МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
ФАКУЛЬТЕТ ІНФОРМАТИКИ
КАФЕДРА ІНФОРМАТИКИ

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: «**DIFFUSION MODELS FOR MUSIC GENERATION**»

Виконав студент 4-го року навчання,
Спеціальності
Комп’ютерні науки - 122
Савкін Гліб Ігорович
Керівник: Крюкова Галина Віталіївна
кандидат наук, доцент
Рецензент:
_____

Кваліфікаційна робота захищена з оцінкою _____

Секретар ЕК _____
(підпис)

«___» _____ 2024 р.

Київ - 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

ФАКУЛЬТЕТ ІНФОРМАТИКИ

КАФЕДРА ІНФОРМАТИКИ

ЗАТВЕРДЖУЮ

Завідувач кафедри інформатики

доцент, кандидат наук.

_____ Гороховський С. С.

«___» _____ 2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на кваліфікаційну роботу

студенту Савкіну Глібу Ігоровичу

факультету інформатики 4 курсу бакалаврської програми

Тема: «**DIFFUSION MODELS FOR MUSIC GENERATION**»

Зміст ТЧ до кваліфікаційної роботи:

Abstract

Introduction

1. Domain-Specific Challenges and Terminology

2. Existing Deep Learning Architectures for Music Generation

3. DDPMs and Model Structure

4. Model Implementation and Performance

Results

Conclusion

Sources

Додатки (за необхідністю)

Дата видачі «___» _____ 2024 р.

Керівник _____

Завдання отримав _____

Графік підготовки кваліфікаційної роботи до захисту

Графік узгоджено «___» _____ 2024 р.

№	Назва етапу кваліфікаційної роботи	Термін виконання етапу	Підпис наукового керівника	Дата ознайомлення наукового керівника	Примітка
1.	Отримання теми кваліфікаційної роботи.	20.12.2023			
2.	Ознайомлення з темою кваліфікаційної роботи.	10.03.2024			
3.	Розробка плану та структури роботи.	10.04.2024			
4.	Робота з науковою літературою, опис основних означень.	30.04.2024			
5.	Дослідження результатів отриманих в літературі.	05.05.2024			
6.	Робота над текстовим оформленням результатів.	10.05.2024			
7.	Попередній аналіз кваліфікаційної роботи. Виправлення помилок.	11.05.2024			

8.	Попередній захист кваліфікаційної роботи.	12.05.2024			
9.	Захист кваліфікаційної роботи.	29.05.2024			

Науковий керівник _____

(ПІБ)

Виконавець кваліфікаційної роботи _____

(ПІБ)

Contents

Abstract

Denoising diffusion probabilistic models (DDPMs) are a type of deep learning model that have shown impressive results in various fields. Their unorthodox approach to content generation has allowed them to excel in tasks where other models have struggled to achieve great performance.

However, though DDPMs have been used extensively in the fields of image and video generation, their use in other fields is much less prominent. One of such fields is audio generation. This paper explores the possibility and practicality of using DDPMs for tasks related to the generation of sound. Specifically, we will be looking at their capabilities in generating musical pieces. We will also provide a brief overview of any existing models for music generation. The reason for this is twofold: it will provide a great point of comparison for the results achieved with diffusion models, and some key ideas and developments from other model types will be used in the development of our own diffusion model. We will focus on the three most popular solutions for audio generation, specifically Variational Autoencoders (VAEs), Generative Adversarial Networks (GANs), and Transformer-Based Models.

This comparison is based on both the theoretical advantages and disadvantages of the reviewed models, as well as the results obtained during the training and sampling of our diffusion model. We hope that the combination of these metrics will allow for better analysis and will help highlight the benefits of DDPMs in this field.

Introduction

In recent years, generative networks have exploded in popularity, gaining huge amount of traction, and even a little notoriety. This change is caused by two things - improved hardware capability, that has been steadily growing for year, and by new, advanced model architectures. Those networks, utilizing the power of deep learning, have already caused a significant impact in how generated content is perceived. If previously such content has been viewed as something experimental, restricted to matters of academia, nowadays it became a staple of the day-to-day interactions. Many industries have been revolutionized overnight, going from a tedious work handled by humans, to fully automated processes, operated by machine learning algorithms. The three domains that have received the most changes are undoubtedly, domains of text, images, and spoken audio generation.

In the domain of text generation, Large Language Models like ChatGPT and Gemini have shown to be able to comprehend questions and produce impressive responses. In fact, those models have even been able to even outperform highly trained professionals in exam results[23]. These models are usually based on a transformers architecture. It allows them to excel at capturing long-term connections and context in text data. By learning from large amounts of text, they learn to generate contextually relevant data to prompts. They've already experienced a great amount of improvements in the technologies used, and will likely continue to grow, fueled by a significant attention they tend to garner.

Image generation has seen major developments in a very similar manner. With a relatively recent introduction of diffusion networks, models like Stable Diffusion and DALL-E have been able to produce scarily realistic samples. Both models have enjoyed a massive popularity in both personal and professional environments.

The domain of spoken audio generation, i.e. text-to-speech (TTS), while less sensational than the other two has also had a lot of steady iterative improvements. Some

examples of such models include WaveNet and Tacotron. TTS technology has a plethora of possible applications, ranging from virtual assistants included in most smartphones, to accessibility tools for the visually impaired. Same technology has also been used to elevate LLMs to another level, by giving them a voice.

Comparatively, the domain of music generation, though a part of the audio domain, has not experienced the same level of attention and development as those mentioned above. While there has been a great amount of research in this area, due to its inherent lack of commercial value, the application of various advanced generative models in music synthesis remains less explored. This difference in popularity presents a great opportunity to explore the potential of such models in this domain, which is uniquely challenging in multiple ways. When it comes to music, due to its inherent rigid structure, humans can often intuitively feel that a composition is badly composed, even without extensive theoretical knowledge. As a comparison, it's not unusual to people not trained in visual arts to miss obvious faults in synthesized images. On top of that, both creativity and emotional expression are required to produce useful results.

Also worth mentioning that music is a very complex domain to create new compositions in even for humans. Unlike text or images, where patterns and semantics are relatively well-defined, music encompasses abstract elements such as melody, harmony, rhythm, and timbre. The development of generative models that can produce musically satisfying compositions requires a great understanding of music theory, human perception, and cultural context.

Out of all popular generative models, Denoising Diffusion Probabilistic Models (DDPMs) have recently received a massive surge in popularity, especially in the field of image generation. DDPMs have demonstrated impressive results and are defacto the go-to models for that domain. Their success in image generation is a proof of their potential for handling complex data.

With the successes of DDPMs in image generation in mind, there is a great basis for exploration of their applications in music generation. Music, similarly to images, is a

complex domain with multiple factors determining the perceived quality of the output. The potential of diffusion models to capture the nuances of music is an exciting prospect that warrants an investigation.

In this work, we aim to research the possibility of applications of diffusion models for the task of symbolic audio generation. We will implement and train a diffusion model, comparing its performance against other popular models for music generation. By providing results and analysis, this study aims to demonstrate the advantages of DDPMs for music generation and to create a foundation for future research in the use of generative models in music generation.

1. Domain-Specific Challenges and Terminology

As mentioned before, the domain of music presents some unique challenges for the synthetic generation, even compared to some of the more complex domains. Unlike most data, which usually has only a single possible representation (for example - pixels for images, or text tokens for text), musical data can be represented in various forms, each with their unique properties, benefits, and drawbacks. In this chapter we explore some of the differences between those different types of data, as it is crucial for working in this domain.

1.1 Symbolic and Non-Symbolic Data

On a basic level, musical data can be categorized into two types: symbolic and non-symbolic data.

Symbolic music data is a type of data that represents music in a format, similar to a musical score. It encodes information about the notes, their duration, intensity, and sometimes even the instruments to be used. Symbolic data does not contain the actual audio but rather a list of instructions about how to recreate the musical piece. In other words, symbolic data represent music notation, but not the music itself. This data format is much easier to work with, especially in the context of machine learning, and is widely used for the purposes of generational networks. Generative networks, especially in the domain of music generation, often prefer symbolic data as it is much smaller than the actual sound waves, and can be used to accurately recreate a piece of music. This way even a simple network can sound crystal clear. On the other hand though, datasets for symbolic data are limited and are unlikely to ever be significantly expanded, due to the difficulty of acquiring new notations of musical pieces. Symbolic data is the type that the DDPM described in this work will train on and generate new samples of.

Non-symbolic music data, on the other hand, refers to the actual audio recordings. This data is much better at capturing the tiny details of music, such as timbre, tone, and the other imperceptible intricacies of a live performance. Compared to the symbolic data,

it has more personality of the performer, their personal touch and charm, and captures minor details of the composition that symbolic data is likely to miss. However, when it comes to machine learning, processing and generating new samples comes at the price of vastly increased calculational expenses. Regardless, this type of musical data shouldn't be disregarded. It has a huge advantage in the amount of data it has available. State-of-the-art models for non-symbolic music generation produce great results even now, and will likely only improve as better and better hardware becomes available. As a matter of fact, during the writeup of this paper a new model called Suno AI has been made freely available, and it has once again demonstrated the huge potential of non-symbolic music approach. Other noteworthy examples are WaveGAN, an architecture proposed by researchers at NVIDIA, and Jukebox, that was developed by OpenAI.

