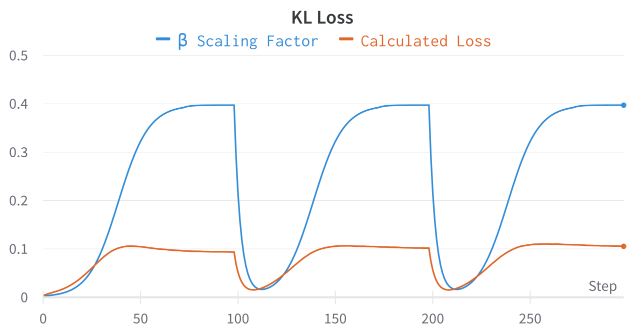


**Figure 22:** Example of the cyclical annealing method from a training run, showing both the scaling factor $\beta$ and the resultant loss $\mathcal{L}_{reg}$.

truth continuous elements into quantized one-hot encodings in a 10-element vector. The loss of each parameter's corresponding adversarial model is thus calculated as the Binary Cross-Entropy between the parameter prediction $\hat{f}_p$ from the latent $z$ and target parameter $p$.

$$\mathcal{L}_{\text{parameter}} = \text{BCE}(p, \hat{f}_p(z)) \tag{3.7}$$

Each model is trained independently on its own loss calculations, thus improving its capability to accurately predict the given parameter. To create the adversarial loss $\mathcal{L}_{\text{adv}}$ for the VAE encoder, we sum the three loss values, thus modeling a representation of the total level of parameter information embedded within the latent space:

$$\mathcal{L}_{\text{adv}} = \mathcal{L}_{\text{density}} + \mathcal{L}_{\text{intensity}} + \mathcal{L}_{\text{genre}} \tag{3.8}$$

### VAE Loss

Detailed in figure 23, the final loss applied to the VAE is a combination of the reconstruction loss (accuracy of predictions), regularization (meaningful distribution of the latent space), and adversarial (minimize latent parameter information). The adversarial loss is scaled with $\gamma$ and applied as a Gradient Reversal Layer (GRL). Whilst $\mathcal{L}_{\text{recon}}$ is backpropagated through the full VAE network, $\mathcal{L}_{\text{reg}}$ and $\mathcal{L}_{\text{adv}}$ are only applied to the Encoder layer.

$$\mathcal{L}_{\text{VAE}} = \mathcal{L}_{\text{recon}} + \beta\mathcal{L}_{\text{reg}} - \gamma\mathcal{L}_{\text{adv}} \tag{3.9}$$

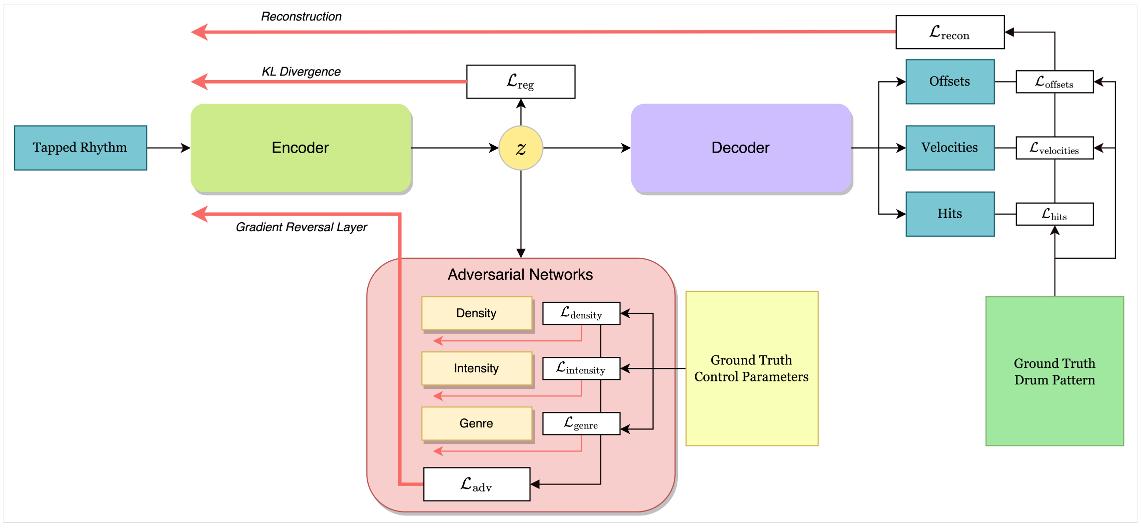

Reconstruction
KL Divergence
Gradient Reversal Layer
$\mathcal{L}_{\text{recon}}$
$\mathcal{L}_{\text{reg}}$
Tapped Rhythm
Encoder
$z$
Decoder
Offsets
Velocities
Hits
$\mathcal{L}_{\text{offsets}}$
$\mathcal{L}_{\text{velocities}}$
$\mathcal{L}_{\text{hits}}$
Adversarial Networks
Density
Intensity
Genre
$\mathcal{L}_{\text{density}}$
$\mathcal{L}_{\text{intensity}}$
$\mathcal{L}_{\text{genre}}$
$\mathcal{L}_{\text{adv}}$
Ground Truth Control Parameters
Ground Truth Drum Pattern

**Figure 23:** Visualisation of the full set of loss functions utilised in the model.

## 3.7 Hyperparameter Tuning

Transformers, VAEs and adversarial networks are all highly sensitive to minor adjustments in their hyperparameter settings. With each experiment we perform *sweeps* wherein a large batch of models are trained with various hyperparameter settings to understand the level of importance and optimal values of each element in relation to final performance. We generally aim to isolate a small number of elements as hyperparameters in each training round, allowing us to more accurately identify correlations and levels of importance. This approach is done iteratively; if we identify an optimal value for a given parameter in a training round, we will anchor it to that value in the next round of experiments.

To facilitate this approach we use the library and API of Weights and Biases (W&B)[39]. W&B allows us to specify hyperparameters and their respective ranges, and analyze the results for each model. This enables the capacity to rapidly iterate through a series of tests and pinpoint the most important set of hyperparameters in regards to key indicators such as reconstruction accuracy, controllability, and KL loss. For each training sweep, we link to a summary report, where you can view a set of analyses as well as further details on individual models.

### 3.7.1 Analysis & Early Stopping

We utilise a number of analysis methods during the training process of each model to determine the capabilities. We detail them here as they will be referenced throughout chapter 4.

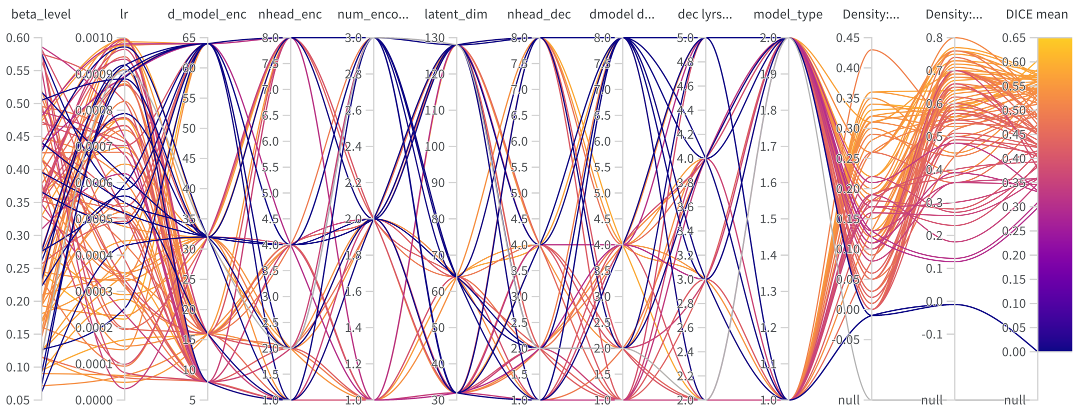


**Figure 24:** An example of hyperparameter model training sweeps with W&B. Each line represents a single model.

- **Latent UMAP**: Utilizing UMAP[40] dimensionality reduction, we create 2D plots with ground-truth parameter data. This helps visualize the level of disentanglement; tight clusters of identical values indicate that the encoder is still embedding this information into $z$.
- **Piano Rolls**: We select a variety of test set examples, and provide them as inputs to the model throughout training. In addition to ground truth parameter data, we test different values of intensity and density, to visualize how these values are impacting its predictions.
- **Sørensen–Dice coefficient (DICE)**: A statistical measurement to determine the similarity between the test set and model predictions. A value of 1.0 indicates that predictions are identical to the ground truth. This serves as a go-to metric for identifying reconstruction accuracy.
- **Density/Intensity [value]**: Every 20 epochs, we choose a single parameter and value - e.g. [density 0.01]. We run the entire test set through the model, using ground-truth control parameters but overriding our selected metric and value. We then calculate the same metric from its predictions, and report the averages. We provide values of 0.01 and 0.99 for density and intensity separately. This helps us determine the *spread* of outputs; a high-quality model can create accurate outputs at the extreme ends of the spectrum.

We provide savable checkpoints at each 20 epochs. Many models were prone to overfitting or latent collapse later in their training runs. We aim to strike a balance between control parameter accuracy and reconstruction (as measured by DICE)

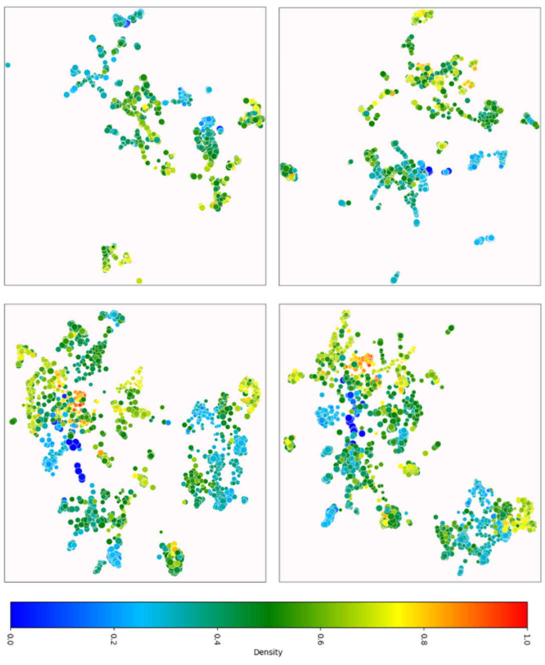


**Figure 25:** The evolution of a model's latent space, color-coded with ground-truth density labels. **Epoch 0**: The initialized state. **Epoch 80**: KL $\beta$ introduced. **Epoch 160**: Adversarial networks are engaged. **Epoch 220**: The selected checkpoint for evaluations.

when determing the optimal epoch. Therefor, early stopping used frequently, and the specific epoch is detailed in each W&B report.

## 3.8 Deployment

We believe that generative AI will have profound effects, both positive and negative, on creative communities. In order to encourage a more collaborative approach and mitigate the harmful impacts of such technologies, it is crucial to involve artists in the evaluation and discussion of such tools early in the development process. However, there are practical difficulties, as the software environments and skills required to develop deep learning models differ greatly from those needed to implement traditional audio-creation tools. Audio software development requires knowledge of topics such as multi-threading, real-time safe procedures, and inter-thread communication techniques - to name a few. We have heard from a number of researchers that the time and domain-specific knowledge needed to deploy a symbolic model in a C++ audio plugin is a massive barrier. We believe that this is one of the reasons that a large proportion of music generation models cannot be utilised in a Digital Audio Workstation (DAW).

We have developed and open-sourced a set of software tools intended to reduce this technical barrier. First we developed NeuralMidiFx, an audio plugin which can host deep learning symbolic models directly within modern DAWs. In addition to this, we released a Software Development Kit (SDK) in collaboration with Neutone Inc.[2] allowing researchers to *wrap* trained models into a more deployment-friendly format. The combination of the SDK and plugin allows researchers to convert and load their model into a musician-friendly format without getting bogged down in the intricacies of audio software development. Together, these tools have allowed us to take trained PyTorch model, and have them loaded in a DAW with realtime performance and controllable parameters in less than ten minutes.

### 3.8.1 NeuralMidiFX Plugin

Modern audio production software typically functions in a centralized manner where the producer is encouraged to continuously work in a single enviroment - the DAW. To this end, the Virtual Studio Technology (VST) format, developed by Steinberg[3], allow developers to package software in an environment that directly connects to modern DAWs. NeuralMidiFX[4] is a VST3 plugin which allows for the deployment of real-time deep learning models that work with MIDI data. By utilising the libtorch[5] C++ API of PyTorch, researchers can serialize their models in a C++ format and include them as a component of the plugin, thus bringing the model directly into the music-creation environment.

