Aurélien Manté
aurelien.mante@ircam.fr

Mémoire de stage

22/08/2024

Génération automatique de jingles audio

Tuteur Ircam
Guillaume Doras
guillaume.doras@ircam.fr

Tuteur Ircam Amplify
Frédéric Amadu
frederic.amadu@ircamamplify.com

Organisme d'accueil
Ircam Amplify
7 rue de Turbigo, 75001 Paris

Rapport non confidentiel

Stage effectué du 01/03/2024 au 31/08/2024

Résumé / Abstract

Ce document présente les recherches menées durant un stage de 6 mois à Ircam Amplify, dans le cadre du parcours ATIAM. Le sujet étant la génération automatique de jingles audio, permettant à l'utilisateur de contrôler temporellement l'audio généré, différents paradigmes de diffusion ont été explorés. Le format utilisé pour représenter les données audio a également été étudié, notamment en terme d'efficacité de la génération. L'intégration du contrôle à la génération, composante primordiale à l'ergonomie d'un tel outil, fait aussi parti de l'analyse.

Une comparaison quantitative et qualitative permet de déterminer que le cadre alliant meilleurs résultats et plus grande efficacité est le paradigme EDM dans un espace latent continu. Toutefois, il faudrait consacrer plus de temps à affiner ces résultats. En effet, les entraînements et l'inférence pour certains modèles prennent énormément de temps (de l'ordre de plusieurs jours chacun pour certains modèles), ce qui a limité notre développement.

L'implémentation a été rendue la plus complète et modulaire possible, pour en permettre une réutilisation presque telle quelle dans de futurs travaux. Parmi les pistes qui nous semblent prometteuses, nous avons identifié l'amélioration de la qualité des données, la définition de paramètres de contrôle plus pertinents, et l'utilisation de couches d'attention avec des *transformers* pour intégrer le contrôle.

Nous avons fait le choix de rédiger ce document en anglais par souci de cohérence : la discipline présente de nombreux termes techniques anglophones qu'il aurait été difficile de traduire sans perdre en intelligibilité.

Mots-clé Apprentissage automatique, diffusion audio, génération conditionnelle, espaces latents

---

This document presents the research conducted during a 6-month internship at Ircam Amplify, as part of the ATIAM program. The focus was on the automatic generation of audio jingles, allowing the user to control the generated audio temporally. Various diffusion paradigms were explored, and the format used to represent audio data was studied, particularly in terms of generation efficiency. The integration of control into the generation process, a crucial component for the usability of such a tool, was also analyzed.

A quantitative and qualitative comparison determined that the framework combining the best results with the greatest efficiency is the EDM paradigm in a continuous latent space. However, more time is needed to refine these results. The training and inference for some models take an enormous amount of time (several days for some models), which limited our development.

The implementation was made as comprehensive and modular as possible, to allow for almost direct reuse in future work. Among the promising directions we have identified are improving the quality of the data, defining more relevant control parameters, and using attention layers with transformers to integrate control.

We chose to write this document in English for the sake of consistency: the field includes many technical terms in English that would have been difficult to translate into French without losing clarity.

Keywords Machine learning, audio diffusion, conditional generation, latent spaces

Acknowledgements

I would like to warmly thank Guillaume for his unwavering support and insightful advice throughout this project. I also want to express my gratitude to Frédéric for the trust he placed in me by entrusting me with this project.

My thanks also go to my colleagues at Ircam, whose advice was extremely valuable in bringing this research to fruition. A big thank you to the sound design team at Ircam Amplify for their valuable feedback on their technical needs and for our more musical discussions.

I extend my thanks to the ATIAM coordination team, especially Cyrielle, for their caring support throughout these six months of internship. Finally, I want to thank my fellow interns for all the lunch breaks, always rich in exchanges and conviviality, which greatly contributed to making this experience even more enjoyable.

Contents

Chapter 1

Introduction

1.1 Subject overview

Creating an audio jingle is an artistic process, requiring significant control for sound designers. However, there are often similarities between different jingles, making it interesting to try to speed up the preliminary phases of sound design. Here, a "jingle" refers to a short soundtrack (less than 10 seconds) composed to accompany a video and embody a sonic identity. The main difference between a jingle and a music track lies in the time scale: while a track requires coherence over several minutes, with broad control of parameters, a jingle aims to develop recognizable sounds with very precise control of parameters over its few seconds. Examples of jingles are available on the Ircam Amplify website[1].

The goal of the internship is to explore various solutions for automatically generating basic elements that make up audio jingles. This generation must be of high quality and highly controllable by the user. Specifically, we aim to enable temporal control over the generated audio, meaning the ability to trace the evolution of a parameter over time.

Audio generation is a well-explored topic in machine learning, with several approaches, each offering its own advantages and disadvantages. Autoregressive models like *WaveNet* [Oord et al. 2016] can generate long sequences but may lack coherence over time. GANs (e.g., *Fre-GAN* [Kim et al. 2021], *MP3net* [Broek 2021], *EVA-GAN* [Liao, Lan, and Zachariah 2024]) offer high-quality sampling but can be challenging to train due to instability and lack diversity (mode coverage). Diffusion methods (such as *WaveGrad* [Chen et al. 2020], *DiffWave* [Kong et al. 2021], *EDMSound* [Zhu et al. 2023], *DiffSound* [D. Yang et al. 2023]) are also widely used, providing both quality and coherence, but they often require long sampling times. The third and last category of the *generative learning trilemma* (Figure 1.1) is the VAEs[2], with fast sampling and good mode coverage but worse sample quality. Given our goal of generating audio clips of just a few seconds, the diffusion approach appeared to be the most suitable (reasonable inference time).

[1] https://soundexperience.ircamamplify.com/
[2] Variational Auto-Encoders

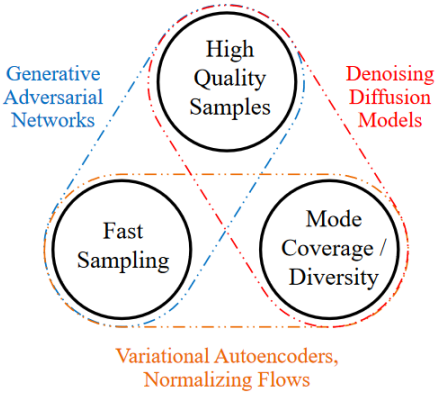

Figure 1.1: Generative learning trilemma. Credit: *Tackling the Generative Learning Trilemma with Denoising Diffusion GANs* [Xiao, Kreis, and Vahdat 2022]

In this report, we begin by reviewing the various diffusion paradigms that currently exist, as well as the common control methods. Following this overview, we present our experiments comparing these different methods, using data that prioritizes modularity over quality. This implementation was made as modular as possible to facilitate reuse in other contexts. Finally, we present the results obtained, which were limited by the time available.

1.2 What is diffusion?

Generative diffusion models were inspired by thermodynamics [Sohl-Dickstein et al. 2015], more precisely by the diffusion of particles in a fluid. The goal is to learn to reverse the diffusion process to recover the original data from its corrupted version. As developed in [Sohl-Dickstein et al. 2015], the goal may be to learn the data's distribution's probability density (continuous state-space) or mass (discrete state-space). In the first case, a diffusion model will navigate back and forth between the data's probability distribution and a gaussian distribution (Figure 1.2); in the second one, between the data's probability mass and a uniform categorical distribution.

Figure 1.2: Example of diffusion process, the model moving to and from data and noise. Credit: *Improving Diffusion Models as an Alternative To GANs, Part 1*[3], A. Vahdat and K. Kreis

Diffusion models have rapidly evolved into a powerful class of generative models for image synthesis [Ho, Jain, and Abbeel 2020, Nichol and Dhariwal 2021, Karras et al. 2022]. Extensive research has been conducted to make it more stable, realistic, and efficient, for instance by positioning it in a latent space [Rombach et al. 2022].

Very quickly, diffusion models were used to generate audio. *WaveGrad* [Chen et al. 2020] and *DiffWave* [Kong et al. 2021] adopted the waveform synthesis approach, but this proved to

[3] https://developer.nvidia.com/blog/improving-diffusion-models-as-an-alternative-to-gans-part-1/

be very resource-intensive and therefore very slow. Other models have adopted the approach of synthesizing spectrograms, either complex [Zhu et al. 2023] or mel-spectrograms [D. Yang et al. 2023, Wu et al. 2023], followed by a phase vocoder to generate the audio. To reduce the dimensionality of the data, latent diffusion has been introduced by [D. Yang et al. 2023].

1.3 Mode coverage vs sample quality

Controlling the diffusion process is key in making those models useful. Those models then require a good balance between mode coverage (to better follow the control) and sample fidelity (synthesis quality). Some solutions have been developed to enable better control of image synthesis while maintaining sample fidelity, like combining a conditional and an unconditional model [Ho and Salimans 2022] or using a LLM[4] [Saharia et al. 2022].

We can distinguish between two types of control: global and local. Global control involves influencing generation at a macro level, often based on textual input. Local control has a different meaning depending on whether we are considering image or audio generation. In the case of images, it involves influencing specific areas in term of the x and y dimensions [Zhao et al. 2023]. In the case of audio, local conditioning acts along the time dimension to influence synthesis differently at each point in time [Wu et al. 2023].

