Marin Gudahl Tufte

Vector Quantized Time Series Generation using Diffusion Models

Hovedoppgave i Fysikk og Matematikk
Veileder: Erlend Aune
Medveileder: Daesoo Lee
Juli 2023

Hovedoppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for matematiske fag

Marin Gudahl Tufte

Vector Quantized Time Series Generation using Diffusion Models

Hovedoppgave i Fysikk og Matematikk
Veileder: Erlend Aune
Medveileder: Daesoo Lee
Juli 2023

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for matematiske fag

NTNU
Norwegian University of
Science and Technology

Department of Mathematical Sciences

TMA4900 Industrial Mathematics, Master's Thesis

Vector Quantized Time Series Generation using Diffusion Models

Authors:
Tufte, Martin Gudahl
Aune, Erlend (supervisor)
Lee, Daesoo (co-supervisor)

Spring, 2023

Abstract

Time series generation (TSG) is a type of generative modelling focusing on learning the distribution of time series. A novel approach to TSG, called TimeVQVDM, is proposed by combining vector quantization with diffusion models. It is motivated by recent success in image generation, where improvement gains has been seen from switching to diffusion models. The proposed model uses vector quantized variational autoencoders (VQ-VAEs) for finding efficient latent representations of the time series, split into low and high frequencies. The latent distributions are then modelled using diffusion models. Sampling from TimeVQVDM is performed by sampling the low frequencies first and then sampling the high frequencies afterwards. This allows the model to capture the overall structure of time series, as well as fine details. The proposed model is evaluated on the UCR time series archive by reporting FID, IS and CAS metrics. For unconditional sampling, TimeVQVDM achieves scores comparable with state-of-the-art alternatives, showing that diffusion models are capable of improvement gains also for the task of generating time series.

Keywords: Vector Quantization, Diffusion Models, Time Series Generation

Sammendrag

Tidsrekkegenerasjon (TSG) er en type generativ modellering som fokuserer på å lære fordelingen av tidsrekker. En ny tilnærming til TSG, kalt TimeVQVDM, er foreslått for å kombinere vektorkvantisering med diffusjonsmodeller. Modellen er motivert av nyere suksess innen bildegenerering, hvor man har sett forbedringsgevinster ved å bytte til diffusjonsmodeller. Den foreslåtte modellen bruker vektorkvantiserte variasjonsautokodere (VQ-VAEs) for å finne effektive latente representasjoner av tidsrekker, delt inn i lave og høye frekvenser. De latente fordelingene blir deretter modellert ved hjelp av diffusjonsmodeller. Generering fra TimeVQVDM utføres ved å generere de lave frekvensene først og deretter generere de høye frekvensene etterpå. Dette gjør at modellen kan fange opp den generelle strukturen til tidsrekker, samt fine detaljer. Den foreslåtte modellen er evaluert på UCR-tidsrekkearkivet ved å rapportere FID-, IS- og CAS-beregninger. For ubetinget generasjon oppnår TimeVQVDM resultateter som kan sammenlignes med toppmoderne alternativer, og viser at diffusjonsmodeller er i stand til å gi forbedringer for tidsrekkegenerasjon.

Nøkkelord: Vektorkvantifisering, Diffusjonsmodeller, Tidsrekkegenerasjon

Preface

I would like to thank my supervisor Erlend Aune for being a creative, interesting and open-minded supervisor for helping me with this thesis. Furthermore, a huge thanks is given to my co-supervisor Daesoo Lee, which has been resource-full with helping me with coding-related problems and always being available to answer questions. I enjoyed our discussions and explorations of new research papers and ideas. I would also thank the Machine Learning for Irregular Time Series (ML4ITS) research team, which provided thoughtful conversations during workshops. Finally, I would like to thank my friends and family for support and motivation during the writing of this thesis.

This master thesis is submitted as *TMA4900 Industrial Mathematics, Master Thesis* in the spring of 2023 in the field of statistical machine learning.

Table of Contents

List of Figures

List of Tables

Abbreiviations

List of important abbreviations in alphabetic order:

- **CAS** classification accuracy score
- **ELBO** evidence lower bound
- **FCN** fully convolutional network
- **FID** Frechét inception distance
- **HF** high frequency
- **IS** inception score
- **LF** low frequency
- **SNR** signal-to-noise ratio
- **STFT** short-time Fourier transform
- **TRTR** train on real; test on real
- **TSG** time series generation
- **TSTR** train on synthetic; test on real
- **VAE** variational autoncoder
- **VDM** variational diffusion model
- **VQ** vector quantization

1 Introduction

Time series are everywhere. A time series is a sequence of data points, usually taken at equally spaced times. They are found in numerous fields, including quantitative finance, weather forecasting, control engineering, astronomy, and any applied science and engineering which involves temporal measurements. Examples of time series are temperature readings, yearly counts of populations, and the daily closing value of a stock. The unique characteristic of time series lies in its temporal correlation, where the value at a particular time step is influenced by values at previous time steps. This autocorrelation property makes analysis of time series distinct from other forms of data analysis.

This thesis focuses on *time series generation* (TSG). It is a branch of generative modelling in AI that focuses on creating synthetic time series that closely resembles some observed data. This data collection can be viewed as arising from a probability distribution. The goal of TSG is to learn this distribution, which enables us to be able to synthetically generate new time series. In general time series have too complex temporal structures too model directly, thereby requiring highly flexible probability distributions. For these tasks, machine learning offer promising methods. Generative modelling have already been successful in several domains, including photorealistic image generation with text-to-image models (Ramesh et al., 2022), (Rombach et al., 2021), (Oppenlaender, 2022) and natural language processing with chat bots (Liu et al., 2023) showing deep understanding and knowledge of languages. Since time series share some structural similarity with images and natural language, many of the proposed methods in generative AI for these domains can be used for time series. However research into generative models for time series have been less explored than their counterparts. This thesis combines ideas from generative AI for images combined with recent advances in the time series literature to tackle the TSG problem.

1.1 Why is it important to synthesize time series?

Overcome insufficient data. In many areas, analysing time series data could be hard due to privacy regulations or difficulty with acquiring enough training data. For instance, consider recordings of vital systems of patients in the medical sector. This data might be illegal to handle directly. However, it could be possible to learn the underlying data distribution and generate synthetic time series with similar statistical properties while preserving individual privacy. Another example is when data collection is expensive or time-consuming. Having computational ways to synthesise new data could overcome these limitations.

Creating robust algorithms. Synthetic time series could also aid other areas of time series analysis. In fields such as quantitative finance and meteorology, the primary goal is forecasting. An advantage of using machine learning for TSG lies in their ability to produce realistic-looking samples. Classical time series models, such as ARIMA (Box et al., 1974) and GARCH (Engle, 1982), often only provide expected forecasts with accompanied confidence intervals, without explicit horizons. By leveraging the stochasticity in TSG, more realistic time series can enhance quantifying uncertainties and the reliability of the forecasts. In signal processing and communication engineering, the goal is usually signal detection. Synthetic data could help in data imputation, enabling the replacement of missing values in the signals. Additionally, synthetic data can help creating more robust algorithms for data mining tasks such as time series clustering, anomaly detection and classification by including synthetic data in the training procedure.

Synthetic audio. In the same way that generative modelling is used for visual data like images and videos, time series can be used for auditory data, like speech and music (Dhariwal, Jun et al., 2020, Zeghidour et al., 2021). Audio generation is an active research field. This is a difficult task due to the high dimensionality of digital audio, usually having 44100 sampling points per second. Machine learning algorithm are already being used for efficient audio compression and modelling (Kumar et al., 2023), allowing artists to generate music and service industries for text-to-speech generation.

1.2 Background for generative modelling of time series

Image generation

The last decade saw a huge breakthrough in machine learning with the invention of *generative adversarial networks* (GANs) (I. J. Goodfellow et al., 2014). This class of methods was a major steppingstone for generative modelling. Data analysts could now look beyond what was already present in the data by synthesizing new data. GANs quickly became popular with their success and dominance in image synthesis. However, they plateaued due to two main problems arising from their adversarial formulation. The first issue is that they are not easy to train, as both a generator and a discriminator has to be trained together. The second issue is mode collapse, where the generator gets stuck in local modes thus having problems learning the full data distribution, leading to lack of diversity in the generated samples.

Diffusion models where first introduced in (Sohl-Dickstein et al., 2015) with a motivation from non-equilibrium thermodynamics. In recent years, they gained significant popularity due to their impressive results on image synthesis. A handful of papers released in the 2020s alone have shown what diffusion models are capable of, such as beating GANs on image synthesis (Dhariwal and Nichol, 2021) and generating photo-realistic images from text prompts (Saharia et al., 2022). Some of these models have been open-sourced, such as *Stable Diffusion* (Rombach et al., 2021), which allows practitioners to create high-resolution images from text prompts and preform in- and out-painting. Diffusion models have also yielded record-breaking performance across other domains, such as video generation and molecule design, emerging as a powerful new family of deep generative models (Yang et al., 2022).

Time series generation

TSG has previously been based on GANs combined with *recurrent neural network* (RNN) (Yoon et al., 2019) and (Ni et al., 2020). However, due to the adversarial nature of GANs, the difficulty with training and mode collapse remains. In addition, RNNs often lack the ability to capture long temporal dependencies. The problem with temporal dependencies has been tried to be fixed using transformer alternatives, such as TTS-CGAN (Transformer Time-Series Conditional GAN) in (Li et al., 2022). The image generation literature have shown significant performance gains by switching to diffusion models instead of GAN alternatives, as seen with Imagen (Saharia et al., 2022), Stable diffusion (Rombach et al., 2021), and Midjourney (Oppenlaender, 2022). According to (Lee et al., 2023), adapting diffusion models should expect a similar performance gain for TSG. Diffusion models are both flexible and tractable, making them suitable also for modelling time series. Their strength lies in being able to create high quality samples with good mode coverage (Xiao et al., 2021).

Frameworks using *vector quantization* (VQ) allows for effective discrete representations of data (Oord et al., 2017). Lee et al., 2023 proposes TimeVQVAE, as the first application of VQ techniques for TSG. The authors uses a two stage modelling approach, where the first stage involves creating latent representations of the time series using a *vector-quantized variational autoencoder* (VQVAE) (Oord et al., 2017). One notable contribution of their work is the utilization of a bi-directional transformer (Chang et al., 2022) to learn the priors of the latent space. The transformer architecture (Vaswani et al., 2017), popularized by their successful in natural language processing, allows for capturing the global temporal consistency more effectively than RNN alternatives. Lee et al., 2023 further propose to model the time series in time-frequency domain, which creates samples with sharp changes in modularity. For natural audio, Kumar et al., 2023 proposes Improved RVQGAN, a universal audio compression model with 90x compression rate, capable of handling speech, music and environmental sounds. This model also VQ for efficient encoding of audio, and the prior is modelled using a GAN architecture. A natural direction of research into TSG is to use recent advances of diffusion model in the image generation domain and combining it with advances using VQ for time series.

1.3 Thesis overview

This thesis analyses time series generation using diffusion models and vector quantization. To the best of our knowledge, this approach has not been done before. This work is mainly inspired by the following four papers:

- Diederik P. Kingma et al., 2021: Variational Diffusion Models
- Lee et al., 2023: Vector Quantized Time Series Generation with a Bidirectional Prior Model
- Rombach et al., 2021: High-Resolution Image Synthesis with Latent Diffusion Models
- Ho, Saharia et al., 2021: Cascaded Diffusion Models for High Fidelity Image Generation

The proposed model is called TimeVQVDM. It is short for vector-quantized variational diffusion models for time series. It is based on the two stage modelling approach in TimeVQVAE (Lee et al., 2023). The first stage involves using vector quantized variational autoencoders for creating latent representations of the time series data. Similar to TimeVQVAE, the time series are analysed in the time-frequency domain which are further split into *low frequencies* (LF) and *high frequencies* (HF). The second stage models the latent distribution with diffusion models. This is analogous with Stable Diffusion (Rombach et al., 2021), where the diffusion model operates in a latent space. For this thesis, the *variational diffusion model* (VDM) (Diederik P. Kingma et al., 2021) is used, as it offers greater theoretical understanding of the log likelihood behind the diffusion process. During the generation process, the low frequencies are generated first and then the high frequencies are generated conditioned on the low frequencies. This cascading generating process is inspired by (Ho, Saharia et al., 2021), whereby high-resolution images are sampled by first generating a low-resolution image and generating the higher resolution afterwards by conditioning on the low resolution image. The same conditioning mechanism used in (Ho, Saharia et al., 2021) is adapted for TimeVQVDM, where generation of HF components is conditioned on generated LF components.

Section 2 is **Theory**. This section covers the necessary theory for the proposed method. The first part states the assumptions for the data set and introduces how variational inference is used for optimizing the models parameters using a loss function. The section then proceeds with presenting the variational autoencoder and how this is combined with vector quantization for creating the latent representations. Diffusion models are then covered, including the sampling procedure. The next part of the theory covers modelling of time series in the time-frequency domain and introduces relevant neural network modules. These include stage one of TimeVQVAE, the U-Net (Ronneberger et al., 2015), and the *fully convolutional network* (FCN) (Z. Wang et al., 2016). The last part of the theory covers evaluation metrics, along with their interpretations. Three evaluation metrics are introduced; the Fréchet inception distance (FID), inception score (IS) and classification accuracy score (CAS). The reader is expected to have a moderate knowledge of statistics and machine learning, as the basics is not covered.

Section 3 is **Experimental Setup**. Details regarding TimeVQVDM, the training data and the loss function is provided here. In addition, the algorithms for training and sampling are given. Several implementation tricks has been made for TimeVQVDM for combining diffusion models with VQ for modelling of time series. These include conditioning mechanisms for the low and high frequency diffusion models and making them work with the stage one of TimeVQVAE. The experimental setup also covers how the model is configured and trained. Next the UCR time series archive (Dau et al., 2018) is covered. It containing a wide range of different real world as well as generated time series from various domains.

Section 4 and 5 is **Results** and **Discussion**. The **Results** section analyses the model by evaluating the performance of TimeVQVDM for both unconditional and conditional sampling. Both qualitative assessment, using visual inspections, and quantitative, using the evaluation metrics, are given. Finally, the **Discussion** section discusses the relative performances of TimeVQVDM on different UCR data sets and compares it with TimeVQVAE, identifying the strength and weaknesses of the proposed model.

2 Theory

2.1 Inference for Time Series Generation

Inference in probabilistic models for time series is often challenging due to their complex temporal structure. It is generally impossible to find the marginal distribution of time series exactly, forcing us to search for approximations. The problem of approximating the distribution of some time series data is referred to as the *inference problem*.

For some data sets the inference problem can be approximated using sampling-based methods, where the generation process involves sampling of random variables as subroutines. A popular type is *Markov Chain Monte Carlo* (MCMC), with instances like the Metropolis-Hastings algorithm (Nicholas et al., 1953; Hastings, 1970) and the Gibbs sampler (S. Geman and D. Geman, 1984). Sampling-based methods can be utilized whenever the dependencies among the components of the random variables are known, such that the inference problem can be formulated as an integral over a known distribution. In practice, the resulting integral is often approximated with numerical methods. MCMC can be difficult to implement efficiently, as it could be hard to pick good proposal distributions and to tell if the Markov chain has converged properly before it can be used for inference.

Variational inference is today a more widely used inference technique for complex data distributions (Blei et al., 2016). It is more applicable for time series since it can be used for the general inference problem, without any explicit assumptions for the dependencies in the time series.

2.1.1 Data set assumptions

The data sets contain time series with corresponding classified labels. Formally, assume that $\boldsymbol{X}$ and Y are observable random variables on the spaces $\mathcal{X}$ and $\mathcal{Y}$ and let q denote the underlying true probability distributions of these variables. For instance, $q(\boldsymbol{x})$ is the marginal distribution of $\boldsymbol{X}$ and $q(\boldsymbol{x}, y)$ is the joint distribution of $\boldsymbol{X}$ and Y. Assume that we have observed n independently and identically distributed data points from the true distribution. This data is referred to as the training data and will also be denoted *the ground truth.* The training data can be written as a set of data points

$$\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_n, y_n)\}, \tag{1}$$

where each time series $\boldsymbol{x} \in \mathcal{X} \subseteq \mathbb{R}^l$ is univariate with fixed length l and the corresponding class label $y \in \mathcal{Y} = \{1, \dots, C\}$, where C is the total number of classes. We assume that the range of $\boldsymbol{X}$ is not bounded, meaning that $\boldsymbol{x}$ can take any value in $\mathbb{R}^l$. In reality, the space $\mathcal{X}$ can be limited. For instance, real time series from counts can be restricted to only having non-negative values or the time series can only take discrete values since measurements of them are rounded. Each time series $\boldsymbol{x}$ can be represented by a sequence of observations as $\boldsymbol{x} = (x_i : i \in I)$, where $I = \{1, 2, \dots, l\}$ is the index set.

The simplest way to approximate $q(\boldsymbol{x})$ is to use the empirical distribution, defined by placing mass $1/n$ on each observation in the training data. Mathematically, it is a discrete probability distribution given by

$$q^*(\boldsymbol{x}) = \frac{1}{n} \sum_{i=1}^{n} \delta_{\boldsymbol{x}_i}, \tag{2}$$

where $\delta_{\boldsymbol{x}_i}$ is the Dirac mass centered at observation $\boldsymbol{x}_i$. The empirical distribution is useful for many purposes, e.g. it is used in bootstrapping for estimating statistical properties of the data. One shortcoming is that this distribution lacks the ability to generate new samples, as it can only output samples seen in the training data. A more sophisticated approach is needed to be able to generate novel content.

2.1.2 Variational inference

The main idea in variational inference is to cast the inference problem as an optimization problem over a family of tractable probability distributions. For this we define a sufficiently large parametric family $\{p_\theta\}_{\theta\in\Theta}$ of distributions, where θ denotes the model parameters. The letter p is used to denote the estimated model probabilities and to distinguish them from the true probabilities, where the letter q is used. Finding the best model then amounts to choosing the parameters

$$\hat{\theta} = \underset{\theta\in\Theta}{\operatorname{argmin}}\ L(p_\theta(\boldsymbol{x}),\ q(\boldsymbol{x})), \tag{3}$$

for some loss function L. The distribution p_θ is good if $p_\theta(\boldsymbol{x}) \approx q(\boldsymbol{x})$, i.e. if it approximates the underlying distribution of time series well. There are two main challenges with variational inference, choosing the loss function L which to minimize and choosing the model architecture for p_θ. The model architecture is presented in Section 3.

The loss function is commonly chosen to be the inclusive *Kullback-Leibler* (KL) divergence, a measure for how one probability distribution differs from a reference distribution. It is defined as

$$D_{\mathrm{KL}}(q(\boldsymbol{x}) \parallel p_\theta(\boldsymbol{x})) := \mathbb{E}_{\boldsymbol{x}\sim q(\boldsymbol{x})}\left[\log\frac{q(\boldsymbol{x})}{p_\theta(\boldsymbol{x})}\right], \tag{4}$$

and can be interpreted as the expected excess surprise of using $p_\theta(\boldsymbol{x})$ as a model when the true model is $q(\boldsymbol{x})$. A basic result in variational inference is that minimizing the KL divergence is equivalent to maximizing the log-likelihood of the data, as seen by the relationship

$$\mathbb{E}_{\boldsymbol{x}\sim q(\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x})\right] = -\mathbb{H}(q) - D_{\mathrm{KL}}(q(\boldsymbol{x}) \parallel p_\theta(\boldsymbol{x})), \tag{5}$$

where $\mathbb{H}(q) := -\mathbb{E}_{\boldsymbol{x}\sim q(\boldsymbol{x})}\left[\log q(\boldsymbol{x})\right]$ is the Shannon entropy of $q(\boldsymbol{x})$, which measures the average level of information inherent in the true data. This term is constant can be omitted. In practice, the true distribution $q(\boldsymbol{x})$ is not available, so we optimize with respect to the empirical distribution $q^*(\boldsymbol{x})$, thereby finding the parameters which maximizes the log-likelihood of the observed data,

$$\hat{\theta} = \underset{\theta\in\Theta}{\operatorname{argmax}}\ \log p_\theta(\boldsymbol{x}). \tag{6}$$

2.1.3 Evidence lower bound

Explicitly parameterized distribution families, such as the exponential family, are too simplistic to model time series data, so we instead consider implicitly parameterized distributions. Let the parameterized distribution p_θ contain a *latent random variable* $\boldsymbol{Z}$ with *prior distribution* $p_\theta(\boldsymbol{z})$. The prior distribution is assumed to be easy to sample from. Also, define a way to convert any $\boldsymbol{z} \sim p_\theta(\boldsymbol{z})$ into a simple distribution over the observable random variable $\boldsymbol{X}$. Then p_θ defines a family of joint distributions over $(\boldsymbol{X}, \boldsymbol{Z})$. Using Bayesian terminology, $\boldsymbol{X}$ is the observed *evidence*, $p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$ is the *likelihood function* and $p_\theta(\boldsymbol{z} \mid \boldsymbol{x})$ is the *posterior distribution* over $\boldsymbol{Z}$.

The *evidence lower bound* (ELBO) is used to maximize Equation (6). The standard formulation of the ELBO is

$$\log p_\theta(\boldsymbol{x}) \geq \mathbb{E}_{z\sim q_\phi(\boldsymbol{z}\mid\boldsymbol{x})}\left[\log\frac{p_\theta(\boldsymbol{x},\boldsymbol{z})}{q_\phi(\boldsymbol{z}\mid\boldsymbol{x})}\right], \tag{7}$$

where $p_\theta(\boldsymbol{x},\boldsymbol{z})$ denotes the *generative model* and $q_\phi(\boldsymbol{z}\mid\boldsymbol{x})$ denotes the *inference model.* Likewise with the true distribution $q(\boldsymbol{x})$, the inference model is written with the letter q to symbolize that it is the true distribution of the latent $\boldsymbol{z}$ conditioned on the time series $\boldsymbol{x}$. The subscript ϕ denotes the parameters in the inference model. Combining Equation (6) with (7), the loss function is chosen as

$$L_{\mathrm{ELBO}}(\theta,\phi\mid\boldsymbol{x}) := \mathbb{E}_{z\sim q_\phi(\boldsymbol{z}\mid\boldsymbol{x})}\left[-\log\frac{p_\theta(\boldsymbol{z},\boldsymbol{x})}{q_\phi(\boldsymbol{z}\mid\boldsymbol{x})}\right]. \tag{8}$$

After optimizing the loss in Equation (8), p_θ can be used for generation in two steps. The first step is to sample the latent variable from the prior distribution, $\boldsymbol{z} \sim p_\theta(\boldsymbol{z})$. The second step is to sample conditionally $\boldsymbol{x} \sim p_\theta(\boldsymbol{x} \mid \boldsymbol{z})$.

2.2 Vector Quantized Variational Autoencoder

TimeVQVDM uses vector quantized variational autoencoders (VQ-VAEs) to create efficient latent representations of time series. VQ-VAE consists of two parts, a variational autoencoder and a vector-quantization model. These are introduced in the following two sections, before presenting the VQ-VAE model.

2.2.1 Variational autoencoder

Variational autoencoders (VAEs) was first introduced in (Diederik P Kingma and Welling, 2013). VAEs are variational Bayesian methods belonging to the family of probabilistic graphical models. The model has a structural similarity with autoencoders, but with a different goal. While auto-encoders learns efficient codings of data using a latent space encoding, the variational autoencoder learns to encode data to a variational distribution, which can be used to generate new samples from.

An autoencoder consists of two neural networks, an encoder network E and decoder network D. The encoder maps the input sample $\boldsymbol{x}$ from the sample space to a latent variable $\boldsymbol{z} = E(\boldsymbol{x})$ in latent space. The decoder then maps from latent space back to sample space, $\hat{\boldsymbol{x}} = D(\boldsymbol{z})$. Usually, there is a dimensionality reduction when encoding the input, such that $\boldsymbol{z}$ has lower dimension than $\boldsymbol{x}$. This forces the autoencoder to find an efficient representation of the signal, often leading to lossy compression, i.e. $\hat{\boldsymbol{x}} = D(E(\boldsymbol{x})) \neq \boldsymbol{x}$.

A variational autoencoder is illustrated in Figure 1. In addition to the encoder and decoder, a VAE also has a third component, a variational distribution in the latent space. The encoder is repurposed to output a set of variables that corresponds to the parameters of the variational distribution. The decoder has the opposite purpose, it inputs the parameters from the latent space and generates data points in sample space. In this way, VAE could be used as a generative model by sampling from the variational distribution and decoding to sample space. The variational distribution is often chosen to be a multivariate Gaussian distribution. In this setting the encoder predicts the mean vector and covariance matrix,

$$E(\boldsymbol{x}) = \{\boldsymbol{\mu}(\boldsymbol{x}), \boldsymbol{\Sigma}(\boldsymbol{x})\}. \tag{9}$$

The latent representation can then be calculated as

$$\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu}(\boldsymbol{x}), \boldsymbol{\Sigma}(\boldsymbol{x})), \tag{10}$$

which can be efficiently computed with $\boldsymbol{z} = \boldsymbol{\mu}(\boldsymbol{x}) + \mathbf{L}(\boldsymbol{x})\boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $\boldsymbol{\Sigma}(\boldsymbol{x}) = \mathbf{L}(\boldsymbol{x})\mathbf{L}(\boldsymbol{x})^\top$ is the Choleksy decomposition of the covariance vector. The VAE is optimized using the evidence lower bound from Equation (8). The inference model is the variational distribution from the encoder, $q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{\mu}(\boldsymbol{x}), \boldsymbol{\Sigma}(\boldsymbol{x}))$, the prior distribution is chosen as a multivariate isotropic Gaussian $p_\theta(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$, and the likelihood function becomes $p_\theta(\boldsymbol{x} \mid \boldsymbol{z}) = D(\boldsymbol{z})$.