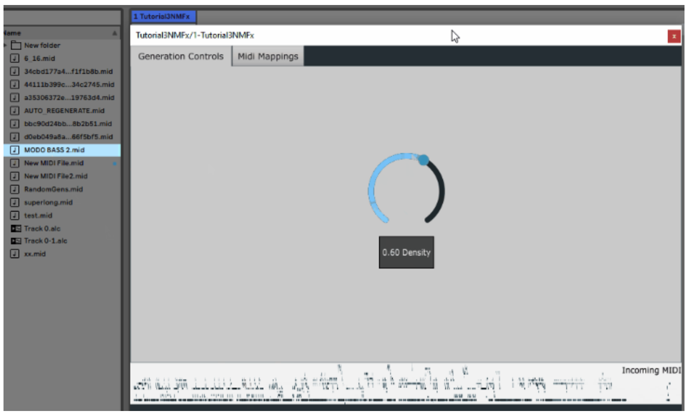


**Figure 26:** One of our early Density models deployed by NeuralMidiFX within Ableton Live for interactive testing.

A separate paper[41] detailing the plugin architecture and examples has been presented at the 2023 Artificial Intelligence Music Conference (AIMC). Developed with

[2]https://neutone.space/
[3]https://www.steinberg.net/technology/
[4]NeuralMidiFX documentation, instructions, and examples are available here.
[5]https://pytorch.org/cppdocs/

the JUCE[6] framework, our plugin enables both real-time and offline inference models by providing a safe multi-threaded environment. We give the developers an easy, modular environment that allows for rapid Graphic User Interface (GUI) prototyping, and a number of functions to access key information such as the tempo and time signature.

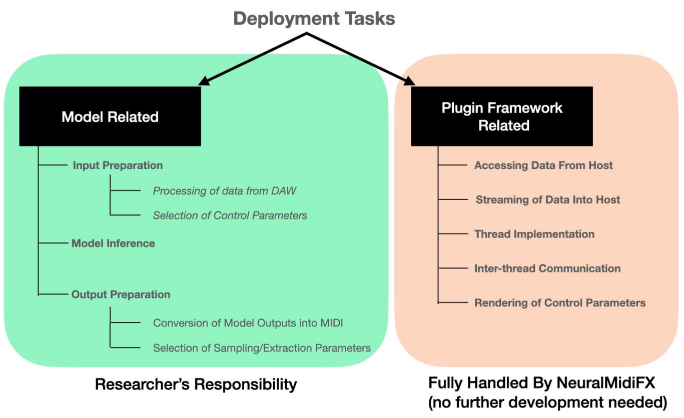

Deployment Tasks
Model Related
Input Preparation
Processing of data from DAW
Selection of Control Parameters
Model Inference
Output Preparation
Conversion of Model Outputs into MIDI
Selection of Sampling/Extraction Parameters
Researcher's Responsibility
Plugin Framework Related
Accessing Data From Host
Streaming of Data Into Host
Thread Implementation
Inter-thread Communication
Rendering of Control Parameters
Fully Handled By NeuralMidiFX
(no further development needed)

**Figure 27:** NeuralMidiFX provides implementations for key tasks, such as data streaming, User Interface generation and multi-threading.

### 3.8.2 NeutoneMIDI SDK

Throughout the development of NeuralMidiFX, we observed that one of the primary challenges of integrating symbolic models within VST plugins was that of the data format. In DAWs and plugins, MIDI messages are typically transmitted in their original hex format, such as in fig. 28. On the other hand, symbolic models often rely on various forms of *tokenization*, where each event is translated into a unique integer (a *token*).

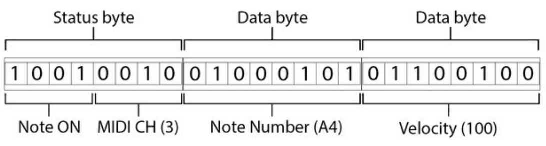


**Figure 28:** A typical MIDI message, transmitting a Note-On event of Velocity 100, from [42].

There are a growing number of tokenization methods for MIDI data, such as REMI[43], CPWord[44], and Octuple[45], each of which translates MIDI events differently. To add to the confusion, a single tokenizer can have a number of configurable options; e.g. one can utilize REMI with different timing sub-divisions. As a result of this, accurately translating MIDI data to and from the format expected by a single model is

[6]https://juce.com/

entirely dependent upon the tokenization parameters it was originally trained upon. Implementing these bespoke conversions in a C++ environment has proven to be both technically difficult and time-consuming.

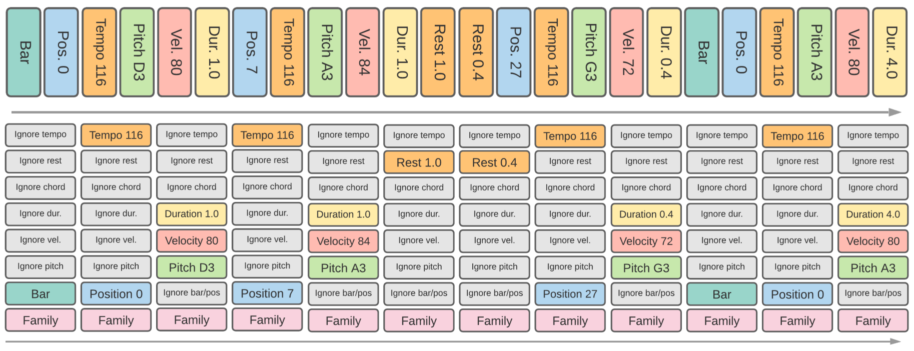


**Figure 29:** Example of the REMI[8] (top) and CPWord[44] (bottom) tokenizations on an identical MIDI passage. Borrowed with permission from Miditok[46].

To address these challenges, we collaborated with Neutone inc. to develop the NeutoneMIDI SDK[7] , which *wraps* a trained model, providing both MIDI-to-Token and Token-to-MIDI translations. To increase the compatibility of NeutoneMIDI with a variety of models, our SDK is built upon the tokenization formats used in Miditok[46], which provides implementations for a variety of popular tokenization methods.

Symbolic models often require some form of control input(s). To accomodate this we provide functionality for a custom *generate* function. Researchers can specify exactly what data they need - e.g. *density*, and how to utilize it within the context of their model's inference process. They can add any number of supported PyTorch operations, thus enabling rapid prototyping of different sampling procedures, data augmentation methods, latent space modulations, and more. NeutoneMIDI is in a purely python environment, allowing researchers to trial these ideas without needing to implement them in C++.

As a result of our collaboration, a researcher simply needs to provide their trained model and a single JSON[8] file (created by Miditok to specify tokenization parameters), and the SDK will produce a *wrapped* model file that handles all tokenization conversions in the style their model was trained upon. This file can then be deployed in a C++ Plugin of their choice, such as NeuralMidiFX.

[7]`https://github.com/QosmoInc/neutone_sdk/`
[8]https://www.json.org/json-en.html

We hope that by developing and open-sourcing both the NeuralMidiFX plugin and NeutoneMIDI SDK, the process of deploying symbolic models into audio production environments has become significantly easier. Both tools are in the early stages of development, and we welcome community contributions and feedback to help improve and expand upon their respective capabilities.

# Chapter 4

# Results & Evaluation

This thesis relied on an iterative methodology, in which we gradually added more parameters and disentanglement techniques, which often introduced new complications and the need to re-optimize various hyperparameters. For the sake of brevity we will briefly summarize the order of experiments, issues encountered, and results. This culminated in a selection of five models, which all demonstrated promising quantitative results. We then exposed them to an informal set of *jam sessions*, in which we tested each model with a variety of inputs and control settings, to determine which single model had the most enjoyable predictions. The chosen model, earthy-armadillo-149, was then identified as our **base model**, and subjected to a series of ablation studies for final evaluations.

## 4.1 Model Selection

**Parameter Injection**

Prior to focusing on disentanglement, we aimed to identify the optimal way to inject a single, continuous control parameter of *density* into our model. We experimented with two separate methods, which were utilised at the encoder stage:

- **1D**: The continuous density value is projected with matrix $W_d \in \mathbb{R}^{1,3}$ to the same dimension of a single input timestep. This is concatenated at the beginning of the input HVO tapped input, thus forming input matrix $HVO \in \mathbb{R}^{(t+1),3}$
- **2D**: The method described in section 3.5.1, albeit with just the single control parameter of density.

In addition to the injection methods, we wanted to understand the optimal configuration for $\mathcal{L}_{\text{reg}}$, specifically the scaling $\beta$ value, the value of cyclical annealing, and when to begin introducing the regularization loss.

We additionally tested several common hyperparameters, such as the number of transformer layers, number of heads, and dimensions of the encoder and decoder models. A full report on the hyperparameters and their results can be accessed here[1]

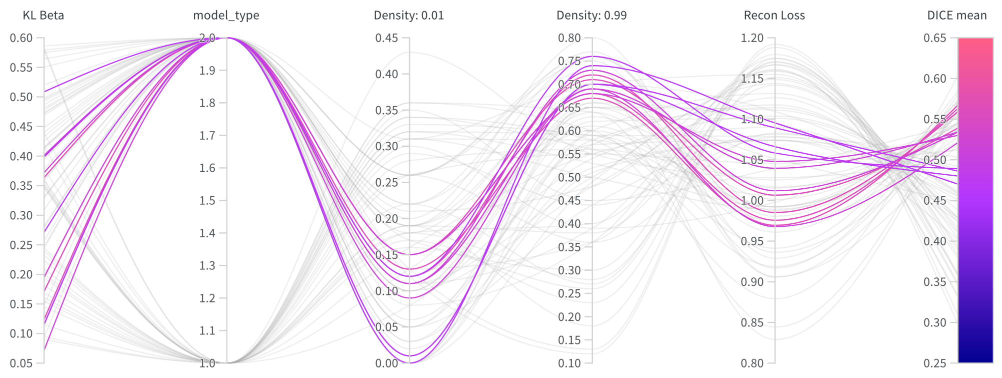


**Figure 30:** W&B sweep to determine the effectiveness of 2D vs. 1D parameter injection methods for *Density*. Highlighted models, all of which utilise the 2D method, had the greatest balance of low and high density separation whilst maintaining favorable results on reconstruction loss and DICE test-set evaluations.

We determined that the 2D method provided substantially better results; when exposed to the density tests detailed in 3.7.1, there was consistently higher separation on our 2D models. Additionally, the KL regularizer term, particularly when combined with cyclical annealing, provided a surprisingly high degree of disentanglement on its own. In line with similar research in symbolic music such as [35], [33] and [34] our models tended to perform most optimally with a $\beta$ value between 0.1 - 0.6. Many models with higher values, such as the one highlighted in figure 31, resulted in complete latent space collapse, where they would predict a generic rock groove regardless of the inputs.

[1]https://api.wandb.ai/links/mmil_julian/tmoi1ctt

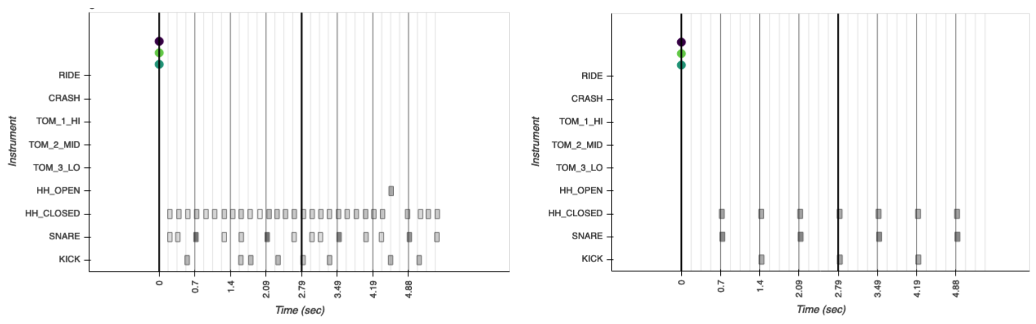


**Figure 31:** Example of latent space collapse from brisk-sweep-74 which has a $\beta$ of 1.13. Regardless of the ground-truth (left), it always predicts a generic rock groove (right).

## Adversarial Networks & Additional Parameters

We then implemented the remaining additional parameters of *Intensity* and *Genre*, introduced the adversarial networks, and moved the 2D injection method into our decoder. As is often the case with adversarial networks, we found them to be highly sensitive to certain parameters, and required a great deal of tuning so as to provide a meaningful impact on the encoder's behavior without overpowering it. Based on the results of the previous experiments, we kept $\beta$ annealing activated and limited the range of between 0.05 and 0.6.

A key discovery was that the adversarial networks needed a semi-structured latent space before their predictions could improve. On the other hand, once $\mathcal{L}_{\text{adv}}$ was introduced, $\mathcal{L}_{\text{recon}}$ would typically flatten-out. We posit that the encoder focuses primarily on hiding the parameters from $z$, and can make little improvements to its reconstruction capabilities once the adversarial loss is applied. With these findings we introduced a delay to the $\gamma$ term similar to that of $\beta$, in order for the encoder and adversarial networks to gain a degree of coherence prior to adversarial engagement. We provide an example of $\mathcal{L}_{\text{VAE}}$ loss in comparison to $\mathcal{L}_{\text{adv}}$ from one of our first balanced training runs in figure 32. At epoch 50, we begin increasing $\gamma$, which results in an increase in the individual adversarial network's losses, and a flattening of $\mathcal{L}_{\text{VAE}}$.