For effective control, the model must demonstrate good mode coverage. To ensure this does not compromise sample fidelity, it is necessary to develop metrics that quantify the fidelity of the generated data. In image synthesis models, the FID[5] is commonly used. For audio synthesis, this metric has been adapted as the FAD[6] [Kilgour et al. 2019]. For reference, the state-of-the-art *MusicControlNet* model [Wu et al. 2023] achieves a FAD ranging between 1.12 and 1.51.

[4] Large Language Model
[5] Fréchet Inception Distance, https://en.wikipedia.org/wiki/Frechet_inception_distance
[6] Fréchet Audio Distance

Chapter 2

Diffusion

2.1 Theoretical background

As seen, diffusion models define a forward and a backward process [Sohl-Dickstein et al. 2015]. The forward process consists in diffusing / adding noise to the original data, to convert it from a complex distribution to a simple one. No learning is involved in this process. Considering a data point sampled from a real data distribution $\mathbf{x}_0 \sim q(\mathbf{x}_0)$, the forward process defines a Markov chain $q(\mathbf{x}_t|\mathbf{x}_{t-1})$.

The learned backward process consists in restructuring / denoising the data, from the fully diffused version to, ideally, the original sample. A model $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ is trained to approximate the probability $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$. Then, by using a random noise as the fully diffused distribution, the model can generate a new and unseen sample from the data distribution it was trained on.

Here, the amount of noise added defines the "diffusion time" (linking to physics). This time can be viewed as either a continuous parameter or as predefined (discrete) steps, hence defining a markov chain (Figure 2.1). The state space of the data also impacts the equations governing diffusion. This space can be either discrete (mass functions) or continuous (probability density functions).

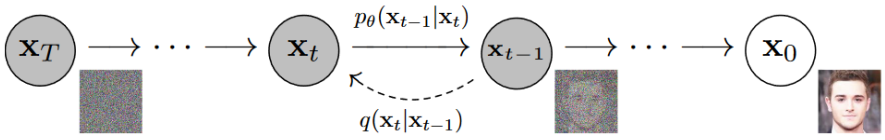

Figure 2.1: Forward / noising process ($q(\mathbf{x}_t|\mathbf{x}_{t-1})$) and backward / denoising process with a learned denoising model ($p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$) for discrete diffusion timesteps between 0 and T. Credit: *Denoising Diffusion Probabilistic Models* [Ho, Jain, and Abbeel 2020]

In 2020, the DDPM[1] framework [Ho, Jain, and Abbeel 2020] defined a novel denoising strategy and a simplified training objective. This model has been used as a foundation in many other papers [Kong et al. 2021, Saharia et al. 2022, Rombach et al. 2022], thereby becoming a reference.

A different denoising strategy is introduced, shifting the focus from predicting the noise parameters (gaussian or binomial) at each step to predicting the noise that was added during the forward diffusion process. In this approach, the model is trained to estimate the noise component present in a noisy sample at each step of the reverse process.

[1] Denoising Diffusion Probabilistic Models

The training objective is reformulated to the mean squared error between the noise added during the forward process and the noise predicted by the model in the reverse process. This reweighted variational bound simplifies the training process, making it more efficient and leading to improved sample quality. This paradigm is presented in detail in subsection 2.2.1 and implemented under the name *DDPM* in our code.

In 2022, EDM[2] models were developed by [Karras et al. 2022]. They unify diffusion models by proposing a continuous-time formulation, based on solutions of SDEs[3] in the case of gaussian noise, allowing efficient sampling, enabling denoising in just a few steps (state-of-the-art is achieved with 18 steps in the paper). While those models are equivalent to DDPM models with infinite timesteps, the latters may be seen as EDM models evaluated at discrete timesteps. They enable stochastic sampling as well as deterministic sampling, which is equivalent to solving the ODE[4] formulation of the diffusion process. This paradigm is presented in detail in subsection 2.2.2 and implemented under the name *EDM* in our code.

In discrete state spaces, while [Sohl-Dickstein et al. 2015] developed diffusion models for binomial distributions [Sohl-Dickstein et al. 2015], they were extended to multinomial distributions by [Hoogeboom et al. 2021], and later adapted to the DDPM architecture (D3PM[5]) by [Austin et al. 2023]. This last paradigm is presented in subsection 2.3.2 and implemented under the name *DiscDPM* in our code. Table 2.1 summarizes the different formalisms we have implemented, classified based on their time and state-space caracteristics.

	Discrete time	**Continuous time**
Discrete state-space	*DiscDPM* [Hoogeboom et al. 2021] (2.3.2)	—
Continuous state-space	*DDPM* [Ho, Jain, and Abbeel 2020] (2.2.1)	*EDM* [Karras et al. 2022] (2.2.2)

Table 2.1: The diffusion formalisms we have implemented, classified based on their time and state-space caracteristics

2.2 Diffusion time

2.2.1 Discrete time

The most common discrete-time approach was developed by [Ho, Jain, and Abbeel 2020]. This formalism (*DDPM*) builds upon the original diffusion probabilistic models [Sohl-Dickstein et al. 2015] to improve the learning and sampling processes.

Diffusion process

The forward process defines T latent variables $\mathbf{x}_1, \ldots, \mathbf{x}_T$, noised according to a variance schedule $\beta_1, \ldots, \beta_T$. For continuous density data, Gaussian noise is usually used, and the joint distribution is:

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \tag{2.1}$$

$$= \mathcal{N}\left(\sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}\right), \text{ with } \beta_t \in [0,1] \forall t \in 1, \ldots, T \tag{2.2}$$

[2] Elucidated Diffusion-based generative Models
[3] Stochastic Differential Equations
[4] Ordinary Differential Equation
[5] Discrete Denoising Diffusion Probabilistic Models

The reparametrization trick lets us derive a close-form expression to sample from $q(\mathbf{x}_t|\mathbf{x}_0)$. With $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{T} \alpha_i$:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon} \text{ where } \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}) \tag{2.3}$$

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}) \tag{2.4}$$

Training

The DDPM formalism uses a network to predict the noise added at each timestep instead of the parameters of this noise (mean and variance). Let $\epsilon_\theta(\mathbf{x}, t)$ be this noise estimator, then the objective function defined by [Sohl-Dickstein et al. 2015] may be minimized by minimizing the simplified loss function defined in eq. 2.5 [Ho, Jain, and Abbeel 2020].

$$\mathcal{L} = \mathbb{E}_{t,q(\mathbf{x}_0),\boldsymbol{\epsilon}}[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t)\|^2] \tag{2.5}$$

Sampling

The goal of the backward (or reverse) process is to be able to sample from the distribution $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$. The model was developed such that $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \sigma_t^2\mathbf{I})$ with $\boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)\right)$. We can thus start with $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$ and sample for T steps following eq. 2.6.

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_{\boldsymbol{\theta}}(\mathbf{x}_t, t)\right) + \sigma_t\boldsymbol{\epsilon} \quad \text{with } \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}) \tag{2.6}$$

2.2.2 Continuous time

The *EDM* formalism [Karras et al. 2022] builds on the continuous-time DPM[6] equations [Sohl-Dickstein et al. 2015] to derive SDEs, proposing an effective noise schedule and sampling method.

Diffusion process

In this approach, instead of using a fixed number of discrete time steps, as in traditional DDPM (see subsection 2.2.1), the diffusion process is modeled as a continuous trajectory, allowing for SDE / ODE formulation instead of a Markov chain. The SDE formulation for the diffusion process was precedently developed by [Song et al. 2021] (eq. 2.7).

$$\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w} \tag{2.7}$$

where $\mathbf{w}$ is the Brownian motion, $\mathbf{f}(;t)$ and $g(t)$ are respectively the *drift* and *diffusion* coefficients of $\mathbf{x}$.

Stochastic and deterministic sampling

This formulation allows for stochastic sampling (reverse SDE) as well as deterministic sampling (reverse ODE), which can both be computed by estimating the score function. With the re-parametrization presented in eq. 2.8 and 2.9, and by choosing the scale factor $s(t) = 1$, the reverse ODE simplifies to eq. 2.10.

[6] Diffusion Probabilistic Models

$$\mathbf{f}(t) = \frac{\dot{s}(t)}{s(t)} \tag{2.8}$$

$$g(t) = s(t)\sqrt{2\dot{\sigma}(t)\sigma(t)} \tag{2.9}$$

$$\mathrm{d}\mathbf{x} = -\dot{\sigma}(t)\sigma(t)\underbrace{\nabla_x \log p_{\sigma(t)}(\mathbf{x}(t), t)}_{\text{score function}}\,\mathrm{d}t \tag{2.10}$$

In the EDM formalism, [Karras et al. 2022] define a *denoiser* $D_\theta(\mathbf{x}, \sigma)$ that minimizes the expected L_2 denoising error (eq. 2.11). The target data and the denoiser are linked through the score function (eq. 2.12).

$$\mathcal{L} = \mathbb{E}_{\mathbf{y}\sim p_{\text{data}}}\mathbb{E}_{\mathbf{n}\sim\mathcal{N}(0,\sigma^2\mathbb{I})}\|D_\theta(\mathbf{y}+\mathbf{n},\sigma) - \mathbf{y}\|_2^2 \tag{2.11}$$

$$\nabla_{\mathbf{x}} \log p(\mathbf{x}, \theta) = \frac{D_\theta(\mathbf{x}, \sigma(t)) - \mathbf{x}}{\sigma(t)^2} \tag{2.12}$$

This allows deterministic sampling by solving the probability flow ODE. For stochastic sampling, [Song et al. 2021] propose to add a Langevin diffusion SDE to the ODE in eq. 2.10. Figure 2.2 presents a 1D overview of stochastic vs deterministic sampling, as well as a simplified visual representation of the diffusion and reverse processes.