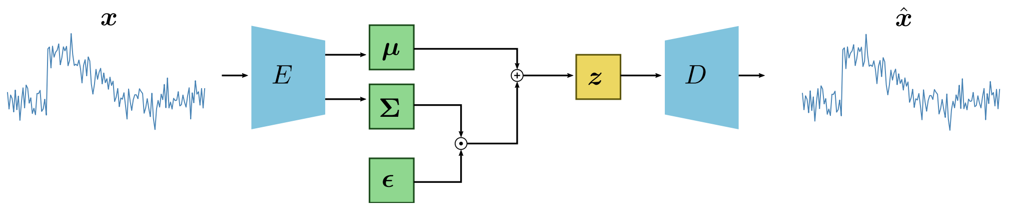


Figure 1: Variational autoencoder. An input signal $\boldsymbol{x}$ is encoded to a latent representation $E(\boldsymbol{x}) = \{\boldsymbol{\mu},\ \boldsymbol{\Sigma}\}$, consisting of the parameters of the variational distribution. Sampling from the VAE is done by sampling a latent vector from the variational distribution. The latent vector can then be decoded to the sample space, $\hat{\boldsymbol{x}} = D(\boldsymbol{z})$.

2.2.2 Vector quantization

Vector Quantization (VQ) is a classical quantization technique in signal processing that involves quantizing data into a finite set of codebook vectors, also called tokens (Gray, 1984). It is a lossy data compression algorithm, and allows modeling of probability density functions using the vectors in the codebook. VQ tries to minimize the distortion between the input signal and the quantized output, thus achieving a compact representation of the data while keeping as much information as possible. It works by dividing a set of training vectors into groups having approximately the same number of vectors closets to them, thus working as a type of clustering algorithm.

The codebook is an essential component in VQ. It is defined as

$$\mathcal{Z} = \{\boldsymbol{e}_1, \dots, \boldsymbol{e}_K\}, \qquad \boldsymbol{e}_k \in \mathbb{R}^d,\ 1 \le k \le K, \tag{11}$$

and contains K vectors with dimension d. $\mathcal{Z}$ can now be used to quantize an encoding of the input signal $\boldsymbol{x}$. Assume that $\boldsymbol{x}$ is encoded to $\boldsymbol{z} = E(\boldsymbol{x}) \in \mathbb{R}^{N \times d}$, where $N \times d$ is the shape of the encoding. In general, N could have more than one dimension, i.e. it can be represented as the Cartesian product $N = N_1 \times \cdots \times N_{\dim(N)}$, where $\dim(N)$ is number of dimensions of N. The corresponding quantization/tokenization process maps each index $i \in N$ of $\boldsymbol{z}$ to

$$(\boldsymbol{z}_q)_i = \underset{\boldsymbol{e}_k \in \mathcal{Z}}{\operatorname{argmin}}\ \|(\boldsymbol{z})_i - \boldsymbol{e}_k\|_2, \tag{12}$$

where $\boldsymbol{z}_q$ represents the quantized version of $\boldsymbol{z}$. The quantization process in Equation (12) compares each continuous token in $\boldsymbol{z}$ with each discrete token $\boldsymbol{e}_k$ in the codebook in terms of a similarity measure and replaces them with the closest. The similarity measures used here is the Euclidean distance. After the quantization, there is a finite number of representations for $\boldsymbol{z}_q$. Since each index is mapped to one of K tokens, then the total number of unique representations is $K^{|N|}$, where $|N|$ is the number of indices. This means that a generative model using VQ can in theory not produce more than this unique number of samples. Figure 2 illustrates a learned codebook for $d = 2$ and $K = 4$. The space $\mathbb{R}^2$ is partitioned into four Voroni cells, each cell consisting of the points closets to the corresponding token inside.

Vector Quantization offer several benefits for data analysis. It often achieves significant data compression with minimal loss of information, making it suitable for various signals, such as images and time series. VQ effectively clusters similar data points together based on the similarity measure, which could lead to smoother and more robust representations of the original data. This could lead to learning patterns and structures useful for feature extraction and downstream tasks.

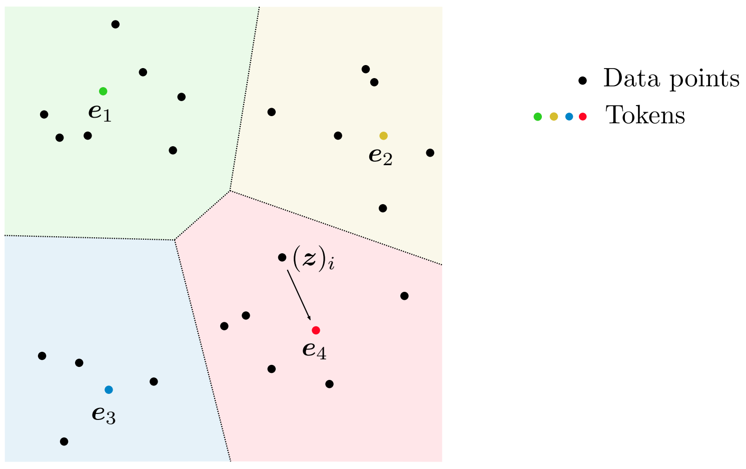


Figure 2: Vector-Quantization in $\mathbb{R}^2$ using 4 tokens and Euclidean distance measure. The space is partitioned into a four regions represented using the tokens in the codebook. The quantization process is illustrated by mapping the continuous vector $(\boldsymbol{z})_i$ to the closest token $\boldsymbol{e}_4$.

2.2.3 Vector quantized variational autoencoder

Vector quantized variational autoencoders (VQ-VAE) was first introduced in (Oord et al., 2017). It is an unsupervised technique for learning discrete representations of data. VQ-VAE differs from VAEs in two ways, the encoder network outputs discrete latent representations, rather than continuous, and the prior distribution is learned, rather than static. The latent representations outputted by the encoder network is quantized using ideas from VQ. This allows the model to learn useful discrete representations of the data and circumvent issues of posterior collapse seen with standard VAEs. VQ-VAEs produces sharper reconstructions than AE and VAE (Bank et al., 2020). An overview of the VQ-VAE model is shown in Figure 3. A VQ-VAE model has an encoder and decoder similar to AEs. In addition, there is a codebook for doing vector quantization in latent space. The input sample $\boldsymbol{x}$ is encoded to a latent space $\boldsymbol{z} = E(\boldsymbol{x})$. The continuous representation is then quantized with the codebook, using the quantization process in Equation (12). Denoting the quantized latent as $\boldsymbol{z}_q$, then the reconstructed sample is $\hat{\boldsymbol{x}} = D(\boldsymbol{z}_q)$.

Similar to VAEs, VQ-VAEs are optimized using the evidence lower bound, which in turn optimizes for the likelihood of the data. Oord et al., 2017 suggests learning the latent distribution, often called the prior distribution, after training the encoder, decoder and codebook. To do this, the authors suggests a two-stage training approach, where training the neural networks is divided into two stages. Stage one involves training the encoder and decoder together with the codebook. In stage one, a uniform prior is assumed for the distribution of the codebook tokens. Since the uniform distribution is the maximum entropy distribution over a categorical distribution such as the codebook tokens, it is non-informative. The inference distribution is the quantized encoded signal, $q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) = E(\boldsymbol{x})_q = \boldsymbol{z}_q$, and the generative model is the joint of the latent variable and the decoded latent variable, $p_\theta(\boldsymbol{x}, \boldsymbol{z}_q) = p_\theta(\boldsymbol{x} \mid \boldsymbol{z}_q)p_\theta(\boldsymbol{z}_q)$, where $p_\theta(\boldsymbol{x} \mid \boldsymbol{z}_q) = D(\boldsymbol{z}_q)$ and $p_\theta(\boldsymbol{z}_q)$ is uniform over the tokens in the codebook. After stage one, the discrete prior distribution of the tokens in the codebook is learned. During the training of the prior, the parameters in the encoder, decoder and codebook are freezed.

Learning the prior distribution can be reduced to learning the indices of the tokens in the codebook. Since the quantization process maps to discrete representations, the indices of the tokens in the codebook can be modelled, rather than the tokens themselves. This could be for instance be paired with autoregressive models or attention-based models, such as transformers (Vaswani et al., 2017). Standard VAE often has good mode coverage, but they lack in sample quality of generated samples (Xiao et al., 2021). Using VQ-VAE combined with a learned prior allows for more flexible distributions, while minimizing the loss of information, which could lead to higher quality in the generated samples. For instance, TimeVQVAE utilizes a bidirectional transformer (Chang et al., 2022) for modelling the indices of the codebook.

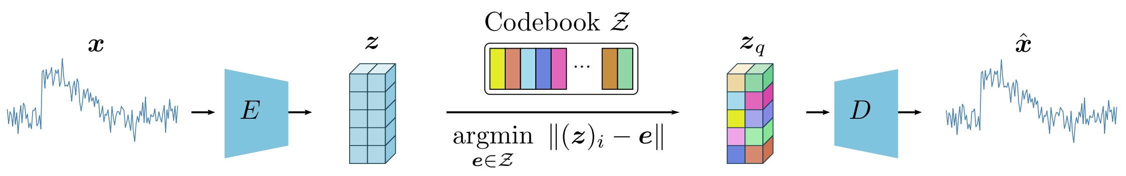

$\boldsymbol{x}$
E
$\boldsymbol{z}$
Codebook $\mathcal{Z}$
$\underset{\boldsymbol{e}\in\mathcal{Z}}{\operatorname{argmin}} \; \|(\boldsymbol{z})_i - \boldsymbol{e}\|$
$\boldsymbol{z}_q$
D
$\hat{\boldsymbol{x}}$

Figure 3: Overview of a vector-quantized variational autoencoder (VQ-VAE). The input $\boldsymbol{x}$ is encoded to a latent representation $\boldsymbol{z} = E(\boldsymbol{x})$. The latent representation is then quantized, before it is reconstructed back to the sample space using a decoder $\hat{\boldsymbol{x}} = D(\boldsymbol{z}_q)$. Paired with a prior model of the latent representation, the VQ-VAE is capable of generative modelling.

2.3 Diffusion Models

TimeVQVDM uses diffusion models to learn the latent distribution of time series. In machine learning, a diffusion model is a latent variable model which maps to a latent space using a *diffusion process*. Diffusion models was introduced in (Sohl-Dickstein et al., 2015) and based the diffusion process on non-equilibrium thermodynamics. The idea is to destroy structure in the data through a forward noising process. A neural network is then trained to reverse this process, essentially denoising the diffused data, thus recovering structure from the noise.

This thesis adapts the *Variational Diffusion Model* (VDM), first described by Google research in (Diederik P. Kingma et al., 2021). It is a continuous time-step diffusion model defined as a latent variable model with latents $\boldsymbol{z} = \{\boldsymbol{z}_t \mid t \in [0,1]\}$ that obey a *forward process* $q_\phi(\boldsymbol{z} \mid \boldsymbol{z}_0)$ starting at the data distribution encoded to latent space, $\boldsymbol{z}_0 \sim q_\phi(\boldsymbol{z}_0 \mid \boldsymbol{x})$. This forward process is a Gaussian noising process on $[0,1]$ that satisfies the continuous Markovian structure

$$q_\phi(\boldsymbol{z}_t \mid \boldsymbol{z}_0) = \mathcal{N}\left(\boldsymbol{z}_t;\ \alpha_t \boldsymbol{z}_0,\ \sigma_t^2 \mathbf{I}\right), \qquad q_\phi(\boldsymbol{z}_t \mid \boldsymbol{z}_s) = \mathcal{N}\left(\boldsymbol{z}_t;\ \alpha_{t|s} \boldsymbol{z}_s,\ \sigma_{t|s}^2 \mathbf{I}\right), \tag{13}$$

where $0 \le s < t \le 1$ are time steps and the conditional mean and variance are defined as

$$\alpha_{t|s} := \frac{\alpha_t}{\alpha_s}, \qquad \sigma_{t|s}^2 := \sigma_t^2 - \alpha_{t|s}^2 \sigma_s^2. \tag{14}$$

Note that α_t and σ_t are non-negative scalar-valued smooth functions of t, such that the noising schedule is differentiable. The forward process in Equation (13) can be formulated as a reversible *stochastic differential equation* (SDE), as outlined in Appendix A. The *reverse process* is defined as $q_\phi(\boldsymbol{z}_s \mid \boldsymbol{z}_t)$ and is the true distribution of the latent $\boldsymbol{z}_s$ conditioned on a more noisy latent $\boldsymbol{z}_t$. It can in general not be written in an analytical form and must be estimated. A useful property is that the Markov property also holds for the reverse process, allowing sampling from the estimated reverse process. The goal of diffusion models is to learn this process, thus being able to recover structure from noise.

2.3.1 Signal-to-noise ratio

From Equation (13) we can write the latent at time $t \in [0,1]$ as

$$\boldsymbol{z}_t = \alpha_t \boldsymbol{z}_0 + \sigma_t \boldsymbol{\epsilon}, \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{15}$$

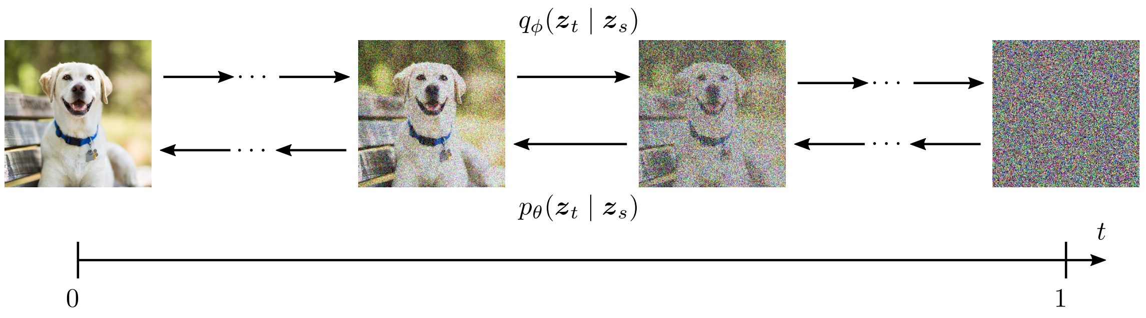
using the reparameterization trick. It consists of two components, referred to as the signal component and the noise component. They are given as $\alpha_t \boldsymbol{z}_0$ and $\sigma_t \boldsymbol{\epsilon}$, respectively. The *signal-to-noise ratio* (SNR) is the ratio of the power of the signal component to the power of the noise component, and is a common measure in electrical and communication engineering. Since power is equal to expected squared amplitude, the SNR can be written as

$$\mathrm{SNR}(t) := \frac{\mathbb{E}[(\alpha_t \boldsymbol{z}_0)^2]}{\mathbb{E}[(\sigma_t \boldsymbol{\epsilon})^2]} = \frac{\alpha_t^2}{\sigma_t^2} \cdot \frac{\mathbb{E}[\boldsymbol{z}_0^2]}{\mathbb{E}[\boldsymbol{\epsilon}^2]}. \tag{16}$$

Because $\boldsymbol{\epsilon}$ is standard Gaussian, then $\mathbb{E}[\boldsymbol{\epsilon}^2] = \mathbb{V}[\boldsymbol{\epsilon}] + \mathbb{E}[\boldsymbol{\epsilon}]^2 = 1$. In fact, we can also assume that $\mathbb{E}[\boldsymbol{z}_0^2] = 1$ by making adjustments to the parameters ϕ of the encoder $q_\phi(\boldsymbol{z}_0 \mid \boldsymbol{x})$, such that the data distribution encoded to latent space is standardized to have mean zero and unit variance. The SNR is thus simply equal to $\mathrm{SNR}(t) = \alpha_t^2/\sigma_t^2$. Furthermore, we define the *logarithmic signal-to-noise ratio* as

$$\lambda_t := \log \mathrm{SNR}(t) = \log \frac{\alpha_t^2}{\sigma_t^2}. \tag{17}$$

For the forward process to be a proper diffusion process, we must ensures that the latent becomes noisier with time. This is done by assuming that the $\mathrm{SNR}(t)$, and likewise λ_t, is strictly monotonically decreasing on $[0,1]$, i.e. that $\lambda_t < \lambda_s$ whenever $0 \le s < t \le 1$. Figure 4 illustrates a diffusion process for images, where Gaussian noise is added to all three color channels. The decreasing SNR makes the image gradually noiser with time.



$q_\phi(\boldsymbol{z}_t \mid \boldsymbol{z}_s)$
$p_\theta(\boldsymbol{z}_t \mid \boldsymbol{z}_s)$
t
0
1

Figure 4: Forward and backward diffusion processes applied to images. The forward process, given by $q_\phi(\boldsymbol{z}_t \mid \boldsymbol{z}_s)$, adds Gaussian noise to the image following the definition in Equation (13). The backwards process is estimated with $p_\theta(\boldsymbol{z}_t \mid \boldsymbol{z}_s)$ by a neural network with parameters θ. The input becomes gradually noisier through the diffusion process.

2.3.2 Noise schedule

The noising schedule for the diffusion process is fully determined by the two functions α_t and σ_t. We use a *variance conserving* noise schedule, meaning that the variance of the latent variable at any time is constant. In fact, this is without lack of generality, as is shown in Appendix G of (Diederik P. Kingma et al., 2021). We further use the cosine schedule, such that the noising schedule is given by

$$\alpha_t := \cos(\beta_t), \qquad \sigma_t := \sin(\beta_t), \tag{18}$$

where $\beta_t := \frac{\pi t}{2}$. Thus it is easy to verify that $\mathbb{V}[\boldsymbol{z}_t] = \cos^2(\beta_t) + \sin^2(\beta_t) = 1$ for all $t \in [0,1]$, so the variance conserving diffusion is satisfied. With the noising process defined by Equation (18), the completely diffused latent has limiting distribution exactly equal to $\boldsymbol{z}_1 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The distribution of the latent $\boldsymbol{z}_1$ is denoted as the prior distribution, and has a SNR of zero. In other words, all information of the input $\boldsymbol{z}_0$ is lost, i.e. $q_\phi(\boldsymbol{z}_1 \mid \boldsymbol{z}_0) = q_\phi(\boldsymbol{z}_1) = \mathcal{N}(\mathbf{0}, \mathbf{I})$.

2.3.3 Approximation of the reverse process

Learning the reverse diffusion process can be reduced to learning to denoise $\boldsymbol{z}_t \sim q_\phi(\boldsymbol{z}_t \mid \boldsymbol{z}_s)$ into an estimate $p_\theta(\boldsymbol{z}_s \mid \boldsymbol{z}_t) \approx q_\phi(\boldsymbol{z}_s \mid \boldsymbol{z}_t)$. For this the auxiliary distribution $q_\phi(\boldsymbol{z}_s \mid \boldsymbol{z}_t, \boldsymbol{z}_0)$, where $0 \le s < t \le 1$, is used. In Appendix B it is shown that $q_\phi(\boldsymbol{z}_s \mid \boldsymbol{z}_t, \boldsymbol{z}_0)$ is isotropic Gaussian,

$$q_\phi(\boldsymbol{z}_s \mid \boldsymbol{z}_t, \boldsymbol{z}_0) = \mathcal{N}(\boldsymbol{\mu}_{s|t,0}(\boldsymbol{z}_t,\ \boldsymbol{z}_0),\ \sigma^2_{s|t,0}\mathbf{I}), \tag{19}$$

with mean and variance given by

$$\boldsymbol{\mu}_{s|t,0}(\boldsymbol{z}_t,\ \boldsymbol{z}_0) = e^{\lambda_t - \lambda_s}\frac{\alpha_s}{\alpha_t}\ \boldsymbol{z}_t + (1 - e^{\lambda_t - \lambda_s})\alpha_s\ \boldsymbol{z}_0, \qquad \sigma^2_{s|t,0} = (1 - e^{\lambda_t - \lambda_s})\sigma_s^2. \tag{20}$$

If $\boldsymbol{z}_0$ is known, then there is an analytical form for the reverse process. A noisy latent $\boldsymbol{z}_t$ arising from diffusing $\boldsymbol{z}_0$ can then be used to find the true distribution of the less noisy latent $\boldsymbol{z}_s$. However, $\boldsymbol{z}_0$ is not available. Instead the reverse process is approximated as

$$p_\theta(\boldsymbol{z}_s \mid \boldsymbol{z}_t) := q_\phi(\boldsymbol{z}_s \mid \boldsymbol{z}_t, f_\theta(\boldsymbol{z}_0; \boldsymbol{z}_t)) \approx q_\phi(\boldsymbol{z}_s \mid \boldsymbol{z}_t), \tag{21}$$

where $f_\theta(\boldsymbol{z}_0; \boldsymbol{z}_t) \approx \boldsymbol{z}_0$ is the model prediction of $\boldsymbol{z}_0$ when the input is the diffused sample $\boldsymbol{z}_t$. The notation $f_\theta(\cdot)$ is used to denote the model's predictions, which differs from the model's distributions $p_\theta(\cdot)$. To be clear, p_θ is used for the generative models probability distributions, while f_θ is used for any specific prediction made by the generative model.

2.3.4 Alternative parameterizations

In practice, diffusion models are often parameterized to predict other quantities than $\boldsymbol{z}_0$. The authors of (Ho, Jain et al., 2020) found that predicting the standardized noise component, $\boldsymbol{\epsilon}$, produces higher sample quality. Using Equation (15), the predicted noise at time t is

$$f_\theta(\boldsymbol{\epsilon}) = \frac{1}{\sigma_t}\left(\boldsymbol{z}_t - \alpha_t f_\theta(\boldsymbol{z}_0)\right) \approx \frac{1}{\sigma_t}\left(\boldsymbol{z}_t - \alpha_t \boldsymbol{z}_0\right) = \boldsymbol{\epsilon}. \tag{22}$$

In this way, the diffusion model is instead trained to predict $\boldsymbol{\epsilon}$ and uses it to calculate the prediction for $\boldsymbol{z}_0$. Substituting $\boldsymbol{z}_0 = \frac{1}{\alpha_t}\left(\boldsymbol{z}_t - \sigma_t \boldsymbol{\epsilon}\right)$ into Equation (20), the mean and variance of the reverse process can be expressed as

$$\boldsymbol{\mu}_{s|t,0}(\boldsymbol{z}_t,\ \boldsymbol{\epsilon}) = \frac{\alpha_s}{\alpha_t}\boldsymbol{z}_t + (1 - e^{\lambda_t - \lambda_s})\frac{\alpha_s}{\alpha_t}\sigma_t\ \boldsymbol{\epsilon}, \qquad \sigma^2_{s|t,0} = (1 - e^{\lambda_t - \lambda_s})\sigma_s^2. \tag{23}$$

Another popular parameterization, which is designed for variance conserving diffusion, is the diffusion velocity, defined by

$$\boldsymbol{v}_t := \frac{\partial \boldsymbol{z}_t}{\partial \beta_t} = \alpha_t \epsilon - \sigma_t \boldsymbol{z}_0, \tag{24}$$

where $\beta_t = \pi t/2$ is interpreted as the diffusion angle. Predicting the diffusion velocity was first described in (Salimans and Ho, 2022) and has been successfully used in some diffusion models such as (Saharia et al., 2022) and (Ho, Chan et al., 2022). The velocity has some interesting properties, such as having unit variance, $\mathbb{V}[\boldsymbol{v}_t] = 1$. Lin et al., 2023 recommends always picking the $\boldsymbol{v}$-parameterization for diffusion models. This is further discussed in section 5.

A geometric visualization of the different parameterizations is shown in Figure 5. The figure shows how each point in latent space can be decomposed in a signal- and noise-component. Each point in this space is in itself a probability distribution, weighted by the signal- and the noise-coordinates. The latents in the forward diffusion process, $\boldsymbol{z}_t$, for a $t \in [0, 1]$, can be plotted as points on the form (α_t, σ_t). From the definitions of α_t and σ_t, this corresponds to locations on the unit sphere in the first quadrant. $\boldsymbol{z}_0$ has only signal components, so it is located on $(1, 0)$, while $\boldsymbol{\epsilon}$ is a Gaussian distribution, so it is located on $(0, 1)$. In this space, the reason behind the diffusion velocity became clear. It can be visualized as the direction of the diffusion process, similar to circular motion in mechanics. Note that when $t = 0$, then the velocity is equal to the noise component. Also, when $t = 1$, then the velocity is the signal multiplied by -1.

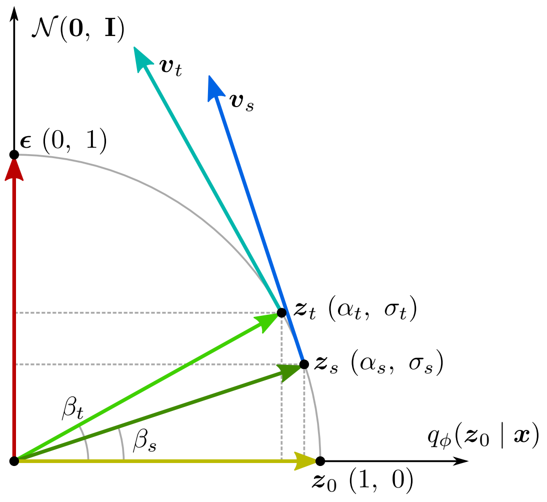

$\mathcal{N}(\mathbf{0},\ \mathbf{I})$
$\boldsymbol{v}_t$
$\boldsymbol{v}_s$
$\boldsymbol{\epsilon}\ (0,\ 1)$
$\boldsymbol{z}_t\ (\alpha_t,\ \sigma_t)$
$\boldsymbol{z}_s\ (\alpha_s,\ \sigma_s)$
β_t
β_s
$q_\phi(\boldsymbol{z}_0 \mid \boldsymbol{x})$
$\boldsymbol{z}_0\ (1,\ 0)$

Figure 5: Geometric visualization of a continuous time-step diffusion process with variance-conserving noise schedule. The signal amplitude is on the x-axis and noise amplitude is on the y-axis. The forward diffusion process morphs the signal distribution into a noise distribution. The latent $\boldsymbol{z}_s$ contains more signal and less noise than the latent $\boldsymbol{z}_t$.