From our initial evaluation metrics it became apparent that $\mathcal{L}_{\text{adv}}$ was indeed encouraging the encoder to disentangle our control parameters from $z$ beyond the levels previously attained from regularization loss alone. As detailed in figure 33, most models developed a tangibly higher degree of accuracy on both density and intensity predictions after the adversarial loss was introduced. This was further confirmed by our ablation studies in section 2.3.

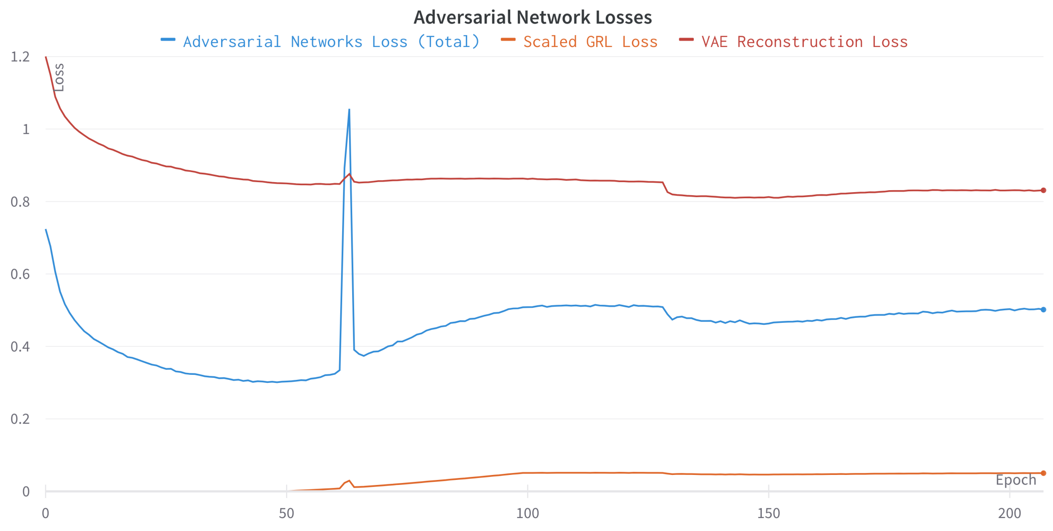


**Figure 32:** Sparkling-bird-129 was one of the initial models with balanced adversarial networks. We disable the GRL for the first 50 epochs, and then linearly increase $\gamma$ to the maximum level.

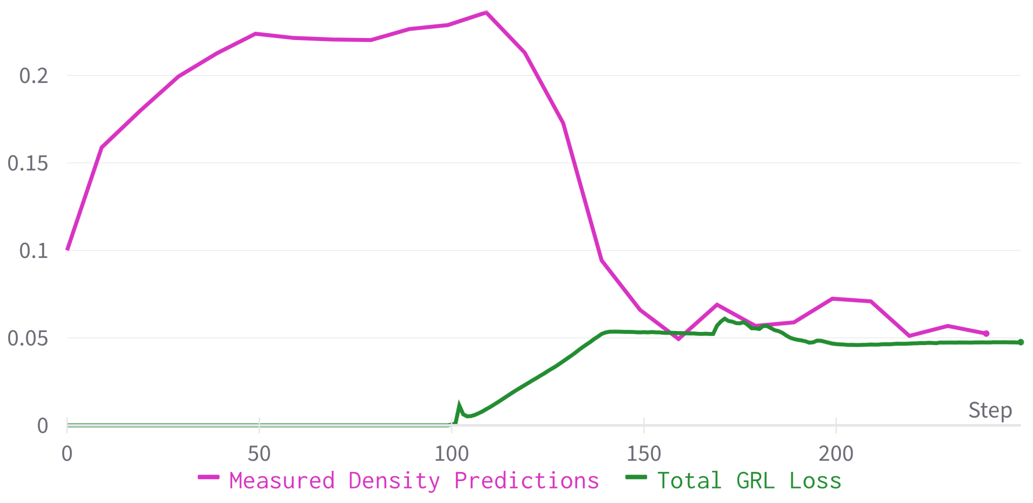


**Figure 33:** One of the final models, classic-sweep-31, demonstrates the impact of GRL loss on *Density* disentanglement. Here we provide the full test-set as input with a *Density* setting of 0.01. As $\gamma$ rises the measured density output rapidly decreases, reducing the delta to the desired control value. The decoder is able to more accurately model the desired density, as the encoder has learned to remove this information from the latent vector.

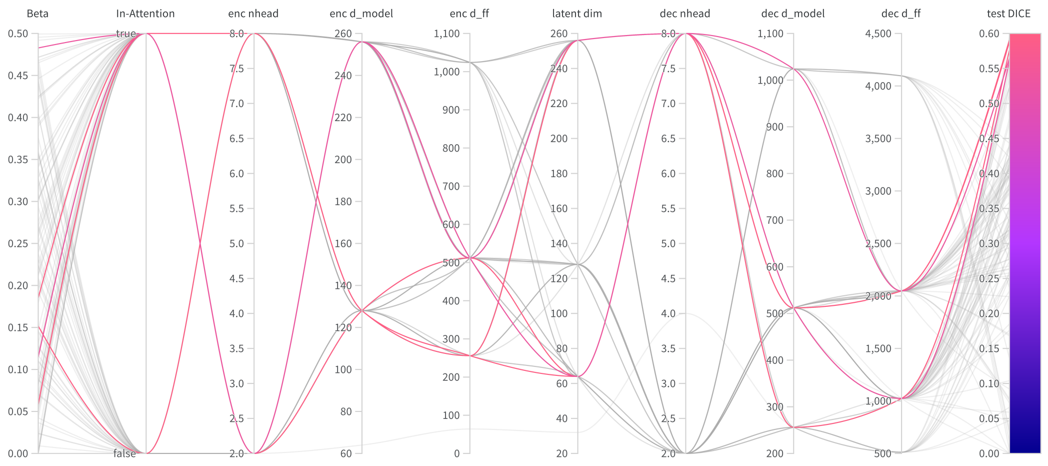


**Figure 34:** The final W&B sweep, in which we trained a variety of models with adversarial networks, KL regularization and in-attention decoders. Highlighted are the five models chosen for the final round of evaluations.

### Decoder In-Attention

With all three parameters present and the adversarial networks balanced, we introduced the Decoder In-Attention mechanism. With $\mathcal{L}_{\text{reg}}$ and $\mathcal{L}_{\text{adv}}$ creating an invariant latent space, this would help remind the transformer layers of our intended control parameters throughout the decoding process.

Additionally, from interactive sessions with our previous models, we identified potential overfitting behavior. Specifically, predictions seemed highly sensitive to the offsets and velocities of our inputs. Adjusting a single note by a few milliseconds seemed to dramatically alter its outputs, which is not reflective of a real drummer's behavior. We therefore introduced two additional dropout measures, *velocity dropout* and *offset dropout*, to reduce the model's dependence on individual velocity and offset values.

We found a substantial improvement after these additions; the models trained with an In-Attention Decoder mechanism gained significant improvements in their capability to separate *Density* and *Intensity* values, whilst retaining their capability to generate accurate recreations. We configured the decoder-type as a hyperparameter, with the option of *In-Attention* or *Standard* - e.g. a traditional transformer encoder. As such, we were able to clearly identify the impact the decoder in-attention injection method had on our parameter accuracies. You can view all of the models, and their respective hyperparameters in this report.

Detailed in figure 35, the models trained with the in-attention decoder mechanism averaged a higher spread on intensity predictions. Simply put, higher intensity values resulted in higher intensity outputs, and lower values created lower outputs.

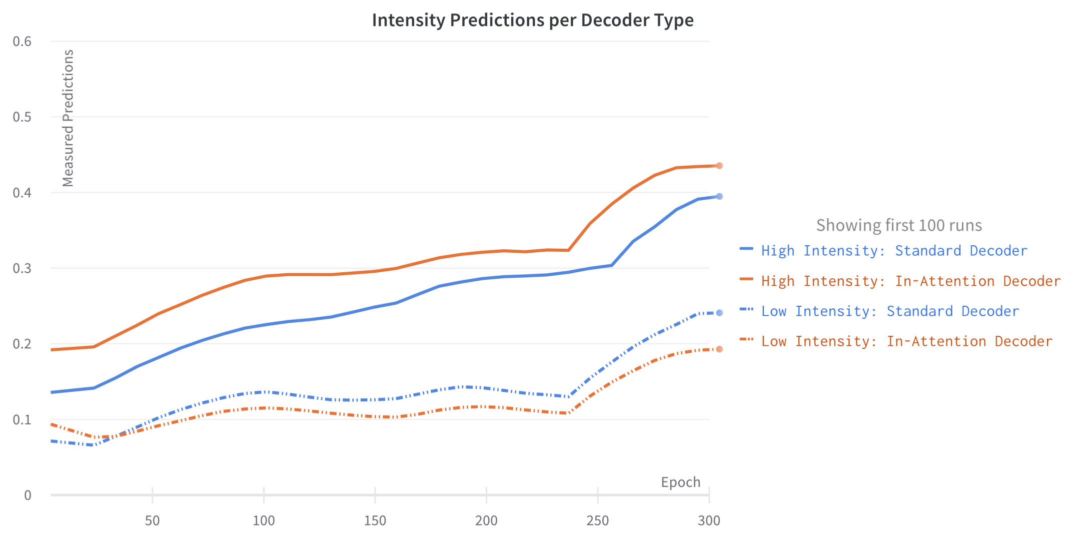


**Figure 35:** Average predicted outputs for Low and High *Intensity* evaluations. Model's that utilised the Decoder In-Attention method typically produced a wider range of outputs that were more closely aligned with the specified intensity.

From this sweep we chose five models that had high accuracy on both density and intensity predictions, as well as high DICE scores; thus demonstrating strength in both disentanglement and reconstruction. The models and their key hyperparameters are detailed in table 1.

| Name | Decoder | Beta | enc d. model | enc n. heads | enc n. layers | dec d. model | dec n. heads | dec nlayers | latent dim. |
|---|---|---|---|---|---|---|---|---|---|
| **elated-sweep-33** | In-Attention | 0.11 | 256 | 2 | 2 | 512 | 8 | 2 | 64 |
| **jumping-sweep-22** | Standard | 0.15 | 128 | 8 | 3 | 512 | 8 | 3 | 64 |
| **rose-sweep-43** | In-Attention | 0.48 | 256 | 2 | 4 | 1024 | 8 | 8 | 256 |
| **classic-sweep-31** | In-Attention | 0.19 | 128 | 8 | 2 | 512 | 8 | 6 | 256 |
| **revived-sweep-3** | In-Attention | 0.23 | 128 | 2 | 2 | 256 | 8 | 8 | 64 |

**Table 1:** Models selected for further musical evaluations.

## Final Tuning

Using the deployment tools described in section 3.8, we loaded our chosen models into Ableton live. We then created our own set of tapped inputs that covered a variety of styles and tempos. For each model, we tested its ability to create outputs at each combination of density and intensity values, as well as the genres of rock, jazz, latin and hip-hop. Although informal, these evaluations proved highly beneficial. For example, rose-sweep-43 had a DICE of 56% (relatively high), yet produced consistently *ugly* patterns, filled with strange voice selections, such as repeating toms. It also had a poor density/intensity separation, and would often get *more* dense as you reduced the control value. Revived-sweep-3, on the other hand, provided decent results, but used a max-velocity kick drum regardless of the intensity setting. It also rarely used the cymbals/rides, even when set to jazz mode.

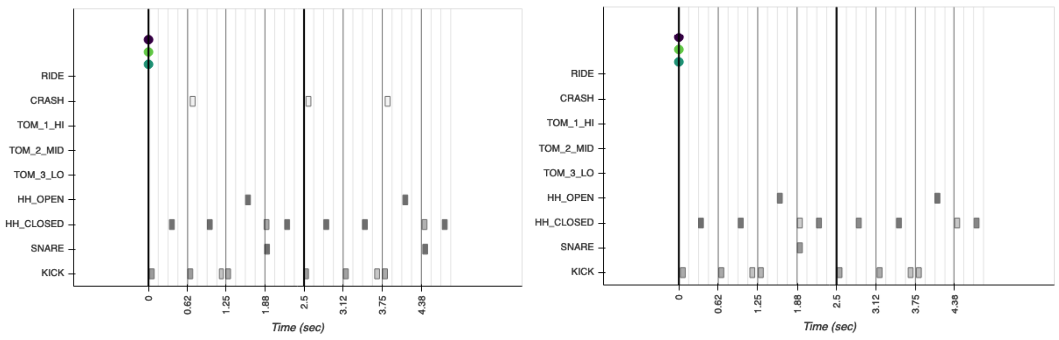


**Figure 36:** (**left**) Ground truth, **(right)** Prediction. Classic-sweep-31 has a consistent ability to accurately predict unseen rhythms.