1.2 MIDI

One of the most common formats for symbolic music data is MIDI (Musical Instrument Digital Interface). MIDI is a standard that describes a protocol, a digital interface, and connectors. It was originally developed to allow musical instruments, computers, and other music related devices to communicate with each other. It's also one of the more prominent data formats to perform machine learning training on.

MIDI does not contain the exact audio signal, it instead consists of messages about a musical performance. For example, when a keyboard is played, it sends messages about which keys are pressed, how hard they are pressed, and when they are released.

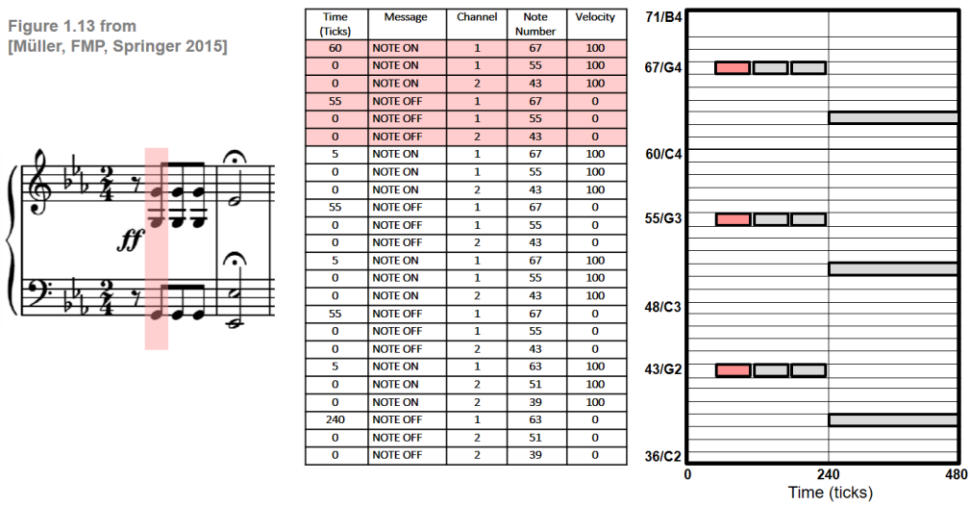

Figure 1.13 from
[Müller, FMP, Springer 2015]

Time (Ticks)	Message	Channel	Note Number	Velocity
60	NOTE ON	1	67	100
0	NOTE ON	1	55	100
0	NOTE ON	2	43	100
55	NOTE OFF	1	67	0
0	NOTE OFF	1	55	0
0	NOTE OFF	2	43	0
5	NOTE ON	1	67	100
0	NOTE ON	1	55	100
0	NOTE ON	2	43	100
55	NOTE OFF	1	67	0
0	NOTE OFF	1	55	0
0	NOTE OFF	2	43	0
5	NOTE ON	1	67	100
0	NOTE ON	1	55	100
0	NOTE ON	2	43	100
55	NOTE OFF	1	67	0
0	NOTE OFF	1	55	0
0	NOTE OFF	2	43	0
5	NOTE ON	1	63	100
0	NOTE ON	2	51	100
0	NOTE ON	2	39	100
240	NOTE OFF	1	63	0
0	NOTE OFF	2	51	0
0	NOTE OFF	2	39	0

Figure 1. Example of MIDI file structure, compared to a note sheet

A typical MIDI message contains several pieces of information[20]. The most important ones are the note number (which corresponds to a specific pitch), velocity (how hard the note is played), and duration (how long the note is held). On top of this, MIDI can include other data like tempo, control changes (like volume or modulation), and program changes (which can specify changes not directly related to the music performance, such as instrument changes), though most of these are irrelevant for the purposes of machine learning.

Parameter	**Description**
Note Number	Corresponds to a specific pitch
Velocity	Indicates how hard the note is played
Duration	Specifies how long the note is held
Tempo	Defines the speed or pace of the music
Control Changes	Include adjustments like volume or modulation

Program Changes	Specify changes unrelated to music performance

Table 1. Key MIDI Data Parameters

MIDI's format makes it a great choice for composing and editing music, as it allows for very easy changes to the key components of a musical piece compared to full on recordings. However, it is quite limited when it comes to the nuances of a live performance, it doesn't convey audio properties like timbre or the subtle variations of a human performer.

One of the most significant applications of MIDI in machine learning, and one that is of interest for the purposes of this work, is the task of symbolic music generation. Compared to alternatives (piano roll, ABC notation, etc.), MIDI is the most detailed and intricate, allowing it to get as close as possible to the level of quality of non-symbolic data. Additionally, due to the popularity of the format, there is a huge number of datasets available with different styles of musical performances to choose from.

Representation	**Description**	**Advantages**	**Disadvantages**
MIDI	Digital protocol for music data transmission	Detailed and intricate	Requires MIDI-compatible software/hardware
Piano Roll	Grid-based notation with notes and time	Visual representation	Lacks additional metadata like velocity
ABC Notation	Text-based format using letters and symbols	Lightweight and human-readable	Very simplistic, can only describe chords

Table 2 Comparison between MIDI and alternative music representation formats

There are a lot of MIDI datasets to choose from. For example, The Maestro dataset is a curated collection of classical piano performances captured in MIDI format. It has over 200 hours of recordings of various pianists playing a diverse set of classical pieces.

Maestro is well regarded when it comes to research for its high-quality recordings. It also focuses primarily on the classical music, which is generally the most popular genre of music to train models on. However that is also its major weakness, as any other genre of music is simply not possible to create with a model trained on Maestro.

The Lakh MIDI dataset, or the "Large-scale MIDI Dataset," is a collection of MIDI files sourced from various online sources. It has a sizeable collection of more than 175 thousand MIDI compositions, that belong to numerous musical genres and styles. In this Lakh provides a sizeable advantage of high variability, allowing synthesized compositions to be noticeably unique. It covers popular music, classical compositions, folk tunes, and more.

Another good dataset for the task would be the MetaMIDI dataset. It is larger than both of the previously mentioned datasets, and consists of a more than 400 thousand MIDI files, with an even higher variability in genres, styles, and time periods. However, MetaMIDI is not freely available to the public. Because of this it is only used in some of the most advanced projects, where researchers are able to gain access to it.

2. Existing Deep Learning Architectures for Music Generation

The field of synthetic content generation, and specifically the domain of music generation, is very diverse and has a large number of useful architectural solutions. Each one of those is approaching the task of the music synthesis in a different and unique way. Though certain models have established themselves as the go to choice in some areas, when it comes to symbolic data there is not a single concrete option, but rather a set of different approaches. For the purposes of this paper it is crucial to carefully examine those to properly assess the standing of diffusion based approach among them. This chapter aims to achieve that, by performing a brief exploration into the workings of several alternative methods in music synthesis, namely Variational Autoencoders (VAEs), Transformer Models, and Generative Adversarial Networks (GANs). Each of these models will approach the task from a different angle, not only by providing a baseline for future comparison, but also by establishing some crucial features that will be used by the final model.

2.1 Variational Autoencoders

Variational Autoencoders (VAEs) are a class of relatively simple generative models that have been a staple in generative machine learning for a very long time. They have been used to solve a huge range of tasks[19], including music generation among others. Despite their simplicity variational autoencoders can quickly produce great results.

The strength of VAEs lies in their ability to learn continuous representations of complex data and how to efficiently reduce it to a lower dimensionality and to restore the reduced data to its original form. By optimizing a variational lower bound on the data likelihood, VAEs capture only the most important features of the input data distribution in the latent space.

2.1.1 VAE Architecture

At its core, a VAE consists of two primary components: an encoder and a decoder. The encoder transforms the input data into a latent (hidden) space representation, while the decoder reconstructs the input data from this latent representation. The latent space is a lower-dimensional space that captures the essential characteristics of the input data.

In the encoding phase, the input data x is passed through the encoder, which outputs parameters of a probability distribution, typically a Gaussian distribution. These parameters are the mean μ and variance σ^2 (or log-variance $log\ \sigma^2$ for numerical stability). This distribution represents the encoder's belief about the data's latent representation.

In music generation, a VAE can learn to encode musical pieces into a latent space that captures their primary features. By sampling different points in this latent space and decoding them, the VAE can generate new musical pieces that share characteristics with the training data.