2.3.5 Sampling

To be able to sample from a diffusion model, time must be discretized. Let T be a finite number of denoising steps. Time is discretized uniformly on $[0, 1]$ using the grid $\boldsymbol{\tau} = \{\tau_i\}_{i=0}^{T}$, where $\tau_i = i/T$. Since that the Markov property also holds for the reverse diffusion process, the joint distribution for the reverse Markov process on the discretized space can be written as

$$q_\phi(\boldsymbol{z}_{\boldsymbol{\tau}}) = q_\phi(\boldsymbol{z}_1) \prod_{i=1}^{T} q_\phi(\boldsymbol{z}_{\tau_{i-1}} \mid \boldsymbol{z}_{\tau_i}). \tag{25}$$

Assuming a trained diffusion model with $p_\theta(\boldsymbol{z}_s \mid \boldsymbol{z}_t) \approx q_\phi(\boldsymbol{z}_s \mid \boldsymbol{z}_t)$ for $0 \leq s < t \leq 1$, we can approximate the joint distribution of the reverse process as

$$p_\theta(\boldsymbol{z}_{\boldsymbol{\tau}}) = p(\boldsymbol{z}_1) \prod_{i=1}^{T} p_\theta(\boldsymbol{z}_{\tau_{i-1}} \mid \boldsymbol{z}_{\tau_i}), \tag{26}$$

where the prior $p(\boldsymbol{z}_1) = q_\phi(\boldsymbol{z}_1) = \mathcal{N}(\mathbf{0}, \mathbf{I})$. Each term in the product in Equation (26) corresponds to a single denoising step. In total there are T denoising steps.

A sample from the learned distribution $p_\theta(\boldsymbol{x}) \approx q(\boldsymbol{x})$ is generated by starting with a sample from the prior distribution, $\tilde{\boldsymbol{z}}_1 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Then T denoising steps of the form $p_\theta(\tilde{\boldsymbol{z}}_{\tau_{i-1}} \mid \tilde{\boldsymbol{z}}_{\tau_i})$ for $i = T, \ldots, 1$ are preformed to get successively less noisy estimates. The last prediction $\tilde{\boldsymbol{x}} \sim p_\theta(\boldsymbol{x} \mid \tilde{\boldsymbol{z}}_0)$ is then a sample generated by the diffusion model. The sampling process is preformed serially, i.e. one after the other. It is sometimes referred to as the *ancestral sampler*. Inserting the approximate reverse distribution given in Equation (21), the reverse process approximated by the diffusion model can be written as

$$p_\theta(\boldsymbol{z}_{\boldsymbol{\tau}}) = p(\boldsymbol{z}_1) \prod_{i=1}^{T} q_\phi(\boldsymbol{z}_{\tau_{i-1}} \mid \boldsymbol{z}_{\tau_i}, f_\theta(\boldsymbol{z}_0; \boldsymbol{z}_{\tau_i})), \tag{27}$$

where $f_\theta(\boldsymbol{z}_0; \boldsymbol{z}_{\tau_i})$ is the model prediction of $\boldsymbol{z}_0$ when inputting the latent $\boldsymbol{z}_{\tau_i}$. Sampling from $q_\phi(\boldsymbol{z}_{\tau_{i-1}} \mid \boldsymbol{z}_{\tau_i}, f_\theta(\boldsymbol{z}_0; \boldsymbol{z}_{\tau_i}))$ is preformed by sampling

$$\boldsymbol{z}_{\tau_{i-1}} = \boldsymbol{\mu}_{s|t,0}(\boldsymbol{z}_{\tau_t},\ f_\theta(\boldsymbol{z}_0; \boldsymbol{z}_{\tau_i})) + \sigma_{s|t,0}\ \boldsymbol{\delta}, \qquad \boldsymbol{\delta} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{28}$$

where the mean and variance is given in Equation (20) and $\boldsymbol{\delta} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is the stochasticity of the sampler. Since sampling from this model can be expensive, a lower number of denoising steps can be chosen to get lower quality samples. If one instead wants higher quality samples, a larger number of denoising steps can be used.

2.3.6 Classifier-free guidance

Similar to other types of generative models, such as GANs and transformers, diffusion models are capable of modeling conditional distributions of the form $p(\boldsymbol{x} \mid \boldsymbol{c})$. This can be implemented by either using a separate classifier to guide the generation process or directly with a conditional diffusion model and paves the way to controlling the synthesis process through inputs $\boldsymbol{c}$ such as labels, text, semantic maps or other conditions. Conditional generation has been successfully used in many text-to-image generators, for instance in (Saharia et al., 2022), where the conditional argument is a text prompt.

Dhariwal and Nichol, 2021 proposes *classifier guidance*, which uses a separate classifier to guide the sampling process of the diffusion model. A drawback of this is that one also need to define and train a classifier alongside the diffusion model. One could also define the diffusion model so that it incorporates a conditional argument, so that there is no need for a separate classifier. This is called *classifier-free* guidance, and the idea is to let the diffusion model be trained with the conditional arguments. The forward noising process remains unchanged. This approach was first used by (Ho, Saharia et al., 2021) for high quality fidelity image generation using cascaded diffusion models, and was also found to work better than classifier guidance in (Nichol et al., 2021).

In this thesis, the class labels are an example of a conditional argument. The class labels y are encoded using a label encoder mapping the class to the set $\{0, 1, \cdots, C-1\}$. To ensure that the diffusion model still can be used for unconditional sampling, there is a dropout probability $p_{\text{unconditional}} = 0.1$ of not conditioning on the class label during training, such that the unconditional class is used to update the parameters in the model in 10% of the training iterations. Let $f_\theta(\boldsymbol{\epsilon}; \boldsymbol{z}_t, t, y)$ be the model prediction of the noise when inputting the latent $\boldsymbol{z}_t$, the time t and the corresponding class y. Similarly, let $f_\theta(\boldsymbol{\epsilon}; \boldsymbol{z}_t, t, \emptyset)$ be the model prediction without conditioning on y. Sampling can then be done using a *guidance weight* w, such that the updated noise estimate instead is

$$f_{\theta,\text{cfg}}(\boldsymbol{\epsilon}; \boldsymbol{z}_t, t, y) = (1-w)\ f_\theta(\boldsymbol{\epsilon}; \boldsymbol{z}_t, t, \emptyset) + w\ f_\theta(\boldsymbol{\epsilon}; \boldsymbol{z}_t, t, y). \tag{29}$$

After training, this empowers the model with progressive control over the degree of alignment between the conditional arguments and the sample by varying the guidance weight. Setting $w = 0$ corresponds to unconditional sampling, while setting $w = 1$ results in conditional sampling.

2.3.7 Overexposure problem and rescale classifier-free guidance

In Equation (29), one might think that $w \in [0, 1]$. In practice, researchers have found it beneficial to use large guidance weights. Saharia et al., 2022 found that using large guidance weights produces higher text-to-image alignment for image generation, but that it damages the fidelity by producing highly saturated images. The overexposure problem can be fixed using a *thresholding* technique, where the latents are scaled down or clipped to the proper intervals in-between each sampling step. Saharia et al., 2022 introduces *dynamic thresholding* to fix the overexposure problem. The images are scaled to the interval $[-1, 1]$ prior to the diffusion process. During sampling, the dynamic threshold technique first clips the images to $[-s, s]$ for a $s > 1$ at a certain absolute percentile pixel value and then divide by s. This pushes each pixel values outside of the range $[-1, 1]$ inwards and prevents over-saturation at each step, producing significantly higher photorealism.

The overexposure problem is different for time series data, as they generally have no bounded range. E.g. some time series have outliers with amplitude much higher than the standard deviation of the time series. Lin et al., 2023 finds that when the SNR approaches zero, classifier-free guidance becomes very sensitive, causing overexposure and instabilities during sampling. Inspired by dynamic thresholding, which is only designed for image-space models, they introduce a thresholding technique which rescales the estimates after applying classifier-free guidance. This method is applicable in both sample- and latent-space models. A modification of this procedure is used in this thesis. The rescaling procedure is to calculate a standardized version of the classifier-free guidance estimate. A standardized estimate of the denoised latent using classifier-free guidance is given as

$$f_{\theta,\text{rescaled}}(\boldsymbol{z}_s; \boldsymbol{z}_t, t, y) = \frac{f_{\theta,\text{cfg}}(\boldsymbol{z}_s; \boldsymbol{z}_t, t, y)}{\text{std}(f_{\theta,\text{cfg}}(\boldsymbol{z}_s; \boldsymbol{z}_t, t, y))}. \tag{30}$$

Note that Equation (30) rescales the latent $\boldsymbol{z}_s$, not the noise $\boldsymbol{\epsilon}$. Similar to (Lin et al., 2023), a rescaling parameter $\gamma \in [0, 1]$ is used for the adjusting the estimate. The final estimate is

$$f_{\theta,\text{final}}(\boldsymbol{z}_s; \boldsymbol{z}_t, t, y) = (1-\gamma)\ f_{\theta,\text{cfg}}(\boldsymbol{z}_s; \boldsymbol{z}_t, t, y) + \gamma\ f_{\theta,\text{rescaled}}(\boldsymbol{z}_s; \boldsymbol{z}_t, t, y). \tag{31}$$

Note that the rescaling parameter γ decides how much the model enforces the latents to have unit variance. Choosing $\gamma = 0$ means there are no rescaling, while $\gamma = 1$ means that the latents are scaled to have unit variance in-between each denoising step. Using the rescaling procedure in Equation (31) can avoid the latents from being deviating far away from having unit variance.

2.4 Time-Frequency Modelling

2.4.1 Discrete short-time Fourier transform

Instead of working with time series in the time domain, the signals are converted to time-frequency domain, which offers a richer representation of the signals. To do this, the *short-time Fourier transform* (STFT) is used. STFT is a Fourier-related invertible transformation used to access local frequency and phase content of a signal as it changes over time. The discrete STFT is defined as

$$\mathrm{STFT}(\boldsymbol{x})_{\omega,m} := \sum_{k=0}^{n_{\mathrm{fft}}-1} w_k \boldsymbol{x}_{m\cdot h+k} \exp\left(-i\frac{2\pi\cdot\omega k}{n_{\mathrm{fft}}}\right), \tag{32}$$

where $\boldsymbol{x}$ is the input signal, w_k is the window function, n_{fft} is the window length and h is the hop length. The indices (ω, m) are the height and width of the spectrogram, corresponding to the frequency and time dimensions. For each time-index m, Equation (32) is the *fast Fourier transform* (FFT) applied to a section of the signal $\boldsymbol{x}$ over a window length n_{fft} weighted by the window function. The time resolution depends on the length l of the time series and the hop length h. The STFT offers a trade-off between the frequency and time resolutions of the spectrogram. If the window length is wide, the spectrogram has a high frequency resolution but low time resolution. If the window length is narrow, the spectrogram has a low frequency resolution but high time resolution.

The *inverse short-time Fourier transform* (ISTFT) is defined as the inverse of Equation (32). It is performed using the overlap-add method, where overlapping regions, corresponding to the same time step after applying the inverse FFT for each time index in the spectrogram, are added. The ISTFT can be calculated as

$$\mathrm{ISTFT}(\boldsymbol{u})_n = \sum_{\substack{k,m\\ m\cdot h+k=n}} \boldsymbol{U}_{k,m}, \qquad \boldsymbol{U}_{k,m} = \frac{1}{n_{\mathrm{fft}}}\sum_{\omega=0}^{n_{\mathrm{fft}}-1} w_k^{-1}\boldsymbol{u}_{\omega,m}\exp\left(i\frac{2\pi\cdot\omega k}{n_{\mathrm{fft}}}\right). \tag{33}$$

Figure 6 illustrates the STFT for a time series using a bell shaped window function. The spectrogram produced has $n_{\mathrm{fft}} = 8$ and $h = 1$. Since $h < n_{\mathrm{fft}}$, the windows for where the FFT is applied are overlapping. This means that the frequencies over the overlapping section of the signal contributes to multiple indices of the time domain in the spectrogram. Because the Fourier transform produces complex numbers, the spectrogram is visualized as having two channels, corresponding to the has real and imaginary values.

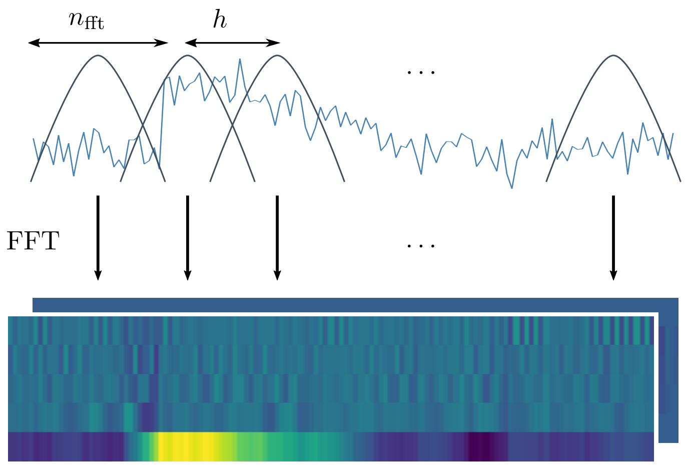


Figure 6: Short-time Fourier transform. The fast Fourier transform is applied at regions of the signal of length n_{fft} every h time step, creating a time-frequency representation with local frequency and phase content through time. The produced spectrogram has $n_{\mathrm{fft}} = 8$ and $h = 1$.

2.4.2 Splitting signals into low and high-frequencies

For an input signal $\boldsymbol{x}$, let $\boldsymbol{u} = \mathrm{STFT}(\boldsymbol{x})$ denote the spectrogram, where the definition of the STFT is given in Equation (32). Following the work of (Lee et al., 2023), we split the spectrogram into low and high frequencies using the two zero-padding operations defined as

$$\mathcal{P}_{\mathrm{LF}}(\boldsymbol{u})_{\omega,m} := \begin{cases} \boldsymbol{u}_{\omega,m} & \text{if} \quad |\omega| \leq \omega_0 \\ 0 & \text{else} \end{cases}, \qquad \mathcal{P}_{\mathrm{HF}}(\boldsymbol{u})_{\omega,m} := \begin{cases} 0 & \text{if} \quad |\omega| \leq \omega_0 \\ \boldsymbol{u}_{\omega,m} & \text{else} \end{cases}, \tag{34}$$

where ω_0 is the cutoff frequency. $\mathcal{P}_{\mathrm{LF}}$ removes high frequency components, while $\mathcal{P}_{\mathrm{HF}}$ removes the low frequency components. A spectrogram can now be split into $\boldsymbol{u} = \boldsymbol{u}^{\mathrm{LF}} + \boldsymbol{u}^{\mathrm{HF}}$, where

$$\boldsymbol{u}^{\mathrm{LF}} = \mathcal{P}_{\mathrm{LF}}(\boldsymbol{u}), \qquad \boldsymbol{u}^{\mathrm{HF}} = \mathcal{P}_{\mathrm{HF}}(\boldsymbol{u}). \tag{35}$$

Using the ISTFT, we can convert back the low and high frequency spectrograms into the low and high frequency signals. The input signal can now be written as $\boldsymbol{x} = \boldsymbol{x}^{\mathrm{LF}} + \boldsymbol{x}^{\mathrm{HF}}$, where

$$\boldsymbol{x}^{\mathrm{LF}} = \mathrm{ISTFT}(\boldsymbol{u}^{\mathrm{LF}}), \qquad \boldsymbol{x}^{\mathrm{HF}} = \mathrm{ISTFT}(\boldsymbol{u}^{\mathrm{HF}}). \tag{36}$$

The reason for splitting the signals is that the low frequencies corresponds to overall structure of the time series, while the high frequencies correspond to finer details and noise. The idea is to generate the lower frequencies first, and then generate the higher frequencies afterwards conditioned on the lower frequencies. Figure 7 illustrates how a signal is split into low and high frequencies.

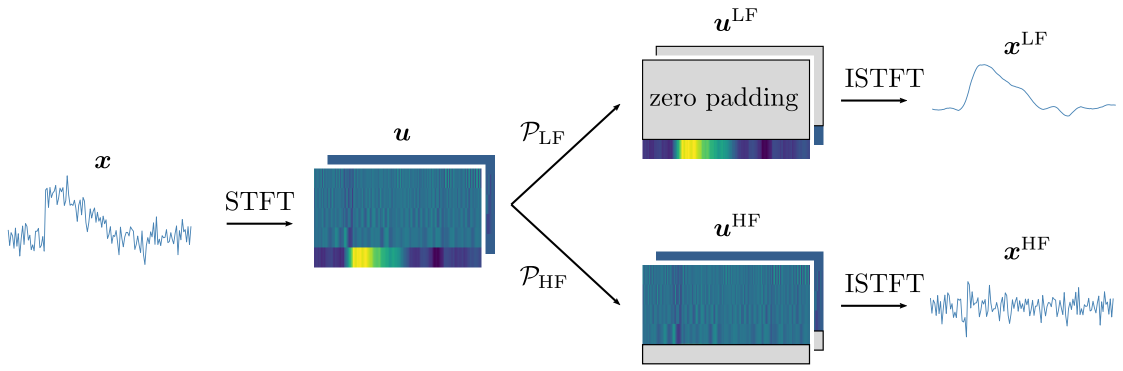

$\boldsymbol{x}$
STFT
$\boldsymbol{u}$
$\mathcal{P}_{\mathrm{LF}}$
$\mathcal{P}_{\mathrm{HF}}$
$\boldsymbol{u}^{\mathrm{LF}}$
zero padding
ISTFT
$\boldsymbol{x}^{\mathrm{LF}}$
$\boldsymbol{u}^{\mathrm{HF}}$
ISTFT
$\boldsymbol{x}^{\mathrm{HF}}$

Figure 7: Splitting a time series into low and high frequencies. Using the short-time Fourier transform, an input signal $\boldsymbol{x}$ is converted to time-frequency domain which is further separated into low and high frequencies. Applying the inverse transform results in a low and high frequency signal.

2.5 Neural Network Architectures

Different neural architectures are used for stage one and two. The VQ-VAE models in stage one is similar to stage one of TimeVQVAE. Therefore, the neural architecture for stage one of TimeVQVAE is presented. Specifics of the implementation are presented in Section 3. For the diffusion models, the neural architecture for the denoising module is implemented using U-Nets (Ronneberger et al., 2015). In addition to these two architectures, the architecture for the fully convolutional network (Z. Wang et al., 2016) is presented. This is a classifier network which is useful for evaluating the performance of TimeVQVDM.

2.5.1 Stage 1 of TimeVQVAE

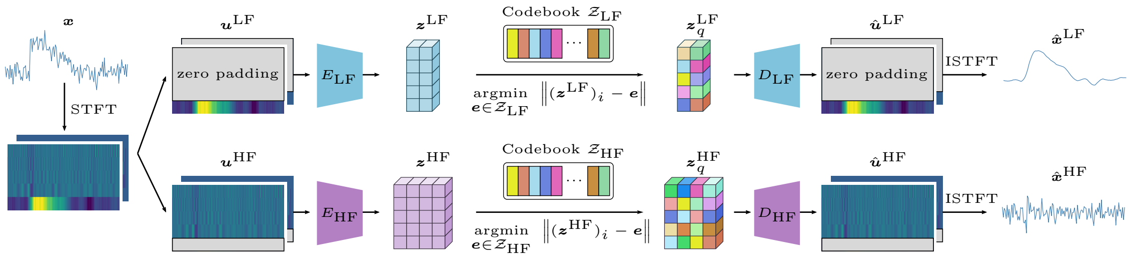
The model architecture for stage one of TimeVQVAE is illustrated in Figure 8. An input signal $\boldsymbol{x}$ is split into LF and HF spectrograms $\boldsymbol{u}^{\mathrm{LF}}$ and $\boldsymbol{u}^{\mathrm{HF}}$ using Equation (35). The spectrograms are then fed through two separate VQ-VAEs, producing latents $\boldsymbol{z}^{\mathrm{LF}}$ and $\boldsymbol{z}^{\mathrm{HF}}$ and tokenized latents $\boldsymbol{z}_q^{\mathrm{LF}}$ and $\boldsymbol{z}_q^{\mathrm{HF}}$. The quantized latents are then reconstructed to time domain. The reconstructed LF and HF time series are denoted as $\hat{\boldsymbol{x}}^{\mathrm{LF}}$ and $\hat{\boldsymbol{x}}^{\mathrm{HF}}$, respectively.

Encoder and decoder

The architecture for the encoder and decoder is the same as in (Oord et al., 2017). The encoder consists of n downsampling convolutional blocks with 2D convolutions, batch normalization and leaky ReLU activation. The downsampling convolutional layers are implemented with kernel size (3,4), stride (1,2) and padding (1,1). Note that this only downsamples the temporal axis of the spectrograms. It is followed by a *residual network* (ResNet) (He et al., 2015) with m blocks. A residual network is a network containing a skip connection that perform the identity mapping. The input is then added to the output of the blocks inside the network. The ResNet blocks are implemented with 2D convolutions, batch normalization and leaky ReLU activation. After the encoder, the latent is downsampled with a rate of 2^n. The decoder is similar to the encoder. It has first m ResNet blocks, followed by n up-sampling convolutional layers. The up-sampling layers is implemented with transposed 2D convolutions with kernel size, stride and padding identical to the encoder.

VQ

The implementation of the VQ follows (P. Wang et al., 2022). It contains the K tokens with dimension d for doing vector-quantization. If the latent dimension from the encoder has dimension d_E and it does not match the dimension of the tokens, the VQ includes an input and output projection layer. The projections are implemented using a linear layer. The input projection has d_E incoming channels and d outgoing channels, while the output projection has the number of input and output channels reversed.



$\boldsymbol{x}$
STFT
$\boldsymbol{u}^{\mathrm{LF}}$
zero padding
E_{LF}
$\boldsymbol{z}^{\mathrm{LF}}$
Codebook $\mathcal{Z}_{\mathrm{LF}}$
$\underset{\boldsymbol{e}\in\mathcal{Z}_{\mathrm{LF}}}{\operatorname{argmin}} \left\|(\boldsymbol{z}^{\mathrm{LF}})_i - \boldsymbol{e}\right\|$
$\boldsymbol{z}_q^{\mathrm{LF}}$
D_{LF}
$\hat{\boldsymbol{u}}^{\mathrm{LF}}$
zero padding
ISTFT
$\hat{\boldsymbol{x}}^{\mathrm{LF}}$
$\boldsymbol{u}^{\mathrm{HF}}$
E_{HF}
$\boldsymbol{z}^{\mathrm{HF}}$
Codebook $\mathcal{Z}_{\mathrm{HF}}$
$\underset{\boldsymbol{e}\in\mathcal{Z}_{\mathrm{HF}}}{\operatorname{argmin}} \left\|(\boldsymbol{z}^{\mathrm{HF}})_i - \boldsymbol{e}\right\|$
$\boldsymbol{z}_q^{\mathrm{HF}}$
D_{HF}
$\hat{\boldsymbol{u}}^{\mathrm{HF}}$
ISTFT
$\hat{\boldsymbol{x}}^{\mathrm{HF}}$

Figure 8: Neural architecture for stage 1. An input signal is split into low and high frequencies using the short-time Fourier transform and zero-padding operations. Two VQ-VAEs are used to learn efficient representations of the low and high frequency latents.

2.5.2 U-Net

U-Net is a popular type of *convolutional neural network* (CNN) first introduced in (Ronneberger et al., 2015) for image segmentation. The name comes from its unique U-shaped architecture, where the input signal is first contracted and then expanded. This enables it to capture contextual information effectively while maintaining fine-grained details. The U-Net preforms several downsampling steps to compress the input signal, which increases the receptive field at downstream layers. At each successive down layer the number of feature channels is increased. After the down layers in the contracting part, the network preforms the same number of upsampling steps to arrive at the same dimension as the input signal. The U-Net has skip connections at each down and up-layer. Skip connections allows gradients to propagate easier during training, but it also can encourage reuse of features captured at different levels of abstraction. The U-Net is visualized in Figure 9.

Down- and upsampler

Between each layer in the contracting part of the U-Net, the signal is downsampled. The downsampler is a 1D convolutional layer with kernel size 4, stride 2 and padding 1. This downsampling is similar to the temporal downsampling in the VQ-VAE model. An input of length l will have length $\lfloor l/2 \rfloor$ after the downsampling layer. Similarly, between each layer in the expanding part, the signal is up-sampled. For the up-sampler, an up-sampling layer with scale factor 2 is used. This means that an input of length l will have length $2 \cdot l$ after the up-sampling layer. Applying a downsampling layer together with an up-sampling layer can create a shape mismatch. A signal of length l is downscaled to length $\lfloor l/2 \rfloor$ first and then upscaled to length $2\lfloor l/2 \rfloor$. Whenever l is odd, this does not equal l, so there is a mismatch. The shape mismatch problem is resolved by changing the upsampling procedure to include an extra up-scaling and interpolation layer whenever l is odd.

Residual network with linear attention

For the implementation used in this thesis, the U-Net processes the features at each layer using a ResNet (He et al., 2015) with linear attention (Katharopoulos et al., 2020). The residual network is composed of two ResNet blocks and linear attention. Each ResNet block contains two weight-standardized 1D-convolutions with group normalization (Qiao et al., 2019). After the two convolutional blocks, the signal is layer-normalized before inputting to the linear attention block.

Conditional arguments

There are two types of conditional arguments for the U-Net, scalar conditions and input conditions. The U-Net is used to predict the noise using the diffused latent. This can be written as $\hat{\boldsymbol{\epsilon}} = f_\theta(\boldsymbol{\epsilon}; \boldsymbol{z}_t, \boldsymbol{c})$, where $\boldsymbol{c}$ are the conditional arguments. For the LF diffusion model, the conditions are the time step and class label, $\boldsymbol{c} = (t, y)$. For the HF diffusion model, the conditions include the denoised LF latent, $\boldsymbol{c} = (t, y, \boldsymbol{z}_0^{\mathrm{LF}})$. Of these conditions, the diffusion time step and class label are scaler. Both of these are embedded to a vector and then concatenated to produce a single embedding. The diffusion time t is embedded using a learned sinusoidal positional embedding (Vaswani et al., 2017), while the classes have a learned embedding vector for each class. To guide the prediction from the U-Net, the scaler conditions are provided at every ResNet block, where it is used to compute a scale and shift vector for guiding the output after the group normalization layers. The input conditions for the HF diffusion model is concatenated with the input to the U-Net. This follows the work of (Ho, Saharia et al., 2021).