After several hours of testing, we determined that classic-sweep-31 and jumping-sweep-22 were the most fun and appealing. After directly comparing these two models on a number of identical settings, it became apparent that classic-sweep-31 had provided more realistic outputs on the extreme control parameters (e.g. density at 98% and intensity at 10%). Furthermore, it altered its velocity patterns more drastically in response to genre inputs.

It was observed during these tests that none of the models were *great* at changing their outputs in response to genre changes. Specifically, every drum groove sounded vaguely *rock-ish*. We hypothesized this was due to the dataset; with rock accounting for 31% of the GMD, it is harder for the model to learn the patterns of other genres.

## 4.2 Evaluations

In this section we will present the quantitative metrics used to make a final set of evaluations on our baseline model. As is often the case with generative models, objective metrics do not necessarily give a complete indication of the artistic quality of the outputs. With that said, they still provide an overall indication of the model's capabilities, and provide targeted insights as to its weaknesses. We selected classic-sweep-31 as our model, due to its high DICE score, accurate density and intensity predictions, and appealing patterns throughout the interactive qualitative tests.

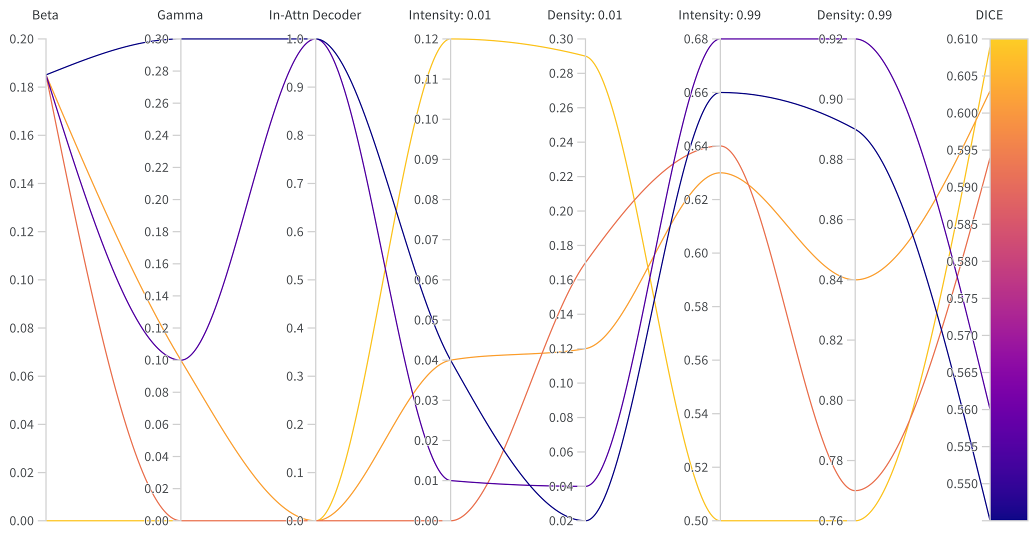


**Figure 37:** The W&B sweep of our final evaluation models.

With no known baseline evaluations on controllable, expressive rhythm generation modelling, we format our evaluations as an ablation study. With genre-adaptation identified as a weakness of classic-sweep-31, we trained an identical model with a genre-weighting tensor added, to account for the imbalance of genres represented in the GMD. This sweep, called earthy-armadillo-149, has been identified as our **base model**. We therefore present our ablation study models, which have key parameters modified and removed to better understand their impact on reconstruction and controllability in table 2. You may also view the full W&B report here.

| Sweep Name | Model Identifier | KL Beta | Adv. Gamma | Decoder |
|---|---|---|---|---|
| **earthy-armadillo-149** | **Base Model** | 0.185 | 0.1 | In-Attention |
| **fanciful-cherry-153** | **Higher Gamma** | 0.185 | 0.3 | In-Attention |
| **lazy-hedgehog-19** | **No In-Attention** | 0.185 | 0.1 | Standard |
| **zesty-river-160** | **No Adversarial or In-Attention** | 0.185 | 0 | Standard |
| **dry-rain-154** | **No KL, Adversarial or In-Attention** | 0 | 0 | Standard |

**Table 2:** Base model and variations for final evaluations.

### 4.2.1 Reconstruction

The primary objective of the model is to accurately convert a tapped pattern into a drum performance. Here we will detail various quantitative evaluations and their results to assess the overall reconstruction accuracy. It is important to recognize that the model is a VAE, therefore placing an upper limit on its reconstruction accuracies.

This is appealing from a musical standpoint, as it means that it will (almost) never generate the same pattern twice. We further sub-categorize the section into two parts: Hit metrics, to determine *composition* accuracy, and Velocity and Offset metrics, to determine *performance* efficacy.

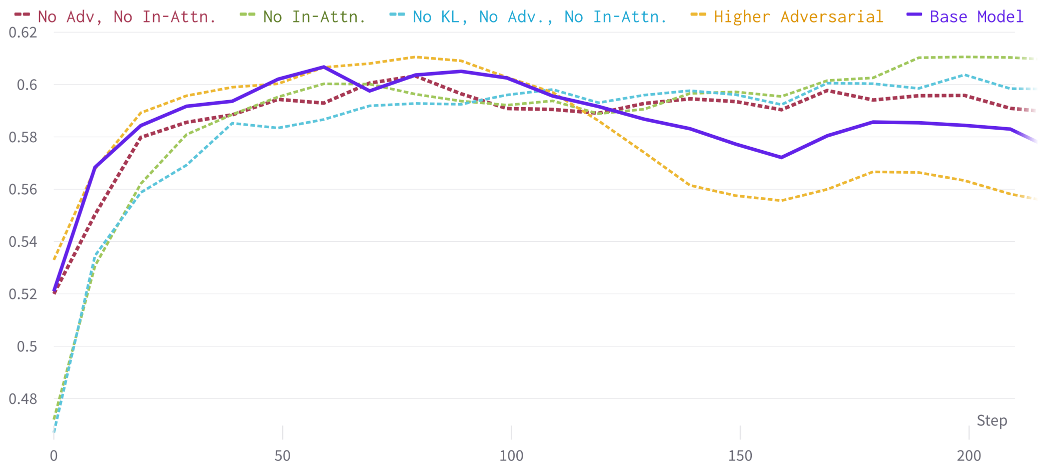


**Figure 38:** DICE test set results per model. This highlights the tension between disentanglement and reconstruction quality; the baseline and high-gamma model have slightly lower DICE scores.

### Hit Accuracy

Provided the HVO tapped input, the model must calculate which drum voices are active at a given time-step. Detailed in figures 39 are a series of metrics to analyze the base model's hit density prediction accuracy in relation to the test set ground truth. **Step density** measures, per pattern, the number of time-steps with an active hit as a ratio to the total time-steps. This measure is indifferent to voicing; as such it provides a collapsed perspective on pattern density distributions. **Number of Instruments** calculates the total number of voice-types active in a single pattern. Our base model has a distribution that is nearly identical to the test set in both metrics; this indicates a high capacity to estimate the general density and number of voices present for a given pattern.

We present the **Hit Prediction** analyses in **figure 40**. Detailed are the distribution of total predicted hits per pattern. **True Predictions** indicate that the correct voice was predicted at the correct timestep, and **False Predictions** counts the number of occurences when a hit was predicted for a voice and time that the ground truth had silence. This analysis helps further visualize the overall density accuracy of the model, as well as its capability to correctly predict which voice(s) are active at a

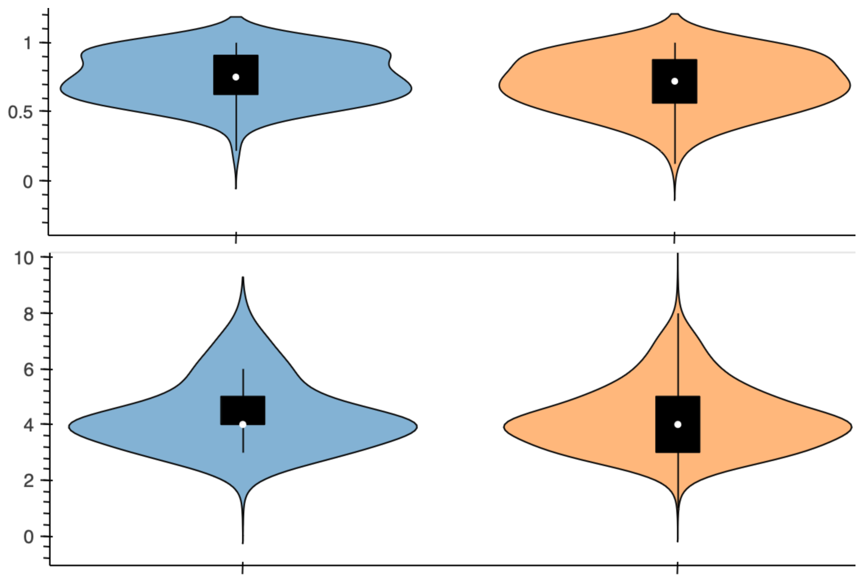


**Figure 39:** Base model predictions in comparison to the full test set. In both examples, the ground truth is on the left, and predictions on the right. **Top**: Step density. **Bottom**: number of instruments

given time. As evidenced through Total Predictions, the model's output densities are closely aligned with ground truth distributions, albeit with a number of incorrect high-density (>60) patterns.

It is notable that there are a significant quantity of False Predictions, indicating that the model is often predicting a hit of a given voice when it should be silence. Considering the favorable density results, this likely points to a difficulty in voice selection; for example, it may play a hi-hat when the ground truth has a snare. Some degree of this is to be expected from a VAE given the sampling-based generation. Figure 41 gives a single example of this, where the overall predicted density and voice-types are similar, but the individual voicings are not aligned with the ground truth.

### Velocity & Offset Accuracy

In addition to hits, the model is tasked with *humanizing* its outputs by providing Velocity and Offset values. Accurately capturing these nuances is key to the creation of fun, engaging rhythmic ideas. In this section we will provide global metrics on the velocity and offset accuracies. As these are also critical to conveying the difference in various styles, we report a more detailed set of genre-specific metrics in **section 4.2.2**.

Detailed in **figure 42** are the means and standard deviations of Hit, Velocity and

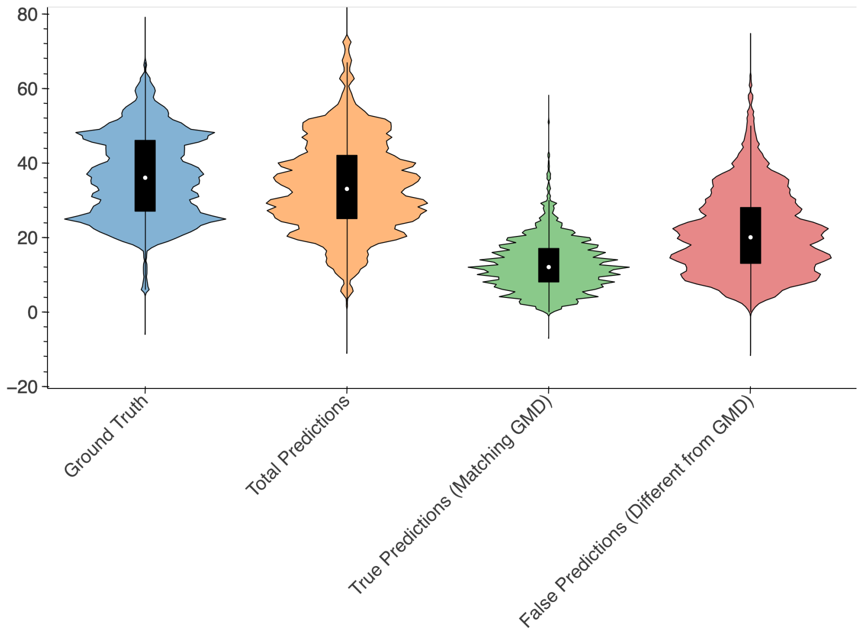


**Figure 40:** Hit count distributions vs. ground truth for the base model.

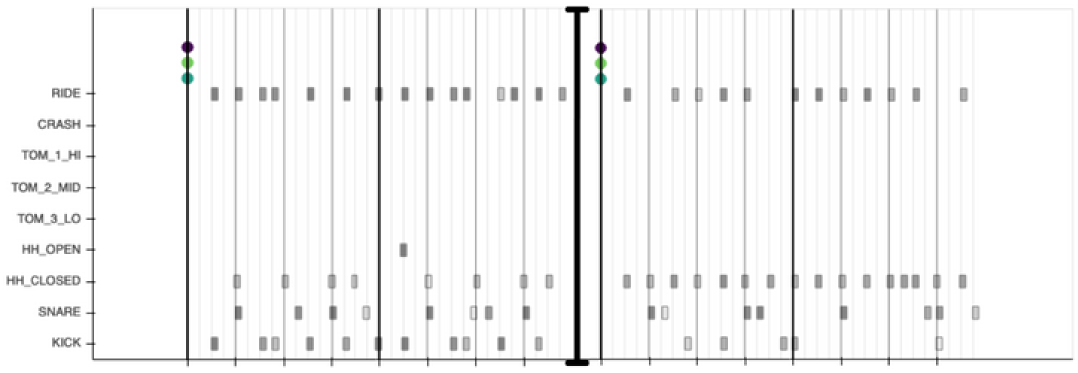


**Figure 41: (Left)** Ground truth, **(right)** prediction. A piano roll visualization of the base model predicting a test set pattern. While the overall density and voice selection is similar, the voices used at a specific time index often differ from the ground truth.