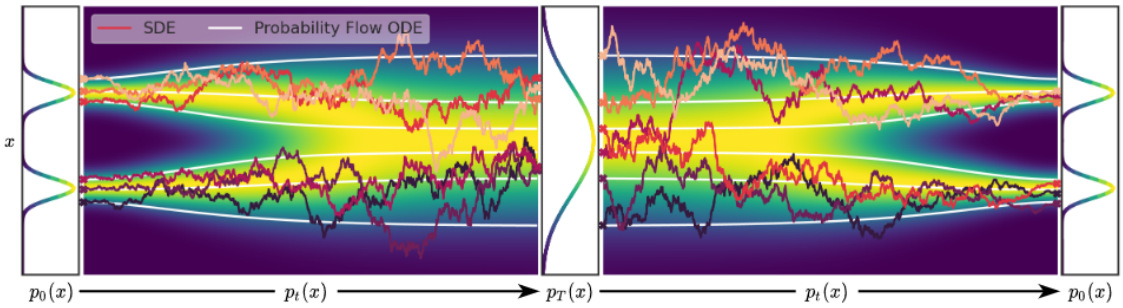

Figure 2.2: Overview of score-based generative modeling through SDEs. The data distribution $p_0(x)$ is mapped to a noise distribution with an SDE that can be reversed for sampling. Credit: *Score-Based Generative Modeling through Stochastic Differential Equations* [Song et al. 2021]

Optimized sampling

When estimating the flow lines for deterministic sampling, a second order correction of the estimated gradient proves to allow for bigger time steps, thus faster and better sampling. [Karras et al. 2022] chose the Heun's 2nd order method, which only doubles the NFE[7] for the denoiser. This function, taking the form of a trained neural network, represents the most computational cost, so limiting the NFE to two by denoising step is key for fast sampling.

Choosing the scale factor $s(t)$ and noise schedule $\sigma(t)$ enables to change the shape of the flow curves. It is interesting to make them as straight as possible, to allow for bigger time steps (them being linear). [Karras et al. 2022] suggest to set $s(t) = 1$ and $\sigma(t) = t$ in this purpose.

[7] Number of Function Evaluations

Preconditioning plays a key role in the quality of the training. [Karras et al. 2022] define weights that regulate the variance of the samples (c_{in}, c_{noise}), and allow a form of skip-connection (c_{skip}, c_{out}). With $F_\theta(\mathbf{x}, \sigma)$ the raw network, the denoiser takes the new form presented in eq. 2.13.

$$D_\theta(\mathbf{x}, \sigma) = c_{\text{skip}}(\sigma)\mathbf{x} + c_{\text{out}}(\sigma)F_\theta(c_{\text{in}}(\sigma)\mathbf{x}, c_{\text{noise}}(\sigma)) \tag{2.13}$$

2.3 State-space

The state-space the diffusion model is evolving in may be discrete or continuous. In our use case, an example of continous data state-space would be spectrograms (real, complex, ...), data bins being floating points. While discrete state-spaces are often associated with text data, they may be useful for audio generation in latent spaces, when the upstream encoder uses vector quantization. This is for instance the case of *Encodec* [Défossez et al. 2022], which can produce both continuous latent codes and discretized ones.

2.3.1 Continuous state-space

When working in continuous state-spaces, the model evaluates probability distributions. This enables diffusing with Gaussian noise, which simplifies the equations involving the score function. Moreover, the SDE / ODE approach developed for the EDM formalism [Karras et al. 2022] requires the state-space to be continuous, since it uses the properties of Gaussian noise.

Thus, the implementations discussed in subsections 2.2.1 (*DDPM*) and 2.2.2 (*EDM*) are based on continuous state-spaces.

2.3.2 Discrete state-space

The original paper on diffusion models included binomial distributions [Sohl-Dickstein et al. 2015], but [Hoogeboom et al. 2021] leveraged this idea to categorical distributions. The key concept is that one data sample $\mathbf{x}$ is encoded as a one-hot vector of dimension K equal to the number of classes (the possible discrete values), which corresponds to the probability mass function for the underlying data distribution.

Diffusion process

The diffusion process adds noise by smoothing the mass function to a uniform distribution. Following the DDPM notations (see 2.2.1), with $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$, $\alpha_i = 1 - \beta_i$, the density probability at timestep t follows the categorical distribution given in eq. 2.14.

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{C}\left(\mathbf{x}_t, \mathbf{p}_{\bar{\alpha}_t}(\mathbf{x}_0) = \bar{\alpha}_t\mathbf{x}_0 + \frac{1-\bar{\alpha}_t}{K}\right) \tag{2.14}$$

Using Bayes' rule and the Markovian nature of q, the reverse process may be explicited as in eq. 2.15. Figure 2.3 illustrates the forward and backward processes.

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{C}\left(\mathbf{x}_{t-1}, \mathbf{p}_{\text{post}}(\mathbf{x}_t, \mathbf{x}_0) = \frac{\mathbf{p}_{\alpha_t}(\mathbf{x}_t) \odot \mathbf{p}_{\bar{\alpha}_{t-1}}(\mathbf{x}_0)}{C}\right) \tag{2.15}$$

where:
$$\mathbf{p}_{\alpha_t}(\mathbf{x}_t) \odot \mathbf{p}_{\bar{\alpha}_{t-1}}(\mathbf{x}_0) = \left(\alpha_t\mathbf{x}_t + \frac{1-\alpha_t}{K}\right) \odot \left(\bar{\alpha}_{t-1}\mathbf{x}_0 + \frac{1-\bar{\alpha}_{t-1}}{K}\right) \tag{2.16}$$

$$C = \sum_{k=1}^{K} \left[\mathbf{p}_{\alpha_t}(\mathbf{x}_t) \odot \mathbf{p}_{\bar{\alpha}_{t-1}}(\mathbf{x}_0)\right]_k \tag{2.17}$$

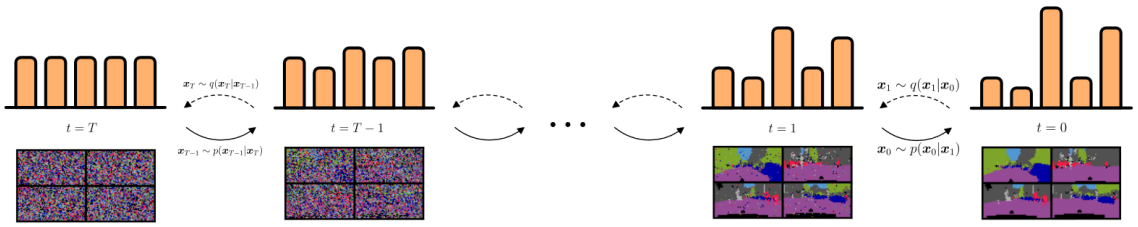

Figure 2.3: Overview of the multinomial diffusion process. From right to left, $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ gradually adds noise to the data. From left to right, a generative model $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ gradually denoises the signal. Credit: *Argmax Flows and Multinomial Diffusion: Learning Categorical Distributions* [Hoogeboom et al. 2021]

Sampling

Considering we have a trained network $F_\theta(\mathbf{x}_t, t)$ modeling $\mathbf{x}_0$, we can model $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ from eq. 2.15 as presented in eq. 2.18. A new latent variable is sampled from this updated categorical distribution at each sampling step.

$$\begin{aligned} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) &= q(\mathbf{x}_{t-1}|\mathbf{x}_t, F_\theta(\mathbf{x}_t, t)) \\ &= \mathcal{C}\left(\mathbf{x}_{t-1}, \mathbf{p}_{\text{post}}(\mathbf{x}_t, F_\theta(\mathbf{x}_t, t)) = \frac{\mathbf{p}_{\alpha_t}(\mathbf{x}_t) \odot \mathbf{p}_{\bar{\alpha}_{t-1}}(F_\theta(\mathbf{x}_t, t))}{C}\right) \end{aligned} \tag{2.18}$$

Training

The network is trained by minimizing the KL divergence[8] between the distributions $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ and $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ (eq. 2.19).

$$\mathcal{L} = \sum_{k=1}^{K} \mathbf{p}_{\text{post}}(\mathbf{x}_t, \mathbf{x}_0)_k \log\left(\frac{\mathbf{p}_{\text{post}}(\mathbf{x}_t, \mathbf{x}_0)_k}{\mathbf{p}_{\text{post}}(\mathbf{x}_t, F_\theta(\mathbf{x}_t, t)_k}\right) \tag{2.19}$$

2.4 Control

Controlling diffusion generation is essential to enable practical applications of the model. To achieve this, several design choices are available. The integration of control into the network layers has a major impact on the effectiveness of this control and the stability of the training. In our research, we identified four methods of control integration: affine modulation, channel-wise concatenation, attention layers using transformers [Vaswani et al. 2023], and conditionally parametrized convolutions (*CondConv*) [B. Yang et al. 2020].

Control via modulation involves modulating the latent variable with the output of a control branch. This modulation can take the form of an affine modulation (*FiLM* [Perez et al. 2017]) or a sum operation (*Controlnet* [Zhang, Rao, and Agrawala 2023]), sometimes including convolutional layers initialized with zero-weights (*zero-convolutions*) so that the control is gradually integrated during training [Zhang, Rao, and Agrawala 2023; Zhao et al. 2023; Wu et al. 2023]. In this case, using a copy of a pre-trained unconditional generator as control branch seems to yield better results than randomly initialized weights [Zhang, Rao, and Agrawala 2023].