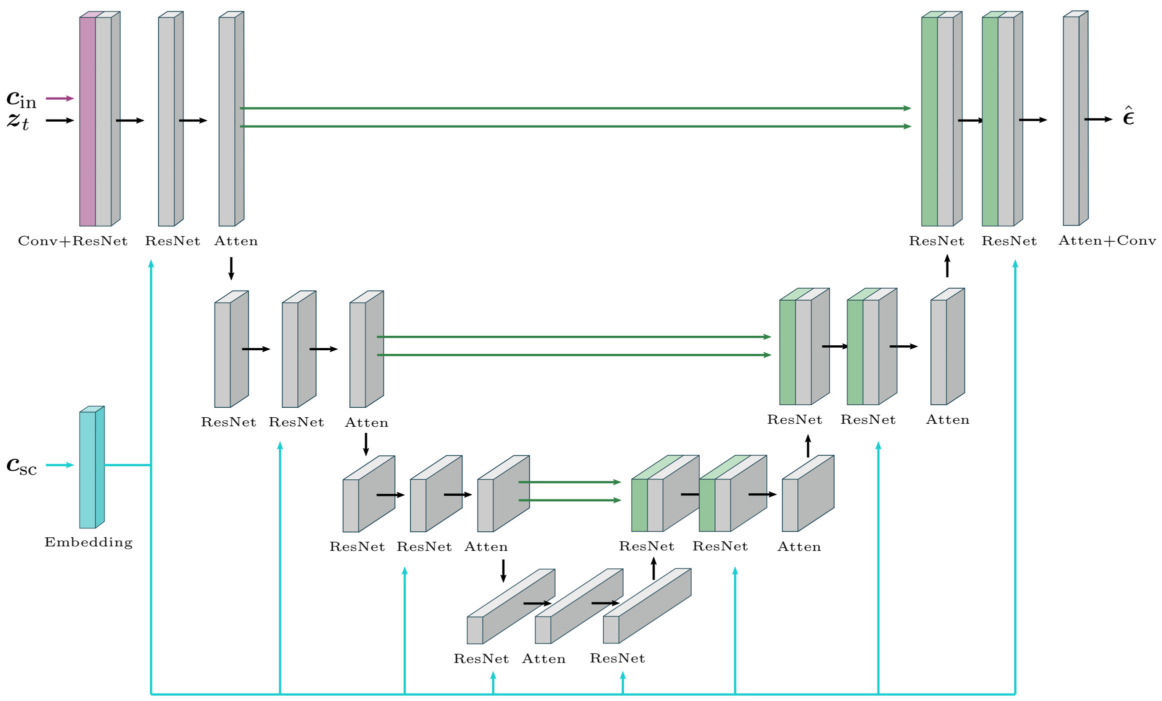


Figure 9: Overview of the U-Net architecture. The input signal $\boldsymbol{z}_t$ is first compressed using a series of downsampling layers, before it is scaled up with an equal amount of up-sampling layers. Each down and up-layer has two ResNet blocks and linear attention. Skip-connections are highlighted using the green arrows. Conditional scaler arguments ($\boldsymbol{c}_{\text{sc}}$) are provided at every level of the U-Net, and input conditions ($\boldsymbol{c}_{\text{in}}$) are concatenated along with the input signal.

2.5.3 Fully Convolutional Network

The *fully convolutional network* (FCN) (Z. Wang et al., 2016) is one of the strongest baselines for time series classification. It can effectively capture features and patterns of time series. The basic block of the FCN is a convolutional block containing three 1D convolutional layers, batch normalization, and ReLU activation. The final network is created by stacking three convolutional blocks after one another with 128, 256 and 128 channels, respectively. After the convolution blocks, the features are fed into a *global average pooling* (GAP) layer and a linear layer. The Softmax function is then applied to produce the final class predictions. The FCN is illustrated in Figure 10.

In this thesis, the FCN network is used for classification and feature representation. It is represented by p_η, where η is the network parameters. As a classifier, the predicted class probabilities are given as $\hat{y} = p_\eta(y \mid \boldsymbol{x})$. For evaluating generated time series, the FCN model is used for extracting useful representations of the time series of which it was trained on. This is used for the evaluation metrics.

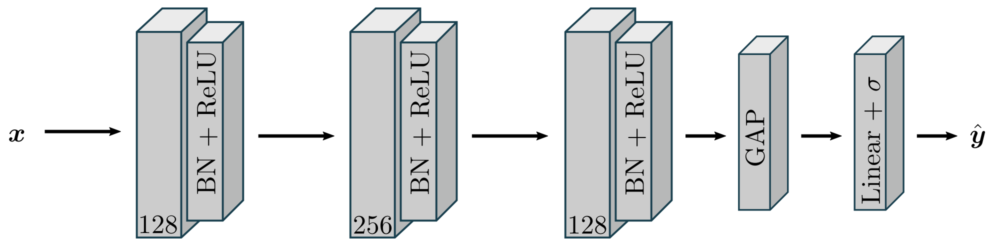


Figure 10: Fully Convolutional Network. An input signal $\boldsymbol{x}$ is fed through three convolutional blocks before predicting the class label. The convolutional blocks contain a 1D convolutions, batch normalization and ReLU activation. The convolutional blocks follows by global average pooling and a linear layer with Softmax activation. The Softmax function is represented using σ.

2.6 Evaluation metrics

Having ways to evaluate generative models is essential for measuring their performances. There have been many quantitative techniques suggested in the literature for analysing different machine learning algorithms. Classification, regression and clustering tasks all have established metrics, relying on measuring the models using the observed data. E.g. for classification it is common to evaluate using the accuracy or confusion matrix and for regression the mean squared error or coefficient of determination can be used. Generative modelling is a fundamentally different task. It tries to learn the probability distribution of the observed data and then uses it to generate samples looking similar to what it has learned. It is however hard to define good performance measures for generative models because generation is fundamentally a qualitative task.

2.6.1 Visual inspection

According to (Lee et al., 2023) there has been a lack of proper evaluation protocols to measure quality of generated samples in the TSG literature. Traditionally, qualitative protocols have involved visual inspection. One way is to plot the run charts of the synthetic time series and comparing them with real time series. This allows for a visual impression for assessing sample quality, variability and other qualitative aspects. Another way to visually compare feature representations is by using *principal component analyses* (PCA) or *t-distributed stochastic neighbor embeddings* (t-SNE) created using synthetic and real samples (Brophy et al., 2021; Yoon et al., 2019; Li et al., 2022). However, these methods cannot be reduced to scaler metrics. Lee et al., 2023 proposes to use three established metrics from the image generation literature. These are the *inception score* (IS), the *Fréchet inception distance* (FID) and *classification accuracy score* (CAS). For evaluating TimeVQVDM, these tree metrics is used in combination with visual inspection of the generated samples.

2.6.2 Inception Score (IS)

The IS measures how realistic the generative model outputs are by measuring the diversity and sharpness of generated samples. It was proposed in (Salimans, I. Goodfellow et al., 2016) and is motivated by two desiderata for generative models, good mode coverage and high quality. Diversity is the variability of the samples while sharpness is the clarity. The inception score is defined as

$$d_{\mathrm{IS}}(\eta) := \exp\left\{ \mathbb{E}_{\boldsymbol{x}\sim p_\theta}\left[D_{\mathrm{KL}}(p_\eta(y \mid \boldsymbol{x}) \parallel p_\eta(y))\right]\right\}, \tag{37}$$

where $p_\eta(y \mid \boldsymbol{x})$ is the distribution from a classifier with parameters η and $p_\eta(y)$ is the marginal distribution of the classes from the classifier. The probability of each class from of the classifier is continuous, rather than discrete. The KL divergence in Equation (37) can be calculated as

$$D_{\mathrm{KL}}(p_\eta(y \mid \boldsymbol{x}) \parallel p_\eta(y)) = \sum_{i=1}^{n} p_\eta(y \mid \boldsymbol{x}_i)\left(\log p_\eta(y \mid \boldsymbol{x}_i) - p_\eta(y)\right),$$

where n is the number of generated samples.

A high IS indicates that the generative model has high performance. It is achieved if the entropy of the distribution of classes given the generated samples is minimized and predictions of the classifier are evenly distributed according to the ground truth. In other words, the classifier should confidently predict a single class for each time series and the distribution of classes should be representative. Unlike the computer vision field, there are no clearly established classification models for time series. To calculate the representation vector $p_\eta(y \mid \boldsymbol{x})$ for IS, the FCN classifiers (Z. Wang et al., 2016) can be used. This method was first introduced in (K. E. Smith and A. O. Smith, 2020), but their FCN models are not available. Instead, the pre-trained classifiers from (Lee et al., 2023) are used.

IS has some limitations, such as inability to capture feature relevance present in the samples and intra-class diversity. Additionally, if the classifier is overfitted to the true data, then there could be a bias towards it. Memorising and replicating the training data could then lead to high IS, while generating something not present in the training data could lead to low IS. According to (Borji, 2021), the IS has to some degree been superseded by the FID measure.

2.6.3 Fréchet Inception Distance (FID)

The Fréchet inception distance is a common measure for assessing the quality of samples created by generative models. While the inception score only evaluates the distribution of generated samples, FID compares the distribution of generates samples with a set of samples from the ground truth. It was introduced in (Heusel et al., 2017) and is the current standard for assessing the quality of generative models (Borji, 2021). For any two probability distributions q_1, q_2 over $\mathbb{R}^l$ having finite mean and variance, the Fréchet inception distance is defined as the Wasserstein-2 metric between q_1 and q_2,

$$d_{\mathrm{FID}}(q_1, q_2) := \left(\inf_{\gamma \in \Gamma(q_1, q_2)} \int_{\mathbb{R}^l \times \mathbb{R}^l} \|x - y\|^2 \, \mathrm{d}\gamma(x, y) \right)^{1/2},$$

where $\Gamma(q_1, q_2)$ is the set of all couplings on $\mathbb{R}^l \times \mathbb{R}^l$ with marginals q_1 and q_2 on the first and second factors. According to Dowson and Landau, 1982, it is explicitly solvable for two multivariate Gaussian distributions, $q_1 = \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $q_2 = \mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$, as

$$d^2_{\mathrm{FID}}(q_1, q_2) = \|\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\|^2 + \mathrm{tr}\left(\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2 - 2\left(\boldsymbol{\Sigma}_1 \boldsymbol{\Sigma}_2\right)^{1/2} \right), \tag{38}$$

where $\mathrm{tr}(\cdot)$ is the trace.

Rather than directly comparing each part of the samples, such as every pixel for images or each time step in a time series, the FID compares the mean and standard deviations of the distribution of samples compared with the ground truth. The intuition behind FID comes from the optimal transport problem, where we wish to morph one of the probability distributions into the other by moving the probability masses around. The cost of movement is equal to the Euclidean distance between the two points. As a result, this metric mimics human perception of similarity.

The feature representation for calculating FID follows (Lee et al., 2023). Instead of fitting two Gaussian distributions to a generated distribution of time series and real time series, the two distributions are fitted to the feature representation from the FCN classifiers. The representation vector is extracted right after the GAP layer.

2.6.4 Classification Accuracy Score (CAS)

A separate classifier can be used to evaluate the quality of synthetic data generated using class-conditional sampling. Assume that the synthetic data is given as a set of data points $\{(\tilde{\boldsymbol{x}}_i, \tilde{y}_i)\}_{i=1}^n$, similarly to the ground truth $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$. A classification network $p_\eta(y \mid \boldsymbol{x})$ can then be trained on the synthetic data, and validated on test data. Then it can be compared with a reference classification network trained on the real training data. The classification accuracy is used for the measure of relative performance of each network. If the synthetic data resembles the training data, then the classification accuracy on the test set will be similar to the classification accuracy when training the classifier on real data. This procedure allows us to test if class-conditional generation works properly and indicate that the synthetic data has proper alignment of samples/class pairs.

The pre-trained FCN models in (Lee et al., 2023) are used for the reference classifiers trained on the real data. An identical network with the same hyperparameters is then used to fit a new FCN model on the synthetic conditionally sampled data and to compare with the reference model. The two CAS are reported as *train on real; test on real* (TRTR) and *train on synthetic; test on real* (TSTR), respectively. A high TSTR accuracy indicates that the model was able to learn the conditional distribution of the data.

3 Experimental Setup

3.1 Proposed model

The proposed model is called TimeVQVDM. It is illustrated in Figure 11. It is a generative model for time series which splits the signals into low and high frequencies, and models them in the time-frequency domain using vector quantization. TimeVQVDM uses the two-stage modelling approach in TimeVQVAE (Lee et al., 2023). Stage one involves finding efficient latent representations for the low and high frequencies using VQ-VAEs (Oord et al., 2017). Stage two is modelling of the latent distributions using a prior model. In contrast to TimeVQVAE, which uses transformer models for the prior, TimeVQVDM uses diffusion models. The code is provided in https://github.com/martintufte/TimeVQ-VDM.

Instead of learning the marginal distribution of the time series, $q(\boldsymbol{x})$, it learns the joint distribution of low and high frequencies, $q(\boldsymbol{x}^{\mathrm{LF}}, \boldsymbol{x}^{\mathrm{HF}})$. Sampling from TimeVQVDM is preformed by sampling from the learned distribution $p_\theta(\boldsymbol{x}^{\mathrm{LF}}, \boldsymbol{x}^{\mathrm{HF}})$ and adding the generated low and high frequencies together. The prior diffusion models are trained using classifier-free guidance, allowing TimeVQVDM for conditional sampling of the form $p_\theta(\boldsymbol{x} \mid \boldsymbol{c})$, where $\boldsymbol{c}$ is a conditional argument.

Stage 1: VQ-VAE modelling

The input time series $\boldsymbol{x}$ is split into low and high frequency spectrograms using the STFT and zero-padding operations. The spectrograms are then modelled using VQ-VAEs, similar to (Lee et al., 2023). The encoder, decoder and codebook of the VQ-VAEs are denoted as E_{LF}, D_{LF} and $\mathcal{Z}_{\mathrm{LF}}$ for low frequencies and E_{HF}, D_{HF} and $\mathcal{Z}_{\mathrm{HF}}$ for high frequencies. The two VQ-VAE models are independently parameterized, such that the learned latent representation for LF can capture other features than the latent representation for HF. Low frequencies captures overall structure of the time series, while high frequencies corresponds to finer details and noise. From Equation (35), the low and high frequency spectrograms in time-frequency domain are given as

$$\boldsymbol{u}^{\mathrm{LF}} = \mathcal{P}_{\mathrm{LF}}(\mathrm{STFT}(\boldsymbol{x})), \qquad \boldsymbol{u}^{\mathrm{HF}} = \mathcal{P}_{\mathrm{HF}}(\mathrm{STFT}(\boldsymbol{x}), \tag{39}$$

where the zero-padding operations are defined in Equation (34). Using the encoders in the VQ-VAE models, the continuous latent representations for LF and HF are given as

$$\boldsymbol{z}^{\mathrm{LF}} = E_{\mathrm{LF}}(\boldsymbol{u}^{\mathrm{LF}}), \qquad \boldsymbol{z}^{\mathrm{HF}} = E_{\mathrm{HF}}(\boldsymbol{u}^{\mathrm{HF}}), \tag{40}$$

and their quantized latent representations are found with Equation (12),

$$(\boldsymbol{z}_q^{\mathrm{LF}})_i = \underset{\boldsymbol{e}_k \in \mathcal{Z}_{\mathrm{LF}}}{\operatorname{argmin}} \left\|(\boldsymbol{z}^{\mathrm{LF}})_i - \boldsymbol{e}_k\right\|_2, \qquad (\boldsymbol{z}_q^{\mathrm{HF}})_i = \underset{\boldsymbol{e}_k \in \mathcal{Z}_{\mathrm{HF}}}{\operatorname{argmin}} \left\|(\boldsymbol{z}^{\mathrm{HF}})_i - \boldsymbol{e}_k\right\|_2. \tag{41}$$

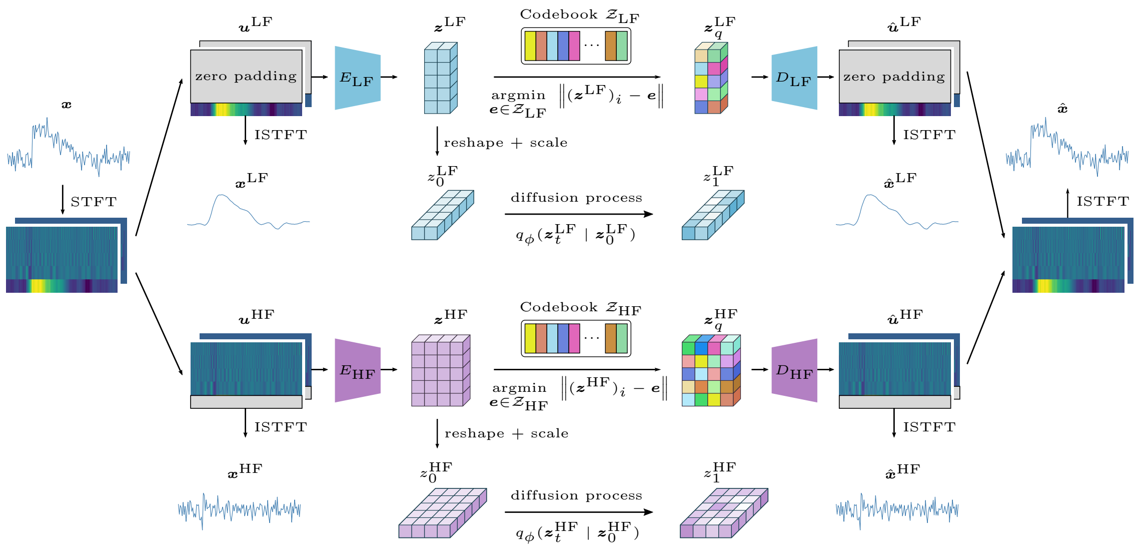
The reconstructed low and high frequency spectrograms follows from decoding the quantized latents using the decoders in the VQ-VAE models,

$$\hat{\boldsymbol{u}}^{\mathrm{LF}} = \mathcal{P}_{\mathrm{LF}}(D_{\mathrm{LF}}(\boldsymbol{z}_q^{\mathrm{LF}})), \qquad \hat{\boldsymbol{u}}^{\mathrm{HF}} = \mathcal{P}_{\mathrm{HF}}(D_{\mathrm{HF}}(\boldsymbol{z}_q^{\mathrm{HF}})). \tag{42}$$

Note that the reconstructed spectrograms from the decoders are zero-padded, which prevents the reconstructed low frequency spectrogram to contain high frequencies and vice versa. The reconstructed low and high frequency signals are then given by using the inverse STFT in Equation (33), such that

$$\hat{\boldsymbol{x}}^{\mathrm{LF}} = \mathrm{ISTFT}(\hat{\boldsymbol{u}}^{\mathrm{LF}}), \qquad \hat{\boldsymbol{x}}^{\mathrm{HF}} = \mathrm{ISTFT}(\hat{\boldsymbol{u}}^{\mathrm{HF}}). \tag{43}$$

Finally, the reconstructed time series is the reconstructed low and high frequency time series added together, $\hat{\boldsymbol{x}} = \hat{\boldsymbol{x}}^{\mathrm{LF}} + \hat{\boldsymbol{x}}^{\mathrm{HF}}$.



$\boldsymbol{x}$
STFT
$\boldsymbol{u}^{\mathrm{LF}}$
zero padding
ISTFT
$\boldsymbol{x}^{\mathrm{LF}}$
E_{LF}
$\boldsymbol{z}^{\mathrm{LF}}$
reshape + scale
z_0^{LF}
Codebook $\mathcal{Z}_{\mathrm{LF}}$
$\underset{\boldsymbol{e}\in\mathcal{Z}_{\mathrm{LF}}}{\operatorname{argmin}} \left\|(\boldsymbol{z}^{\mathrm{LF}})_i - \boldsymbol{e}\right\|$
diffusion process
$q_\phi(\boldsymbol{z}_t^{\mathrm{LF}} \mid \boldsymbol{z}_0^{\mathrm{LF}})$
$\boldsymbol{z}_q^{\mathrm{LF}}$
z_1^{LF}
D_{LF}
$\hat{\boldsymbol{u}}^{\mathrm{LF}}$
zero padding
ISTFT
$\hat{\boldsymbol{x}}^{\mathrm{LF}}$
$\hat{\boldsymbol{x}}$
ISTFT
$\boldsymbol{u}^{\mathrm{HF}}$
ISTFT
$\boldsymbol{x}^{\mathrm{HF}}$
E_{HF}
$\boldsymbol{z}^{\mathrm{HF}}$
reshape + scale
z_0^{HF}
Codebook $\mathcal{Z}_{\mathrm{HF}}$
$\underset{\boldsymbol{e}\in\mathcal{Z}_{\mathrm{HF}}}{\operatorname{argmin}} \left\|(\boldsymbol{z}^{\mathrm{HF}})_i - \boldsymbol{e}\right\|$
diffusion process
$q_\phi(\boldsymbol{z}_t^{\mathrm{HF}} \mid \boldsymbol{z}_0^{\mathrm{HF}})$
$\boldsymbol{z}_q^{\mathrm{HF}}$
z_1^{HF}
D_{HF}
$\hat{\boldsymbol{u}}^{\mathrm{HF}}$
ISTFT
$\hat{\boldsymbol{x}}^{\mathrm{HF}}$

Figure 11: TimeVQVDM architecture. Time series are split into low and high frequency spectrograms, which are encoded to latent space using VQ-VAEs. The latent representations are then modelled using variational diffusion models. Sampling from TimeVQVDM is done by sampling from the low frequency diffusion model and then sampling from the high frequency diffusion model conditioned on the low frequencies.

Stage 2: Diffusion prior

Variational diffusion models (VDM) (Diederik P. Kingma et al., 2021) are used to model the continuous latent representations after encoding to latent space. There are two diffusion processes, one for the low frequency latent and the other for high frequency latent. For the low frequencies, the forward diffusion process is given as

$$q_\phi(\boldsymbol{z}_t^{\mathrm{LF}} \mid \boldsymbol{z}_0^{\mathrm{LF}}) = \mathcal{N}\left(\boldsymbol{z}_t^{\mathrm{LF}};\ \alpha_t \boldsymbol{z}_0^{\mathrm{LF}},\ \sigma_t^2 \mathbf{I}\right),$$

which continuously destroys structure in the latent. The diffusion starts at $t = 0$ and ends at $t = 1$. Here $\boldsymbol{z}_0^{\mathrm{LF}}$ is the non-diffused low frequency latent, equal to $\boldsymbol{z}^{\mathrm{LF}} = E_{\mathrm{LF}}(\boldsymbol{u}^{\mathrm{LF}})$. In practice, $\boldsymbol{z}_0^{\mathrm{LF}}$ is implemented as a reshaped and scaled version of $\boldsymbol{z}^{\mathrm{LF}}$. The noise schedule is variance conserving and given by the two function $\alpha_t = \cos(\pi/2 \cdot t)$ and $\sigma_t = \sin(\pi/2 \cdot t)$ in Equation (18). The fully diffused latent has limiting distribution $\boldsymbol{z}_1^{\mathrm{LF}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. A similar diffusion process is defined for the high frequency latent representation.

The latents of the diffusion model are denoted as $\boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{LF}}$ for LF and $\boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{HF}}$ for HF, where $\boldsymbol{\tau} = [0, 1]$ is the continuous time where the diffusion process is defined. The diffusion models are trained to reverse the noising processes with 1D denoising U-Nets (Ronneberger et al., 2015). The U-Nets are trained with conditional arguments, which allows a cascading sampling from TimeVQVDM by first sampling the low frequencies, and then sampling the high frequencies later. The sampling procedure follows from discretizing $\boldsymbol{\tau}$ as $\{\tau_i\}_{i=0}^{T}$ with $\tau_i = i/T$. Sampling the low frequencies are then performed by starting with noise $\boldsymbol{z}_{\tau_T}^{\mathrm{LF}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and performing T denoising steps. Each denoising step is given in Equation (19) as

$$\boldsymbol{z}_{\tau_{i-1}}^{\mathrm{LF}} \sim q_\phi(\boldsymbol{z}_{\tau_{i-1}}^{\mathrm{LF}} \mid \boldsymbol{z}_{\tau_i}^{\mathrm{LF}}, f_\theta(\boldsymbol{z}_0^{\mathrm{LF}}; \boldsymbol{z}_{\tau_i}^{\mathrm{LF}}, \boldsymbol{c})),$$

where $f_\theta(\boldsymbol{z}_0^{\mathrm{LF}}; \boldsymbol{z}_{\tau_i}^{\mathrm{LF}}, \boldsymbol{c})$ is the diffusion model's prediction of $\boldsymbol{z}_0^{\mathrm{LF}}$ when inputting the noisy latent and a conditional argument $\boldsymbol{c}$. After sampling the low frequency latent, a similar procedure is used to sample the high frequencies. For the high frequency diffusion model, the conditional arguments includes the low frequencies. This makes the sampling process align the generated low and high frequency components.

3.2 Loss function

TimeVQVDM is trained by finding the parameters that maximize the likelihood of the training data. In practice, the likelihood is maximized by using the evidence lower bound in Equation (8). Before deriving the loss function, two assumptions for the inference model and the generative model are made:

1. The inference models for low and high frequencies are independent.
2. The generative model for high frequencies is conditioned on the low frequencies.