Offset losses on the training set for all models in our ablation study. This highlights a recurring issue throughout the training processes. Namely, the models experience a substantial improvement in the Hit and Velocity calculations, but almost no loss reduction in the offsets. This was prevalent throughout all sweeps and experiments; various measures were undertaken to improve it, such as the Velocity and Offset loss masking detailed in section 3.6. Despite these attempts, the issue clearly persisted, and provides an opportunity for future research.

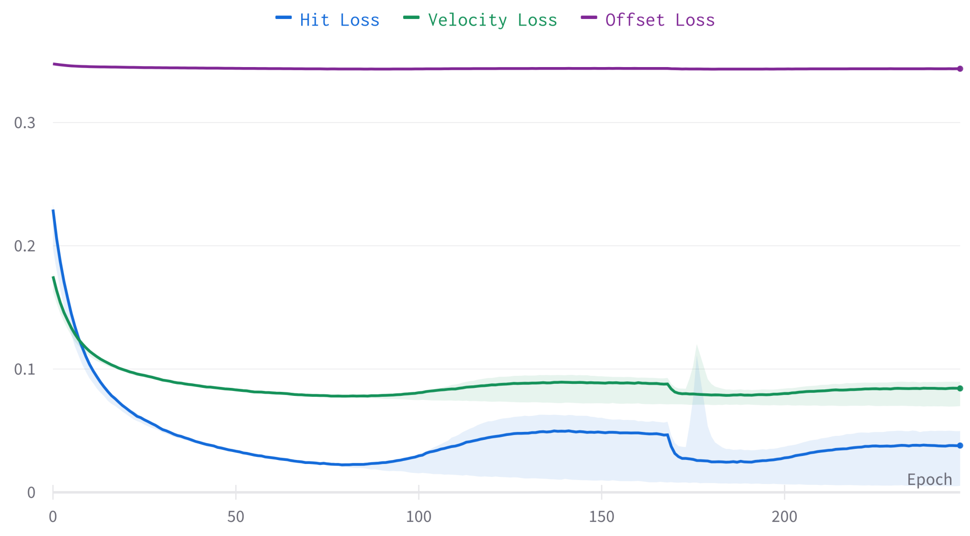


**Figure 42:** The distribution of Hit, Velocity, and Offset train losses across all five evaluation models. While Hit and Velocity losses would decrease, Offsets losses remained near their initial values.

To measure our base model's velocity prediction accuracies, we highlight in figure 43 the **Polyphonic Velocity** distributions on test set predictions. There is a noticeable difference in the upper distributions, indicating that the model is missing some higher-velocity predictions. This ties in to our observation that the models typically struggled with upper-intensity parameter separations. Despite this discrepancy, the distributions are still remarkably similar. This is in line with our perceptual evaluations, where it was evident that a variety of soft and loud notes will be reasonably predicted with each pattern.

In figure 44 we present a graphical evaluation, in which heat maps are plotted for placement and velocities of kick, snare and closed hi-hat voicings for the full test set. This allows us to visually inspect the distribution of velocities, per voice, to understand the general accuracy of the model in its modelling of velocity distributions.

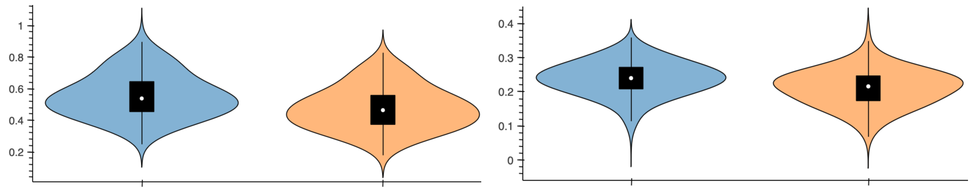


**Figure 43:** Polyphonic Velocity distributions, with ground truths represented as blue and predictions in orange. **Left**: Mean. **Right**: Standard Deviation

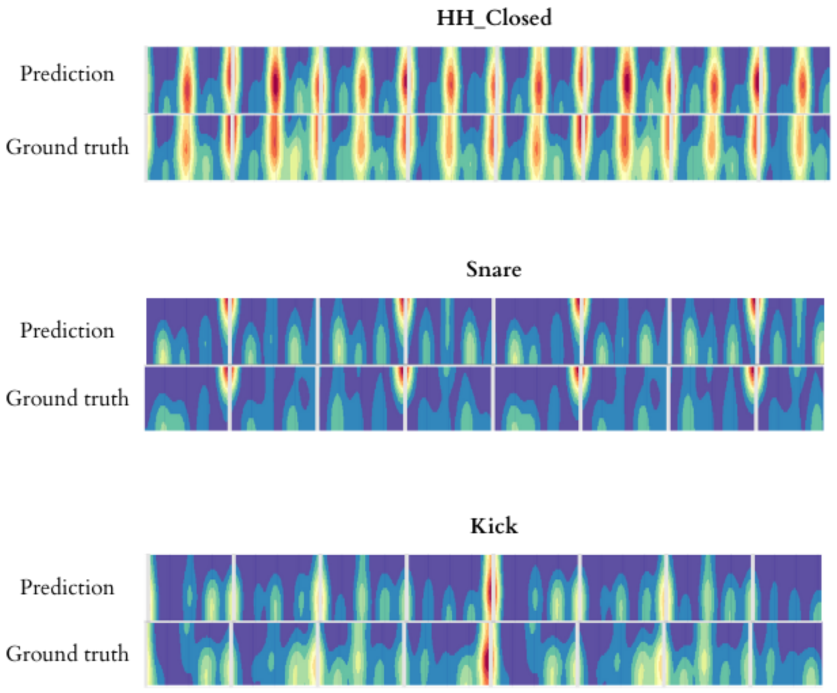


**Figure 44:** Velocity heat maps, ground truth and prediction comparisons for Kick, Snare and Hi-Hat on the test split. Hotter colors indicate a high number of predictions in that time and velocity.

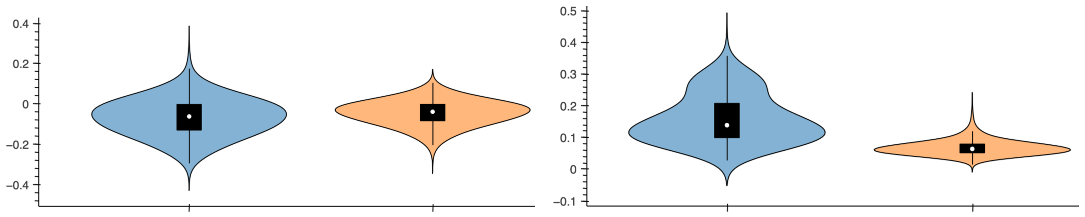


**Figure 45:** Polyphonic Offset distributions, mean (left) and standard deviation (right). Blue represents the test set, and orange the predictions.

A high degree of accuracy in the test set velocity recreations is demonstrated. You can see, for example, that the snare tends towards higher density values on the off-beats, whereas quieter *ghost snares* are present on other timesteps. This behavior is reconstructed to a high degree of similarity in our base model. On the hi-hats there is a nearly uniform distribution of velocity values on each 8$^{\text{th}}$-note, which provides evidence that the model is generally capable of predicting velocities across the full spectrum of values.

Finally, we report on the **Polyphonic Offset** ground truth and prediction distributions of our base model in figure 45. This highlights a point of difficulty for the model; the level of deviation is significantly lower, indicating more *quantized* patterns. Whilst the ground-truth mean is slightly below 0, indicating drummer's tendency to play *ahead* of the beat, our model has a mean tightly clustered exactly at 0. In addition, the standard deviation is noticeably smaller. Musically, this means the model is not recreating accurate micro-timings, and is often skewing towards a *quantized* set of rhythmic predictions.

This is particularly evident in genres such as jazz, which is detailed further in section 4.2.2.

The difficulty in predicting offsets was first documented in [2], and became evident throughout our own training runs when the offset losses failed to reduce significantly. We present this as a key finding and area of opportunity for further research. For example, we hypothesize that a hierarchical model that separately predicts *composition* and *performance* tokens could provide tangible improvements. This is further discussed in section 5.2.

### 4.2.2 Controllability

A properly disentangled model has a latent space that is invariant to the target control parameters, providing the decoder with enough information to make accurate reconstructions without biasing its outputs against the control parameters. For ex-

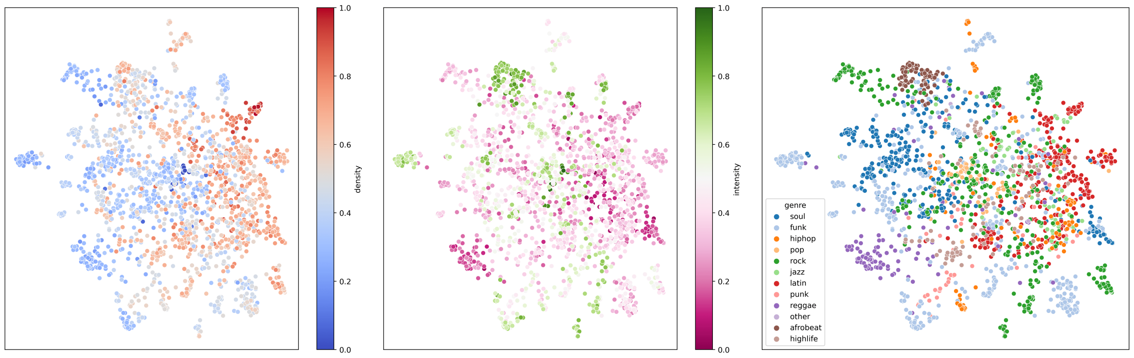


**Figure 46:** UMAP of our model with a heightened adversarial $\gamma$ of 0.3 compared to the base model. Left to right: Density, Intensity, Genre; zoom in for best detail.

ample, if the encoder provides significant information detailing *Low Density*, but the user specifies *High Density* via parameter injections, the decoder will generally fail to follow the density instruction. First, we utilise Uniform Manifold Approximation and Projection (UMAP) dimensionality reduction to provide a visual inspection of our model's latent space. We then report separately on Density, Intensity and Genre prediction accuracies.

### Latent Maps

Using the UMAP python package [40] we perform dimensionality reduction of the latent vector $z$. We provide as input to our target model the full test set, as well as the ground-truth data on control parameters density, intensity and genre. We plot each UMAP separately, so as to allow for detailed visual inspection of each control element in regards to the latent space. In an ideal scenario, $z$ will appear spread out, with the control parameters evenly distributed throughout the space. Tight clumps of a given metric, e.g. a singular genre, indicates that the encoder is still providing this information to the decoder.

We can observe, for example, that there are examples of low/high density and intensity measures spread through the distribution of fanciful-cherry-153 in figure 46. This points towards a tangible impact on the encoder's behavior from the adversarial networks. On the genre chart (right), it is apparent that certain genres - such as rock and funk have a wide distribution. Other genres, such as reggae and afrobeat appear tightly clustered. This pattern appears throughout most UMAP's, and appears correlated to the distribution of genres in the GMD. In other words, genres with a high number of training examples (rock) give the model a range of examples to learn from, with low and high density/intensity ground truth patterns. Genres such as reggae have a smaller number of examples, with a very similar set

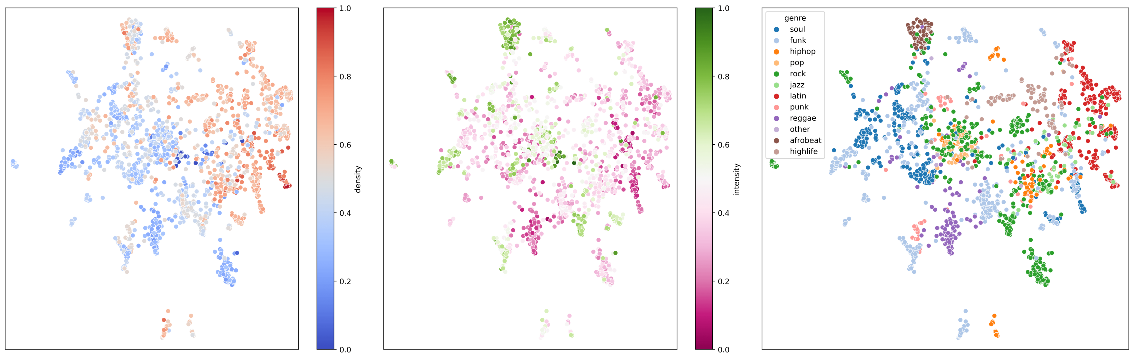


**Figure 47:** UMAP of the base model.

of characteristics, preventing $z$ disentanglement.