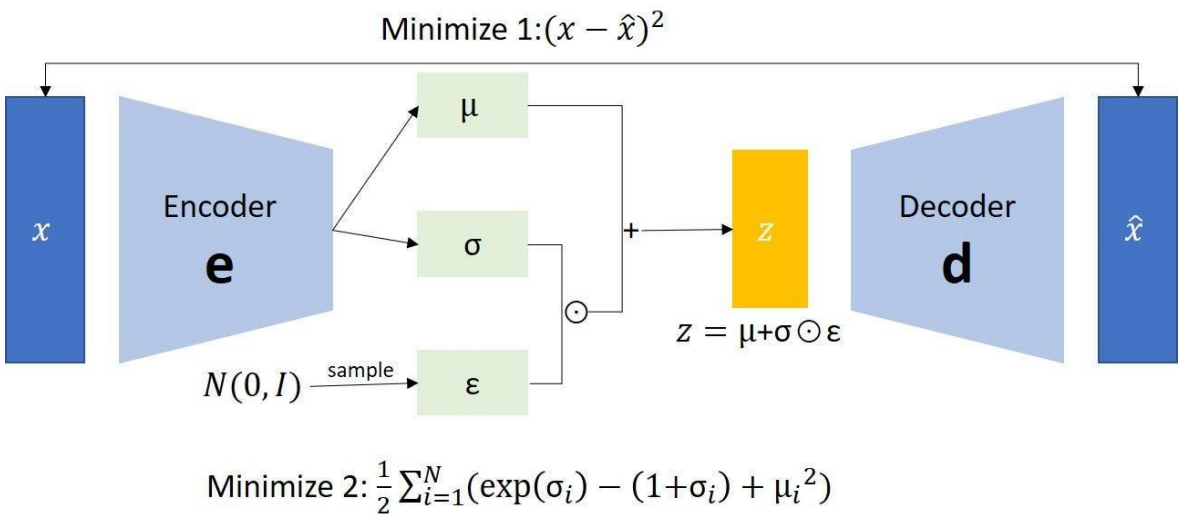

Figure 2. Basic VAE architecture

2.1.2 The Reparameterization Trick

To enable back propagation through random sampling, VAEs use what is known as the reparameterization trick. Instead of sampling z directly from the distribution $N(\mu, \sigma^2)$, a variable ϵ is sampled from a standard normal distribution $N(0,1)$ and is used to compute $z = \mu + \sigma \odot \epsilon$. This reparametrization makes the sampling process differentiable, as ϵ is independent of the model parameters.

The sampled latent variable z is then passed through the decoder to generate the reconstructed output $\hat{x}$. The decoder learns to map the latent representation back to the data space, attempting to reconstruct the input data as accurately as possible.

2.1.3 Loss Function

The training of a VAE involves optimizing two key parts in its loss function: the reconstruction loss and the Kullback-Leibler (KL) divergence. The reconstruction loss (mean squared error for continuous data or binary cross-entropy for binary data) measures how well the decoder can reconstruct the input data from the latent representation. The KL divergence acts as a regularizer, ensuring that the learned distribution $q(z \mid x)$ (the encoder's output) is close to the prior distribution $p(z)$, which is typically is a standard normal distribution $N(0,1)$. The loss function can be represented as:

$$L = -E_{q(z \mid x)}[logp(x \mid z)] + KL(q(z \mid x) \mid\mid p(z)) \quad (1)$$

2.1.4 Common Issues

Variational Autoencoders (VAEs) are a powerful tool in generative modeling, but they have a few challenges that can worsen their results when attempting to produce high-quality outputs. One of the largest issues is the delicate balancing of the reconstruction loss and the KL.

The reconstruction loss attempts to ensure that the generated samples resemble the input data, in order to preserve the structure of the input dataset. However, this can sometimes create an overfitting issue. Alternatively, results can come out being very similar to the original data, without much variance.

At the same time, the KL divergence encourages the latent space to be well-structured and continuous. However, a large focus on this part of the loss calculation can cause images to be overly generic.

In order to solve this balancing issue hyperparameters need to be carefully balanced, and regularization techniques need to be properly used in order to achieve a balanced distribution of the output samples. However, this is often hard to achieve. There are various other methods to achieve this balance - scheduled KL divergence weight, alternative divergence measures, adversarial training schemes etc.

Also, the choice of latent space dimensionality and the architecture of the encoder and decoder have a huge influence on the VAE's performance. A higher-dimensional latent space can allow for more detailed representation, but can also cause overfitting. And a lower-dimensional latent space can prevent model from fully capturing the entire set of data properties.

2.1.5 MusicVAE

MusicVAE is a variation encoder model, developed by researchers at Google's Magenta project[23]. It’s a state-of-the-art encoder for symbolic music, specifically trained to encode and decode MIDI sequences. MusicVAE plays an important role in this project, therefore in this section we want to delve deeper into the structure of this model.

In general MusicVAE has 2 very important aspects, that set it apart from other similar models - how it handles temporal dependencies and its hierarchical latent structure.

Music is an inherently temporal domain, and basic VAEs struggle with capturing long running dependencies in such data. To combat this, MusicVAE uses RNN network to handle those dependencies. Using the RNN network’s hidden state, the MusicVAE is able to capture temporal aspect of the data much better, resulting in a more natural sound.

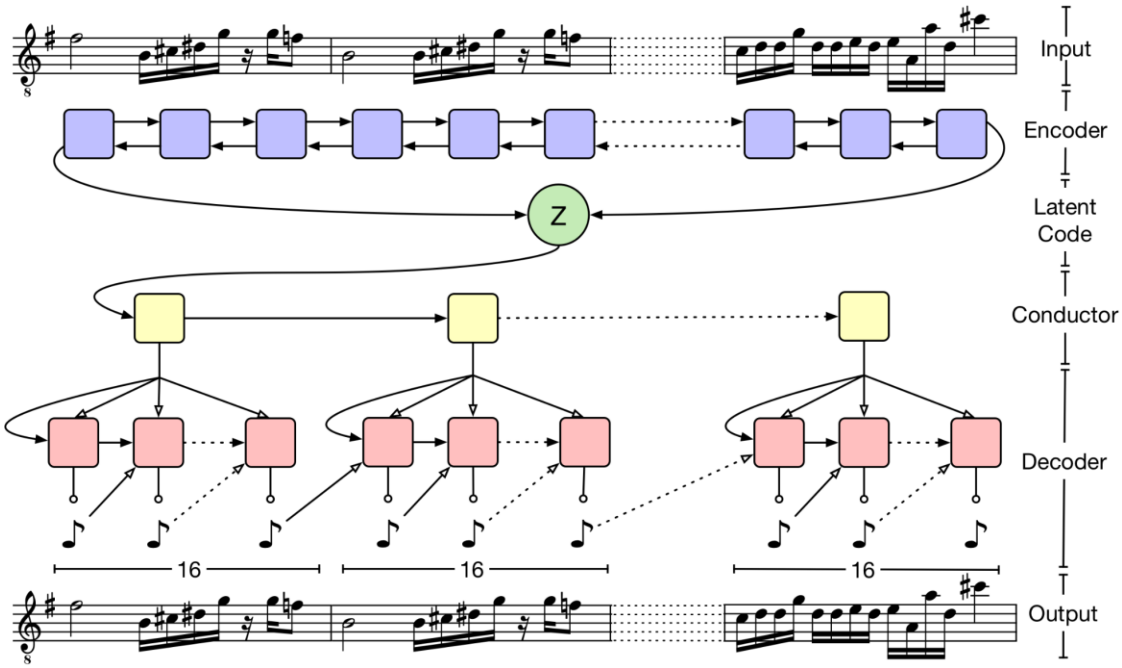

Figure 3. Pre-trained encoder of MusicVAE

The way latent space is set up in MusicVAE is completely unique to this model. Unlike the traditional single-layer latent space, MusicVAE uses a multi layer one, where each layer represents a different level of music - notes, measures, phrases. By using this more complex latent space model is able to generate content in a more ‘thoughtful’

manner - instead of generating each step based on only the note, MusicVAE is aware of the higher level concepts of the piece.

Combined, those two provide a better structured output in both the temporal and the abstract higher level structure. As the result, MusicVAE is able to produce some of the most impressive synthesized results to date.

2.1.5 Conclusion

VAEs are a powerful tool for both a basic generation, and a complex one, when combined with other advanced models and improvements. They can certainly be used for music generation with a great performance and results.

However, harnessing the full capabilities of VAEs in music generation requires some level of skill. Balancing the reconstruction loss and the KL divergence is crucial to obtain satisfactory results. On top of that, a proper dimensionality of latent space has to be selected. Though both of those problems are considerably easier to solve when compared to the issues of other models in this paper, it is important to note them. Finally, the performance of VAEs can be lacking. In order to improve their performance they are often combined with other models, which in turn removes the main benefit of VAEs - ease of training.

2.2 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) have been a popular solution for an assortment of generative tasks for quite some time, and have proven to be a powerful tool when it comes to content synthesis. They are especially prominent in the domain of image generation[25][26]. GANs have an unusual architecture, which consists of two neural networks, engaged in a game like adversarial competition. There have been attempts to use GANs for music generation as well, for example the Magenta team has developed a promising GANSynth model[27]. In this chapter, we will examine the architecture of GANs, understand how they can be applied in the domain of music, and explore their prons and cons for this domain.

2.2.1 Generator and Discriminator Networks

The architecture of GANs is significantly different from the more traditional generative models. A typical GAN consists of two neural networks, the generator, and the discriminator, engaged in a continuous adversarial process.

Generator (G): The generator's role is to create data that is indistinguishable from real-world data. It starts with a random noise vector z, drawn from a standard normal distribution, and transforms this vector into a data instance. In the context of music, this would be either a segment of audio (for non-symbolic data) or a sequence of musical notes, presented in MIDI format (for symbolic data). The transformation is typically achieved through a series of deconvolutional layers (in the case of audio) or fully connected layers (for symbolic music).