The first assumption states that the learned latent representations for the time series is split into finding separate representations for low and high frequencies. The second assumption involves the generation process, where sampling is from the joint distribution $p_\theta(\boldsymbol{x}^{\mathrm{LF}}, \boldsymbol{x}^{\mathrm{HF}})$ by first sampling $p_\theta(\boldsymbol{x}^{\mathrm{LF}})$ and then sampling from $p_\theta(\boldsymbol{x}^{\mathrm{HF}} \mid \boldsymbol{x}^{\mathrm{LF}})$. The generated time series $\boldsymbol{x} \sim p_\theta(\boldsymbol{x})$ is found from adding the low and high frequencies,

$$\boldsymbol{x} = \boldsymbol{x}^{\mathrm{LF}} + \boldsymbol{x}^{\mathrm{HF}}, \qquad \boldsymbol{x}^{\mathrm{LF}}, \boldsymbol{x}^{\mathrm{HF}} \sim p_\theta(\boldsymbol{x}^{\mathrm{LF}}, \boldsymbol{x}^{\mathrm{HF}}).$$

From the definition of the ELBO in Equation (8), recall that the loss function is given by

$$L_{\mathrm{ELBO}}(\theta, \phi \mid \boldsymbol{x}) = \mathbb{E}_{\boldsymbol{z} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[-\log \frac{p_\theta(\boldsymbol{z}, \boldsymbol{x})}{q_\phi(\boldsymbol{z} \mid \boldsymbol{x})}\right],$$

where $p_\theta(\boldsymbol{z}, \boldsymbol{x})$ is the generative model of the signal and the latents and $q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ is the inference model over the latents. The signal consists of low and high frequencies, and we assume that the latents consist of four components corresponding to the latents over the low and high frequency diffusion processes and the vector-quantized representations. This is written as

$$\boldsymbol{x} = (\boldsymbol{x}^{\mathrm{LF}}, \boldsymbol{x}^{\mathrm{HF}}), \qquad \boldsymbol{z} = \left(\boldsymbol{z}_q^{\mathrm{LF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{LF}}, \boldsymbol{z}_q^{\mathrm{HF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{HF}}\right),$$

where $\boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{LF}}$ and $\boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{HF}}$ are the latents in the diffusion models and $\boldsymbol{z}_q^{\mathrm{LF}}$ and $\boldsymbol{z}_q^{\mathrm{HF}}$ are the latent representation in the vector-quantization models. From assumption one, it follows that the joint distribution in the inference model for TimeVQVDM can be split up as

$$\begin{aligned} q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) &= q_\phi(\boldsymbol{z}_q^{\mathrm{LF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{LF}}, \boldsymbol{z}_q^{\mathrm{HF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{HF}} \mid \boldsymbol{x}^{\mathrm{LF}}, \boldsymbol{x}^{\mathrm{HF}}) \\ &= q_\phi(\boldsymbol{z}_q^{\mathrm{LF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{LF}} \mid \boldsymbol{x}^{\mathrm{LF}}, \boldsymbol{x}^{\mathrm{HF}})\; q_\phi(\boldsymbol{z}_q^{\mathrm{HF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{HF}} \mid \boldsymbol{x}^{\mathrm{LF}}, \boldsymbol{x}^{\mathrm{HF}}) \\ &= q_\phi(\boldsymbol{z}_q^{\mathrm{LF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{LF}} \mid \boldsymbol{x}^{\mathrm{LF}})\; q_\phi(\boldsymbol{z}_q^{\mathrm{HF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{HF}} \mid \boldsymbol{x}^{\mathrm{HF}}). \end{aligned} \tag{44}$$

Similarly, from assumption two, it follows that the generative model for TimeVQVDM can be split up as a generative model over the low frequencies and a conditional distribution over the high frequencies,

$$\begin{aligned} p_\theta(\boldsymbol{z}, \boldsymbol{x}) &= p_\theta(\boldsymbol{z}_q^{\mathrm{LF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{LF}}, \boldsymbol{z}_q^{\mathrm{HF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{HF}}, \boldsymbol{x}^{\mathrm{LF}}, \boldsymbol{x}^{\mathrm{HF}}) \\ &= p_\theta(\boldsymbol{z}_q^{\mathrm{LF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{LF}}, \boldsymbol{x}^{\mathrm{LF}}),\; p_\theta(\boldsymbol{z}_q^{\mathrm{HF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{HF}} \boldsymbol{x}^{\mathrm{HF}} \mid \boldsymbol{z}_q^{\mathrm{LF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{LF}}, \boldsymbol{x}^{\mathrm{LF}}). \end{aligned} \tag{45}$$

Inserting Equation (44) and (45) into (8), the loss function can be formulated as

$$\begin{aligned} &L_{\mathrm{ELBO}}(\theta, \phi \mid \boldsymbol{x}) \\ &= \underbrace{\mathbb{E}_{q_\phi}\left[-\log \frac{p_\theta(\boldsymbol{z}_q^{\mathrm{LF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{LF}}, \boldsymbol{x}^{\mathrm{LF}})}{q_\phi(\boldsymbol{z}_q^{\mathrm{LF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{LF}} \mid \boldsymbol{x}^{\mathrm{LF}})}\right]}_{=:L_{\mathrm{ELBO}}^{\mathrm{LF}}(\theta,\phi|\boldsymbol{x})} + \underbrace{\mathbb{E}_{q_\phi}\left[-\log \frac{p_\theta(\boldsymbol{z}_q^{\mathrm{HF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{HF}}, \boldsymbol{x}^{\mathrm{HF}} \mid \boldsymbol{z}_q^{\mathrm{LF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{LF}}, \boldsymbol{x}^{\mathrm{LF}})}{q_\phi(\boldsymbol{z}_q^{\mathrm{HF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{HF}} \mid \boldsymbol{x}^{\mathrm{HF}})}\right]}_{=:L_{\mathrm{ELBO}}^{\mathrm{HF}}(\theta,\phi|\boldsymbol{x})}. \end{aligned} \tag{46}$$

The loss in Equation (46) is a sum over a loss term for low frequencies and a loss term over high frequencies. The only difference between the low and high frequencies terms are the additional conditional arguments passed to the high frequency generative model.

Low frequency terms

The loss is derived assuming that the diffusion process is discretized as $\boldsymbol{\tau} = \{\tau_i\}_{t=0}^{T}$, where $\tau_i = i/T$. The inference model for low frequencies can then be written as

$$q_\phi(\boldsymbol{z}_q^{\text{LF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\text{LF}} \mid \boldsymbol{x}^{\text{LF}}) = q_\phi(\boldsymbol{z}_q^{\text{LF}} \mid \boldsymbol{z}_0^{\text{LF}})\ q_\phi(\boldsymbol{z}_0^{\text{LF}} \mid \boldsymbol{x}^{\text{LF}})\ \prod_{i=1}^{T} q_\phi(\boldsymbol{z}_{\tau_i}^{\text{LF}} \mid \boldsymbol{z}_{\tau_{i-1}}^{\text{LF}}), \tag{47}$$

where $q_\phi(\boldsymbol{z}_0^{\text{LF}} \mid \boldsymbol{x})$ is the inference model of the encoding of low frequencies to latent space and $q_\phi(\boldsymbol{z}_q^{\text{LF}} \mid \boldsymbol{z}_0^{\text{LF}})$ is the inference model for the quantization process of low frequencies. The product in Equation (47) is from the discretized diffusion process for low frequencies. The generative model for the low frequencies can be written as

$$p_\theta(\boldsymbol{z}_q^{\text{LF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\text{LF}}, \boldsymbol{x}^{\text{LF}}) = p_\theta(\boldsymbol{x}^{\text{LF}} \mid \boldsymbol{z}_q^{\text{LF}})\ p_\theta(\boldsymbol{z}_q^{\text{LF}} \mid \boldsymbol{z}_0^{\text{LF}})\ p(\boldsymbol{z}_1^{\text{LF}})\ \prod_{i=1}^{T} p_\theta(\boldsymbol{z}_{\tau_{i-1}}^{\text{LF}} \mid \boldsymbol{z}_{\tau_i}^{\text{LF}}), \tag{48}$$

where $p_\theta(\boldsymbol{x}^{\text{LF}} \mid \boldsymbol{z}_q^{\text{LF}})$ is the reconstruction in sample space from the quantized low frequency latent using the LF decoder and $p_\theta(\boldsymbol{z}_q^{\text{LF}} \mid \boldsymbol{z}_0^{\text{LF}})$ is the quantization process. The product in Equation (48) is from the denoising process of the diffusion model. A detailed derivation of the ELBO for low frequencies is derived in appendix C. Taking the limit as $T \to \infty$ for the discretization of time in the diffusion process, it is shown that the low frequency terms consists of four parts,

$$L_{\text{ELBO}}^{\text{LF}}(\theta, \phi \mid \boldsymbol{x}) = L_{\text{prior}}^{\text{LF}} + L_{\text{diffusion}}^{\text{LF}} + L_{\text{codebook}}^{\text{LF}} + L_{\text{reconstruct}}^{\text{LF}}, \tag{49}$$

corresponding to the steps needed to sample from the model. The first is to draw a diffused latent from the prior distribution. The next is to denoise the latent using the diffusion model. Then, the latent is quantized using the codebook and reconstructed to sample space. Appendix C further derives the four components as

$$\begin{aligned}
L_{\text{prior}}^{\text{LF}} &= D_{\text{KL}}(q(\boldsymbol{z}_1^{\text{LF}} \mid \boldsymbol{x}^{\text{LF}}) \parallel p(\boldsymbol{z}_1^{\text{LF}})),\\
L_{\text{diffusion}}^{\text{LF}} &= -\frac{1}{2}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I}),\ t\sim \text{U}(0,1)}\left[\frac{\text{d}}{\text{d}t}(\lambda_t)\left\|\boldsymbol{\epsilon} - \hat{\boldsymbol{\epsilon}}^{\text{LF}}\right\|_2^2\right],\\
L_{\text{codebook}}^{\text{LF}} &= \left\|\text{sg}\left[\boldsymbol{z}_0^{\text{LF}}\right] - \hat{\boldsymbol{z}}_q^{\text{LF}}\right\|_2^2 + \beta\left\|\boldsymbol{z}_0^{\text{LF}} - \text{sg}\left[\hat{\boldsymbol{z}}_q^{\text{LF}}\right]\right\|_2^2,\\
L_{\text{reconstruct}}^{\text{LF}} &= \left\|\boldsymbol{u}^{\text{LF}} - \hat{\boldsymbol{u}}^{\text{LF}}\right\|_2^2 + \left\|\boldsymbol{x}^{\text{LF}} - \hat{\boldsymbol{x}}^{\text{LF}}\right\|_2^2,
\end{aligned}$$

where $\hat{\boldsymbol{\epsilon}}^{\text{LF}} = f_\theta(\boldsymbol{\epsilon}; \boldsymbol{z}_t^{\text{LF}})$ is the predicted noise added to the LF latent using the diffusion model. In the codebook loss $\hat{\boldsymbol{z}}_q^{\text{LF}}$ is the quantized version of $\boldsymbol{z}_0^{\text{LF}}$ using the LF codebook $\mathcal{Z}_{\text{LF}}$. For the reconstruction loss, $\boldsymbol{u}^{\text{LF}} = \mathcal{P}_{\text{LF}}(\text{STFT}(\boldsymbol{x}))$ and $\hat{\boldsymbol{u}}^{\text{LF}} = \mathcal{P}_{\text{LF}}(D_{\text{LF}}(\boldsymbol{z}_q^{\text{LF}}))$ are in the time-frequency domain, and $\hat{\boldsymbol{x}}^{\text{LF}} = \text{ISTFT}(\hat{\boldsymbol{u}}^{\text{LF}})$ and $\boldsymbol{x}^{\text{LF}} = \text{ISTFT}(\boldsymbol{u}^{\text{LF}})$ are in the time domain.

The first term of Equation (49) is the prior loss, equaling to the KL divergence between the completely diffused low frequency latent and the diffusion prior. Since the diffusion prior does not contain any trainable parameters, this term is constant and can be omitted during training. The second term is a Monte Carlo estimate of the diffusion loss, equaling to a weighted mean squared error of predicting the noise added to the low frequency latent. The diffusion loss is similar to the one presented in (Diederik P. Kingma et al., 2021). The third term is the codebook loss, which is equal to the sum of two loss terms. The codebook loss is equal to the original codebook loss given in (Oord et al., 2017). Here sg[$\cdot$] denotes the stop-gradient operator, which stops the gradients from accumulating further backwards during backpropagation. The second term of $L_{\text{codebook}}^{\text{LF}}$ is the commitment loss, weighted by a parameter β. This term ensures that the variance of the tokens in the codebook don't explode. The forth term is the reconstruction loss, which reconstructs the low frequency sample from the latent space. The reconstruction loss is calculated in both time-frequency and time domain, following the work of (Défossez et al., 2022) and (Lee et al., 2023).

High frequency terms

The derivation for the high frequencies is similar to the derivation for low frequencies. The inference and generative model for HF are

$$q_\phi(\boldsymbol{z}_q^{\mathrm{HF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{HF}} \mid \boldsymbol{x}^{\mathrm{HF}}) = q_\phi(\boldsymbol{z}_q^{\mathrm{HF}} \mid \boldsymbol{z}_0^{\mathrm{HF}})\ q_\phi(\boldsymbol{z}_0^{\mathrm{HF}} \mid \boldsymbol{x}^{\mathrm{HF}})\ \prod_{i=1}^{T} q_\phi(\boldsymbol{z}_{\tau_i}^{\mathrm{HF}} \mid \boldsymbol{z}_{\tau_{i-1}}^{\mathrm{HF}}), \tag{50}$$

$$p_\theta(\boldsymbol{z}_q^{\mathrm{HF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{HF}}, \boldsymbol{x}^{\mathrm{HF}} \mid \boldsymbol{z}_q^{\mathrm{LF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{LF}}, \boldsymbol{x}^{\mathrm{LF}}) = p_\theta(\boldsymbol{x}^{\mathrm{HF}} \mid \boldsymbol{z}_q^{\mathrm{HF}})\ p_\theta(\boldsymbol{z}_q^{\mathrm{HF}} \mid \boldsymbol{z}_0^{\mathrm{HF}})\ p(\boldsymbol{z}_1^{\mathrm{HF}})\ \prod_{i=1}^{T} p_\theta(\boldsymbol{z}_{\tau_{i-1}}^{\mathrm{HF}} \mid \boldsymbol{z}_{\tau_i}^{\mathrm{HF}}, \boldsymbol{z}_0^{\mathrm{LF}}), \tag{51}$$

The only difference to the low frequencies is that the diffusion model for HF is conditioned on the low frequency latent. A similar derivation for the high frequency components shows that

$$L_{\mathrm{ELBO}}^{\mathrm{HF}}(\theta, \phi \mid \boldsymbol{x}) = L_{\mathrm{prior}}^{\mathrm{HF}} + L_{\mathrm{diffusion}}^{\mathrm{HF}} + L_{\mathrm{codebook}}^{\mathrm{HF}} + L_{\mathrm{reconstruct}}^{\mathrm{HF}}, \tag{52}$$

with

$$\begin{aligned}
L_{\mathrm{prior}}^{\mathrm{HF}} &= D_{\mathrm{KL}}(q(\boldsymbol{z}_1^{\mathrm{HF}} \mid \boldsymbol{x}^{\mathrm{HF}}) \parallel p(\boldsymbol{z}_1^{\mathrm{HF}})),\\
L_{\mathrm{diffusion}}^{\mathrm{HF}} &= -\frac{1}{2}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I}),\ t\sim\mathrm{U}(0,1)}\left[\frac{\mathrm{d}}{\mathrm{d}t}(\lambda_t)\left\|\boldsymbol{\epsilon} - \hat{\boldsymbol{\epsilon}}^{\mathrm{HF}}\right\|_2^2\right],\\
L_{\mathrm{codebook}}^{\mathrm{HF}} &= \left\|\mathrm{sg}\left[\boldsymbol{z}_0^{\mathrm{HF}}\right] - \hat{\boldsymbol{z}}_q^{\mathrm{HF}}\right\|_2^2 + \beta\left\|\boldsymbol{z}_0^{\mathrm{HF}} - \mathrm{sg}\left[\hat{\boldsymbol{z}}_q^{\mathrm{HF}}\right]\right\|_2^2,\\
L_{\mathrm{reconstruct}}^{\mathrm{HF}} &= \left\|\boldsymbol{u}^{\mathrm{HF}} - \hat{\boldsymbol{u}}^{\mathrm{HF}}\right\|_2^2 + \left\|\boldsymbol{x}^{\mathrm{HF}} - \hat{\boldsymbol{x}}^{\mathrm{HF}}\right\|_2^2.
\end{aligned}$$

As with the low frequencies, the prior loss is omitted during training. The reconstructed noise for the diffused high frequency latent is conditioned on the low frequency latent, $\hat{\boldsymbol{\epsilon}}^{\mathrm{HF}} = f_\theta(\boldsymbol{\epsilon}; \boldsymbol{z}_t^{\mathrm{HF}}, \boldsymbol{z}_0^{\mathrm{LF}})$.

Combining low and high frequency terms

Combining Equation (49) and (52) gives the ELBO for TimeVQVDM. Minimizing the ELBO with respect to θ then is the same as minimizing the combined diffusion, quantization and reconstruction loss with respect to θ. Using the two-stage modelling approach, the loss can be written as

$$\operatorname*{argmin}_\theta\ L_{\mathrm{ELBO}}(\theta, \phi \mid \boldsymbol{x}) = \operatorname*{argmin}_\theta\ \left(L_{\mathrm{diffusion}} + L_{\mathrm{VQ}}\right), \tag{53}$$

where the training loss in stage one is given as

$$L_{\mathrm{VQ}} = L_{\mathrm{codebook}}^{\mathrm{LF}} + L_{\mathrm{codebook}}^{\mathrm{HF}} + L_{\mathrm{reconstruct}}^{\mathrm{LF}} + L_{\mathrm{reconstruct}}^{\mathrm{HF}},$$

and the training loss in stage two is given as

$$L_{\mathrm{diffusion}} = L_{\mathrm{diffusion}}^{\mathrm{LF}} + L_{\mathrm{diffusion}}^{\mathrm{HF}}.$$

Optimizing the neural networks in two steps does not necessary lead to the joint optimum for θ. However, training the encoder, decoder and vector-quantizer first is practical as it allows for creating efficient representations for the latent space. Then the distribution of the latent representations are learned using the diffusion models.

3.3 Architecture and implementation details

The model is implemented with Pytorch in Python. In the following section, some of the variables are referred to with the shape of their tensors. E.g. an input batch $\boldsymbol{x}$ has shape (B, C, L), where B is the batch dimension, C is the channel dimension and L is the length dimension. For the spectrograms, such as $\boldsymbol{u} = STFT(\boldsymbol{x})$, the shape is (B, H, W, C), where H is the height of the spectrogram and W is the width.

Time-frequency modelling

The implementation of the STFT and ISTFT is using `torch.stft` and `torch.istft` with a window length of `n_fft` $= 8$, which has been experimentally found to lead to good performance (Lee et al., 2023). This is associated with the compression amount of the data. A lower n_{fft} leads to a shorter frequency axis and longer temporal axis of $\boldsymbol{u}$. In addition, the window function is constant with $w_k = 1$ for $1 \leq k \leq n_{\text{fft}}$ and the hop length is set to $h = 1$. This differs from TimeVQVAE, which used $h = 2$. Reducing the hop length creates smoother representations in the time-frequency domain since the Fourier transform is applied at double the amount of time steps. The STFT is implemented with `onesided = True`, which makes the frequency components be in the range $1 \leq \omega < \lfloor n_{\text{fft}} \rfloor / 2 + 1 = 5$. The shape of the spectrogram $\boldsymbol{u} = \text{STFT}(\boldsymbol{x})$ is (B, H, W, C), where $H = 5$ is the height, equal to the number of frequency components, $W = L + 1$ is the width of the spectrogram and $C = 2$ is the real and imaginary channels. When zero-padding the spectrogram using $\mathcal{P}_{\text{LF}}$ and $\mathcal{P}_{\text{HF}}$, the cutoff frequency is sat to $\omega_0 = 1$. This means that the low frequency spectrogram only includes the bottom frequency component, $\omega = 1$, while the high frequency spectrogram includes the rest of the frequency band, $\omega > 1$. The LF and HF spectrograms are then given as $\boldsymbol{u}^{\text{LF}} = \mathcal{P}_{\text{LF}}(\boldsymbol{u})$ and $\boldsymbol{u}^{\text{HF}} = \mathcal{P}_{\text{HF}}(\boldsymbol{u})$.

Encoder and decoder

The implementation of the encoders and decoders are from (Huang et al., 2021). The compression rate for the LF encoder is chosen such that the temporal axis of $\boldsymbol{u}^{\text{LF}}$ is compressed to a down-sampled length between 16 and 31. For instance, a time series of length $l = 192$ is compressed $n = 3$ times to have a down-sampled length $W = \lfloor\lfloor\lfloor 193/2 \rfloor/2\rfloor/2\rfloor = 24$. Similarly, the compression rate for the HF encoder is chosen such that the temporal axis of $\boldsymbol{u}^{\text{HF}}$ is compressed to a downsampled length between 64 and 127. For the length $l = 192$, it is compressed $n = 1$ time to have length $W^{\text{HF}} = \lfloor 193/2 \rfloor / = 84$. If a dataset has temporal length shorter than the downsampled width, the downsampling rate is set to 1, so no downsampling is performed. The number of ResNet blocks is $m = 4$ for both the encoder and the decoder and the number of channels are set to 64, following the implementation in (Lee et al., 2023). After encoding the LF and HF spectrograms, the encoded latents are $\boldsymbol{z}^{\text{LF}} = E_{\text{LF}}(\boldsymbol{u}^{\text{LF}})$ and $\boldsymbol{z}^{\text{HF}} = E_{\text{HF}}(\boldsymbol{u}^{\text{HF}})$. Their tensor representations has shapes (B, H, W^{LF}, C) and (B, H, W^{HF}, C), where $16 \leq W^{\text{LF}} < 32$ and $64 \leq W^{\text{HF}} < 128$ and $C = 64$.

VQ

The implementation of VQ are from (P. Wang et al., 2022). The codebook for both low and high frequencies are implemented with the same number of tokens and dimensions. At first, the codebooks were implemented similar to (Lee et al., 2023) with the number of tokens in $\mathcal{Z}_{\text{LF}}$ and $\mathcal{Z}_{\text{HF}}$ set to $K = 32$, where the dimension of each token was $d = 64$. In (Lee et al., 2023), a transformer model is used to model the token indices of the quantized latents $\boldsymbol{z}_q^{\text{LF}}$ and $\boldsymbol{z}_q^{\text{HF}}$. However, for TimeVQVDM, the prior modelling are of the continuous latents themselves using diffusion models. Before inputting the latents $\boldsymbol{z}^{\text{LF}}$ and $\boldsymbol{z}^{\text{HF}}$ to the models, they are reshaped by stacking the frequency axis along the channel dimension. For instance, this means that $\boldsymbol{z}^{\text{LF}}$, with shape (B, H, W^{LF}, C), are reshaped to $(B, H \cdot C, W^{\text{LF}})$. Since the $H = 5$, and $C = 64$, the number of channels inputted to the diffusion model was initially $H \cdot C = 5 \cdot 64 = 320$. Even though the encoders-decoder achieves a high compression rate along the temporal axis with this setup, the high number of channels in the latent space don't lead to significant dimensionality reduction overall.

Initially, when trying to train the diffusion models using this setup, the models was not able to learn the latent distributions. This issue was fixed by decreasing d significantly and increasing K accordingly. When switching to $d = 4$ and $K = 512$, the diffusion models was able to learn the latent distribution.

When using the low dimension for the tokens, the encoder and decoders was changed accordingly such that the number of channels was set to 4. This reduced the number of parameters in the encoder and decoder and increased the reconstruction loss, impacting the quality of the generated time series. To maintain a low reconstruction loss while keeping the dimensionality of the tokens low, it was found beneficial to let the number of channels in the encoder and decoder be 64, and projecting it to the low dimensional before inputting them to the VQ. The projection layers was implemented using a linear layer with 64 input channels and 4 output channels. Similarly, there are a linear layer before sending the quantized latents to the decoder. The linear layers are implemented using `torch.nn.Linear`.

U-Net

The implementation of the U-Net is based on the implementation in (Phil Wang, 2022), which includes a U-Net for discrete time-steps diffusion models applied to images. More specifically, it is an implementation of the *denoising diffusion probabilistic model* (DDPM) model given in (Ho, Jain et al., 2020) using Pytorch. This implementation was then adapted to be used for continuous time-step diffusion models applied to time series. This involved changing the convolutional layers from 2D to 1D, changing the implementation of the noising process, and fixing the attention mechanism. The DDPM implementation has discrete time-steps for the diffusion process, where time has values $1, 2, \dots, 1000$. When adapting it to use for continuous time-step diffusion, where time has continuous values between 0 to 1, the linear attention mechanism stopped working properly. After looking at the implementation of the VDM (Diederik P. Kingma et al., 2021) provided in (Google, 2022), it became apparent that the authors had multiplied the time variable by a factor of 1000. Implementing this trick resolved the problems with the attention mechanism. The linear attention uses the default implementation provided by (Phil Wang, 2022), with 4 attention heads of dimension 32.

Denote $\mathcal{U}_{\text{LF}}$ and $\mathcal{U}_{\text{HF}}$ the U-Nets for low and high frequency, respectively. $\mathcal{U}_{\text{LF}}$ has 3 down layers, while $\mathcal{U}_{\text{HF}}$ has 4 down layers. The number of channels at the first layer of the U-Net is 64 for LF and 32 for HF. At each successive down layer, the number of channels in the diffusion model is doubled. $\mathcal{U}_{\text{LF}}$ have $(64, 128, 256)$ channels for the down-layers, and $\mathcal{U}_{\text{HF}}$ have $(32, 64, 128, 256)$ channels for the down-layers. The width of the down-sampled latent at the lowest layer of the U-Net is between 2 and 4, while the width of the lowest layer of the HF U-Net is between 4 and 8. With this high level of compression, the models achieves a wide visual field at lower levels of the U-Net. The number of groups in the group normalization is set to 4 in the ResNet blocks.

Conditional arguments for the diffusion models

The implementation for the conditional arguments follows the work of (Ho, Saharia et al., 2021), where scaler conditions are provided at every ResNet block and input conditions are concatenated with the input to the U-Net. The class y and time step t are both embedded to a dimension of 256. To allow for both conditional and unconditional generation, the number of embedding vectors for the class is set to $C + 1$, where C is the number of classes. For unconditional sampling, the extra embedding vector is then used instead of the class embedding. The embeddings for the class condition is implemented using `torch.nn.Embedding`. The time is embedded using a sinusoidal positional embedding layer, following the implementation in (Crowson and AI, 2021). The class and time embeddings are concatenated to a vector of length 512. To produce the scale and shift vectors for guiding the synthesize process at each ResNet block, the scale and shift is implemented with a parameterized ReLU activation function and a linear layer. The resulting tensor is then chunked into a scale and shift vector. Let the output after the group normalization layer in the ResNet block be denoted by $\boldsymbol{x}$. It is scaled and shifted by calculating $\boldsymbol{x}_{\text{guided}} = \boldsymbol{x}(\text{scale} + 1) + \text{shift}$.