For comparison we provide the UMAP of our base model in figure 47. With the lower adversarial $\gamma$ term it is evident that $z$ is moderately less disentangled. Note, for example, the visible partitioning between high and low densities, which is not as defined in our higher-gamma model. You can also observe with certain genres, such as Latin, a higher degree of clustering. For additional insight, we provide a visualisation of our models with fewer disentanglement components in figure 48. With the KL term, adversarial losses and in-attention decoder stripped out, we can see a markedly higher degree of clustering. This provides an initial indication that the techniques presented in the thesis are having a quantitative impact on the encoder's ability to separate control elements from the latent vector.

### Density and Intensity

The metrics of density and intensity provide an excellent baseline for assessing control invariance due to their quantitative nature. Here, we will detail a variety of assessments made to determine the model's capability to adapt its outputs based upon user input.

To gain a visual understanding of the distribution of outputs, we provide heatmaps in figures 50 and 52. We create a Gaussian distribution of random numbers with a mean of 0.5 and standard deviation of 0.15. In addition, we create a random distribution of genres and randomly sample from $z$. We provide these as inputs to our model for decoding and sampling. We then calculate the densities and intensities of the predictions, using the same normalizing function as the training dataset.

It is apparent that our base model is following a similar distribution to that of the input values. With that said, there is a noticeable discrepancy in the upper intensities. The centroid of both density and intensity predictions is skewed to the

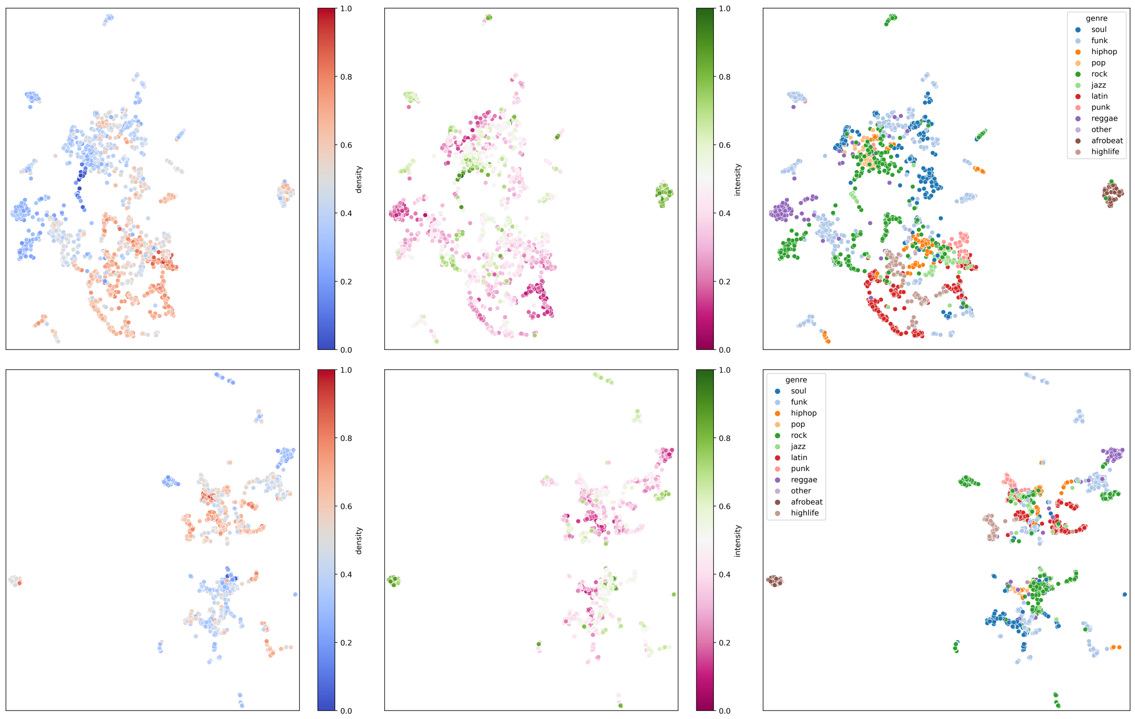


**Figure 48: Top**: Model without adversarial components or in-attention decoder. **Bottom**: Model without KL loss, adversarial components or in-attention decoder.

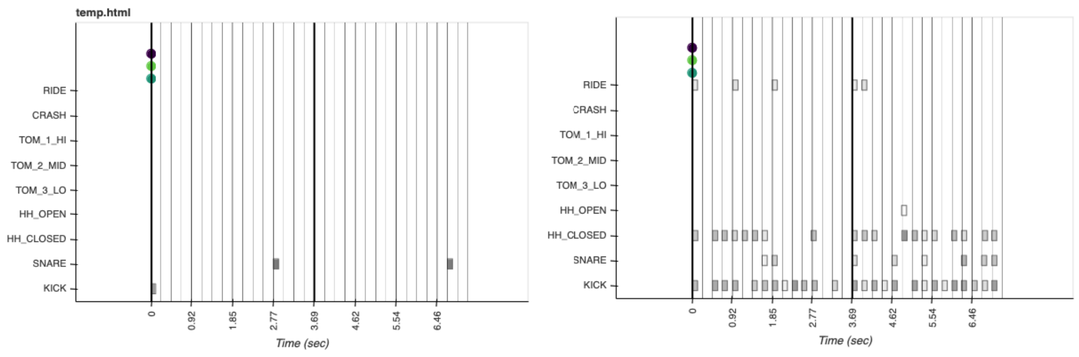


**Figure 49:** Piano rolls demonstrating the capability of earth-149 to respond to control values. Both examples have the same input: **(left)** low density, high intensity. **(right)** high density, medium intensity.

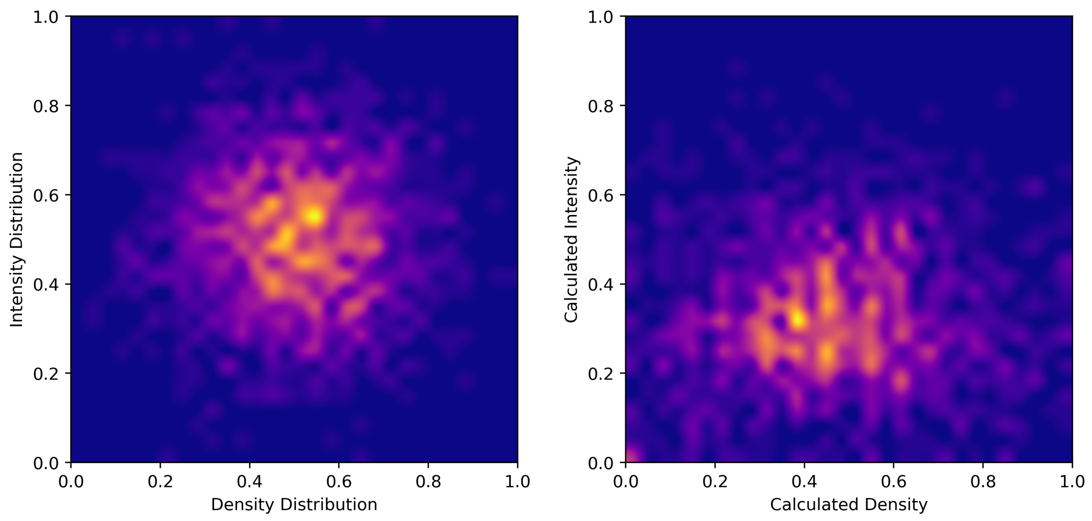


**Figure 50:** Prediction intensities and densities heatmap. **Left**: Ground truth Gaussian distribution of control parameter inputs. **Right**: Base model predictions.

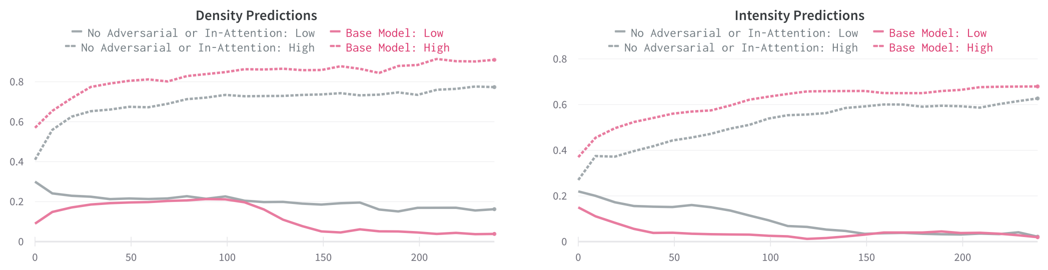


**Figure 51:** Progression of density and intensity predictions over the course of training; comparison of base model (pink) vs. no adversarial, no in-attention decoder variant (gray). At epoch 100, there is a noticeable drop in low-density predictions, which correlates with the activation of the adversarial networks.

lower values, particularly on the latter. This is aligned with our training metrics, which indicated that all models had particular difficulty on generating high-intensity patterns.

On the other hand, the models in figure 52 have a noticeably deteriorated performance. The model that has only KL $\beta$ as a disentanglement method outputs a wider spread of parameters, demonstrating an existent but limited degree of relationship between input and output. On the other hand, removing the $\beta$ term resulted in a heatmap that can only be described as a *minimalist art piece*, with no discernable relationship to the Gaussian distribution.

We present in figure 53 the calculated accuracy for each model on low and high densities and intensities. To calculate this, we choose a single parameter and value to test; e.g. [density: 0.01]. We process the full test through the model, with the

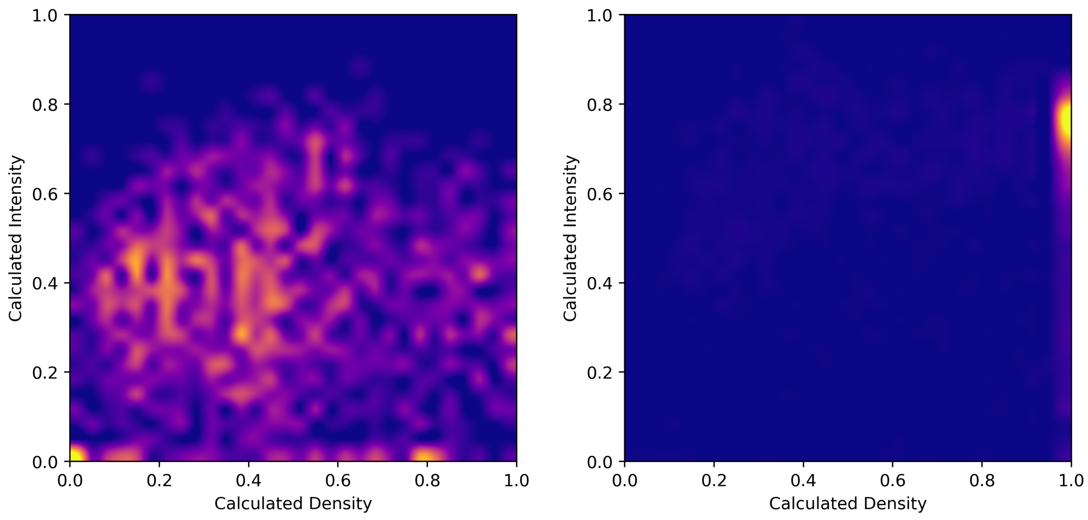


**Figure 52:** Calculated metrics on ablation test models. **Left**: No adversarial networks or in-attention decoder. **Right**: No KL loss, adversarial networks or in-attention decoder.

specified parameter and value, and set the remaining parameters to their ground-truth values. We plot the performance of each model per metric to gain a detailed understanding their various capabilities and weaknesses. Additionally, we calculate the average distance between target-output pairs. This is detailed in table 3 alongside DICE metrics, providing a holistic view of both reconstruction and control performance.

We first observe that the base model has the highest performance on both density and intensity accuracy metrics, averaging 94% and 85% respectively. To our surprise, the higher gamma model saw only a marginal improvement on intensity accuracy, with slightly worse predictions on both high density and intensity tests. This, along with a 2% lower DICE, leads us to conclude that there are diminishing returns from the adversarial loss component. On the other hand, the three models with reduced disentanglement showed weaker performance on nearly every metric. By simply removing the in-attention decoder, we observed a 4% decrease in density accuracies. As the introduction of adversarial networks and the novel in-attention decoder mechanism are key contributions of this thesis, we see the 9% difference in density performance as a promising indicator of their utility in controllable generation models.