[8] Kullback–Leibler divergence, https://en.wikipedia.org/wiki/Kullback-Leibler_divergence

Chapter 3

Experiments

3.1 Development

Each diffusion formalism developed (*DDPM* (subsection 2.2.1), *EDM* (subsection 2.2.2), *DiscDPM* (subsection 2.3.2)) was tested on the *swiss-roll* dataset from *scikit-learn* [Pedregosa et al. 2011]. Results of the *DDPM* framework with a 4-layer 128-wide MLP[1] are presented in Figure 3.1.

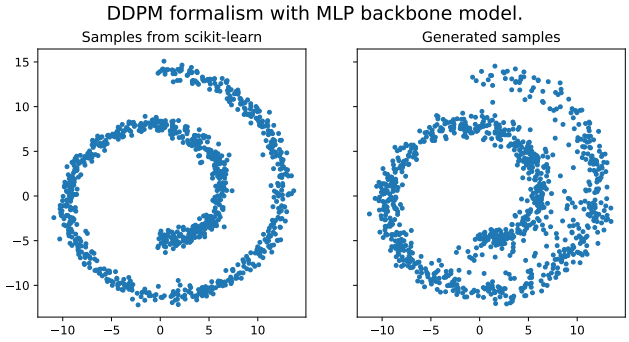

Figure 3.1: Samples generated with the DDPM formalism and a MLP backbone model, trained on the swiss-roll dataset. On the left, samples generated with scikit-learn, on the right, samples generated with the trained model.

For the DDPM framework, which was the first implemented, we stepped up the complexity by switching from a MLP to a CNN[2], this time on the *MNIST* dataset [Deng 2012]. As it lead to very satisfying results (see Figure 3.2), we could start developing larger models on audio data. As the EDM paradigm yielded similar results, we chose not to show them here.

[1] MultiLayer Perceptron
[2] Convolutional Neural Network

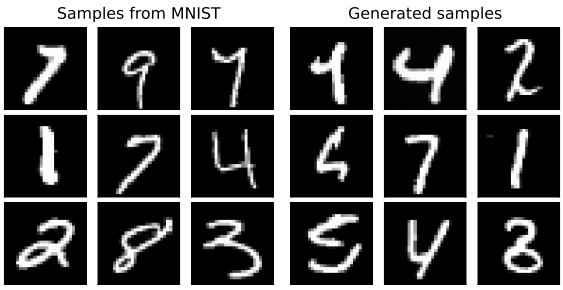

Figure 3.2: Samples generated with the DDPM formalism with a CNN backbone model trained on the MNIST dataset (right) compared to samples from the dataset (left).

The DiscDPM framework was further tested with a small U-Net (see subsection 3.4.1), which yielded very satisfying results. The successive steps of discrete diffusion carried during sampling are presented in Figure 3.3. The data it was trained on consists of 2 discrete codes (x and y axes) of 40 classes, following the *swiss-roll* pattern.

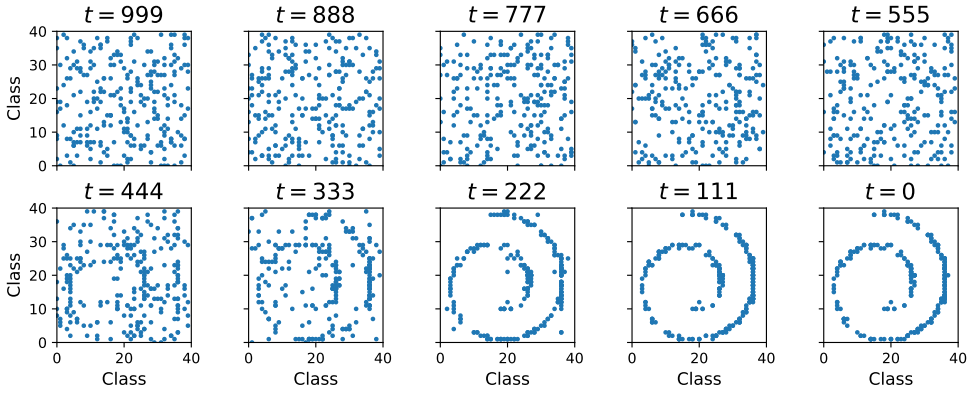

Figure 3.3: The successive diffusion steps carried by the *DiscDPM* framework, trained on 2D data over 40 classes following the *swiss-roll* pattern. The prior (initial noise) follows a uniform distribution over each dimension.

3.2 Control integration

Following the different conditioning strategies determined in section 2.4, we tested different control methods. Even though conditioning with attention layers seems to be a promising approach [Hawthorne et al. 2022], especially since it allows control inputs to be of different size than the variable it is controlling, we did not have time to implement this method nor *CondConv* layers, leaving that for future work.

We separated control into two categories: global control and local control. Global control typically takes the form of style guidance, such as textual input. Local control, in the case of images, refers to conditioning on specific areas pixel-wise (2D control input). When manipulating audio, it refers to conditioning that varies over time (1D control input).

We integrated local control using the channel concatenation approach, which proved to be efficient, while global control was integrated using the *FiLM* modulation method. However, this modulation seems to have too much impact on the generation, which led to massively degraded sampling. Our objective being to enable local control on the generation of basic elements for jingles, we focused on integrating local control rather than global control, which we eventually did not use in our experiments.

3.3 Data

To test different architectures, we needed audio data paired with conditioning data. For this, we decided to synthesize audio clips from a MIDI database, allowing us to easily create conditioning data. The MIDI database we chose is *Lakh* [*The Lakh MIDI Dataset v0.1* n.d.], and we exported the audio using the *Fluidsynth* synthesizer [*FluidSynth/fluidsynth* 2024] with the *General User Soundfont library* [*https://schristiancollins.com/generaluser.php* n.d.]. More information about the generated data is provided in subsection 3.3.3.

3.3.1 Audio format

Signal processing representations

Our initial goal was to avoid any dependency on other pre-trained or to-be-trained models. To achieve this, we used a time-frequency representation, in particular the CQT[3]. The advantage of CQT over STFT[4] is its ability to highlight harmonic relationships, which are crucial in our case of musical instrument sounds. The chosen parameters, offering the best balance between size and quality, are as follows:

- minimum frequency of $130.81\,\mathrm{Hz}$ (corresponding to the C3 MIDI note)
- 36 frequency bins per octave
- 216 bins (range of 6 octaves)
- hop length of 100 samples, with a sampling frequency $f_s = 24\,\mathrm{kHz}$, so a hop time of $4\,\mathrm{ms}$

To not rely on a downstream phase vocoder, the complex CQT data is split across 2 channels (real and imaginary parts). The model thus operates in 3 dimensions. For $8\,\mathrm{s}$ of audio, the data shape is $[C = 2, F = 216, T = 1921]$[5]. This representation may be used with the DDPM and EDM paradigms (see table 2.1).

Continuous latent state space

The memory usage of the CQT data and the computation time it required quickly became limiting factors in the development of the models. Therefore, we turned to latent space methods to compress the information [Rombach et al. 2022]. Though not being the highest-fidelity option for our specific case, the pre-trained *Encodec* model [Défossez et al. 2022] allows for efficient data compression and is a common solution for latent audio spaces.

The encoder part of Encodec outputs 128-dimensional floating point vectors, sampled at $75\,\mathrm{Hz}$ (Figure 3.4). In this state-space, one 8s-sample is 2-dimensional and of shape $[C = 128, T = 600]$, which is around 10 times lighter than the CQT format. The main drawback of using a pre-trained encoder-decoder architecture is the bias introduced by the data it was trained on. Ultimately, for the best results, a specific encodec-decoder model could be trained. This representation may be used with the DDPM and EDM paradigms (see table 2.1).

Discrete latent state space

The Encodec model also includes a vector quantization layer, which further compresses the data into discrete codes referencing a codebook (see Figure 3.4). The available bandwidths B_w and the corresponding number of discrete codes n_q are presented in table 3.1 [Défossez et al. 2022]. The codes are encoded on 1024 classes, so they take integer values from 0 to 1023.

[3] Constant Q Transform, https://en.wikipedia.org/wiki/Constant-Q_transform
[4] Short-Time Fourier Transform, https://en.wikipedia.org/wiki/Short-time_Fourier_transform
[5] *C* stands for *channels*, *F* for *frequencies*, *T* for *time*

B_w	n_q
$1.5\,\text{kbps}$	2
$3\,\text{kbps}$	4
$6\,\text{kbps}$	8
$12\,\text{kbps}$	16
$24\,\text{kbps}$	32

Table 3.1: The different bandwidths available with the Encodec model and their corresponding number of discrete codes

For our experiments, we chose a bandwidth of $12\,\text{kbps}$, which leads to 8s-samples of shape $[C = 16, T = 600]$, 8 times lighter than the continuous latent representation, 80 times lighter than the CQT representation. However, even though this reduction in size is reflected in the database, it does not translate to memory usage during training, due to the one-hot-like representation of categorical distributions (see Section 4.3). This data format is only compatible with the DiscDPM paradigm (see table 2.1).