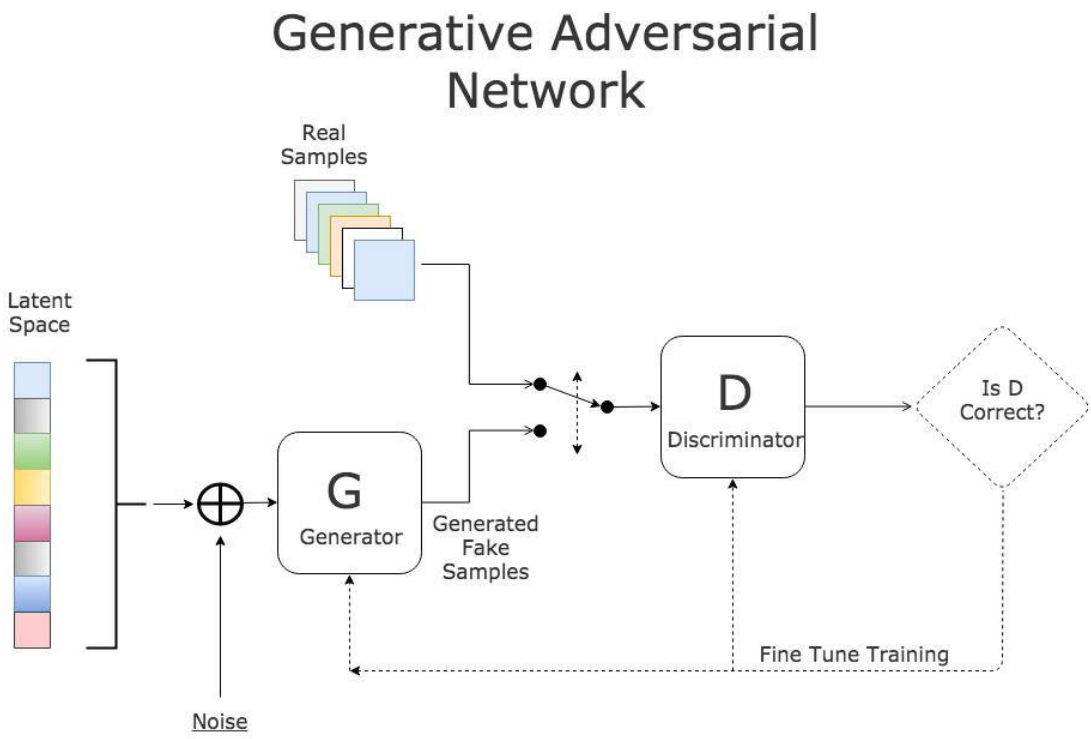

Figure 4. Basic GAN architecture

Discriminator (D): The discriminator acts as a critic, tasked with distinguishing between real data and the fake data generated by G. It receives two types of inputs: real data instances and the outputs produced by G. The discriminator would typically be a convolutional neural network (for non-symbolic data) or a fully connected network (for symbolic data), outputting a single value representing the probability that the given input is real.

The training of GANs involves a game, where G and D are trained simultaneously with opposing goals:

Generator's Objective: To generate data that is so similar to real data that D cannot distinguish it from the real thing.

Discriminator's Objective: To accurately classify the input as real or fake. D aims to maximize its probability of correctly classifying both real and fake data.

Both those objectives can be described with the following formula, which the generator attempts to maximize and the discriminator attempts to minimize:

$$min_G max_D V(D,G) = E_{x\sim p_{data}(x)}[logD(x)] + E_{z\sim p_z(z)}[log(1 - D(G(z)))] \quad (2)$$

In this equation, the first term, $E_{x\sim p_{data}(x)}[logD(x)]$, represents the expected value of the logarithm of the discriminator's output over all real data instances x. Since this term is related to the discriminator's performance on real data, it is not directly influenced by the generator.

The second term, $E_{z\sim p_z(z)}[log(1 - D(G(z)))]$, is where the generator's performance comes into play. Here:

- z is a noise vector sampled from the generator's input noise distribution $p_z(z)$
- $G(z)$ is the output of the generator when given z as input, essentially the fake data generated by the generator.
- $D(G(z))$is the discriminator's output when given this fake data. It represents the probability, according to the discriminator, that the generated data is real.
- $log(1 - D(G(z)))$ is the logarithm of the probability that the generated data is fake, as assessed by the discriminator.

The training process of GANs is a delicate balance. If the discriminator becomes too good, the generator's gradients can vanish, making it hard for G to improve. On the other hand, if the generator is too good, the discriminator's task becomes too difficult. This balance is crucial for the training process to succeed.

2.2.2 Mode Collapse and Solutions

A common issue in GAN training is mode collapse, where the generator starts producing a limited variety of outputs. Several techniques, like introducing more randomness, using different architectures, or using regularization methods, have been developed to mitigate this.

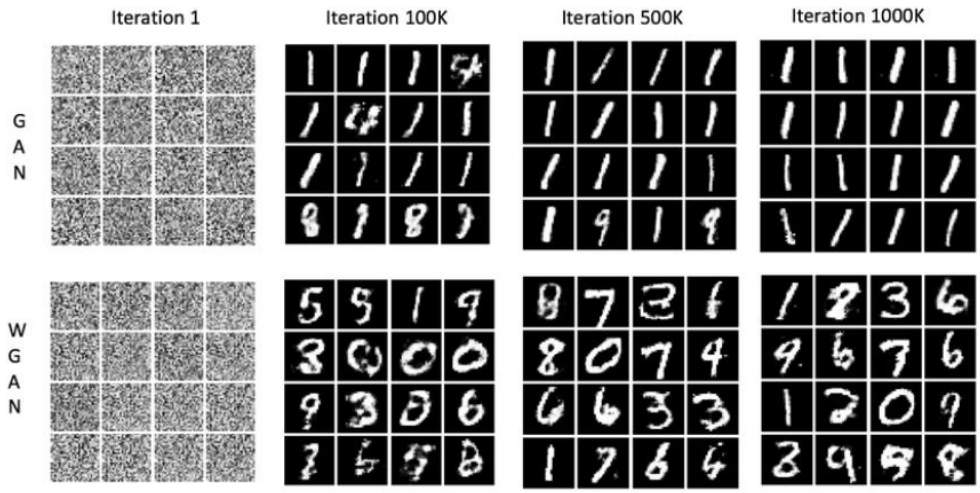

Figure 5. Results from GAN and WGAN (Wasserstein GAN) on MNIST dataset. GAN suffers from mode collapse

2.2.3 Conclusion

The architecture of GANs, with its adversarial competition between the generator and discriminator, is a powerful framework for generating complex data like music. The design of this architecture boosts its ability to learn detailed data distributions, and this makes GANs a powerful tool for AI music generation. However, there are challenges that are inherent to the training process of GANs (such as mode collapse), those require much care and precision to achieve great outcomes.

2.3 Transformer-based Models

Transformer-based models have been introduced comparatively recently[28] as a new groundbreaking development in machine learning, and have since become one of the most prominent architectures. Originally developed for the purposes of natural language processing, these models have demonstrated high capability in dealing with sequential data, making them a topic of interest for music generation. Transformer models are especially prominent in non-symbolic music generation, and are by for the most used architecture for that task.

2.3.1 Fundamentals of Transformer Architecture

The transformer model abandons the conventional recurrent layers used in previous sequence-to-sequence models. Instead, it relies on a structure built around the self-attention mechanism.

Self-attention mechanism allows the model to consider the entire sequence of data at once without overwhelming it. Attention allows it to weigh the importance of different parts of the sequence in relation to each other.

For a given input sequence, the model first computes three vectors for each element: a query vector (Q), a key vector (K), and a value vector (V). These vectors are obtained through learned linear transformations of the input data. The attention weight between two positions in the sequence is computed as the dot product of their query and key vectors, followed by a softmax operation to obtain normalized weights. Formulaically, the attention weights A are computed as:

$$A = softmax(\frac{QK^T}{\sqrt{d_k}}) \tag{3}$$

where d_k is the dimension of the key vectors, and the division by $\sqrt{d_k}$ is a scaling factor to prevent overly large dot products.

The output of the self-attention layer for each position is a weighted sum of the value vectors, with weights given by the attention weights.

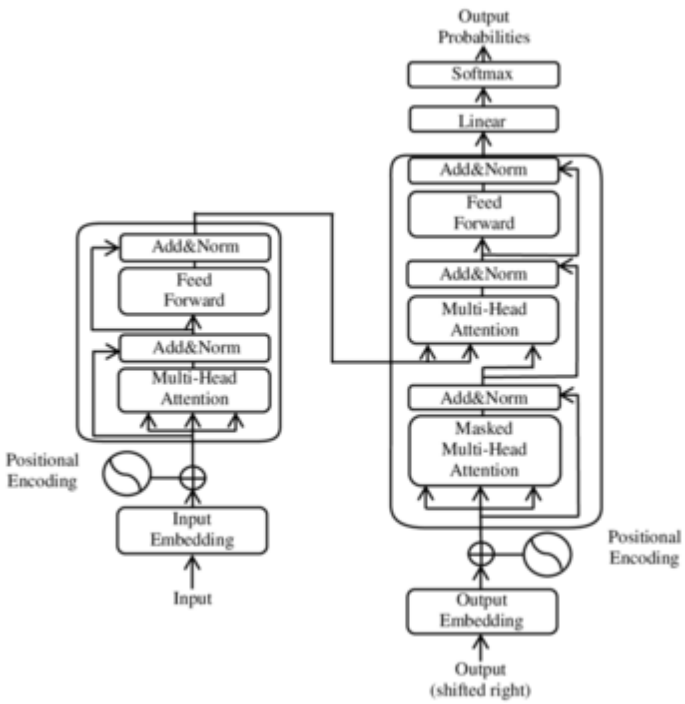

Figure 6. Basic transformer architecture

2.3.2 Transformer Blocks

The transformer model consists of an encoder and a decoder, each consisting of multiple layers. Each layer in both the encoder and decoder contains a self-attention mechanism, and additionally, the decoder layers contain a second attention mechanism that focuses on the output of the encoder stack.