## Genre

We aim to develop a system that can translate an input tapped rhythm into a genre specified by the user. Determining the effectiveness of such a system quantitatively

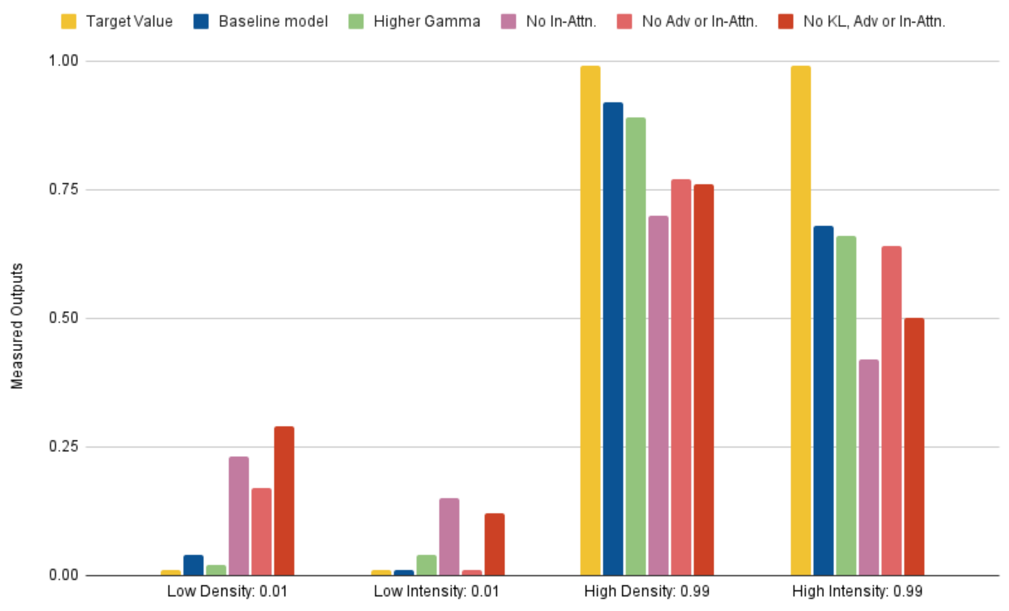


**Figure 53:** Measured accuracy per model of low and high predictions on continuous parameters.

| Model Name | Density Accuracy ↑ | Intensity accuracy ↑ | DICE ↑ |
|---|---|---|---|
| **Base Model** | **0.94** | **0.85** | 0.56 |
| **Higher Gamma** | **0.94** | 0.84 | 0.54 |
| **w/out In-Attention** | 0.90 | 0.82 | 0.6 |
| **w/out Gamma or In-Attention** | 0.85 | 0.83 | 0.59 |
| **w/out KL, Gamma or In-Attention** | 0.81 | 0.74 | **0.61** |

**Table 3:** Ablation study models to determine the impact of various disentanglement and injection techniques on controllability and reconstruction accuracies.

is difficult; unlike a parameter such as density, one cannot calculate how *punk* a beat is. We therefore analyze a series of velocity heat maps to get a general understanding of the model's ability to shape its predictions per genre. We select two voices, snare and hi-hat, as they are well represented across all genres. We analyze five genres in an attempt to capture a diverse set of musical cultures: Rock, Latin, Jazz, Funk, Afrobeat. We also highlight that jazz and Afrobeat make up just 8.4% and 5.2% of the total dataset.

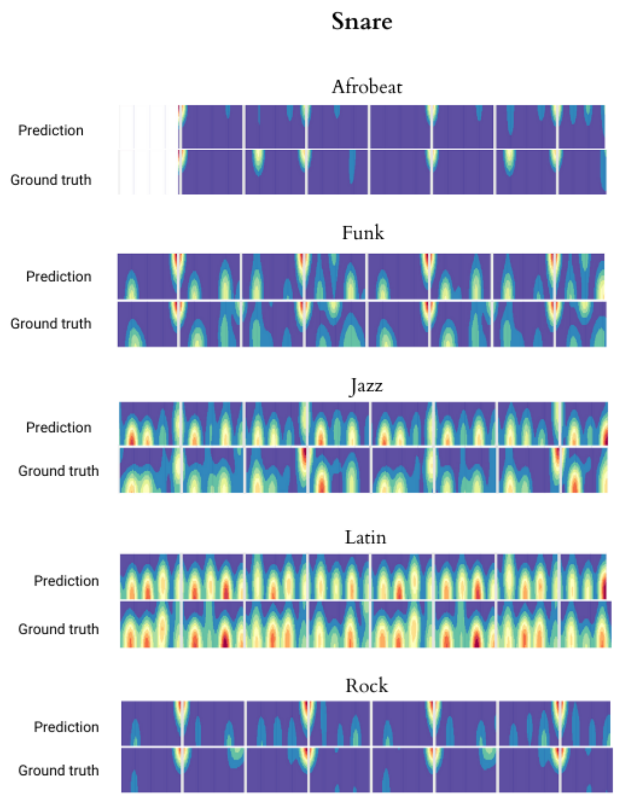


**Figure 54:** Base model snare velocity heat map genre.

We display the velocity heatmaps for snare predictions in figure 54, and closed hi-hat in figure 55. In both examples, the model shows remarkable capability to adapt its hit and velocity predictions in accordance with the given style. For example, on Rock and Funk patterns, the snare typically has a high velocity on the secondary beats (2, 4, 6, 8), whereas in Latin, the velocities are subdued. On the closed hi-hat, we see a stronger emphasis on the off-beats in jazz and Latin patterns, whereas funk and rock place greater importance on the downbeats. This provides evidence that, in terms of velocities, our base model is changing its predictions based upon the input pattern.

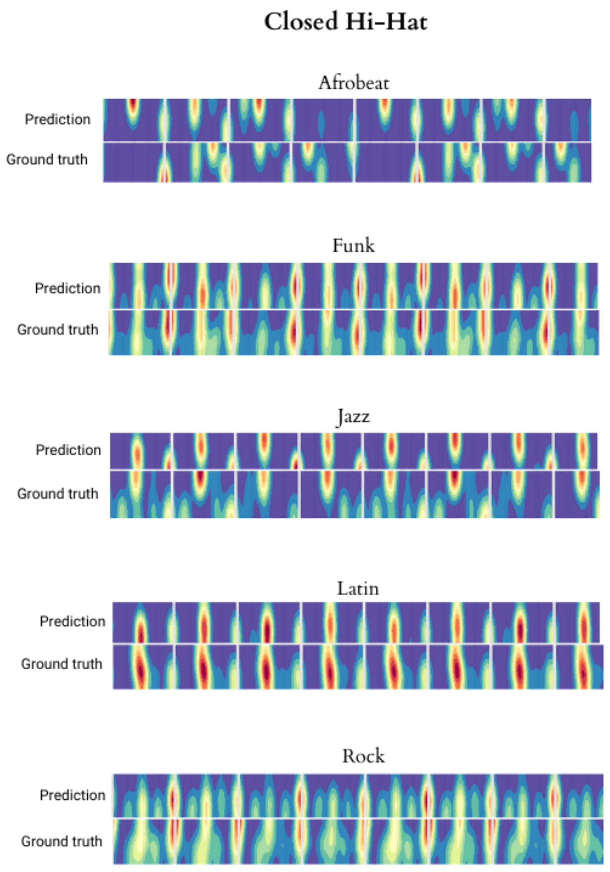


**Figure 55:** Base model closed hi-hat velocity heat map per genre.

Further analysis is provided in figure 56, which details the polyphonic velocity prediction distribution means for Latin, funk, rock and jazz. Similar to the global evaluations, the model has a lower centroid. This means that generally it is predicting lower velocities. Nonetheless, the overall shape of the distribution is quite similar to the ground truth in all four genres, with rock and funk have a particularly accurate representations.

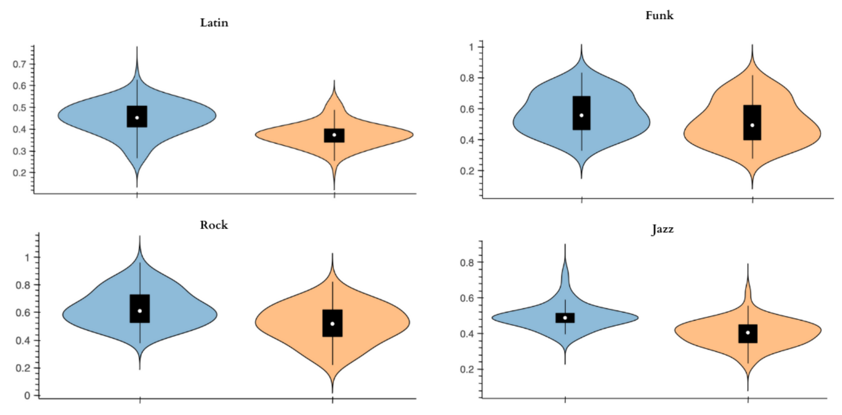


**Figure 56:** Base model polyphonic velocity distributions per genre, with ground truth on the left and predictions on the right.

Finally, in figure 57 the polyphonic offset standard deviation is detailed for the same four genres. It is notable that the model has a significantly smaller spread and lower centroid in all four examples. In line with the global evaluations, this indicates that the predicted rhythmic patterns are more quantized and have significantly less micro-timing variety throughout. Thus, the model's general difficulty in predicting micro-timing information is interfering with its capability to create distinguishable, style-specific rhythmic patterns. In our listening tests, this was particularly apparent in genres such as jazz, where micro-timing is critical providing a sense of *swing*.

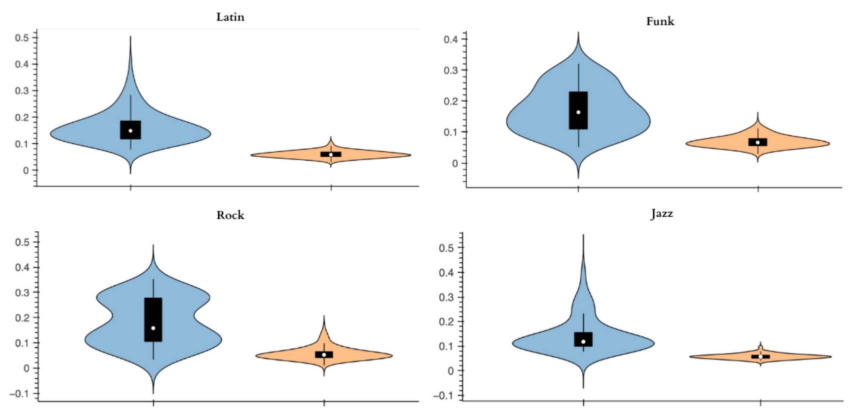


**Figure 57:** Base model polyphonic velocity distributions per genre, with ground truth on the left and predictions on the right.

# Chapter 5

# Conclusions & Future Work

## 5.1 Conclusions

In this thesis we presented VAEDER, a symbolic music generation model that converts tapped inputs into drum grooves whilst adhering to user parameters of density, intensity, and genre. To achieve this, we combined a transformer-based VAE architecture with several disentanglement and parameter injection techniques. More specifically, we trained an array of adversarial classification networks, which in tandem with the $\beta$-scaled regularization loss, provided a latent vector $z$ that is moderately invariant to our target parameters. We subsequently inject these parameters, first with a Pre-Decoder layer, and then a modified In-Attention transformer decoder layer. Ablation studies have demonstrated that every one of these methods contributes to an increased performance on parameter adherence.

In addition to the model itself, we developed and open-sourced a VST3 Plugin, NeuralMIDIFX, as well as the NeutoneMIDI SDK. These tools allow researchers in the symbolic music domain to quickly wrap and deploy their models directly into DAWs. As these tools came to fruition, we experienced first-hand how quick and easy it is to utilise them; once we had a new batch of models trained, we were able to deploy dozens of them in Ableton within an hour. As noted throughout our evaluations, music generation tools (both symbolic and audio) are notoriously difficult to quantitatively assess; we often found promising models turned out to produce musical rubbish. These tools provide a quick and painless approach to deployment, allowing both ourselves and future researchers to actually *use* the models, and share them with others for qualitative evaluations.

### 5.1.1 Cultural Implications

It is impossible to avoid the feeling that this thesis was conducted at a pivotal moment. One year ago, when we began to propose ideas for a new rhythmic generative system, tools such as ChatGPT and Stable Diffusion were not yet publicly released. Since then people have become both enamored and terrified with the capabilities that these models demonstrate. It is only a matter of time before AI with comparable fidelity is developed for the musical realm.

Subjectively, we have observed that many discussions around such tools focus on two objectives: replacing artists and/or making music easier. In other words - *saving cost and time.* We echo the concerns of the authors of Anticipatory Music Transformer[47] in regards to the economic implications on labor markets for creative work, the potential for further cultural homogenisation, and the uncertainty around legal frameworks. We have spoken to many friends and artistic collaborators who are both afraid of and angry at the type of technologies being developed. They have a serious and grounded fear that large-scale music generation systems will wipe out many of the current financial opportunities that exist for creatives.

Like most revolutionary technologies, generative AI systems will have both positive and negative impacts. We harbor concerns about the sizable financial incentives that might favor the negative aspects, specifically the tools engineered to marginalize or altogether eliminate artists from the creative process.

Our aspiration with this thesis was to offer a proposition for an alternative approach, showcasing to researchers and musicians alike the potential of deep learning models to become an invaluable resource to creative communities. By developing a model that requires fine-grained control, and works best in an existing audio production environment, we believe that VAEDER is more akin to an 808 drum-machine than a powerful LLM.

## 5.2 Limitations & Future Work

The VAEDER model was not without its limitations, both technical and artistic, and we wish to summarize them here, as well as potential areas of future research.