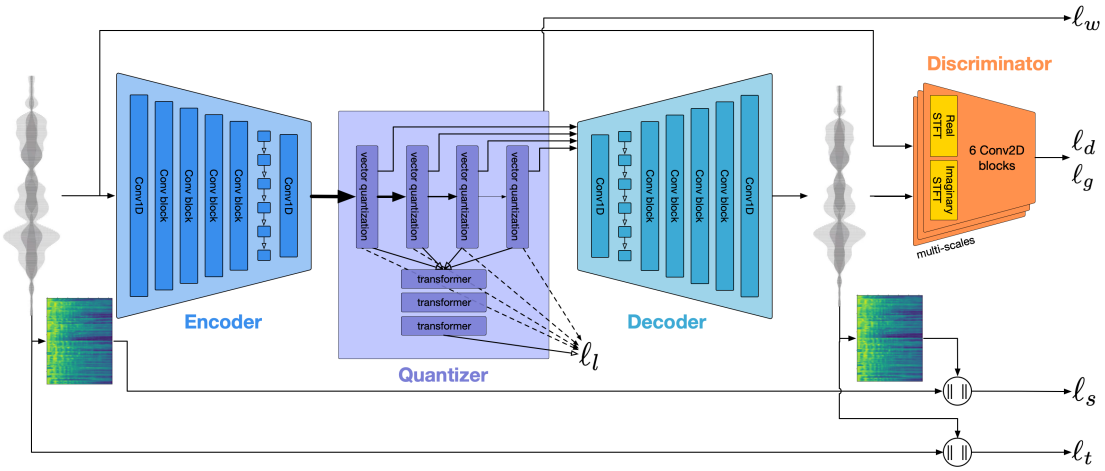

Figure 3.4: The architecture of the Encodec model. For continuous latent variables (3.3.1), the *encoder* and *decoder* parts are used, bypassing the *quantizer* bottleneck. For discrete latent variables (3.3.1), the quantized vectors are used (different compression levels are available). Credit: *High Fidelity Neural Audio Compression* [Défossez et al. 2022]

3.3.2 Control data

The local control features extracted from the MIDI information (see section 3.2) are:

- **velocity** The velocity of the current note, normalized from $[0, 127]$ to $[0, 1]$. 1D vector (scalar along the time dimension). Set to 0 when no note is playing.
- **chromas** The chroma of the current note, normalized from $[0, 11]$ to $[0, 1]$. 1D vector (scalar along the time dimension). Set to 0 when no note is playing.
- **instruments** The instrument playing the current note, value in $[0, 1]$ defining the activation of an instrument. 2D matrix, with one dimension for the different instruments (of size 3 in our case) and one time dimension. Considering our database, the instruments are either fully activated or deactivated (0 or 1), and there is only one instrument for the whole clip.

All these controls may be set to -1 for no conditioning. Three examples of control data are presented in Figure 3.5. It should be noticed that these arbitrary controls have numerous limitations. For instance, the chroma representation cannot include octave information, which creates ambiguity regarding the actual melody played.

We did not use global control even for conditioning on timbre because, as explained in section 2.4, it had too strong an effect on the generation. Additionally, this approach allows for the timbre to vary over time.

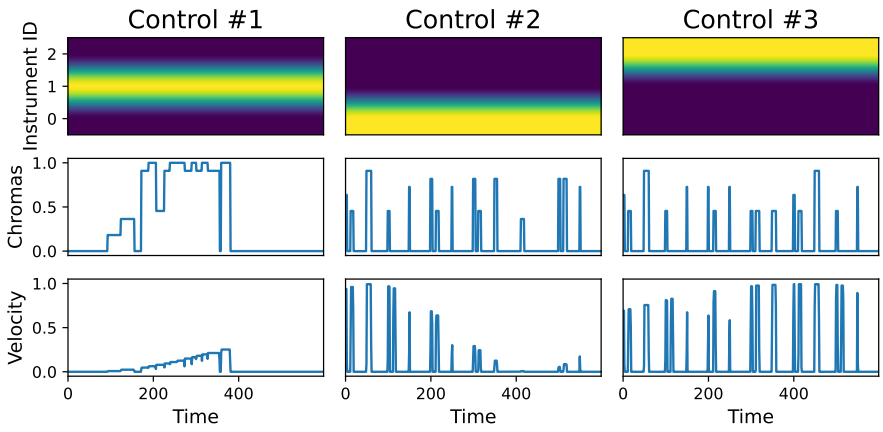

Figure 3.5: Examples of control data used for training. From top to bottom, the instrument matrix, where the ID of the instrument playing is activated while the others are set to 0, the chromas vector, indicating the global variation in pitch, and the velocity vector, indicating the intensity of the note played, and if any note is played at all.

3.3.3 Databases

Throughout our research, numerous datasets were exported. The latest versions, which we use for this report, consist of 8-second mono audio clips. Each clip contains a monophonic melody (only one note played at a time) performed by one of three instruments: piano, violin, or trumpet. The notes range from C3 to C5 (notes outside this range in the MIDI database were transposed), and their velocity is modulated by a sine wave with random frequency and phase to add diversity (with the frequency varying between $\frac{1}{8\times T}$ and $\frac{8}{T}$ with T the duration of the audio clip).

Each of these clips is paired with the corresponding control data (see subsection 3.3.2), sampled at the same rate as the audio format ($24\,\mathrm{kHz}$ for the CQT format, $75\,\mathrm{Hz}$ for the Encodec latent codes). The benefits of both a latent representation that compresses the temporal dimension and the integration of control mechanisms allowing for a different control data size compared to the audio data (such as attention layers) are clear: it is unnecessary to control generation at a precision of $24\,\mathrm{kHz}$, which consumes a significant amount of memory, especially with the CQT representation, which is already by far the most resource-intensive.

In our code, this database configuration is called *PVT*[6], and comes in 4 data formats:

- *PVT-audio*, with raw audio clips, useful for the computation of quantitative metrics to evaluate the quality of the generation (see section 4.2).
- *PVT-cqt*, with audio data exported in the CQT representation (3.3.1)
- *PVT-encodec*, with audio data exported in the continuous codes Encodec latent representation (3.3.1)

[6] Piano-Violin-Trumpet

- *PVT-disc*, with audio data exported in the discrete codes Encodec latent representation (3.3.1)

All examples are 8s-long. The raw audio dataset contains $50\,\mathrm{k}$ examples. The other datasets contain two splits: the *train-dataset* of about $100\,\mathrm{k}$ examples and the *valid-dataset* of about $20\,\mathrm{k}$ examples (ratio of 0.2).

Exporting representations from audio data on-the-fly during training was highly suboptimal for GPU utilization. Therefore, all datasets were exported offline in the compressed *npz* format. Even so, data decompression prevented the GPUs from being used to their full computational capacity. To address this, we switched to TensorFlow's *tfrecord* format for serialization, which allows for saving data in binary and thus enables very fast reading. The only adjustment required was to include a `shapes.txt` file to reshape the matrices during data loading.

3.4 Backbone model

We refer to the trainable model within the diffusion process as the *backbone model*. Our whole backbone architecture is presented in Figure 3.7.

3.4.1 U-Net

We decided to use a U-Net architecture [Ronneberger, Fischer, and Brox 2015] as our backbone model. It appears well-suited for capturing latent representations at different scales through down- and up-sampling, and it is widely used in the literature [Rombach et al. 2022; Ho, Jain, and Abbeel 2020; Wu et al. 2023].

This U-Net can be composed of either 1D or 2D convolutions depending on the data format (2D for CQT, 1D for continuous and discrete codes). The non-linearity function used is *SiLU*, and the normalization is *GroupNorm* with a single group. At each downsampling step, the number of channels is multiplied by a factor, which is set in the configuration file. We tested our U-Net implementation with the DDPM formalism as a mock test-case with the swiss-roll dataset to check its proper functioning (Figure 3.6).

Following [Ronneberger, Fischer, and Brox 2015], the U-Net has a last *OutConv* convolutional layer, allowing to spread the data over C_{out} channels if needed (the original model was developed for pixel-wise classification, so this layer mapped the representation to the different classes). In the case of continuous data $C_{\text{out}} = C_{\text{data}}$, but in the case of discrete codes, as we need to determine mass probability functions, the models needs to output the probability over 1024 classes for each of the 16 input channels (more on this in the subsection 3.4.3).

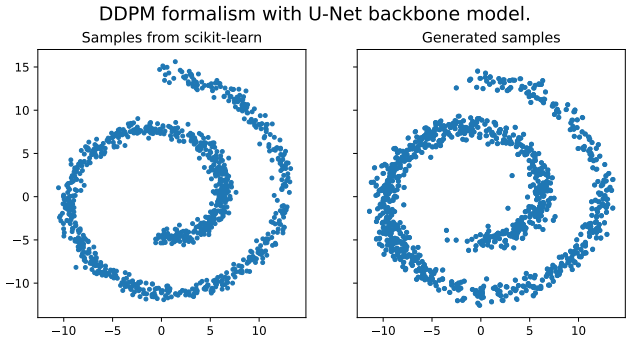

Figure 3.6: Samples generated with the DDPM formalism and a U-Net backbone model (with 1D convolutions), trained on the swiss-roll dataset. On the left, samples generated with scikit-learn, on the right, samples generated with the trained model.

3.4.2 Local and global control

The local control branch follows an architecture similar to the first half of the U-Net. The temporal dimension is halved at each layer to allow for concatenation with the latent representation, but the number of channels differs. Specifically, we set a fixed number of control channels that remains constant at each layer, set to 128 channels in our case (configurable).