Since the transformer does not inherently process sequential data in order, it requires a mechanism to take into account the position of each element in the sequence. This is achieved through positional encodings, which are added to the input embeddings at the base of the encoder and decoder stacks. Positional encodings can be sine and cosine functions of different frequencies.

2.3.3 Conclusion

In music generation, transformers treat a piece of music as a sequence, similar to a sentence in language processing. The self-attention mechanism allows the model to consider entire musical phrases and their relationships, making it adept at understanding complex musical structures and dependencies.

This architecture allows for a nuanced understanding and generation of music, capturing both the immediate and broader context of musical elements. The mathematical structure of transformers enables them to process and generate sequences with a level of complexity that suits the complex nature of music quite well.

3. DDPMs and Model Structure

Denoising Diffusion Probabilistic Models (DDPMs) have been introduced relatively recently in a paper called “Deep Unsupervised Learning using Nonequilibrium Thermodynamics”[2] (2015). In a relatively short period, those models became a prominent model category for a range of machine learning tasks, one of the most notable being the task of image synthesis. They are known for their ability to generate high-quality data through a unique training and sampling process.

The unique training and sampling process used by DDPMs, which uses the principles of nonequilibrium thermodynamics, allows them to generate high-precision images with impressive visual fidelity. By simulating the diffusion process inherent in natural image formation, DDPMs can effectively model the underlying probability distribution of real-world images, enabling the synthesis of diverse and realistic visual content. Therefore, DDPMs have become a cornerstone in the field of image synthesis, offering unparalleled capabilities for creating visually appealing and contextually rich images.

This chapter aims to explore the detailed workings and structure of DDPMs and to demonstrate the possibility of their application for music generation. We will start by looking closely at the ideas behind these models, focusing on the aspects that allow them to turn random noise into organized and meaningful data through a process known as diffusion. Afterward, the chapter will shift its focus to the application of DDPMs in the area of music generation. We will discuss how the DDPM framework can be applied to the complex field of music, addressing challenges and possibilities of using these models to create new, expressive, unique musical pieces.

3.1 Denoising Diffusion Probabilistic Models Overview

As described in the original paper, the training process of a diffusion model has two steps. Firstly, those models “systematically and slowly destroy structure in a data distribution through an iterative forward diffusion process”. Following this, they “learn a

reverse diffusion process that restores structure in data, yielding a highly flexible and tractable generative model of the data".

In essence, this process gradually adds noise to the data over a series of steps, slowly destroying the original. Diffusion models are then trained to learn a reverse diffusion process that restores the data structure. Eventually, the same process can be used to create new original data from randomly sampled noise.

Afterwards, the model is trained to execute a reverse diffusion process, precisely restoring the original data structure. This duality allows for the creation of a highly flexible and tractable generative model capable of synthesizing new data from randomly sampled noise.

3.1.1 Tractability and Flexibility

One of the goals the original paper focuses on is the concept of **"tractability"** and **"flexibility"** of probabilistic models. In this context:

- **Tractability** is how easy it is to work with a model using mathematical methods and to fit the model to actual data. A model is tractable if it can be understood and solved using straightforward calculations and if it does not require complex techniques or a lot of computing power to train it on real data.
- **Flexibility** is how well a model can accurately capture and represent different types of situations or behaviors. A model is flexible if it can adapt to various conditions and complexities in the data. It means the model can effectively handle diverse scenarios and provide reliable results, even when the system or phenomena it represents are complex.

Historically, all probabilistic models excel at one only of those parameters - they are either easy to analyze and train, but not able to map out more complex data structures, or they are able to gain a deep understanding of underlying data, but are very difficult to train up to a sufficient level and are difficult to perform analytical analysis on.

All the models we've reviewed previously fall into one of those two categories. As discussed in previous chapters, GANs and transformer-based models can generate high-quality data, but are infamously difficult to train due to things like mode collapse and non-convergence or due to the sheer number of parameters in a model. On the other hand, VAEs are easy to train but lack the quality and deep understanding of their adversaries.

However, diffusion models are able to strike a balance between tractability and flexibility. They are relatively easy to train and analyze, because "estimating small perturbations is more tractable than explicitly describing the full distribution with a single, non-analytically-normalizable, potential function". At the same time, DDPMs have shown a great level of data comprehension, comparable to those of GANs and transformer models.

This jack-of-all-trades property makes DDPMs especially attractive for the purposes of music generation. High tractability makes it possible to train a competent model even with few computational resources available without sacrificing the quality of the generated samples.

3.2 Forward and Reverse Diffusion Processes

As mentioned previously, the diffusion process can be split into two processes - forward diffusion, during which noise is gradually added to the image, and reverse diffusion, during which the model learns to reverse the process and remove noise from the image.

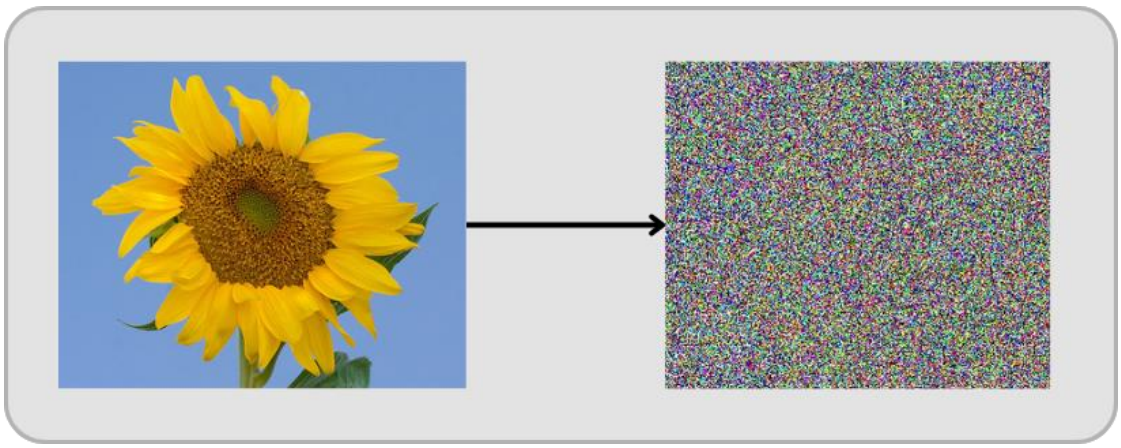

Figure 7. An example of a forward diffusion process

3.2.1 Forward Diffusion

The input image can be thought of as a distribution of some sort. The goal of the forward diffusion process is to transform this unknown distribution into another, known one. Typically the resulting noisy distribution is a multivariate Gaussian distribution. Each step t of forward diffusion adds noise to the input, which can be defined by the following formula:

$$q(x_t \mid x_{t-1}) := N(x_t, \sqrt{1-\beta_t} x_{t-1}, \beta_t I), \tag{4}$$

where x_t is the output on the time step t, x_{t-1} is the input on the time step t and output of the previous step $t-1$, N is normal distribution, $\sqrt{1-\beta_t} x_{t-1}$ is the mean, and $\beta_t I$ is variance.

A significant issue with this equation is that in order to generate sample at step t, sample for each previous step also needs to be generated. This ends up being very inefficient and computationally expensive. Instead, there is a way to reUsing this formula, the entire noise to be added at T can be written down as:

$$q(x_{1:T}|x_0) := \prod_{t=1}^{T} q(x_t|x_{t-1}) \tag{5}$$

After this, by defining $\alpha_t := 1 - \beta_t$ and $\underline{\alpha} := \prod_{s=1}^{t} \alpha_s$, Equation (4) can be rewritten to be only dependent on the input data:

$$q(x_t \mid x_0) := N(x_t, \sqrt{\underline{\alpha}_t} x_0, (1 - \underline{\alpha}_t) I) \quad (6)$$

This can be continued, by modifying Expression (4) using the reparametrization trick, mentioned in 2.1.2: $q(x_t \mid x_{t-1}) = N(x_t, \sqrt{1 - \beta_t} x_{t-1}, \beta_t I) = \sqrt{1 - \beta_t} x_{t-1} + \sqrt{\beta_t} \epsilon$, (7)

And finally:

$$q(x_t \mid x_0) := N(x_t, \sqrt{\underline{\alpha}_t} x_0, (1 - \underline{\alpha}_t) I) = \sqrt{\underline{\alpha}_t} x_{t-1} + \sqrt{1 - \underline{\alpha}_t} \epsilon \quad (8)$$

Here $\underline{\alpha}$ is always known, because β is known. With this, this equation makes it possible to calculate noise for any step t without needing to calculate the previous steps.