#### Microtiming

Similar to [2] and [9] our models consistently struggled to learn offset information. This was first noted as the offset losses never decreased by a tangible amount, and confirmed throughout our evaluations. This is a serious limitation when attempting to model human performance characteristics. It is particularly evident in certain genres, such as jazz and funk, which are characterised by micro-timing patterns, e.g. *swing*. VAEDER has a reasonable understanding of the Hit and Velocity patterns of the genres we trained on, but the lack of meaningful Offsets results in drum patterns that do not sound stylistically-accurate.

It could be beneficial to separate the Hit calculations from the Velocity and Offset components in a hierarchical approach, creating one system to exclusively *compose*, and another to then generate a *performance*. This would solve the issue of predicting velocities and offsets on non-active voices. Furthermore, it would enable researchers to train the *compositional* system on a much larger corpus of data, as there is a considerable amount of open-source quantized score information available.

#### Modelling Intensity

We also noted a consistent limitation in VAEDER's capability to model higher intensity values. This behavior was not identified with density, which uses the same disentanglement and injection techniques. With this discrepancy, we believe there is something either in the initial calculation of intensity itself, or the distribution of the dataset, that leads to this limited capability. Furthermore, measuring intensity as the average velocity of the pattern is not necessarily aligned to human perception; for example, a medium crash cymbal can evoke a higher sense of intensity than a loud snare. We would be curious to see how the model performs on intensity, if trained with a modified parameter calculation, and additional high-intensity genres such as metal or breakcore.

#### Datasets

It is notable that within the realm of performance rhythm generation, there is (to the best of our knowledge) a single, open-source dataset: the Groove MIDI Dataset. This is in stark contrast to other fields, such as natural language processing, vision and many audio-domain tasks such as source separation. As stated by [20], the majority of research in this field is directed towards the development of new models, whereas there could be tremendous value gained from creating new, high-quality symbolic datasets. Whilst we mean no criticism of the Magenta team who kindly created and open-sourced the GMD, there are cultural implications of having a singular dataset that is primarily focused on Western, 4/4 rhythms. This became most apparent in our work when the model struggled to learn certain genre representations, such as

*Jazz*, whilst attaining much higher results on genres such as *Rock*.

#### Data Representation

The HVO data representation technique provides many advantages, namely the ability to encapsulate dynamic performances in a ememory-efficient manner. This is not without its drawbacks however; as noted in [2] the models consistently fail to model triplets, and presumably other rhythmic divisions outside of 16th-note divisions. We again wish to highlight the cultural implications of this, as many of the world's richest rhythmic traditions draw upon an incredible array of divisions that cannot be accurately modeled with a 16th-note grid. We attempted early in the thesis to create one such representation which combined the data-efficiency of HVO with a more flexible token-based approach that could work with any time-signature and beat division. However, it became apparent that an entire re-work of the evaluation methods, model, and training procedure would be necessary to accommodate this change, and we ultimately had to focus on the key objective of parameter disentanglement.

With the above two points in mind, we generally wish to emphasize the importance of investing in a more diverse and culturally-inclusive collection of datasets and representation methods.

### Fine Tuning

Finally, it is noteworthy that VAEDER has managed to create captivating outputs despite training on just 15 hours of data, a small drop compared to standard audio datasets. The biggest barrier to training on more data is the lack of availability; most rhythm datasets are simply *scores*, which contain no information about velocity or microtiming. We hypothesize that, in conjunction with a hierarchical approach, it could be beneficial to first train a foundational rhythm model on a much larger repository of drum scores. This would give it a greater degree of flexibility and accuracy in its Hit calculations. It would then be possible to fine-tune the model on a smaller dataset, such as the GMD or even an individual drummer's performances. Such an approach could both improve the capabilities of the model whilst simultaneously unlocking new, smaller datasets for personalized rhythmic models.

## 5.3 Resources

We provide here a set of links to easily access various elements of the research presented throughout this thesis.

VAEDER model repository, including pre-trained checkpoints, evaluation tools, and training scripts:

`https://github.com/behzadhaki/GrooveTransformer`

Video Demonstration:

`https://youtu.be/v6VtPNv7cXI?feature=shared`

Selection of MIDI and audio files:

`https://github.com/behzadhaki/GrooveTransformer/tree/dev/VAE_Control_Classifiers/demos/vaeder`

NeuralMidiFx VST3 Plugin:

`https://neuralmidifx.github.io/`

NeutoneMIDI SDK:

`https://github.com/QosmoInc/neutone_sdk`

# Bibliography

[1] Gillick, J., Roberts, A., Engel, J. H., Eck, D. & Bamman, D. Learning to groove with inverse sequence transformations. In *ICML*, vol. 97 of *Proceedings of Machine Learning Research*, 2269–2279 (PMLR, 2019).

[2] Haki, B., Nieto, M., Pelinski, T. & Jordà, S. Real-Time Drum Accompaniment Using Transformer Architecture. In *Proceedings of the 3rd Conference on AI Music Creativity* (AIMC, 2022). URL `https://doi.org/10.5281/zenodo.7088343`.

[3] Briot, J., Hadjeres, G. & Pachet, F. *Deep Learning Techniques for Music Generation* (Springer, 2020).

[4] Jabreel, M. & Moreno, A. A deep learning-based approach for multi-label emotion classification in tweets. *Applied Sciences* **9**, 1123 (2019).

[5] Kumar, R. L. *et al.* Recurrent neural network and reinforcement learning model for covid-19 prediction. *Frontiers in Public Health* **9** (2021). URL `https://www.frontiersin.org/articles/10.3389/fpubh.2021.744100`.

[6] Vaswani, A. *et al.* Attention is all you need. *CoRR* **abs/1706.03762** (2017).

[7] Shaw, P., Uszkoreit, J. & Vaswani, A. Self-attention with relative position representations. *CoRR* **abs/1803.02155** (2018).

[8] Huang, C. A. *et al.* Music transformer: Generating music with long-term structure. In *ICLR (Poster)* (OpenReview.net, 2019).

[9] Ramos, T. P. Completing audio drum loops with transformer neural networks. URL `https://zenodo.org/record/5554854`.

[10] Introduction to autoencoders. `https://www.jeremyjordan.me/autoencoders`.

[11] Ji, S., Luo, J. & Yang, X. A comprehensive survey on deep music generation: Multi-level representations, algorithms, evaluations, and future directions. *CoRR* **abs/2011.06801** (2020).

[12] Hadjeres, G., Pachet, F. & Nielsen, F. DeepBach: a steerable model for Bach chorales generation. In Precup, D. & Teh, Y. W. (eds.) *Proceedings of the 34th International Conference on Machine Learning*, vol. 70 of *Proceedings of Machine Learning Research*, 1362–1371 (PMLR, 2017). URL `https://proceedings.mlr.press/v70/hadjeres17a.html`.

[13] Jiang, N., Jin, S., Duan, Z. & Zhang, C. Rl-duet: Online music accompaniment generation using deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, 710–718 (2020).

[14] Liu, J. *et al.* Symphony generation with permutation invariant language model. *arXiv preprint arXiv:2205.05448* (2022).

[15] Roberts, A., Engel, J. H., Raffel, C., Hawthorne, C. & Eck, D. A hierarchical latent vector model for learning long-term structure in music. In *ICML*, vol. 80 of *Proceedings of Machine Learning Research*, 4361–4370 (PMLR, 2018).

[16] Thickstun, J., Hall, D., Donahue, C. & Liang, P. Anticipatory music transformer (2023). `2306.08620`.

[17] Jiang, J., Xia, G. G., Carlton, D. B., Anderson, C. N. & Miyakawa, R. H. Transformer vae: A hierarchical model for structure-aware and interpretable music representation learning. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 516–520 (IEEE, 2020).

[18] Tan, H. H. & Herremans, D. Music fadernets: Controllable music generation based on high-level features via low-level feature modelling. In *ISMIR*, 109–116 (2020).

[19] Shih, Y., Wu, S., Zalkow, F., Müller, M. & Yang, Y. Theme transformer: Symbolic music generation with theme-conditioned transformer. *CoRR* **abs/2111.04093** (2021).

[20] Hernandez-Olivan, C., Hernandez-Olivan, J. & Beltran, J. R. A survey on artificial intelligence for music generation: Agents, domains and perspectives. *arXiv preprint arXiv:2210.13944* (2022).

[21] Yin, Z., Reuben, F., Stepney, S. & Collins, T. Deep learning's shallow gains: a comparative evaluation of algorithms for automatic music generation. *Machine Learning* **112**, 1785–1822 (2023).

[22] Maezawa, A., Yamamoto, K. & Fujishima, T. Rendering music performance with interpretation variations using conditional variational RNN. In *ISMIR*, 855–861 (2019).

[23] Choi, K., Hawthorne, C., Simon, I., Dinculescu, M. & Engel, J. H. Encoding musical style with transformer autoencoders. In *ICML*, vol. 119 of *Proceedings of Machine Learning Research*, 1899–1908 (PMLR, 2020).

[24] Hawthorne, C., Huang, A., Ippolito, D. & Eck, D. Transformer-NADE for piano performances .

[25] Lattner, S. & Grachten, M. High-level control of drum track generation using learned patterns of rhythmic interaction. In *WASPAA*, 35–39 (IEEE, 2019).

[26] Dahale, R., Talwadker, V., Rao, P. & Verma, P. Generating coherent drum accompaniment with fills and improvisations. *CoRR* **abs/2209.00291** (2022).

[27] Makris, D., Guo, Z., Kaliakatsos-Papakostas, M. A. & Herremans, D. Conditional drums generation using compound word representations. In *Evo-MUSART*, vol. 13221 of *Lecture Notes in Computer Science*, 179–194 (Springer, 2022).

[28] Nuttall, T., Haki, B. & Jorda, S. Transformer neural networks for automated rhythm generation URL `https://nime.pubpub.org/pub/8947fhly/release/1`.

[29] Wang, S. *et al.* Controllable data generation by deep learning: A review. *arXiv preprint arXiv:2207.09542* (2022).

[30] Lample, G. *et al.* Fader networks: Manipulating images by sliding attributes. *Advances in neural information processing systems* **30** (2017).

[31] Engel, J. H., Hoffman, M. D. & Roberts, A. Latent constraints: Learning to generate conditionally from unconditional generative models. In *ICLR (Poster)* (OpenReview.net, 2018).

[32] Pati, A. & Lerch, A. *Latent Space Regularization for Explicit Control of Musical Attributes.*

[33] Mezza, A. I., Zanoni, M. & Sarti, A. A latent rhythm complexity model for attribute-controlled drum pattern generation. *EURASIP J. Audio Speech Music. Process.* **2023**, 11 (2023).

[34] Kawai, L., Esling, P. & Harada, T. Attributes-aware deep music transformation. In *ISMIR*, 670–677 (2020).

[35] Wu, S. & Yang, Y. Musemorphose: Full-song and fine-grained music style transfer with just one transformer VAE. *CoRR* **abs/2105.04090** (2021).

[36] Dai, Z. *et al.* Transformer-xl: Attentive language models beyond a fixed-length context. *CoRR* **abs/1901.02860** (2019).

[37] Raffel, C. Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching. URL `https://colinraffel.com/projects/lmd/`.

[38] Fu, H. *et al.* Cyclical annealing schedule: A simple approach to mitigating KL vanishing. *CoRR* **abs/1903.10145** (2019).

[39] Biewald, L. Experiment tracking with weights and biases (2020). URL `https://www.wandb.com/`. Software available from wandb.com.

[40] McInnes, L., Healy, J., Saul, N. & Grossberger, L. Umap: Uniform manifold approximation and projection. *The Journal of Open Source Software* **3**, 861 (2018).

[41] Haki, B., Lenz, J. & Jorda, S. Neuralmidifx: A Wrapper Template for Deploying Neural Networks as VST3 Plugins. *AIMC 2023* Https://aimc2023.pubpub.org/pub/givwzz98.

[42] Breve, B., Cirillo, S., Cuofano, M. & Desiato, D. Perceiving space through sound: mapping human movements into midi. 49–56 (2020).

[43] Huang, Y. & Yang, Y. Pop music transformer: Beat-based modeling and generation of expressive pop piano compositions. In *ACM Multimedia*, 1180–1188 (ACM, 2020).

[44] Hsiao, W.-Y., Liu, J.-Y., Yeh, Y.-C. & Yang, Y.-H. Compound word transformer: Learning to compose full-song music over dynamic directed hypergraphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 178–186 (2021).

[45] Zeng, M. *et al.* Musicbert: Symbolic music understanding with large-scale pre-training. In *ACL/IJCNLP (Findings)*, vol. ACL/IJCNLP 2021 of *Findings of ACL*, 791–800 (Association for Computational Linguistics, 2021).

[46] Fradet, N., Briot, J.-P., Chhel, F., El Fallah-Seghrouchni, A. & Gutowski, N. Miditok: A python package for midi file tokenization. In *22nd International Society for Music Information Retrieval Conference* (2021).

[47] Thickstun, J., Hall, D., Donahue, C. & Liang, P. Anticipatory music transformer (2023). `2306.08620`.