Even though we have not included global control in our experiments, it was implemented in our backbone architecture. It takes the form of linear modulation. The branch is composed of MLP blocks with a Sigmoid function as the output, to serve as an activation mechanism for the different channels. The output of each MLP is the input of the next one. The complete conditional denoiser backbone model is schematized in Figure 3.7.

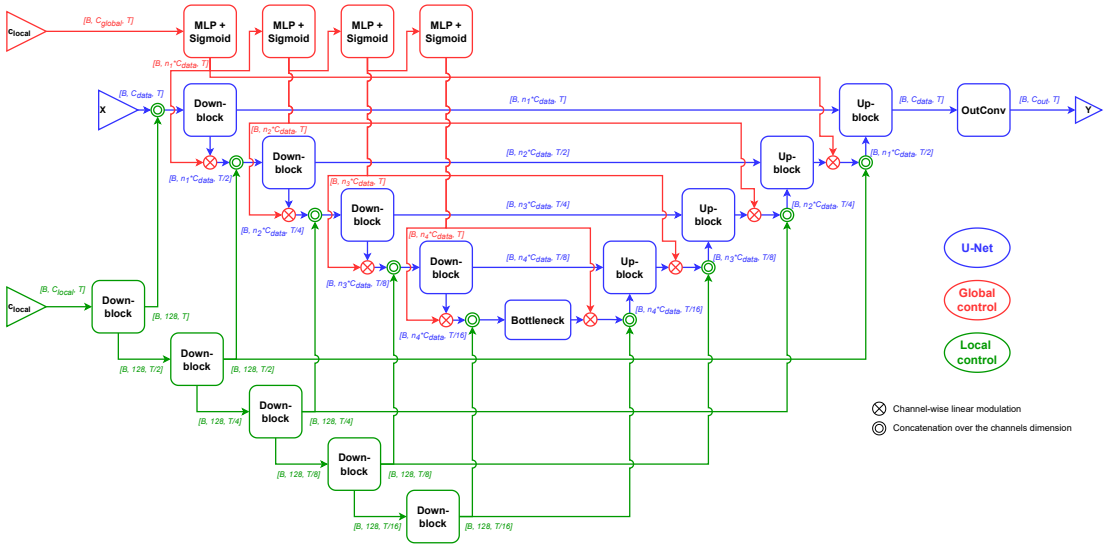

Figure 3.7: Summary of our conditional denoiser architecture, composed of one U-Net, a local control branch and a global control branch. The shapes indicated correspond to 1D data (codes in our case). For 2D data (CQT), a frequency dimension F is added and resampled with the time dimension T. The down- and up-blocks are as described in [Ronneberger, Fischer, and Brox 2015], i.e. 2 convolution layers with a time-dependent modulation in-between.

3.4.3 U-NetExpand

For the special case of discrete data (the *DiscDPM* framework), the architecture presented in Figure 3.7 seemed significantly limited. Indeed, the last upblock outputs C_{data} channels ($C_{\text{data}} = 16$ in the case of discrete codes), and the convolution layer *OutConv* is used to create $C_{\text{data}} \times n_{\text{classes}} = 16 \times 1024 = 16384$ channels. This extreme last layer was considerably limiting the model's performance.

To address this issue, we developed a slightly modified version of the U-Net, where the channel multiplication factors in the upsampling blocks are not the same as those during the downsampling phase. This allows us to increase the channel dimension throughout the U-Net, avoiding a situation where the final layer multiplies the number of channels by 1024. In our code, this architecture is called *UNetExpand*.

Chapter 4

Results

4.1 Training

The training was conducted in epochs of 10000 steps, with a learning rate of 10^{-4}, using a scheduler. We set a maximum of 50 epochs, so $500\,\mathrm{k}$ steps. No model reached convergence within this time, despite the considerable computation time. For each data sample, each control feature had a 30% change of being set to a uniform -1 tensor, which stands for unconditional generation.

Quite quickly in the development process, the memory consumption became prohibitive, forcing us to shard the data across 4 GPUs to maintain a reasonable batch size (with CQT data, the maximum batch size on a single GPU is between 4 and 8). Using the GPU servers at Ircam was limiting, so we utilized the Jean-Zay computing center, which allowed us to run multiple training sessions in parallel. Trainings on Jean-Zay servers are launched with `slurm` scripts, allowing to launch scripts for up to $100\,\mathrm{h}$ (more info on IDRIS' website[1]). The wait time to launch a script depends on the traffic and the migration of Jean-Zay's storage servers during this summer delayed many trainings, but using this computing center was still a huge time saver overall.

4.2 Metrics

To compare the different frameworks, we needed a metric, and we chose the FAD[2] [Kilgour et al. 2019] as it is commonly used in the literature [Zhu et al. 2023; Wu et al. 2023]. An overview of the FAD calculation is provided in Figure 4.1. In the original work this metric was developed for, the goal was to evaluate the performance of an audio denoising model, with the evaluation set being the *enhanced eval. set*. In our case, it is a set of samples generated by our model. The *large database of clean music* corresponds to our *PVT-audio* baseline database (see Subsection 3.3.3), exported with the same parameters as the training dataset. The pre-trained *VGGish* classification model then exports embeddings for each set, and the FAD is calculated from the estimation of multivariate Gaussians for each set of embeddings.

[1] http://www.idris.fr/jean-zay/gpu/jean-zay-gpu-exec_partition_slurm.html
[2] Fréchet Audio Distance

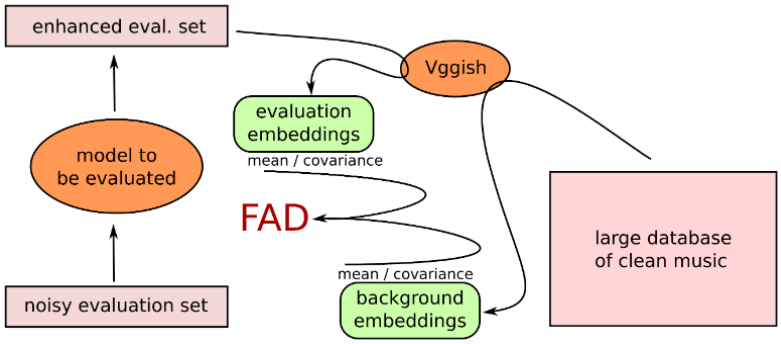

Figure 4.1: FAD computation overview. The *enhanced eval. set* is in our case the set of generated samples, and the *large database of clean music* is the *PVT-audio* database (see subsection 3.3.3). Credit: *Fréchet Audio Distance: A Metric for Evaluating Music Enhancement Algorithms* [Kilgour et al. 2019]

In our case, the baseline set of clean audio consists of $50\,\mathrm{k}$ examples of $4\,\mathrm{s}$ each, and the evaluation set contains 2000 generated examples, also $4\,\mathrm{s}$ each. We were forced to choose these parameters due to inference time and memory usage constraints in the frameworks when using CQT and discrete code data. For consistency, we calculated the FAD for the continuous code frameworks using the same parameters. To obtain more precise values, a much larger evaluation set would be necessary.

For a more comprehensive comparison of the different frameworks, we include the training time (which depends notably on memory usage), the number of trainable parameters, and the inference time for a batch when creating the evaluation set for FAD computation.

4.3 Quantitative

The quantitative results were significantly limited by the time available to us. The CQT data format quickly proved to be too memory-intensive to be usable, so we quickly switched to latent representations. The CQT data format is still included ni the comparison.

The discrete code format required much more memory than we initially anticipated during model development. Although the data is encoded on 16 integers, sampled at $75\,\mathrm{Hz}$, the need to generate a one-hot-like tensor for the probability mass function causes memory usage to increase exponentially with the number of classes. Since Encodec codes are encoded over 1024 classes, the output of the backbone model is 1024 times larger than the input. This limitation impacts the number of parameters in the discrete code model, as well as the training and sampling times. Unfortunately, the EDM paradigm cannot be easily adapted to discrete codes because it relies on solving SDEs, which inherently require continuous data.

The results are presented in Table 4.1. Unfortunately, the lack of time prevented us from properly tuning the hyperparameters of the DDPM with continuous codes framework (we were already mainly using the EDM formalism when we implemented the latent representations). The number of learnable parameters differs from one framework to the other because of the memory usage it yields. The CQT and discrete codes data formats faced memory limitations, so the configuration chosen is the largest possible to train on $16\,\mathrm{GB}$ GPUs. This choice introduced a huge bias in the FAD scores, since the lighter models were closer to convergence after the same number of training steps. We think that is why the EDM-CQT framework scores better FAD-wise even though during development, we found EDM with continous codes to yield better results. We will train lighter models and compute their FAD before the defense in order to present a more relevant comparison. Yet, the training and sampling times being also much shorter, we focused most of our hyperparameter tuning on this framework, leading to the results presented in Section 4.4.

The discrete code format does not seem very suitable for this task and shows a poor FAD score. Although the data itself is lighter, the need to generate from one-hot-like tensors makes it more challenging to model.