The input to the HF diffusion model, $\boldsymbol{z}_t^{\mathrm{HF}}$, is conditioned on $\boldsymbol{z}_0^{\mathrm{LF}}$. Since these latents have different dimensions, $\boldsymbol{z}_0^{\mathrm{LF}}$ has shape $(B, H, W^{\mathrm{LF}}, C)$ and $\boldsymbol{z}_t^{\mathrm{HF}}$ has shape $(B, H, W^{\mathrm{HF}}, C)$, simply concatenating them along the channel dimension is not possible. To make the LF have the same shape as the HF latent, the LF latent is up-scaled and interpolated to the same dimension as the HF. This makes the LF features align with the HF features and was found to be essential for aligning the low and high frequencies during sampling. Without providing $\boldsymbol{z}_0^{\mathrm{LF}}$ to the HF diffusion model, then the generation process would not be able to sample jointly from $q(\boldsymbol{x}^{\mathrm{LF}}, \boldsymbol{x}^{\mathrm{HF}})$. As a result, the generated low and high frequencies would not align.

Scaling of the latents before the diffusion process

In the theory section on diffusion models, the input to the diffusion models are assumed to be standardized. This assumption is not necessarily satisfied after encoding $\boldsymbol{u}^{\mathrm{LF}}$ and $\boldsymbol{u}^{\mathrm{HF}}$ to the latent space. To make sure that the latents $\boldsymbol{z}^{\mathrm{LF}} = E(\boldsymbol{u}^{\mathrm{LF}})$ and $\boldsymbol{z}^{\mathrm{HF}} = E(\boldsymbol{u}^{\mathrm{HF}})$ have unit variance, a LF and HF standard-scaler are learned. Let these be denoted as $\mathcal{S}_{\mathrm{LF}}$ and $\mathcal{S}_{\mathrm{HF}}$, respectively. They contain only two parameters each, a mean and standard deviation parameter. After training the VQ-VAEs in stage 1, $\mathcal{S}_{\mathrm{LF}}$ and $\mathcal{S}_{\mathrm{HF}}$ learns the standard deviation and mean of $q_\phi(\boldsymbol{z}^{\mathrm{LF}} \mid \boldsymbol{x})$ and $q_\phi(\boldsymbol{z}^{\mathrm{HF}} \mid \boldsymbol{x})$ by encoding the whole training data set to latent space and calculating the standard deviation and mean. Thus the input to the LF diffusion model is $\boldsymbol{z}_0^{\mathrm{LF}}$, a reshaped and scaled version of $\boldsymbol{z}^{\mathrm{LF}}$. Similarly, $\boldsymbol{z}_0^{\mathrm{HF}}$ is a reshaped and scaled version of $\boldsymbol{z}^{\mathrm{HF}}$.

3.4 Overview of variables and neural networks

TimeVQVDM has many variables and neural network modules operating at different parts of the model. The shapes of the variables vary depending on the batch size (B) and the the length of the input time series (L). The number of channels (C) is fixed for every variable, and the frequency axis, equal to the height (H) of the spectrograms, are constant for all variables. The downsampled width (W) is between 16 and 31 for the low frequencies and between 64 and 127 for high frequencies. Lets look at the data set 'Wafer' from the UCR data set (Dau et al., 2018), which has length $l = 152$ and contains 2 classes.

A summary of the variables and neural network modules are provided in Table 1 and Table 2. In stage one, the VQ-VAE models are trained, which corresponds to training the encoders, decoders and codebooks. Then in stage two, the low and high frequency diffusion models are trained. The two scalers, $\mathcal{S}_{\mathrm{LF}}$ and $\mathcal{S}_{\mathrm{HF}}$, are fitted before training the diffusion models by learning the standard deviation and mean of the continuous latents.

Table 1: Summary of the variables in TimeVQVDM. (Data set: Wafer)

Variable	**Description**	**Shape of tensor**
$\boldsymbol{x}$, $\boldsymbol{x}^{\mathrm{LF}}$, $\boldsymbol{x}^{\mathrm{HF}}$	time series	$(B, C = 1, L = 152)$
$\boldsymbol{u}$, $\boldsymbol{u}^{\mathrm{LF}}$, $\boldsymbol{u}^{\mathrm{HF}}$	spectrograms	$(B, H = 5, W = 153, C = 2)$
$\boldsymbol{z}^{\mathrm{LF}}$, $\boldsymbol{z}_q^{\mathrm{LF}}$	LF continuous and quantized latents	$(B, H = 5, W = 19, C = 4)$
$\boldsymbol{z}^{\mathrm{HF}}$, $\boldsymbol{z}_q^{\mathrm{HF}}$	HF continuous and quantized latents	$(B, H = 5, W = 76, C = 4)$
$\boldsymbol{z}_0^{\mathrm{LF}}$, $\boldsymbol{z}_t^{\mathrm{LF}}$, $\quad t \in [0, 1]$	LF diffusion latents	$(B, C = 20, W = 19)$
$\boldsymbol{z}_0^{\mathrm{HF}}$, $\boldsymbol{z}_t^{\mathrm{HF}}$, $\quad t \in [0, 1]$	HF diffusion latents	$(B, C = 20, W = 76)$

Table 2: Summary of the modules in TimeVQVDM. (Data set: Wafer)

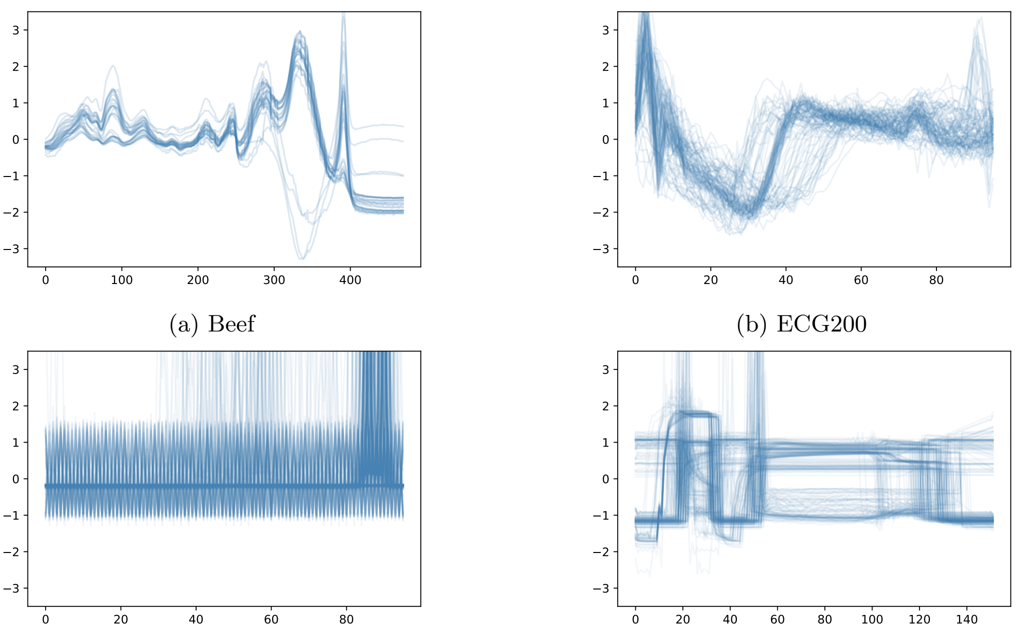
Module	**Description**	**Stage**	**Num. parameters**
E_{LF}, D_{LF}, $\mathcal{Z}_{\mathrm{LF}}$	LF encoder, decoder and codebook	1	534 k
E_{HF}, D_{HF}, $\mathcal{Z}_{\mathrm{HF}}$	HF encoder, decoder and codebook	1	336 k
$\mathcal{S}_{\mathrm{LF}}, \mathcal{U}_{\mathrm{LF}}$	LF scaler and diffusion model	2	4.12 mill
$\mathcal{S}_{\mathrm{HF}}, \mathcal{U}_{\mathrm{HF}}$	HF scaler and diffusion model	2	3.97 mill

3.5 The UCR Time Series Archive

The performance of TimeVQVDM is tested on all time series data from the UCR time series archive (Dau et al., 2018). It is a collection of 128 different data sets provided by the university of California, Riverside. Originally provided for the data mining community for benchmarking different methods for time series classification, it has become a resource for other disciplines in time series analysis, such as clustering, and time series generation. The UCR archive encompass a diverse range of data sets, including:

- Image-derived data: This involves tracking the trajectory of objects in videos.
- Sensory data: This could be of natural phenomena like earthquakes and lightning, as well as other sensory measurements such as chlorine concentration, power consumption, traffic data, and readings from vehicles.
- Device data: This are from electrical devices, such as computers and kitchen applications.
- Health sector data: This includes electrocardiograms, hemodynamic data, and electrooculography data.
- Spectro-analysis data: Spectroscopics of food items like ham, beef, wine, and coffee.
- Simulated data: Artificially generated through computer simulations.

Each of the data sets comes in a predefined training and test split. These data sets are standardized, such that the means is zero-centered and the variance across each data set is one. The length of the time series vary a lot, with the shortest being **SmoothSubspace** with a length of 15, and the longest being **Rock**, having a length of 2844. The number of training samples also varies, with some data sets having very few samples. For instance, **DiatomSizeReduction** only has 16 training samples. A summary of all 128 data sets is provided in Appendix E. Figure 12 displays four of the data sets and their distribution by plotting the run chart of 256 training samples together.



(a) Beef (b) ECG200

(c) ElectricDevices (d) Wafer

Figure 12: Examples from the UCR time series archive. 12a is an example of spectroscopic data, 12b is an example of electrocardiogram from the health sector, 12c is an example of device data and 12d is an example of sensory data. The time series have different temporal structures.

3.6 Experiments

Initial testing on 'Wafer'

Many of the implementation tricks for TimeVQVDM was discovered empirically by trial and error. The model was initially tested using a couple of the data sets from the UCR time series archive, allowing for quick testing of different designs. Most of the discoveries occurred while testing one particular data set in particular, namely on the data set 'Wafer'. This data set, depicted in Figure 12d, contains 1000 training samples, has length 152 and 2 classes. It contains recorded measurements from various sensors during the processing of silicon wafers. The time series has sharp changes in modularity, and in some places there are high frequency outliers. Results of the initial testing is outlined in Section 4.1.

Training on all data sets in the UCR time series archive

After finding an architecture for TimeVQVDM that works reasonably well, some other data sets was tested to make sure that the proposed model generalizes. TimeVQVDM was then trained and evaluated on all 128 data sets in the UCR time series archive. The training of stage one and two was conducted with a batch size was set to 128 using 2000 epochs for stage 1 and 1000 epochs for stage 2. The optimizer used was AdamW (Loshchilov and Hutter, 2017) with cosine learning rate scheduler and initial learning rate of $1.0 \cdot 10^{-3}$. Additional training details are provided in Appendix D. Each experiment, including training, sampling and evaluation, took around 1-2 hour per data set using a NVIDIA GeForce RTX 3080 Laptop GPU. In total, the experiments were run over a span of two weeks.

For evaluation the models, both unconditional and class-conditional sampling is preformed. Unconditional sampling is performed with 128 denoising steps for both low and high frequency diffusion models and rescale parameter $\gamma = 0.1$. The FID and IS score for unconditional sampling are calculated by generating 1024 samples, averaged over three runs. Conditional sampling is performed with 32 denoising steps for both diffusion models, guidance weight $w = 3.0$ and the rescale parameter $\gamma = 0.1$. The FID and IS score for conditional sampling are calculated by generating 1024 samples from one run. For conditional sampling, the class labels are selected according to the class proportions in the training data. This means that if a training set has 100 samples for class '0' and 300 samples for class '1', then the synthetic generated data contains 256 samples generated with class '0' and 768 samples generated with class '1'. This protocol is used for making the evaluated FID and IS scores comparable with the reported scores of TimeVQVAE.

Classification accuracy scores are calculated from training a classifier on synthetic data and testing it on real. The TSTR classification accuracy scores are reported on the real test data, not the real training data. I.e. the classifiers are tested on unseen data. The TRTR classification accuracy scores are from using the pre-trained classifiers provided by (Lee et al., 2023).

3.7 Algorithms

Training

Algorithm 1 and 2 displays the algorithms used for training TimeVQVDM. Algorithm 1 is for training stage one, while Algorithm 2 is for training stage two.

Algorithm 1 Stage 1 training

```
repeat
    $\boldsymbol{x} \sim q(\boldsymbol{x})$    ▷ sample time series
    $\boldsymbol{u}^{\mathrm{LF}},\ \boldsymbol{u}^{\mathrm{HF}} \leftarrow \mathcal{P}_{\mathrm{LF}}(\mathrm{STFT}(\boldsymbol{x})),\ \mathcal{P}_{\mathrm{HF}}(\mathrm{STFT}(\boldsymbol{x}))$    ▷ split high/low frequencies, Eq. (39)
    $\boldsymbol{z}^{\mathrm{LF}},\ \boldsymbol{z}^{\mathrm{HF}} \leftarrow E_{\mathrm{LF}}(\boldsymbol{u}^{\mathrm{LF}}),\ E_{\mathrm{HF}}(\boldsymbol{u}^{\mathrm{HF}})$    ▷ encode to latent space, Eq. (40)

    $(\boldsymbol{z}_q^{\mathrm{LF}})_i \leftarrow \operatorname*{argmin}_{\boldsymbol{e}_k \in \mathcal{Z}_{\mathrm{LF}}} \left\|(\boldsymbol{z}^{\mathrm{LF}})_i - \boldsymbol{e}_k\right\|$    ▷ quantize, Eq. (41)
    $(\boldsymbol{z}_q^{\mathrm{HF}})_i \leftarrow \operatorname*{argmin}_{\boldsymbol{e}_k \in \mathcal{Z}_{\mathrm{HF}}} \left\|(\boldsymbol{z}^{\mathrm{HF}})_i - \boldsymbol{e}_k\right\|$

    $\hat{\boldsymbol{u}}^{\mathrm{LF}}, \hat{\boldsymbol{u}}^{\mathrm{HF}} \leftarrow \mathcal{P}_{\mathrm{LF}}(D_{\mathrm{LF}}(\boldsymbol{z}_q^{\mathrm{LF}})),\ \mathcal{P}_{\mathrm{HF}}(D_{\mathrm{HF}}(\boldsymbol{z}_q^{\mathrm{HF}}))$    ▷ decode to sample space, Eq. (42)

    $\hat{\boldsymbol{x}}^{\mathrm{LF}},\ \hat{\boldsymbol{x}}^{\mathrm{HF}} \leftarrow \mathrm{ISTFT}(\hat{\boldsymbol{u}}^{\mathrm{LF}}),\ \mathrm{ISTFT}(\hat{\boldsymbol{u}}^{\mathrm{HF}})$    ▷ transform to time domain, Eq. (33), (43)
    $\boldsymbol{x}^{\mathrm{LF}},\ \boldsymbol{x}^{\mathrm{HF}} \leftarrow \mathrm{ISTFT}(\boldsymbol{u}^{\mathrm{LF}}),\ \mathrm{ISTFT}(\boldsymbol{u}^{\mathrm{HF}})$

    $L_{\mathrm{codebook}} \leftarrow \left\|\mathrm{sg}[\boldsymbol{z}^{\mathrm{LF}}] - \boldsymbol{z}_q^{\mathrm{LF}}\right\|_2^2 + \left\|\mathrm{sg}[\boldsymbol{z}^{\mathrm{HF}}] - \boldsymbol{z}_q^{\mathrm{HF}}\right\|_2^2 + \beta\left\|\boldsymbol{z}^{\mathrm{LF}} - \mathrm{sg}[\boldsymbol{z}_q^{\mathrm{LF}}]\right\|_2^2 + \beta\left\|\boldsymbol{z}^{\mathrm{HF}} - \mathrm{sg}[\boldsymbol{z}_q^{\mathrm{LF}}]\right\|_2^2$
    $L_{\mathrm{reconstruct}} \leftarrow \left\|\boldsymbol{x}^{\mathrm{LF}} - \hat{\boldsymbol{x}}^{\mathrm{LF}}\right\|_2^2 + \left\|\boldsymbol{x}^{\mathrm{HF}} - \hat{\boldsymbol{x}}^{\mathrm{HF}}\right\|_2^2 + \left\|\boldsymbol{u}^{\mathrm{LF}} - \hat{\boldsymbol{u}}^{\mathrm{LF}}\right\|_2^2 + \left\|\boldsymbol{u}^{\mathrm{HF}} - \hat{\boldsymbol{u}}^{\mathrm{HF}}\right\|_2^2$

    take gradient descent on $\nabla_\theta\ (L_{\mathrm{codebook}} + L_{\mathrm{reconstruct}})$
until converged
```

Algorithm 2 Stage 2 training

```
repeat
    $\boldsymbol{x},\ y \sim q(\boldsymbol{x},\ y)$    ▷ sample time series and class label
    $\boldsymbol{u}^{\mathrm{LF}},\ \boldsymbol{u}^{\mathrm{HF}} \leftarrow \mathcal{P}_{\mathrm{LF}}(\mathrm{STFT}(\boldsymbol{x})),\ \mathcal{P}_{\mathrm{HF}}(\mathrm{STFT}(\boldsymbol{x}))$    ▷ split high/low frequencies, Eq. (39)
    $\boldsymbol{z}_0^{\mathrm{LF}},\ \boldsymbol{z}_0^{\mathrm{HF}} \leftarrow E_{\mathrm{LF}}(\boldsymbol{u}^{\mathrm{LF}}),\ E_{\mathrm{HF}}(\boldsymbol{u}^{\mathrm{HF}})$    ▷ encode to latent space, Eq. (40)

    $t^{\mathrm{LF}},\ t^{\mathrm{HF}} \sim \mathrm{U}(0,1)$    ▷ random diffusion times
    $\boldsymbol{\epsilon}^{\mathrm{LF}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$    ▷ random noise
    $\boldsymbol{\epsilon}^{\mathrm{HF}} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

    $\boldsymbol{z}_t^{\mathrm{LF}} \leftarrow \alpha_{t^{\mathrm{LF}}} \boldsymbol{z}_0^{\mathrm{LF}} + \sigma_{t^{\mathrm{LF}}} \boldsymbol{\epsilon}^{\mathrm{LF}}$    ▷ diffuse LF latent, Eq. (15)
    $\boldsymbol{z}_t^{\mathrm{HF}} \leftarrow \alpha_{t^{\mathrm{HF}}} \boldsymbol{z}_0^{\mathrm{HF}} + \sigma_{t^{\mathrm{HF}}} \boldsymbol{\epsilon}^{\mathrm{HF}}$    ▷ diffuse HF latent, Eq. (15)

    $c^{\mathrm{LF}} \leftarrow y$ if $\mathrm{U}(0,1) > p_{\mathrm{unconditional}}$ else $\emptyset$    ▷ class guidance signal
    $c^{\mathrm{HF}} \leftarrow y$ if $\mathrm{U}(0,1) > p_{\mathrm{unconditional}}$ else $\emptyset$

    $L_{\mathrm{diffusion}} \leftarrow \left\|\boldsymbol{\epsilon}^{\mathrm{LF}} - f_\theta\left(\boldsymbol{\epsilon}^{\mathrm{LF}};\ \boldsymbol{z}_t^{\mathrm{LF}},\ t^{\mathrm{LF}},\ c^{\mathrm{LF}}\right)\right\|_2^2 + \left\|\boldsymbol{\epsilon}^{\mathrm{HF}} - f_\theta\left(\boldsymbol{\epsilon}^{\mathrm{HF}};\ \boldsymbol{z}_t^{\mathrm{HF}},\ \boldsymbol{z}_0^{\mathrm{LF}},\ t^{\mathrm{HF}},\ c^{\mathrm{HF}}\right)\right\|_2^2$

    take gradient descent on $\nabla_\theta\ (L_{\mathrm{diffusion}})$
until converged
```

Sampling

Algorithm 3 and 4 displays how we can sample new time series from the TimeVQVDM. Algorithm 3 is the sample loop from one diffusion model. It performs T denoising steps using the class y and input condition $\boldsymbol{z}_{\text{in}}$. Classifier-free guidance with rescaling is performed using the guidance weight w and rescale-parameter γ.

Algorithm 4 is the sampling procedure of TimeVQVDM. It inputs the number of denoising steps for LF and HF, along with optional class condition and guidance parameters.

Algorithm 3 Ancestral sampling from diffusion model

```
input $T, \boldsymbol{z}_{\text{in}}, y, w, \gamma$        ▷ denoising steps, input condition, class, guidance parameters

sample $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$        ▷ sample from prior
for $i = T, \dots, 1$ do        ▷ sampling loop, Eq. (27)
    $s,\ t \leftarrow (i-1)/T,\ i/T$

    $\hat{\boldsymbol{\epsilon}} \leftarrow (1-w) f_\theta(\boldsymbol{\epsilon}; \boldsymbol{z}, \boldsymbol{z}_{\text{in}}, t, \emptyset) + w f_\theta(\boldsymbol{\epsilon}; \boldsymbol{z}, \boldsymbol{z}_{\text{in}}, t, y)$        ▷ classifier-free guidance, Eq. (29)

    $\boldsymbol{\delta} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$        ▷ denoising step, Eq. (28)
    $\boldsymbol{z}_{\text{cfg}} \leftarrow \frac{\alpha_s}{\alpha_t}\left(\boldsymbol{z} + (1 - e^{\lambda_t - \lambda_s})\sigma_t \hat{\boldsymbol{\epsilon}}\right) + (1 - e^{\lambda_t - \lambda_s})^{1/2} \sigma_s \boldsymbol{\delta}$

    $\boldsymbol{z} \leftarrow (1-\gamma)\ \boldsymbol{z}_{\text{cfg}} + \gamma\ \boldsymbol{z}_{\text{cfg}} / \text{std}(\boldsymbol{z}_{\text{cfg}})$        ▷ rescale latent, Eq. (31)
end for
return $\boldsymbol{z}$
```

Algorithm 4 Sample from TimeVQVDM

```
input $T^{\text{LF}},\ T^{\text{HF}},\ y,\ w,\ \gamma$        ▷ denoising steps, class, guidance weight, rescale factor

$\boldsymbol{z}^{\text{LF}} \sim p_\theta(\boldsymbol{z}^{\text{LF}} \mid y)$        ▷ sample LF using Algorithm 3 with $T = T^{\text{LF}}$, $\boldsymbol{z}_{\text{in}} = \emptyset$
$\boldsymbol{z}^{\text{HF}} \sim p_\theta(\boldsymbol{z}^{\text{HF}} \mid \boldsymbol{z}^{\text{LF}}, y)$        ▷ sample HF using Algorithm 3 with $T = T^{\text{HF}}$, $\boldsymbol{z}_{\text{in}} = \boldsymbol{z}^{\text{LF}}$

$(\boldsymbol{z}_q^{\text{LF}})_i \leftarrow \underset{\boldsymbol{e}_k \in \mathcal{Z}_{\text{LF}}}{\text{argmin}} \left\|(\boldsymbol{z}^{\text{LF}})_i - \boldsymbol{e}_k\right\|$        ▷ quantize, Eq. (41)
$(\boldsymbol{z}_q^{\text{HF}})_i \leftarrow \underset{\boldsymbol{e}_k \in \mathcal{Z}_{\text{HF}}}{\text{argmin}} \left\|(\boldsymbol{z}^{\text{HF}})_i - \boldsymbol{e}_k\right\|$

$\boldsymbol{u}^{\text{LF}}, \boldsymbol{u}^{\text{HF}} \leftarrow \mathcal{P}_{\text{LF}}(D_{\text{LF}}(\boldsymbol{z}_q^{\text{LF}})),\ \mathcal{P}_{\text{HF}}(D_{\text{HF}}(\boldsymbol{z}_q^{\text{HF}}))$        ▷ decode to sample space, Eq. (42)

$\boldsymbol{x}^{\text{LF}},\ \boldsymbol{x}^{\text{HF}} \leftarrow \text{ISTFT}(\boldsymbol{u}^{\text{LF}}),\ \text{ISTFT}(\boldsymbol{u}^{\text{HF}})$        ▷ transform to time domain, Eq. (33)

return $\boldsymbol{x}^{\text{LF}} + \boldsymbol{x}^{\text{HF}}$
```

4 Results

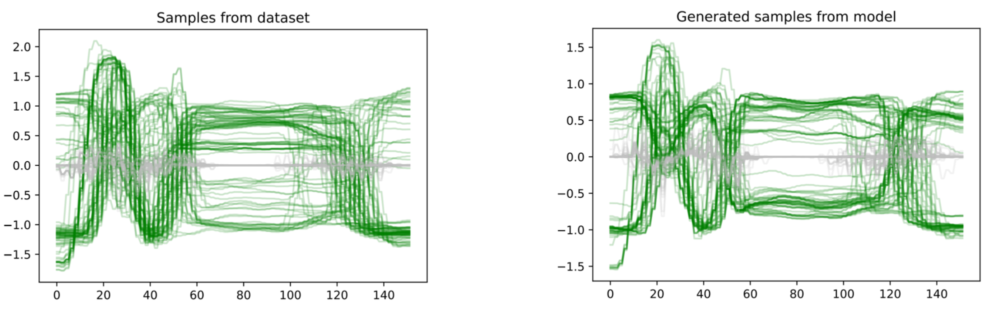
4.1 Results on Wafer

The initial testing on Wafer yielded five important implementation tricks for TimeVQVDM:

- Low dimensionality tokens
- High temporal resolution in STFT
- Scaling the latents before the diffusion models
- Aligning LF and HF with input conditions
- Halting the diffusion process at $t = 0.99$

First trick: Low dimensionality tokens

As outlined under architecture and implementation details in Section 3, it was found necessary to reduce the dimensionality of the tokens. The diffusion models were not able to learn the latent distribution of Wafer with the number of tokens being $K = 32$ and dimensionality $d = 64$, as is used in TimeVQVAE. This could be because the dimensionality of the tokens was stacked along the channel dimension, which creates very many channels for the 1D denoising U-Nets. The first major implementation trick was to decrease the dimensionality of the tokens drastically. We found that choosing $d = 4$ worked great. To keep a similar compression rate, the number of tokens was increased accordingly to 512. In this way the number of parameters in the codebooks, $K \cdot d$ are kept constant.