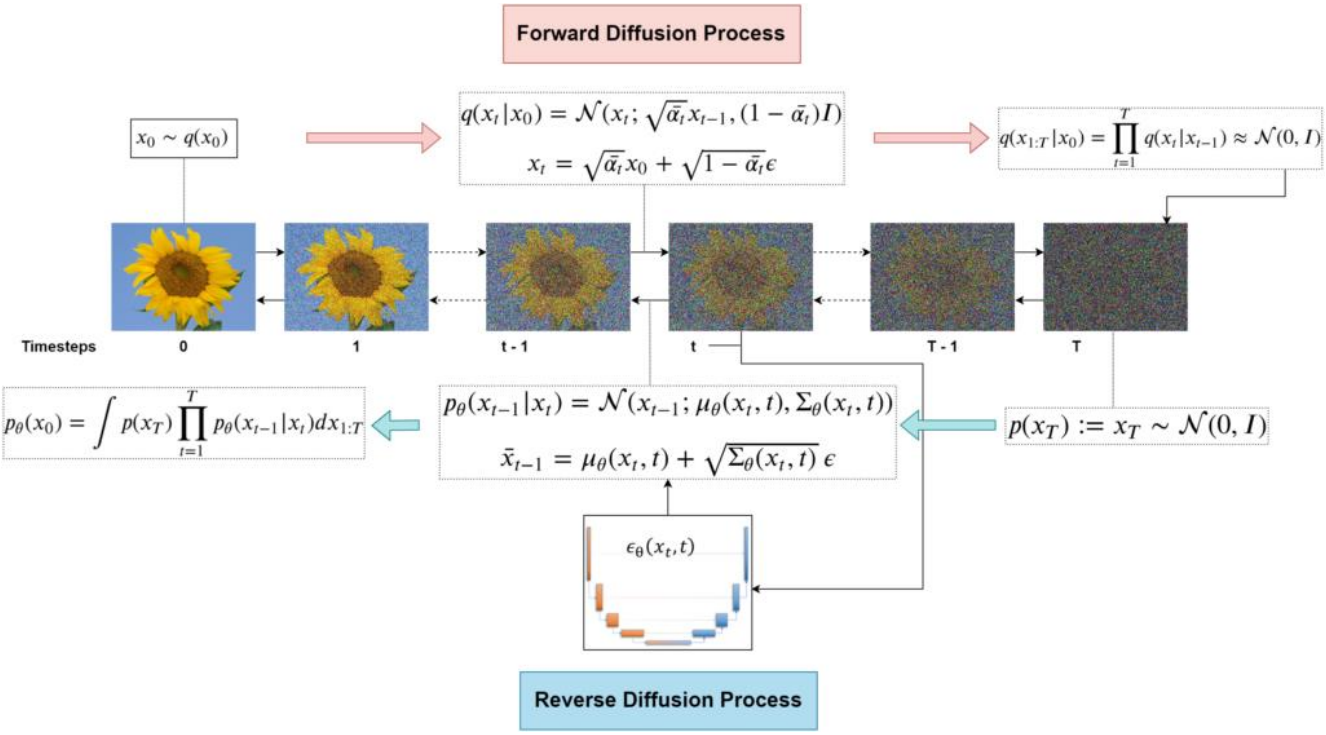

Figure 8. Forward diffusion process demonstration

3.2.2 Scheduling

Scheduling is the plan that decides how to gradually add or remove noise from data over several steps. Well-selected scheduling is extremely important for DDPMs[https://arxiv.org/pdf/2301.10972.pdf], as it determines the amount of noise removed at every step. There are a couple of popular scheduling methods to select from.

Linear Scheduling - one of the most popular scheduling methods, due to its simplicity. Noise levels are increased linearly, gradually increasing with the same speed.

This method can expressed in the following way:

$$\beta_t = \beta_1 + \frac{t-1}{T-1}(\beta_T - \beta_1) \tag{9}$$

Here, T is the total number of steps in the diffusion process, t is the current step of diffusion. With a linear schedule, the initial (β_1) and the final (β_T) noise levels have to be provided, alongside the total step count. In *Ho et al., 2020*, for example, the authors set $T = 1000$, $\beta_1 = 10^{-4}$ and $\beta_T = 0.02$

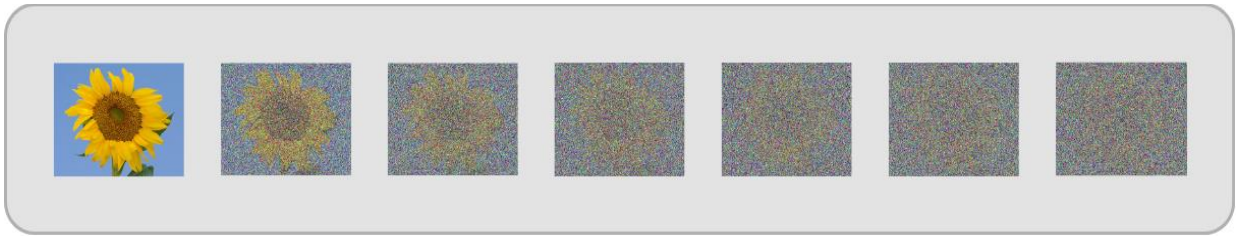

Figure 9. Linear noise scheduling example

Cosine Scheduling - introduced in Improved Denoising Diffusion Probabilistic Models, cosine scheduling aims to reduce the amount of very noisy samples that make up the last half of those generated by linear scheduling. The formula for calculating b_t can be expressed in the following way:

$$\beta_t = 1 - \frac{\underline{\alpha}_t}{\underline{a}_{t-1}},$$

$$\underline{\alpha}_t = \frac{f(t)}{f(0)},$$

$$f_t = cos(\frac{t/T + s}{1+s} \cdot \frac{\pi}{2})^2 \tag{10}$$

Here, s is the 'offset', which prevents β_t from being too small when t is very small. Such offset prevents the first few steps from adding too little noise, as this has been shown to confuse the model. As can be seen from the formula, $\underline{\alpha}_t$, the variance scaling factor at time t, is reduced during the middle section of the process. Thus, noise is added slower during this time, while the rate at both ends is much faster.

Sigmoid Scheduling - first introduced in Scalable Adaptive Computation for Iterative Generation[29], this method remains less popular than others. In essence, it's very similar to cosine scheduling, the only difference being the function used. However, the authors claim that using the sigmoid function results in better stability for higher-resolution training data.

$$x_t = \sqrt{\alpha_t} x_{t-1} + \sqrt{1 - \alpha_t} \epsilon \tag{11}$$

where x_t is the data at time step t, α_t is a predetermined variance schedule, and ϵ is a sample from a standard Gaussian distribution. The variance schedule a_t controls the rate at which noise is added.

3.2.3 Reverse Diffusion Process

The reverse process is where the DDPM learns to generate data. It involves learning to reverse the diffusion process, basically denoising the data to recover the original distribution. In a way, this is similar to any other network architecture reviewed so far. For example, in GANs the generator receives a random noise as input, and tries to generate a new sample based on that. DDPMs follow the same exact process, by attempting to generate a new image from the noise.

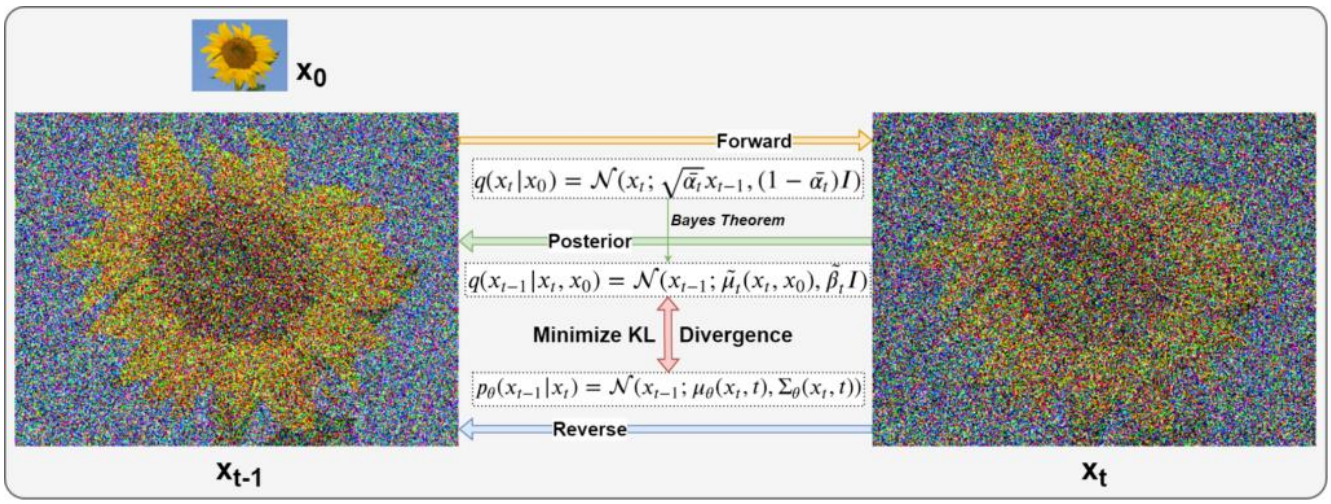

Figure 10. Forward and reverse process on a single step

The goal during reverse diffusion is to be able to predict the noise that has to be removed from a sample. An important point to note is that even though the model is trained on a sequence of images with more and more noise added, it is trained with the goal of removing the entire noise from the image, not just the noise between two timesteps.