Framework	DDPM	EDM	DDPM	EDM	DiscDPM
Data format	CQT	CQT	Float. codes	Float. codes	Disc. codes
Learnable parameters	$56.9\,\mathrm{M}$	$56.9\,\mathrm{M}$	$225.8\,\mathrm{M}$	$225.8\,\mathrm{M}$	$87.9\,\mathrm{M}$
Batch size / GPU	4	4	16	16	16
Training time ($500\,\mathrm{k}$ steps)	$75.7\,\mathrm{h}$	$80.2\,\mathrm{h}$	$36.3\,\mathrm{h}$	$37.3\,\mathrm{h}$	$77.2\,\mathrm{h}$
Inference time ($8 \times 4\,\mathrm{s}$)	$1216.2\,\mathrm{s}$	$45.0\,\mathrm{s}$	$39.6\,\mathrm{s}$	$1.6\,\mathrm{s}$	$112.8\,\mathrm{s}$
FAD	7.23	4.63	73.84	7.48	48.55

Table 4.1: Quantitative metrics for the different diffusion frameworks, all with a U-Net backbone model, trained for $500\,\mathrm{k}$. Due to memory limitations and high computational times, the evaluation set generated for each framework was made up of 2000 examples of $4\,\mathrm{s}$ each. The inference time was measured for batches of size 8 (largest batch size possible for inference with the CQT format on Ircam's $12\,\mathrm{GB}$ GPUs).

Given the significant limitations on quantitative results due to constraints in certain frameworks, we trained an EDM model on continuous codes with more trainable parameters and for $1\,\mathrm{M}$ steps. We refer to this model as our *best model*. Its metrics are presented in Table 4.2. The efficiency of the EDM framework on continuous codes enabled us to calculate the FAD using an evaluation set of $20\,\mathrm{k}$ examples of $8\,\mathrm{s}$ each, providing a more accurate result.

Framework	EDM
Data format	Float. codes
Learnable parameters	$231.6\,\mathrm{M}$
Batch size / GPU	32
Training time ($1\,\mathrm{M}$ steps)	$118.4\,\mathrm{h}$
Inference time ($64 \times 8\,\mathrm{s}$)	$12.7\,\mathrm{s}$
FAD	4.06

Table 4.2: Quantitative metrics for our *best model*, trained for $1\,\mathrm{M}$ steps with a batch size of 32 per GPU. The inference time is provided for reference and should not be compared to the inference time in Table 4.1, as the batch size and sampling duration are different. The evaluation set was made up of $20\,\mathrm{k}$ generated samples.

4.4 Qualitative

Since we did not implement a metric to monitor the control, we leave this to qualitative appreciation. A comparison between audio samples generated by our models and examples from the database will be made available on https://anasynth.demos.ircam.fr/jingles. Figure 4.2 shows mel-spectrograms of samples generated by our *best model* and their associated hand-made control features. It appears clear that the control feature have a great impact on the generation.

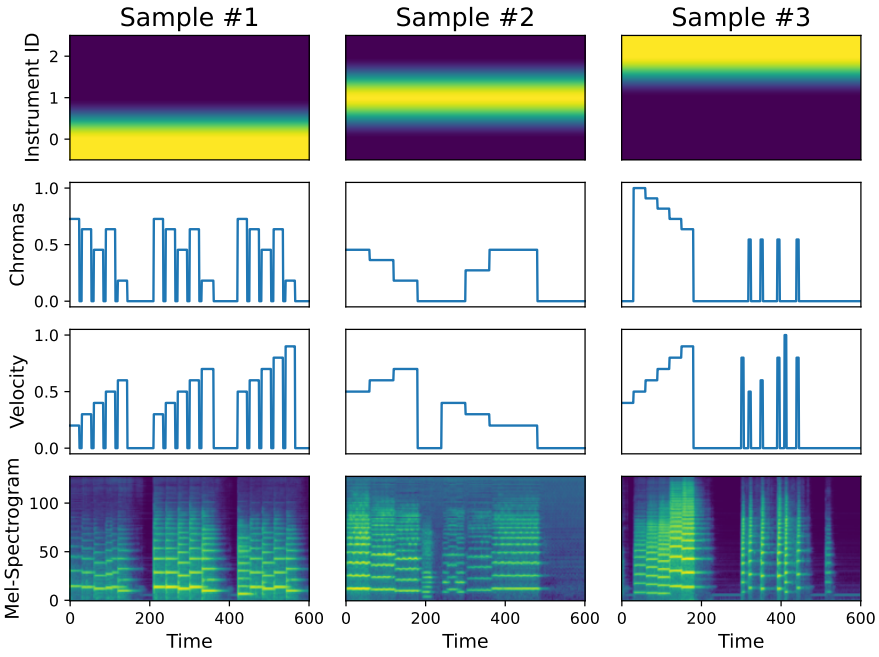

Figure 4.2: Samples generated by our *best model*, conditionned by a hand-made control. This model operating at $75\,\mathrm{Hz}$, the sample duration is $8\,\mathrm{s}$.

Partially unconditional generation (Figure 4.3) highlights the limitations of our control features. Specifically, in addition to the octave ambiguity in chromas, we observed a strong interdependence between the chroma and velocity features. This likely stems from the fact that the absence of a note is represented by both a velocity of 0 and a chroma of 0, which corresponds to the C note. Therefore, conditioning on the 0 chroma has a distinct status compared to other chromas.

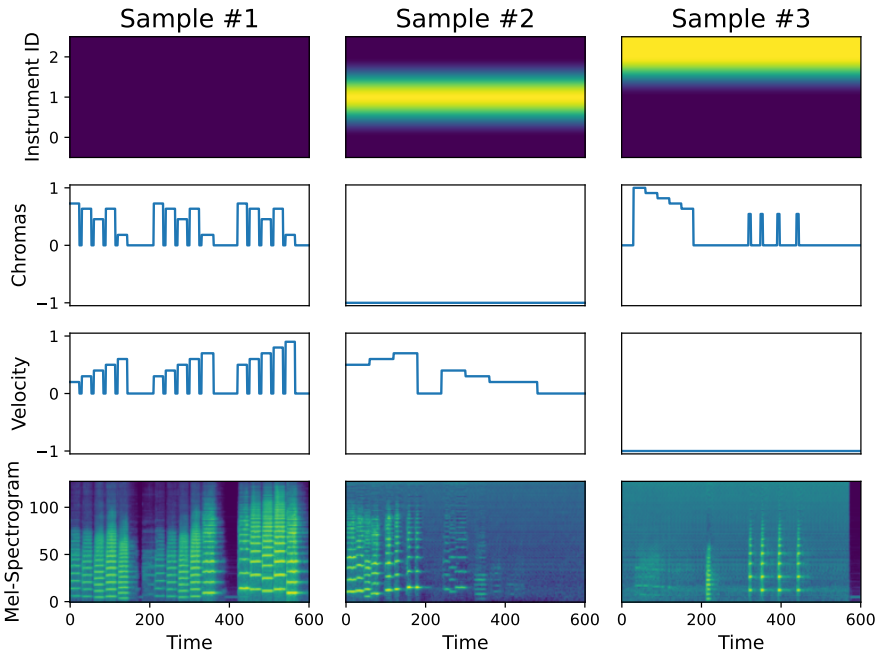

Figure 4.3: $8\,\mathrm{s}$-long samples generated by our *best model*, partially unconditional, one control feature being set to -1. This model operating at $75\,\mathrm{Hz}$, the sample duration is $8\,\mathrm{s}$.

4.5 Future work

EDM has been identified as the most effective paradigm, yielding the highest quality results, but we have only minimally explored the sampling parameters offered by this framework. Specifically, we only performed deterministic sampling, whereas [Karras et al. 2022] demonstrated that adding stochasticity can result in more realistic generation. Here are the different sampling parameters from [Karras et al. 2022] and the value we gave them:

- $n_{\text{steps}} = 18$, the number of denoising steps
- $\sigma_{\text{min}} = 0.002$, the minimum noise level
- $\sigma_{\text{max}} = 80$, the maximum noise level
- $\rho = 7$, parameter controlling the curvature of the gradient lines
- $S_{\text{churn}} = 0$, parameter controlling the overall amount of stochasticity
- $S_{\text{min}} = 0$, minimum noise level at which stochasticity is added
- $S_{\text{max}} =$ inf, maximum noise level at which stochasticity is added
- $S_{\text{noise}} = 1$, gain for the noise added for stochasticity

Samples generated with different S_{churn} parameters should be compared to see the effect of stochasticity.

In the pursuit of generating basic elements for audio jingles, it would be necessary to gather more suitable and higher quality data, as well as to define audio descriptors that are valuable to sound designers. For example, [Peeters et al. 2011] presents numerous descriptors that could be useful for this task.

Developing a dedicated encoder-decoder specifically for this task would also be worthwhile. As we have seen, operating in a latent space is essential for building sufficiently large models, so this latent space must be well-suited to the data. Tailoring the encoder-decoder to capture the specific characteristics of audio jingles could lead to better performance, enabling the model to generate high-quality audio elements.

Conclusion

Over the course of the 6-month internship, we explored various diffusion-based generative frameworks, considering both diffusion paradigms (DDPM, EDM, DiscDPM) and data format (CQT, continuous codes, discrete codes). Unfortunately, we did not have the time to fully develop each framework to a satisfactory level, but the limitations encountered during development provide valuable lessons for future work.

One key takeaway is that diffusion on time-frequency representations like CQT proved to be computationally infeasible due to excessive memory requirements. Regarding diffusion paradigms, EDM offers significantly more efficient sampling than DDPM on continuous state spaces, making it a preferable option. While diffusion on discrete spaces is attractive for many domains, in the case of vector quantized codes, the large number of classes results in significantly higher memory usage compared to continuous codes. When given the choice between a continuous and discrete space, the continuous space generally appears more suitable.