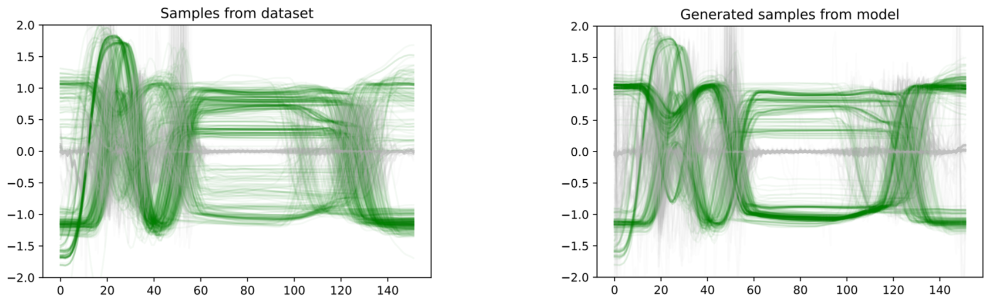


(a) Reconstructions of real samples. (b) Reconstructions of generated samples.

Figure 13: Comparison of reconstructions of low and high frequencies of Wafer using 13a real time series and 13b generated time series. Low frequencies are in green and high frequencies are in gray. The STFT has a hop length $h = 2$. The input to the diffusion models are not scaled.

Second trick: High temporal resolution in STFT

Although decreasing the dimensionality of the tokens made the diffusion models happy, it damaged the reconstruction loss of the VQ-VAE models. Figure 13 shows the reconstructed low and high frequencies of Wafer when using the STFT defined in TimeVQVAE. It has parameters `n_fft = 8` and `hop_length = 2`. Using low dimensionality tokens and this setup for the STFT, the reconstructions for both low and high frequencies becomes jagged. Changing `hop_length = 1` increases the temporal resolution in time-frequency domain and made the reconstructions much better. Figure 14 shows the reconstructed low and high frequencies of Wafer when using the shorter hop length for STFT. The LF was smoother and the high frequencies was able to capture finer details such as the spikes at time steps 20 and 50 in the signals.



(a) Reconstructions of real samples.

(b) Reconstructions of generated samples.

Figure 14: Comparison of reconstructions of low and high frequencies of Wafer using 14a real time series and 14b generated time series. Low frequencies are in green and high frequencies are in gray. The STFT has a hop length $h = 1$. The input to the diffusion models are standardized.

Third trick: Scaling the latents before the diffusion models

The empirical distribution is standardized to having variance $\mathbb{V}[q^*(\boldsymbol{x})] = 1$ and mean $\mathbb{E}[q^*(\boldsymbol{x})] = 0$. In the theory section on the diffusion models, it is assumed that the non-diffused latent distribution for the diffusion process is also standardized. However $\boldsymbol{z}^{\text{LF}}$ and $\boldsymbol{z}^{\text{HF}}$ are not necessary standardized after encoding $\boldsymbol{x}$ to time-frequency domain and encoding using E_{LF} and E_{HF}. which was a problem for the sampling process. Initially, the rescaling parameter was set to 1, such that the latents in the diffusion models always had unit variance in-between each sampling step. This produced synthetic time series with different amplitudes than the training data. This effect is seen in Figure 13b, where the y-axis has a slightly different scale than the y-axis in Figure 13a. The generated samples in Figure 14 incorporates a learned standard-scalers for both LF and HF, making the generated time series have the same amplitudes. After standardizing the inputs to the diffusion models, the rescaling parameter γ was no longer necessary for unconditional sampling. It was however kept at 0.1 to prevent the variance of the latents from deviating to far from 1.

Forth trick: Aligning LF and HF with input conditions

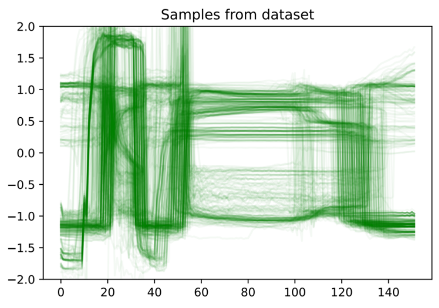
Until now, the low and high frequencies had been generated separately. To align the generated frequencies, it was ultimately found that the best approach was using the conditioning method in (Ho, Saharia et al., 2021) by concatenating the input signal along with the input to the denoising U-Nets. For TimeVQVDM, this is to letting $\boldsymbol{z}_0^{\text{LF}}$ be concatenated along the channel dimension of the input to the high frequency diffusion model. But since $\boldsymbol{z}_t^{\text{HF}}$ has a lower compression rate than $\boldsymbol{z}_0^{\text{LF}}$, $\boldsymbol{z}_0^{\text{LF}}$ needs to be up-scaled to the same dimensionality as $\boldsymbol{z}_t^{\text{HF}}$. As outlined in the implementation details, this was performed by up-scaling $\boldsymbol{z}_0^{\text{LF}}$ and use an interpolation layer to match the correct shape of $\boldsymbol{z}_t^{\text{HF}}$. Figure 15 shows generated samples after implementing the conditioning mechanism for generating HF based on LF. The generated time series resembles the data set, but the mode coverage is not great.

Fifth trick: Halting the diffusion process at $t = 0.99$

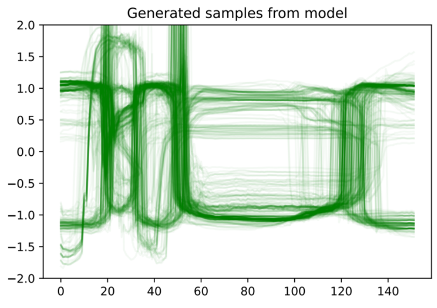
After investigating the equations used for sampling from the diffusion models, it became apparent that the sampling process was very sensitive for low signal-to-noise ratios. From Equation (15), the predicted non-diffused LF latent is $\hat{\boldsymbol{z}}_0^{\text{LF}} = (\boldsymbol{z}_t^{\text{LF}} - \sigma_t \hat{\boldsymbol{\epsilon}}^{\text{LF}})/\alpha_t$, where $\hat{\boldsymbol{\epsilon}}^{\text{LF}}$ is the predicted noise. Since $\alpha_t \to 0$ as $t \to 1$, this creates a division by zero problem. It should make sense that we cannot make an informed prediction of $\boldsymbol{z}_0^{\text{LF}}$ from $\boldsymbol{z}_1^{\text{LF}}$ alone, as $\boldsymbol{z}_1^{\text{LF}}$ contain no information about $\boldsymbol{z}_0^{\text{LF}}$. In testing on Wafer and other data sets, this problem caused the generated samples to have poor mode coverage. Halting the diffusion process at a time $t_{\max}$ resolved this issue. More specifically, the generation process starts denoising from $t = t_{\max}$ and ends at $t = 0$. The value of $t_{\max}$ was empirically determined by letting it take the values in $\{1.0, 0.999, 0.99, 0.98, 0.95\}$ and evaluating the FID score on Wafer using 32 denoising steps for LF and HF, while holding the rescale parameter fixed at 0.1. The results are shown in Table 3 and found that choosing $t_{\max} = 0.99$ worked best. The FID score was then evaluated using 128 and 512 denoising steps, of which 128 denoising steps yielded the lowest FID score. This indicates that halting the diffusion at 0.99 would give higher mode coverage for the generated samples. Indeed, plotting the generated samples from Wafer after halting the diffusion process at 0.99 shows a huge visual improvement in the mode coverage. The generated samples in Figure 16b resembles the real training data better than the generated samples in Figure 15b, even though it has fewer denoising steps.

Table 3: FID scores for unconditional sampling of Wafer by halting the diffusion process at different time steps. The number of generated samples is 1024 and $\gamma = 0.1$.

$t_{\max}$	$\text{SNR}_{\min}$	FID (32 steps)	FID (128 steps)	FID (512 steps)
1.0	0.0	1126.70	-	-
0.999	$2.47 \cdot 10^{-6}$	8.73	-	-
0.99	$2.47 \cdot 10^{-4}$	1.65	**1.15**	1.49
0.98	$9.87 \cdot 10^{-4}$	1.95	-	-
0.95	$6.12 \cdot 10^{-3}$	4.98	-	-

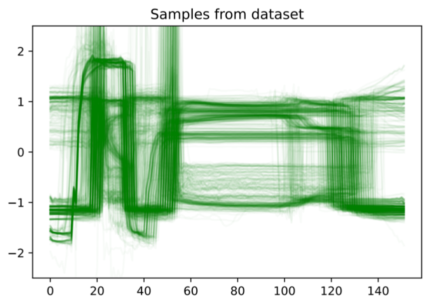


(a) Samples from dataset.

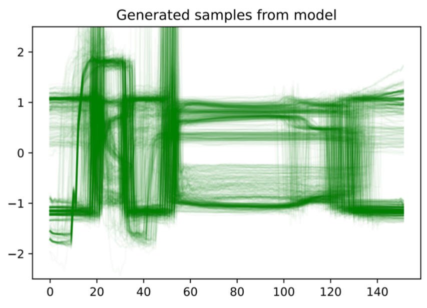


(b) Generated samples.

Figure 15: Comparison of real time series 15a and unconditionally sampled time series 15b of Wafer. The samples are generated using a 1024 sampling steps starting from $t = 1$.



(a) Sampes from dataset.



(b) Generated samples.

Figure 16: Comparison of real time series 16a and unconditionally sampled time series 16b of Wafer. The samples are generated using a 128 sampling steps starting from $t = 0.99$.

4.2 Results on all data sets in the UCR time series archive

The following pages contains full results on the UCR time series archive. Evaluation results from unconditional and conditional sampling are provided in Table 4 and Table 5 respectively. The reported metrics for unconditional sampling are FID and IS. For conditional sampling, the reported metrics are FID, IS and TSTR. After the two tables, the next eight pages contain visual representations from unconditional sampling from all data sets in the UCR archive by plotting the run charts of 256 generated samples next to 256 samples from the ground truth. Since the visual representation from conditional sampling appeares very similar to the unconditional samples, those are not provided.

Table 4: Full IS and FID results for unconditional sampling from the UCR archive with 1024 synthetic time series using 128 sampling steps and $\gamma = 0.1$. The results are averaged over 3 runs.

Name	**IS mean** $\uparrow$	**IS std**	**FID mean** $\downarrow$	**FID std**
Adiac	6.31	0.29	6.76	0.23
ArrowHead	2.13	0.08	3.74	0.05
Beef	2.47	0.06	2.31	0.05
BeetleFly	1.55	0.04	5.1	0.07
BirdChicken	1.57	0.04	6.73	0.03
Car	1.63	0.07	4.93	0.1
CBF	1.89	0.07	14.1	0.46
ChlorineConcentration	1.54	0.08	1.01	0.13
CinCECGTorso	2.02	0.06	4.15	0.03
Coffee	1.36	0.03	10.54	0.03
Computers	1.73	0.03	2.73	0.06
CricketX	2.46	0.08	4.07	0.03
CricketY	2.99	0.19	3.52	0.01
CricketZ	2.37	0.08	4.75	0.06
DiatomSizeReduction	1.56	0.04	13.81	0.03
DistalPhalanxOutlineAgeGroup	1.91	0.07	0.93	0.11
DistalPhalanxOutlineCorrect	1.59	0.07	1.64	0.14
DistalPhalanxTW	2.75	0.18	1.87	0.13
Earthquakes	1.13	0.03	0.94	0.03
ECG200	1.51	0.07	3.47	0.08
ECG5000	2.17	0.12	0.82	0.14
ECGFiveDays	1.62	0.04	7.15	0.19
ElectricDevices	4.45	0.21	2.04	0.12
FaceAll	8.77	0.46	2.03	0.05
FaceFour	1.41	0.02	15.22	0.03
FacesUCR	4.1	0.29	5.34	0.07
FiftyWords	5.92	0.44	4.08	0.16
Fish	3.12	0.17	1.26	0.01
FordA	1.67	0.02	0.65	0.02
FordB	1.72	0.04	0.87	0.04
GunPoint	1.62	0.02	4.45	0.17
Ham	1.42	0.03	1.38	0.03
HandOutlines	1.23	0.04	0.18	0.06
Haptics	1.91	0.07	1.87	0.06
Herring	1.15	0.04	1.26	0.01
InlineSkate	2.31	0.12	11.32	0.25
InsectWingbeatSound	3.72	0.18	1.77	0.04
ItalyPowerDemand	1.84	0.03	2.77	0.17
LargeKitchenAppliances	1.89	0.06	2.07	0.09
Lightning2	1.48	0.04	1.2	0.03
Lightning7	2.57	0.12	5.73	0.1
Mallat	2.98	0.16	2.39	0.1
Meat	1.68	0.06	2.61	0.04

MedicalImages	2.87	0.23	1.26	0.04
MiddlePhalanxOutlineAgeGroup	2.05	0.08	0.7	0.03
MiddlePhalanxOutlineCorrect	1.55	0.09	1.01	0.11
MiddlePhalanxTW	2.8	0.14	1.31	0.03
MoteStrain	1.7	0.02	6.9	0.33
NonInvasiveFetalECGThorax1	10.12	0.45	5.24	0.16
NonInvasiveFetalECGThorax2	13.71	0.47	3.24	0.07
OliveOil	1.04	0.01	1.37	0.02
OSULeaf	1.41	0.03	17.28	0.22
PhalangesOutlinesCorrect	1.46	0.04	0.47	0.04
Phoneme	2.31	0.08	9.85	0.05
Plane	5.49	0.17	3.6	0.14
ProximalPhalanxOutlineAgeGroup	2.43	0.08	0.66	0.05
ProximalPhalanxOutlineCorrect	1.57	0.07	0.45	0.08
ProximalPhalanxTW	2.91	0.1	1.66	0.09
RefrigerationDevices	1.59	0.05	34.1	0.37
ScreenType	1.9	0.08	4.9	0.24
ShapeletSim	1.08	0.01	26.1	0.07
ShapesAll	8.32	0.56	8.58	0.27
SmallKitchenAppliances	1.48	0.06	5.7	0.24
SonyAIBORobotSurface1	1.18	0.04	16.14	0.2
SonyAIBORobotSurface2	1.78	0.03	4.88	0.12
StarLightCurves	2.42	0.11	0.35	0.03
Strawberry	1.7	0.05	0.11	0.02
SwedishLeaf	9.33	0.56	2.84	0.04
Symbols	3.91	0.1	8	0.25
SyntheticControl	4.81	0.2	3.64	0.09
ToeSegmentation1	1.4	0.03	7.24	0.13
ToeSegmentation2	1.43	0.04	6.85	0.15
Trace	2.65	0.07	4.75	0.16
TwoLeadECG	1.52	0.02	7.94	0.17
TwoPatterns	3.23	0.12	1.36	0.04
UWaveGestureLibraryAll	4.1	0.17	1.95	0.08
UWaveGestureLibraryX	4.24	0.2	3.99	0.12
UWaveGestureLibraryY	3.87	0.12	1.73	0.07
UWaveGestureLibraryZ	4.01	0.1	4.74	0.03
Wafer	1.31	0.07	1.03	0.19
Wine	1.39	0.03	0.99	0.02
WordSynonyms	1.74	0.05	4.11	0.02
Worms	2.01	0.13	3.28	0.03
WormsTwoClass	1.5	0.03	0.36	0.05
Yoga	1.35	0.03	0.6	0.02
ACSF1	1.28	0.08	214.2	0.79
AllGestureWiimoteX	2.17	0.13	0.84	0.07
AllGestureWiimoteY	2.0	0.11	1.03	0.05
AllGestureWiimoteZ	1.04	0.02	0.62	0.07
BME	1.92	0.09	17.58	0.58
Chinatown	1.69	0.02	6.36	0.39
Crop	16.44	0.64	1.4	0.18
DodgerLoopDay	1.61	0.05	10.14	0.04
DodgerLoopGame	1.2	0.02	15.78	0.18
DodgerLoopWeekend	1.73	0.04	5.73	0.36
EOGHorizontalSignal	4.98	0.11	1.22	0.02
EOGVerticalSignal	3.75	0.26	1.84	0.35
EthanolLevel	1.19	0.01	0.34	0
FreezerRegularTrain	1.72	0.04	1.75	0.1
FreezerSmallTrain	1.69	0.06	6.63	0.13

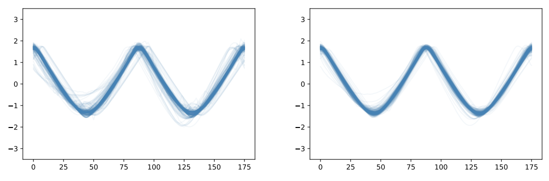
Fungi	3.03	0.19	7.23	0.12
GestureMidAirD1	2.63	0.14	1.76	0.04
GestureMidAirD2	2.74	0.15	4.33	0.62
GestureMidAirD3	2.11	0.11	2.38	0.12
GesturePebbleZ1	1.03	0.01	13.17	0.25
GesturePebbleZ2	1.13	0.05	3.85	0.13
GunPointAgeSpan	1.51	0.03	3.69	0.07
GunPointMaleVersusFemale	1.81	0.04	1.31	0.08
GunPointOldVersusYoung	1.96	0.01	0.95	0.24
HouseTwenty	1.32	0.05	9.89	0.22
InsectEPGRegularTrain	2.85	0.05	20.95	1.79
InsectEPGSmallTrain	2.61	0.06	16.73	0.41
MelbournePedestrian	8.49	0.29	21.19	0.86
MixedShapesRegularTrain	3.4	0.11	3.35	0.04
MixedShapesSmallTrain	2.05	0.06	6.02	0.12
PickupGestureWiimoteZ	1.1	0.04	11.41	1.02
PigAirwayPressure	5.4	0.23	13.44	0.06
PigArtPressure	3.68	0.15	12.45	0.16
PigCVP	4.08	0.16	9.65	0.47
PLAID	1.88	0.09	109.08	47.94
PowerCons	1.79	0.03	0.59	0.11
Rock	2.58	0.09	2.76	0.11
SemgHandGenderCh2	1.32	0.03	1.88	0.05
SemgHandMovementCh2	2.32	0.1	9.75	1.49
SemgHandSubjectCh2	2.19	0.09	6.3	0.21
ShakeGestureWiimoteZ	1.47	0.04	1.65	0.05
SmoothSubspace	2.4	0.05	8.39	0.16
UMD	1.8	0.11	7.64	0.41

Table 5: Full IS, FID, TSTR and TRTR results for conditional sampling from the UCR archive with 1024 synthetic time series using 32 sampling steps, $w = 3.0$ and $\gamma = 0.1$.

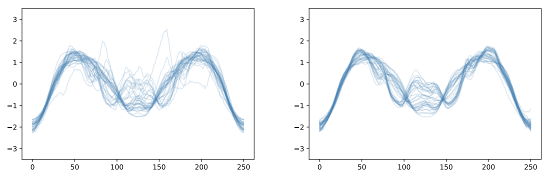
Name	**IS mean** ↑	**FID mean** ↓	**TSTR** (%)	**TRTR** (%)
Adiac	20.68	7.77	67.41	85.42
ArrowHead	2.77	1.41	77.71	80
Beef	4.55	0.62	53.33	73.33
BeetleFly	1.99	0.03	80	80
BirdChicken	1.99	0.03	100	90
Car	2.99	3.7	68.33	91.67
CBF	2.54	7.16	88.64	99
ChlorineConcentration	2.64	1.1	62.89	79.71
CinCECGTorso	2.5	3.02	44	82.46
Coffee	2	5.28	92.86	100
Computers	1.71	1.82	87.2	87.6
CricketX	6.36	1.48	57.46	78.97
CricketY	6.76	1.36	57.46	78.97
CricketZ	5.71	1.56	58.21	77.69
DiatomSizeReduction	3.42	0.42	76	94.12
DistalPhalanxOutlineAgeGroup	2.25	1.24	74.1	71.94
DistalPhalanxOutlineCorrect	1.89	0.98	65	77.54
DistalPhalanxTW	3.66	1.2	66.19	68.35
Earthquakes	1.13	1.24	68.35	73.38
ECG200	1.78	0.54	79	89
ECG5000	2.41	0.48	84.46	94.07

ECGFiveDays	1.8	9.45	52.69	99.19
ElectricDevices	5.33	2.18	48.39	72.56
FaceAll	12.85	0.18	62.34	91.24
FaceFour	2.33	8.8	71.59	90.91
FacesUCR	10.32	0.7	50	93.32
FiftyWords	18.78	1.2	62.81	66.15
Fish	5.55	0.23	93.14	94.86
FordA	1.89	0.18	90	94.47
FordB	1.9	0.38	71.43	79.01
GunPoint	1.79	4.41	86.67	100
Ham	1.86	0.19	67.62	70.48
HandOutlines	1.43	0.12	64.04	90.54
Haptics	2.63	0.72	34.62	49.03
Herring	1.21	1.11	64.06	64.06
InlineSkate	2.33	12.49	31.58	46.18
InsectWingbeatSound	6.04	0.41	30.32	38.94
ItalyPowerDemand	1.99	0.52	80	94.75
LargeKitchenAppliances	2.11	3.52	67.23	90.4
Lightning2	1.65	0.78	67.21	72.13
Lightning7	3.85	3.22	61.64	65.75
Mallat	6.47	0.38	95.12	96.72
Meat	2.74	4.61	76.67	90
MedicalImages	4.56	0.57	63.31	77.24
MiddlePhalanxOutlineAgeGroup	2.46	1	61.04	54.55
MiddlePhalanxOutlineCorrect	1.88	0.93	74.29	82.47
MiddlePhalanxTW	4.26	1.25	52.6	48.7
MoteStrain	1.99	0.04	85.96	91.93
NonInvasiveFetalECGThorax1	24.15	25.4	77.46	93.99
NonInvasiveFetalECGThorax2	22.64	16.17	72.25	93.74
OliveOil	1.07	1.2	43.33	73.33
OSULeaf	4.3	9.07	81.4	96.69
PhalangesOutlinesCorrect	1.85	0.59	82.22	82.4
Phoneme	6.54	3.78	4.81	35.65
Plane	6.5	0.63	100	100
ProximalPhalanxOutlineAgeGroup	2.73	0.97	85.85	82.44
ProximalPhalanxOutlineCorrect	1.85	1.59	80	90.38
ProximalPhalanxTW	3.85	3.25	80.49	79.02
RefrigerationDevices	1.5	32.6	52.94	50.13
ScreenType	2.33	2.06	52.94	64.27
ShapeletSim	1.99	0.73	56.67	85.56
ShapesAll	36.16	0.78	72.73	87.83
SmallKitchenAppliances	1.4	8.26	88.24	78.67
SonyAIBORobotSurface1	1.83	0.04	98.88	95.01
SonyAIBORobotSurface2	1.94	1.37	85.41	97.48
StarLightCurves	2.45	0.34	97.73	96.32
Strawberry	1.91	0.07	92.11	97.84
SwedishLeaf	13.06	0.93	93.81	96.32
Symbols	5.02	1.15	92.51	97.29
SyntheticControl	5.83	2.51	95.45	98.33
ToeSegmentation1	1.7	2.95	67.54	96.05
ToeSegmentation2	1.83	1.72	90	90
Trace	3.69	1.76	90	100
TwoLeadECG	1.97	2.4	100	100
TwoPatterns	3.49	0.52	86.88	86.72
UWaveGestureLibraryAll	5.13	1.29	74.8	83.31
UWaveGestureLibraryX	5.47	2.89	67.72	75.63
UWaveGestureLibraryY	5.27	1.3	56.69	65.91

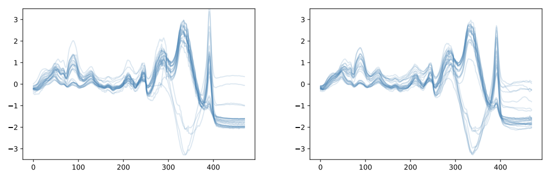
UWaveGestureLibraryZ	5.42	3.1	70.47	73.7
Wafer	1.37	0.34	100	99.66
Wine	1.49	1.55	77.78	74.07
WordSynonyms	3.06	3.03	35.71	49.53
Worms	2.77	1.48	68.83	80.52
WormsTwoClass	1.58	0.16	72.73	79.22
Yoga	1.55	0.39	77.72	87.43
ACSF1	1.33	250.06	49	90
AllGestureWiimoteX	2.84	0.35	37.23	46.71
AllGestureWiimoteY	2.68	0.28	51.06	43.29
AllGestureWiimoteZ	1.06	0.34	24.47	10.71
BME	2.57	18.12	46.67	73.33
Chinatown	1.99	0.16	0	97.96
Crop	22.77	1.58	0	75.39
DodgerLoopDay	1.8	8.86	13.75	43.75
DodgerLoopGame	1.99	0.03	47.83	70.29
DodgerLoopWeekend	1.99	0.03	73.91	96.38
EOGHorizontalSignal	6.57	0.98	8.49	58.29
EOGVerticalSignal	4.85	1.42	16.04	43.92
EthanolLevel	2.11	0.46	51.23	73.6
FreezerRegularTrain	1.96	3.63	79.41	99.12
FreezerSmallTrain	1.79	6.65	76.47	81.68
Fungi	15.87	0.06	12.37	98.39
GestureMidAirD1	3.23	1.05	8.46	20.77
GestureMidAirD2	2.34	11.9	3.85	7.69
GestureMidAirD3	2.22	2.95	6.15	6.15
GesturePebbleZ1	1.54	2.45	17.44	30.81
GesturePebbleZ2	1.87	0.66	15.82	36.71
GunPointAgeSpan	1.65	8.97	0	100
GunPointMaleVersusFemale	1.94	5.29	0	99.68
GunPointOldVersusYoung	1.99	2.23	0	100
HouseTwenty	1.56	6.9	57.98	94.96
InsectEPGRegularTrain	2.88	17.38	47.39	100
InsectEPGSmallTrain	2.61	11.91	35.74	100
MelbournePedestrian	9.29	11.29	14.81	95.74
MixedShapesRegularTrain	4.35	1.67	100	96.74
MixedShapesSmallTrain	4.01	1.62	93.39	92.37
PickupGestureWiimoteZ	1.24	10.56	24	12
PigAirwayPressure	14.68	7.08	2.4	77.88
PigArtPressure	9.72	6.37	1.92	98.08
PigCVP	10.12	15.34	1.92	90.87
PLAID	2.07	316.52	16	12.1
PowerCons	1.92	0.83	50	89.44
Rock	3.47	1.06	18	60
SemgHandGenderCh2	1.56	2.62	19.32	84.17
SemgHandMovementCh2	2.66	11.92	38.66	54.67
SemgHandSubjectCh2	2.49	9.74	14.43	66
ShakeGestureWiimoteZ	1.49	0.42	26	14
SmoothSubspace	2.81	2.58	34.67	98
UMD	2.35	4.58	66.67	96.53