The “Denoising Diffusion Probabilistic Models” paper[3] establishes the formula for a reverse diffusion process as

$$p_\theta(x_{0:T}) := p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t) \tag{12}$$

where

$$p_\theta(x_{t-1} \mid x_t) = N(x_{t-1}, \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \tag{13}$$

This formula describes how we simulate a series of Gaussian transitions starting at the initial distribution $p(x_T)$, and repeat it T times using the $p_\theta(x_{t-1} \mid x_t)$ formula. The $p_\theta(x_{t-1} \mid x_t)$ consist of two parts - $\mu_\theta(x_t, t)$ and $\Sigma_\theta(x_t, t)$. The second part was set to an untrainable time-dependent constant, which means that it is constant that is unique for each timestep.

The first part, however, is trainable, and is defined in the following way

$$\mu_\theta(x_t, t) == \frac{1}{\sqrt{a_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\underline{\alpha_t}}}\epsilon_\theta(x_t, t)\right)$$

(14)

from which comes

$$x_{t-1} = N\left(x_{t-1}, \frac{1}{\sqrt{a_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\underline{\alpha_t}}}\epsilon_\theta(x_t, t)\right), \Sigma_\theta(x_t, t)\right) \tag{15}$$

and finally

$$x_{t-1} = \frac{1}{\sqrt{a_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\underline{\alpha}}}\epsilon_\theta(x_t, t)\right) + \sqrt{\beta_t}\epsilon \tag{16}$$

Here the $\epsilon_\theta(x_t, t)$ is the predicted noise of the model

3.2.4 Potential challenges and drawbacks

Though DDPMs are certainly promising, they are not without issues. Some of the most prominent ones are the following:

- **Computational Intensity**: The iterative nature of the reverse diffusion process can be computationally intensive, requiring significant resources for both training and generation. Lucky, with modern improvements, the computational requirements for diffusion networks have been significantly reduced.
- **Hyperparameter Tuning**: The choice of hyperparameters, such as the variance schedule α_t and the architecture of the neural network, can greatly impact the quality of the generated samples.
- **Quality of Generated Music**: Though not unique to DDPMs, quality of generated music is a significant challenge. Ensuring that the generated music is not only novel but also aesthetically pleasing and musically coherent is difficult. This requires careful design and training of the model.

There are other challenges, aside from those listed here, but compared to other models, diffusion networks are a much more feasible solution for small-scale networks that output satisfactory results.

4. Model Implementation and Performance

In this chapter, we will look at an example of a diffusion model that can be used for the generation of various musical pieces. The architecture that was originally planned was heavily inspired by the one, introduced in the Mittal et al. (2021)[1] paper, with some changes in important areas. However, during the development due to a number of technical difficulties, the architecture was completely reimagined. Instead of an established implementation we attempt to develop or own, by attempting to map musical data to a more common for diffusion models domain of images. This model will be unconditional, generating sequences without any configuration or parameters, related to the music style.

4.1 Implementation considerations

4.1.1 PyTorch vs Tensorflow

PyTorch and TensorFlow are two of the most popular deep learning frameworks in the AI community, and each comes with its own set of strengths and weaknesses. Primarily, they have different approaches to computation graphs, PyTorch uses a dynamic one, while Tensorflow utilizes a static graph.

Dynamic graph - PyTorch uses a dynamic computation graph (called define-by-run scheme), which means that the graph is built on the fly. This makes it more intuitive and easier to work with when it comes to debugging and experimenting with complex architectures.

Static graph - TensorFlow uses a static graph architecture, which can be optimized ahead of time. This can lead to better performance and resource utilization in production environments.

Another noteworthy difference is the objectively better scalability and deployment options of TensorFlow, allowing for a proper production setup. In comparison, PyTorch is less enterprise-oriented. On top of that, Tensorflow is natively supported by the Google Colab environment, which we will be using for our training purposes.

Despite all the advantages Tesnorflow provides, we've decided to use PyTorch. There are two main reasons behind such a decision. First, Tensorflow lacks any out of the box implementations for diffusion model architecture. Second, PyTorch is a lot easier to use for smaller research projects like this one.

4.1.2 DDPM Implementation

While it is feasible to build up a diffusion model from the ground up, it's neither efficient nor effective, compared to the established implementations. Considering that the goal of this paper is not to implement a diffusion model, but rather to examine the use of diffusion models for music generation, we will be using an existing library to provide us with the necessary code. Our choice fell on denoising_diffusion_pytorch library[24], as it's a popular and stable PyTorch diffusion model implementation, perfect for our goals.

By using this established library, we gain access to a robust and thoroughly tested implementation of a diffusion model, reducing the resources needed for implementation and enabling us to focus on the core exploration of diffusion models for music generation. The denoising_diffusion_pytorch library provides an extensive set of functionalities, including utilities for training, sampling, and evaluating diffusion models. All of those and more allows us to easily integrate diffusion models into our music generation pipeline. With this pragmatic approach, we aim to streamline our experimentation process and maximize our ability to explore the potential of diffusion models in the domain of music generation.

4.2 Encountered Difficulties During Development

Though using diffusion models for symbolic music synthesis is exciting and promising for the field, in practice, actually developing them comes with a heap of issues and difficulties. In this short section we would like to address those, as they will definitely influence the possibility of using this architecture for any real world solutions.

As we've described previously, the original plan for our model was to adapt and expand on an existing solution[1]. Model described in that paper matches our

requirements perfectly, while still having some flaws that can be improved. It is also quite ingenious in its design - it smartly uses a pretrained variational encoder network, specifically MusicVAE, to encode MIDI files into a latent space and subsequently decode generated data back into MIDI. This allowed it to efficiently use a DDPM with a transformer network below to learn the data distribution.

However, while attempting to use the aforementioned architecture, we've discovered that this project never really got any updates after the initial development phase. And although it was developed only a couple of years prior to this write up, a large majority of stack it was written in has turned out to be outdated. Even worse, MusicVAE, which we hoped to use in our model to encode MIDI data, had the same exact issues - it has long been abandoned, to the point where even the demos the project hosts are not functional. After numerous attempts to fix the compatibility issues we were forced to admit defeat and completely change our plans.

Issues like this are, unfortunately, extremely common. In symbolic music synthesis field, for example, the majority of the scientific community has abandoned any symbolic music generation projects, giving preference to transformer models, trained to generate audio files directly. However, this approach unfortunately leads to a stagnation of anything but the most popular solutions, leaving unique ones behind. When developing this project our partial goal was to contribute to the development of a less mainstream field, in hope of potentially coming up with new and exciting.

4.3 Implemented model structure

Though our original plan of actions fell through, we believe that a different approach might produce a worthwhile results, even though they are likely to be of a worse quilty. In this section we will describe all components of a model we have implemented and trained, using an original approach.

4.3.1 High-Level Overview

Our idea is to try to encode MIDI files into a vector representation. This way they would be functionally identical to a generic black-and-white image that was reduced to a single dimension. As diffusion models perform extremely well when it comes to image generation, we hope that this approach will allow us to utilize this strength for music.

4.3.2 Choice of Dataset

We have discussed available datasets previously, and out of all available options the Maestro dataset was decided to be the best match for our needs. The main reason for this choice is its reduced variability of music genres, which will make it easier to train a decent model for it.

4.3.3 Data Loading

In order to speed up subsequent runs, we will set up a custom dataset with the data already preprocessed. Preprocessing consists of two basic steps:

- Converting MIDI files into event sequences
- Tokenization of event sequences

In order to load and process MIDI files we are using a pretty_midi module - a great tool that simplifies the process of processing MIDI data

```
import pretty_midi
midi_data = pretty_midi.PrettyMIDI(midi_file_path)
```

We will iterate through the extracted data, using a writing down events for ‘note_on’ and ‘note_off’. The space between those events we will fill with a special ‘time_shift’ event, that signifies that no changes have been made.

```python
TIME_SHIFT_RESOLUTION = 0.01

events = []
for instrument in midi_data.instruments:
    for note in instrument.notes:
        events.append(('note_on', note.start, note.pitch, note.velocity))
        events.append(('note_off', note.end, note.pitch))
events.sort(key=lambda x: x[1])

last_time = 0
event_sequence = []
for event in events:
    event_time = event[1]
    time_shift = event_time - last_time

    if time_shift > 0:
        time_shift_steps = int(time_shift // TIME_SHIFT_RESOLUTION)
        for _ in range(time_shift_steps):
            event_sequence.append(('time_shift', TIME_SHIFT_RESOLUTION))
    event_sequence.append(event)
    last_time = event_time
return event_sequence
```