In our experiments, we found that channel concatenation yielded better results for control integration compared to affine modulation. While attention-based control mechanisms have shown promise in other generative models, we did not have the opportunity to thoroughly explore this approach in our study. Future work could investigate the effectiveness of attention layers for more versatile control over audio generation, in particular the ability to sample the control at a lower frequency than the audio data.

Given the importance of operating in a latent space for diffusion, it highlights the potential benefits of developing and training our own autoencoder. This approach would eliminate the need to rely on a pre-trained network like Encodec. By specializing the model on audio jingle elements, we could achieve greater data compression, optimized specifically for the characteristics of jingles, leading to more efficient and targeted generation.

Bibliography

Austin, Jacob et al. (Feb. 2023). *Structured Denoising Diffusion Models in Discrete State-Spaces*. arXiv:2107.03006 [cs]. DOI: 10.48550/arXiv.2107.03006. URL: http://arxiv.org/abs/2107.03006.

Broek, Korneel van den (Jan. 2021). *MP3net: coherent, minute-long music generation from raw audio with a simple convolutional GAN*. arXiv:2101.04785 [cs, eess]. DOI: 10.48550/arXiv.2101.04785. URL: http://arxiv.org/abs/2101.04785.

Chen, Nanxin et al. (Oct. 2020). *WaveGrad: Estimating Gradients for Waveform Generation*. arXiv:2009.00713 [cs, eess, stat]. DOI: 10.48550/arXiv.2009.00713. URL: http://arxiv.org/abs/2009.00713.

Défossez, Alexandre et al. (Oct. 2022). *High Fidelity Neural Audio Compression*. en. arXiv:2210.13438 [cs, eess, stat]. URL: http://arxiv.org/abs/2210.13438.

Deng, Li (2012). "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6, pp. 141–142.

FluidSynth/fluidsynth (Mar. 2024). original-date: 2017-06-24T15:09:55Z. URL: https://github.com/FluidSynth/fluidsynth.

Hawthorne, Curtis et al. (Dec. 2022). *Multi-instrument Music Synthesis with Spectrogram Diffusion*. arXiv:2206.05408 [cs, eess]. DOI: 10.48550/arXiv.2206.05408. URL: http://arxiv.org/abs/2206.05408.

Ho, Jonathan, Ajay Jain, and Pieter Abbeel (Dec. 2020). *Denoising Diffusion Probabilistic Models*. arXiv:2006.11239 [cs, stat] version: 2. DOI: 10.48550/arXiv.2006.11239. URL: http://arxiv.org/abs/2006.11239.

Ho, Jonathan and Tim Salimans (July 2022). *Classifier-Free Diffusion Guidance*. arXiv:2207.12598 [cs]. DOI: 10.48550/arXiv.2207.12598. URL: http://arxiv.org/abs/2207.12598.

Hoogeboom, Emiel et al. (Oct. 2021). *Argmax Flows and Multinomial Diffusion: Learning Categorical Distributions*. arXiv:2102.05379 [cs, stat]. DOI: 10.48550/arXiv.2102.05379. URL: http://arxiv.org/abs/2102.05379.

https://schristiancollins.com/generaluser.php (n.d.). URL: https://schristiancollins.com/generaluser.php.

Karras, Tero et al. (Oct. 2022). *Elucidating the Design Space of Diffusion-Based Generative Models*. arXiv:2206.00364 [cs, stat]. DOI: 10.48550/arXiv.2206.00364. URL: http://arxiv.org/abs/2206.00364.

Kilgour, Kevin et al. (Jan. 2019). *Fr\'echet Audio Distance: A Metric for Evaluating Music Enhancement Algorithms*. arXiv:1812.08466 [cs, eess]. DOI: 10.48550/arXiv.1812.08466. URL: http://arxiv.org/abs/1812.08466.

Kim, Ji-Hoon et al. (June 2021). *Fre-GAN: Adversarial Frequency-consistent Audio Synthesis*. arXiv:2106.02297 [cs, eess]. DOI: 10.48550/arXiv.2106.02297. URL: http://arxiv.org/abs/2106.02297.

Kong, Zhifeng et al. (Mar. 2021). *DiffWave: A Versatile Diffusion Model for Audio Synthesis*. arXiv:2009.09761 [cs, eess, stat]. DOI: 10.48550/arXiv.2009.09761. URL: http://arxiv.org/abs/2009.09761.

Liao, Shijia, Shiyi Lan, and Arun George Zachariah (Jan. 2024). *EVA-GAN: Enhanced Various Audio Generation via Scalable Generative Adversarial Networks*. arXiv:2402.00892 [cs, eess]. DOI: 10.48550/arXiv.2402.00892. URL: http://arxiv.org/abs/2402.00892.

Nichol, Alex and Prafulla Dhariwal (Feb. 2021). *Improved Denoising Diffusion Probabilistic Models*. arXiv:2102.09672 [cs, stat]. DOI: 10.48550/arXiv.2102.09672. URL: http://arxiv.org/abs/2102.09672.

Oord, Aaron van den et al. (Sept. 2016). *WaveNet: A Generative Model for Raw Audio*. arXiv:1609.03499 [cs]. DOI: 10.48550/arXiv.1609.03499. URL: http://arxiv.org/abs/1609.03499.

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Peeters, Geoffroy et al. (Nov. 2011). "The Timbre Toolbox: Extracting audio descriptors from musical signals". In: *The Journal of the Acoustical Society of America* 130.5, pp. 2902–2916. ISSN: 0001-4966. DOI: 10.1121/1.3642604. URL: https://doi.org/10.1121/1.3642604.

Perez, Ethan et al. (Dec. 2017). *FiLM: Visual Reasoning with a General Conditioning Layer*. arXiv:1709.07871 [cs, stat]. DOI: 10.48550/arXiv.1709.07871. URL: http://arxiv.org/abs/1709.07871.

Rombach, Robin et al. (Apr. 2022). *High-Resolution Image Synthesis with Latent Diffusion Models*. arXiv:2112.10752 [cs]. DOI: 10.48550/arXiv.2112.10752. URL: http://arxiv.org/abs/2112.10752.

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (May 2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation*. arXiv:1505.04597 [cs]. DOI: 10.48550/arXiv.1505.04597. URL: http://arxiv.org/abs/1505.04597.

Saharia, Chitwan et al. (May 2022). *Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding*. arXiv:2205.11487 [cs]. DOI: 10.48550/arXiv.2205.11487. URL: http://arxiv.org/abs/2205.11487.

Sohl-Dickstein, Jascha et al. (Nov. 2015). *Deep Unsupervised Learning using Nonequilibrium Thermodynamics*. arXiv:1503.03585 [cond-mat, q-bio, stat] version: 8. DOI: 10.48550/arXiv.1503.03585. URL: http://arxiv.org/abs/1503.03585.

Song, Yang et al. (Feb. 2021). *Score-Based Generative Modeling through Stochastic Differential Equations*. arXiv:2011.13456 [cs, stat]. DOI: 10.48550/arXiv.2011.13456. URL: http://arxiv.org/abs/2011.13456.

The Lakh MIDI Dataset v0.1 (n.d.). URL: https://colinraffel.com/projects/lmd/.

Vaswani, Ashish et al. (Aug. 2023). *Attention Is All You Need*. arXiv:1706.03762 [cs]. DOI: 10.48550/arXiv.1706.03762. URL: http://arxiv.org/abs/1706.03762.

Wu, Shih-Lun et al. (Nov. 2023). *Music ControlNet: Multiple Time-varying Controls for Music Generation*. arXiv:2311.07069 [cs, eess] version: 1. DOI: 10.48550/arXiv.2311.07069. URL: http://arxiv.org/abs/2311.07069.

Xiao, Zhisheng, Karsten Kreis, and Arash Vahdat (Apr. 2022). *Tackling the Generative Learning Trilemma with Denoising Diffusion GANs*. arXiv:2112.07804 [cs, stat]. DOI: 10.48550/arXiv.2112.07804. URL: http://arxiv.org/abs/2112.07804 (visited on 08/20/2024).

Yang, Brandon et al. (Sept. 2020). *CondConv: Conditionally Parameterized Convolutions for Efficient Inference*. arXiv:1904.04971 [cs]. DOI: 10.48550/arXiv.1904.04971. URL: http://arxiv.org/abs/1904.04971.

Yang, Dongchao et al. (Apr. 2023). *Diffsound: Discrete Diffusion Model for Text-to-sound Generation*. arXiv:2207.09983 [cs, eess]. DOI: 10.48550/arXiv.2207.09983. URL: http://arxiv.org/abs/2207.09983.

Zhang, Lvmin, Anyi Rao, and Maneesh Agrawala (Nov. 2023). *Adding Conditional Control to Text-to-Image Diffusion Models*. arXiv:2302.05543 [cs] version: 3. DOI: 10.48550/arXiv.2302.05543. URL: http://arxiv.org/abs/2302.05543.

Zhao, Shihao et al. (Oct. 2023). *Uni-ControlNet: All-in-One Control to Text-to-Image Diffusion Models*. arXiv:2305.16322 [cs] version: 3. DOI: 10.48550/arXiv.2305.16322. URL: http://arxiv.org/abs/2305.16322.

Zhu, Ge et al. (Nov. 2023). *EDMSound: Spectrogram Based Diffusion Models for Efficient and High-Quality Audio Synthesis*. arXiv:2311.08667 [cs, eess]. DOI: 10.48550/arXiv.2311.08667. URL: http://arxiv.org/abs/2311.08667.

List of Figures

List of Tables