We end up with a list of event objects, that can now be converted into integer values to be used for training. To encode the whole information about an event, we will use a flag that signifies what type of event it is (note_on, note_off, or time_shift), and add a pressed key id if it is a note_on or note_off event. As there are only 128 notes in MIDI file format, flag sizes are easy to compute.

```python
NOTE_ON_FLAG = 0
NOTE_OFF_FLAG = 128
TIME_SHIFT_FLAG = 256

1 usage  glib
def tokenize_event(event):
    if event[0] == 'note_on':
        return NOTE_ON_FLAG + event[2]
    elif event[0] == 'note_off':
        return NOTE_OFF_FLAG + event[2]
    elif event[0] == 'time_shift':
        return TIME_SHIFT_FLAG + int(event[1] / TIME_SHIFT_RESOLUTION)
```

Now that the data is converted and ready to be used it is wrapped into a handy MIDIDataset class, that provides a few useful utils and provide an easy way of saving and loading the resulting dataset

4.3.5 Model Definiton

Now that the data is encoded, the model itself is fairly simple. It has a one dimensional U-net as a backbone, with a one dimensional diffusion on top. Both of those are provided by the denoising_diffusion_pytorch library

```
model = Unet1D(
    dim=64,
    dim_mults=(1, 2, 4, 8)
)

diffusion = GaussianDiffusion1D(
    model,
    seq_length=dataset.max_length,
    timesteps=1000,
    beta_schedule='cosine',
)
```

The models we use are very basic, and this is by design. We hope that the largest benefit will come from our encoding process, and that the standart tools used in image generation can produce great results. The only notable thing here is that we are using a cosine beta scheduling, which we have discussed earlier. It can potentially be replaced with a sigmoid scheduling for a slightly different performance.

4.3.6 Analytics

In order to track the progress of our model during training we will need some specialized metrics. The one we are using is the Pitch Class Histogram Distance. For every 1000 processed samples we get one random real piece of data and a generated one. We then decode them back to the event representation, filter out everything except for the key_on events, and generate note lists. We then calculate which pitch class those notes belong to. Pitch class in this context is an arbitrary group of notes that are close to each other, and are not directly linked to, for example, octaves. We then calculate histograms of how frequent each pitch class is. Those histograms are saved, and once an epoch has finished training, we can calculate the distance between the real ones and the

generated histograms using Jensen–Shannon divergence. The mean of all distances between all pairs of histograms is the final metric.

```
# Calculate histograms every 1000 iterations
if index % 1000 == 0:
    generated_sample = diffusion.sample(batch_size=1)[0]
    real_histograms.append(calculate_pitch_class_distance_histogram(generated_sample))
    real_histograms.append(calculate_pitch_class_distance_histogram)
```

This metric allows us to see whether the generated samples tend to have a distribution of pitches, similar to real musical pieces. While an increase or a decrease might not mean much on its own, we expect that during training this distance will get lower and lower.

4.3.7 Training and Sampling

With all components defined, training and sampling processes are the only ones remaining. They don't introduce anything new or special, and are just the most basic version of themselves. The only point worth mentioning is that during training we are using Adam optimizer with a learning rate of 1e-4

4.3.8 Summary

Our hope was that this model, while structurally very simple, could perform well on the task of music generation. Our training was performed on the Google Colab platform. The examples of generated samples and training results can be found at https://github.com/Xymeterid/musicgen-diffusion.

Conclusion

In this thesis, we have explored the potential of Denoising Diffusion Probabilistic Models (DDPMs) for symbolic audio generation, focusing on the domain of music generation. Despite their widespread use and success in image generation, DDPMs have not received the same level of recognition in the domain of music synthesis. Our investigation aimed to bridge this gap by implementing and training a diffusion model for generating symbolic audio samples, and comparing its performance against other popular AI music generation models such as Variational AutoEncoders (VAEs), Generative Adversarial Networks (GANs), and Transformer-Based Models.

Through our experiments and evaluations, we have demonstrated that DDPMs have a great potential when it comes to symbolic audio generation, primarily due to the great balance of flexibility and tractability those models proved. However, because DDPMs are primarily used for image synthesis, there is a distinct lack of support for other domains in existing diffusion model implementations.

By comparing the performance of DDPMs against other architectures, we have demonstrated the theoretical advantages of diffusion models for the task of music generation. On top of that, the model we developed and trained demonstrates the possibility of application of diffusion models in this domain of machine learning.

Overall, this work contributes to the existing body of research on the application of generative models in symbolic music generation, and demonstrates the potential of DDPMs in this domain.

References

[1] Symbolic Music Generation With Diffusion Models / Mittal Gautam, 2013. *https://archives.ismir.net/ismir2021/paper/000058.pdf*

[2] Sohl-Dickstein J. Deep Unsupervised Learning using Nonequilibrium Thermodynamics / Sohl-Dickstein Jascha, 2015. *https://arxiv.org/pdf/1503.03585.pdf*

[3] Ho J. Denoising Diffusion Probabilistic Models / Ho Jonathan, 2020. *https://arxiv.org/pdf/2006.11239.pdf*

[4] Rogge N. The Annotated Diffusion Model [Електронний ресурс] / N. Rogge, K. Rasul // HuggingFace. – 2022. – Режим доступу до ресурсу: https://huggingface.co/blog/annotated-diffusion

[5] Erdem K. Step by Step visual introduction to Diffusion Models [Електронний ресурс] / Kemal Erdem. – 2023. – Режим доступу до ресурсу: https://erdem.pl/2023/11/step-by-step-visual-introduction-to-diffusion-models

[7] Nichol A. Improved Denoising Diffusion Probabilistic Models / Nichol Alex, 2021. https://arxiv.org/pdf/2102.09672

[8] Prafulla Dhariwal, Alex Nichol Diffusion Models Beat GANs on Image Synthesis, https://arxiv.org/pdf/2105.05233

[9] Jonathan Ho, Tim Salimans Classifier-Free Diffusion Guidance. https://arxiv.org/pdf/2207.12598

[10] What are Diffusion Models? [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

[11] Olaf Ronneberger, Philipp Fischer, Thomas Brox, U-Net: Convolutional Networks for Biomedical Image Segmentation, arXiv:1505.04597, 2015
https://arxiv.org/pdf/1505.04597

[12] Alec Radford, et al. Learning Transferable Visual Models From Natural Language Supervision, PMLR, 2021.
https://arxiv.org/pdf/2103.00020

[13] LING YANG, ZHILONG ZHANG,YANG SONG, SHENDA HONG, RUNSHENG XU, YUE ZHAO, WENTAO ZHANG, BIN CUI, MING-HSUAN YANG - Diffusion Models: A Comprehensive Survey of Methods and Applications https://arxiv.org/pdf/2209.00796.pdf

[14] Akruti Acharya - An Introduction to Diffusion Models for Machine Learning https://encord.com/blog/diffusion-models/

[15] Catherine F. Higham, Desmond J. Higham, Peter Grindrod CBE - Diffusion Models for Generative Artificial Intelligence: An Introduction for Applied Mathematicians https://arxiv.org/pdf/2312.14977.pdf

[16] Karagiannakos S. How diffusion models work: the math from scratch [Електронний ресурс] / Segios Karagiannakos. – 2022. – Режим доступу до ресурсу: https://theaisummer.com/diffusion-models/.

[17] Colin Raffel. "Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching". *PhD Thesis*, 2016.

[18] Payne, Christine. "MuseNet." *OpenAI*, 25 Apr. 2019, openai.com/blog/musenet

[19] Shiv Kumar. Generative Modeling with Variational Autoencoders (VAEs). https://www.linkedin.com/pulse/generative-modeling-variational-autoencoders-vaes-shiv-kumar/

[20] Standard MIDI-File Format Spec. 1.1 © Copyright 1999 David Back. https://www.cs.cmu.edu/~music/cmsip/readings/Standard-MIDI-file-format-updated.pdf

[22] Weiss D. Latest version of ChatGPT aces bar exam with score nearing 90th percentile [Електронний ресурс] / Debra Weiss. – 2023. – Режим доступу до ресурсу: https://www.abajournal.com/web/article/latest-version-of-chatgpt-aces-the-bar-exam-with-score-in-90th-percentile.

[23] Adam Roberts, Jesse Engel, “A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music”, https://arxiv.org/pdf/1803.05428

[24] Denoising Diffusion Probabilistic Model in Pytorch [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/lucidrains/denoising-diffusion-pytorch

[25] Wentao Wang, Xuanyao Huang, “DeepArt: A Benchmark to Advance Fidelity Research in AI-Generated Content”, https://arxiv.org/pdf/2312.10407

[26] Tero Karras, Samuli Laine, “A Style-Based Generator Architecture for Generative Adversarial Networks”, https://arxiv.org/pdf/1812.04948

[27] Jesse Engel, Kumar Krishna Agrawal, GANSYNTH: ADVERSARIAL NEURAL AUDIO SYNTHESIS, https://openreview.net/pdf?id=H1xQVn09FX

[28] Ashish Vaswani, Noam Shazeer, “Attention Is All You Need”, https://arxiv.org/pdf/1706.03762

[29] Allan Jabri, David J. Fleet, Ting Chen, “Scalable Adaptive Computation for Iterative Generation”, https://arxiv.org/pdf/2212.11972