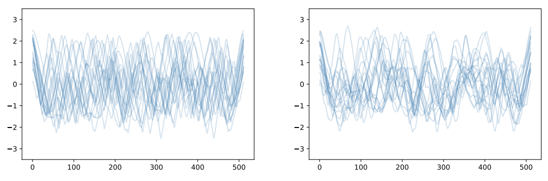
(a) Adiac



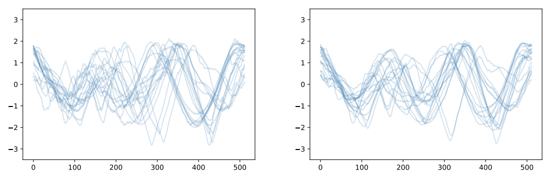
(b) ArrowHead



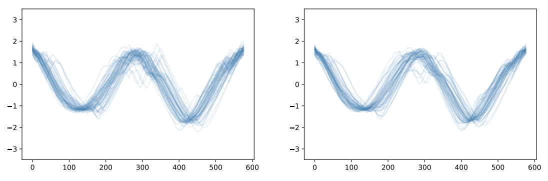
(c) Beef



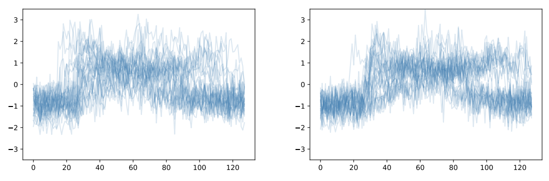
(d) BeetleFly



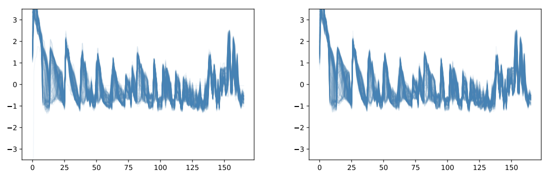
(e) BirdChicken



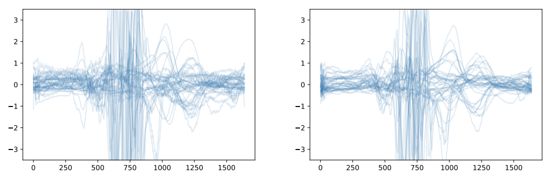
(f) Car



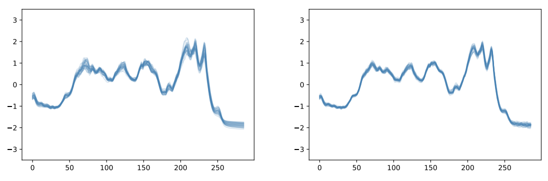
(g) CBF



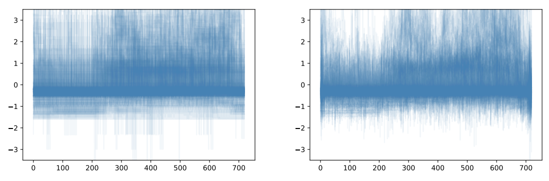
(h) ChlorineConcentration



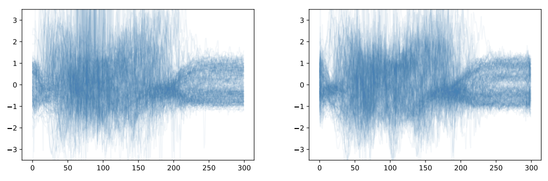
(i) CinCECGTorso



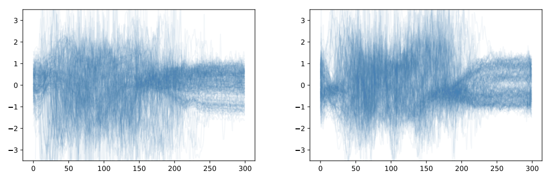
(j) Coffee



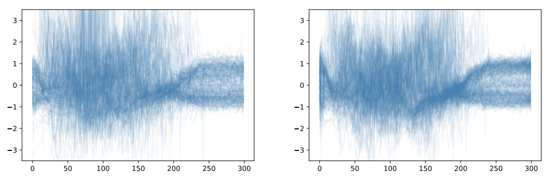
(k) Computers



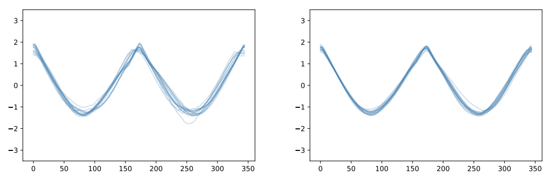
(l) CricketX



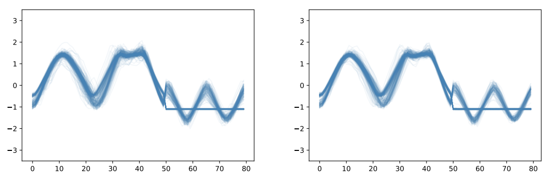
(m) CricetY



(n) CricetZ

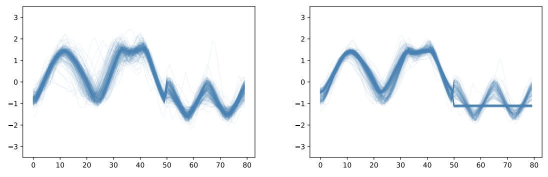


(o) DiatomSizeReduction

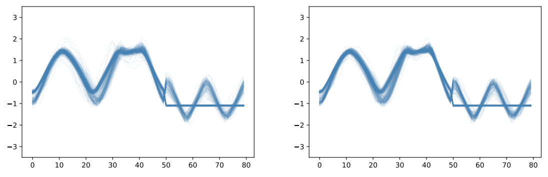


(p) DistalPhalanxOutlineAgeGroup

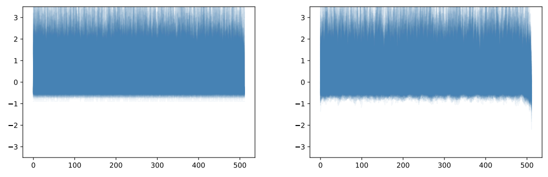
Figure 17: Real / synthetic samples from UCR data sets 1 to 16.



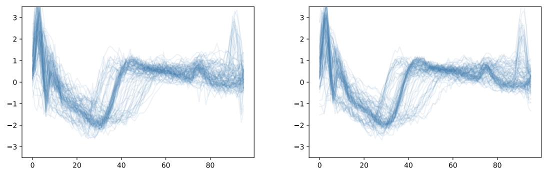
(a) DistalPhalanxOutlineCorrect



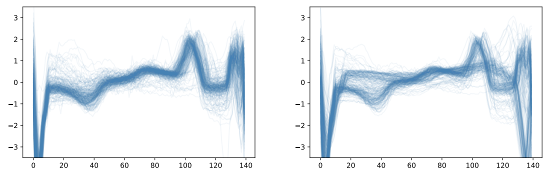
(b) DistalPhalanxTW



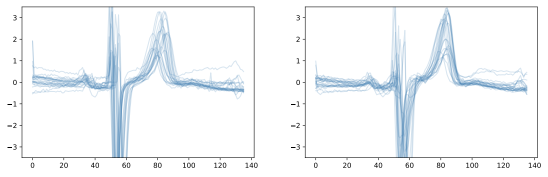
(c) Earthquakes



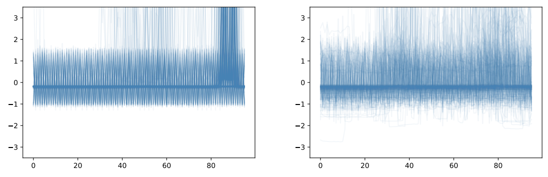
(d) ECG200



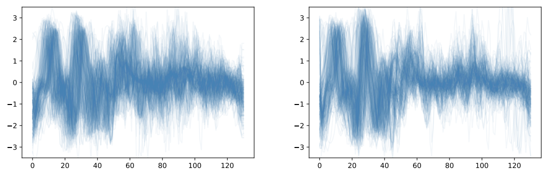
(e) ECG5000



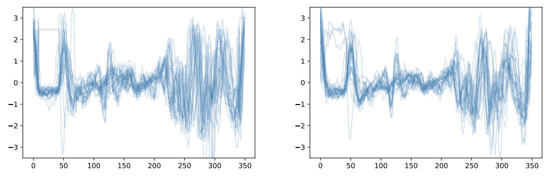
(f) ECGFiveDays



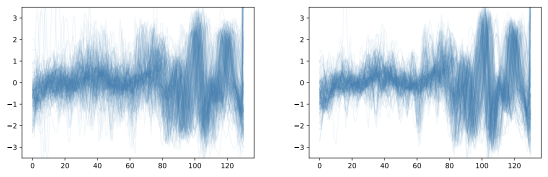
(g) ElectricDevices



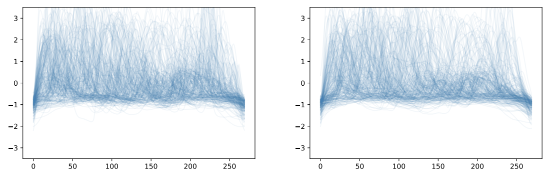
(h) FaceAll



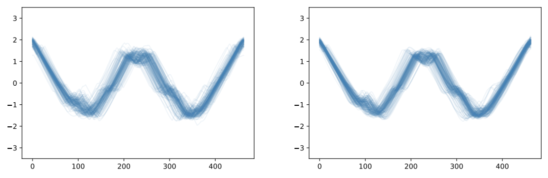
(i) FaceFour



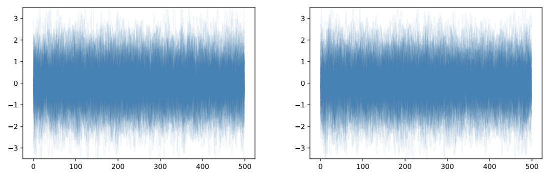
(j) FacesUCR



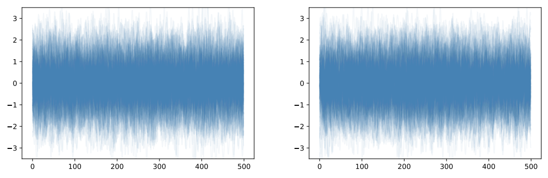
(k) FiftyWords



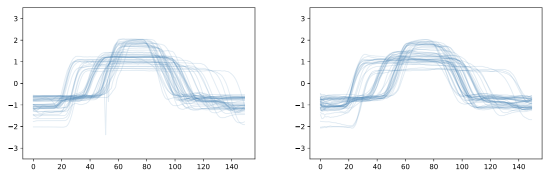
(l) Fish



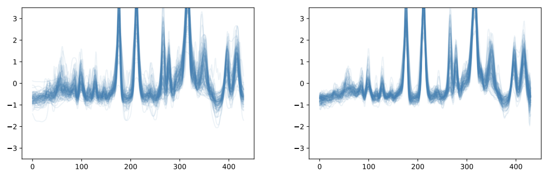
(m) FordA



(n) FordB

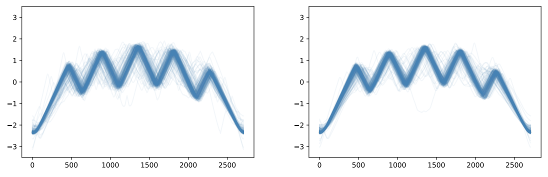


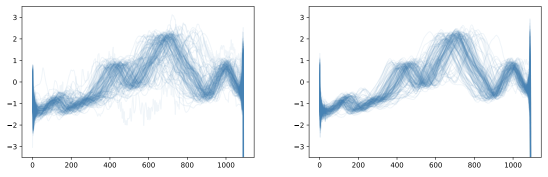
(o) GunPoint

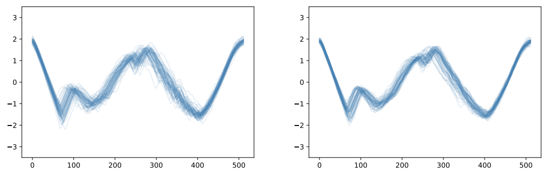


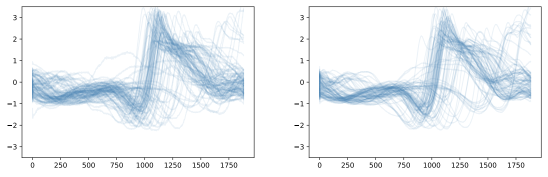
(p) Ham

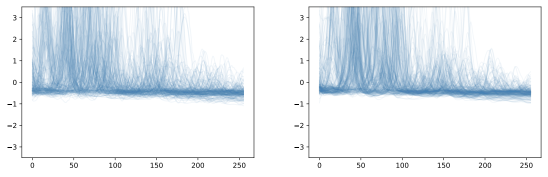
Figure 18: Real / synthetic samples from UCR data sets 17 to 32.

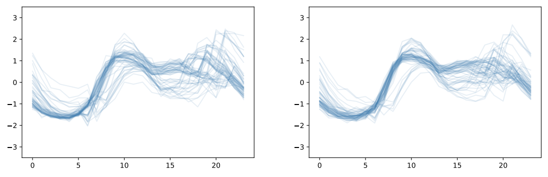

(a) HandOutlines

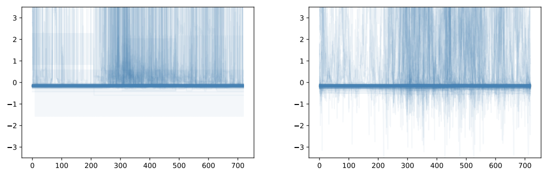

(b) Haptics

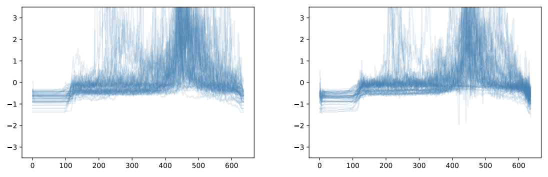

(c) Herring

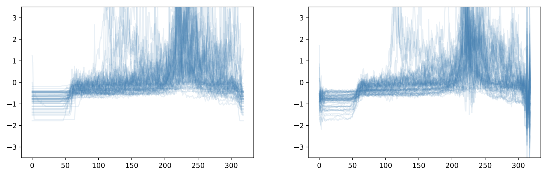

(d) InlineSkate

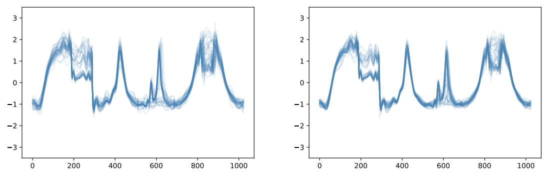

(e) InsectWingbeatSound

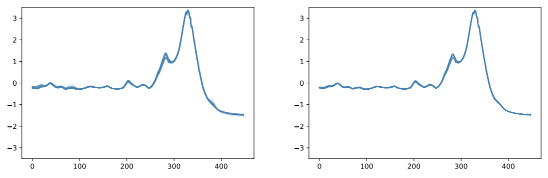

(f) ItalyPowerDemand

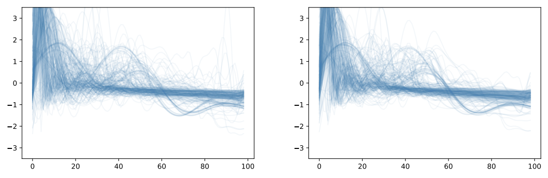

(g) LargeKitchenAppliances

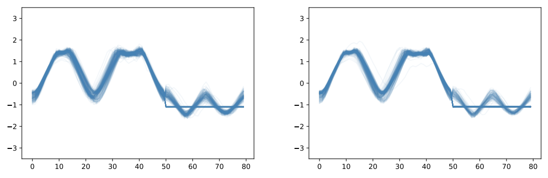

(h) Lightning2

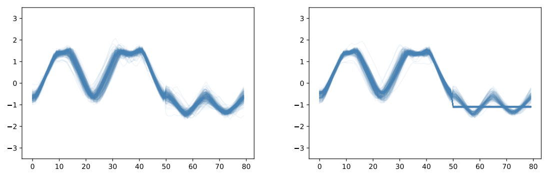

(i) Lightning7

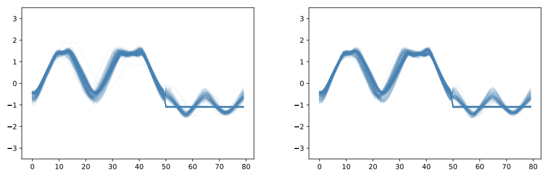

(j) Mallat


(k) Meat


(l) MedicalImages


(m) MiddlePhalanxOutlineAgeGroup


(n) MiddlePhalanxOutlineCorrect


(o) MiddlePhalanxTW

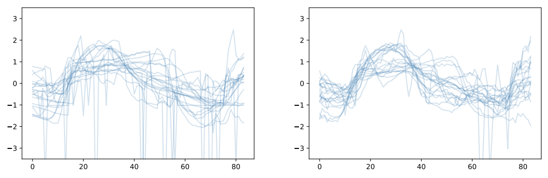
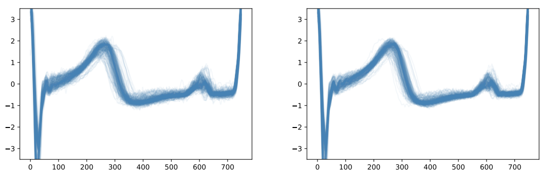
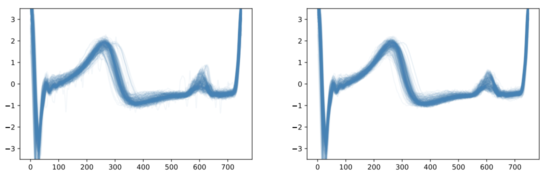
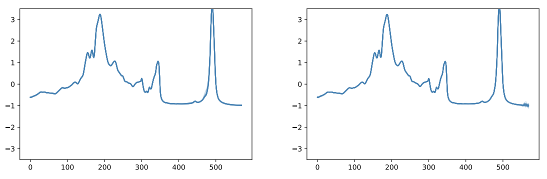
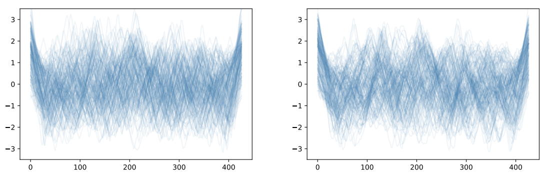

(p) MoteStrain

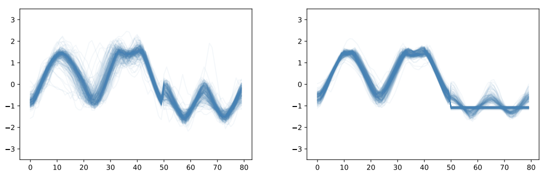
Figure 19: Real / synthetic samples from UCR data sets 33 to 48.

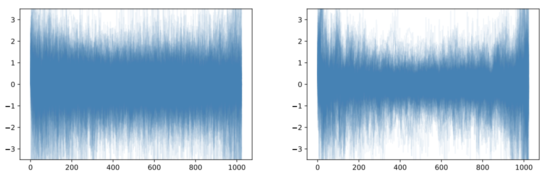

(a) NonInvasiveFetalECGThorax1

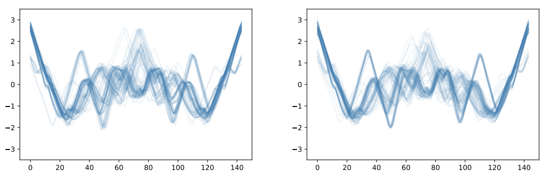

(b) NonInvasiveFetalECGThorax2

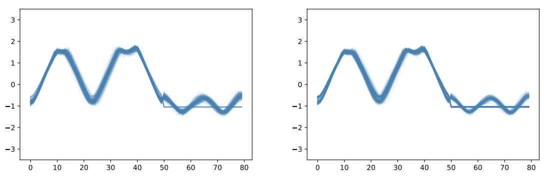

(c) OliveOil

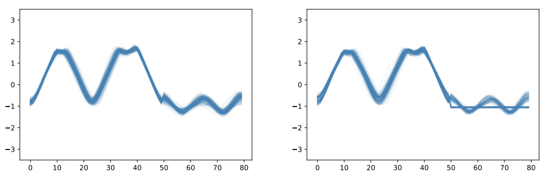

(d) OSULeaf

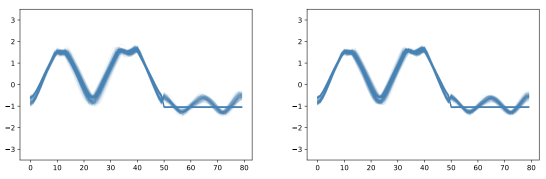

(e) PhalangesOutlinesCorrect

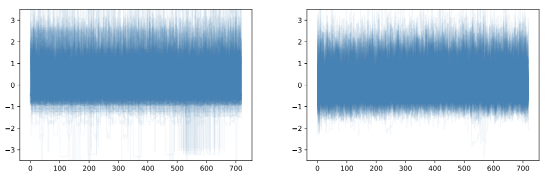

(f) Phonome

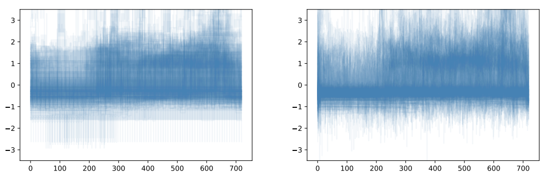

(g) Plane

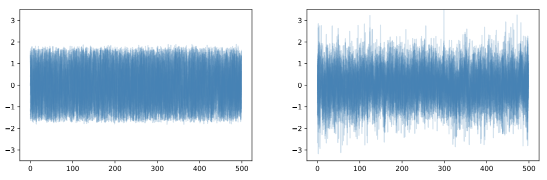

(h) ProximalPhalanxOutlineAgeGroup

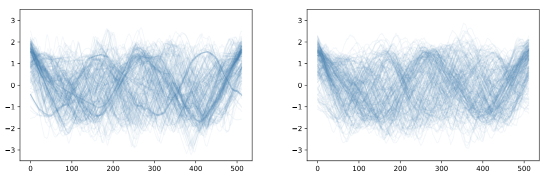

(i) ProximalPhalanxOutlineCorrect

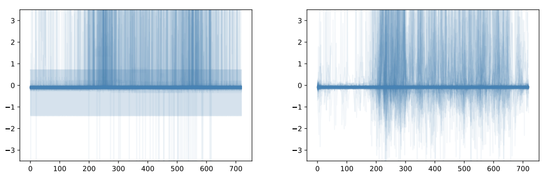

(j) ProximalPhalanxTW


(k) RefrigerationDevices


(l) ScreenType


(m) ShapeletSim


(n) ShapesAll


(o) SmallKitchenAppliances

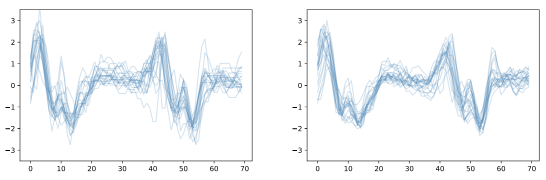
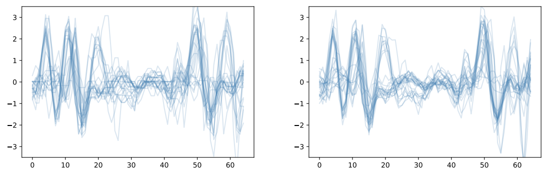
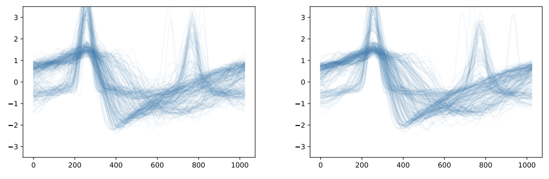
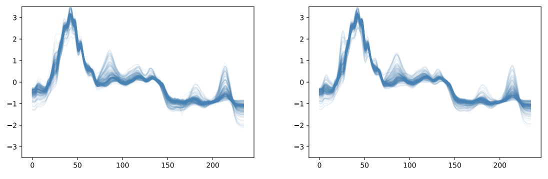
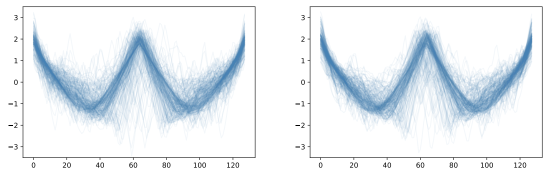

(p) SonyAIBORobotSurface1

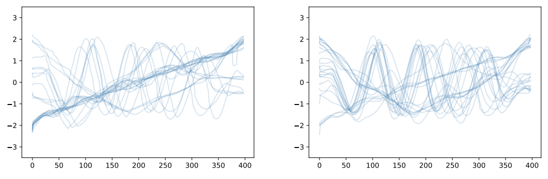
Figure 20: Real / synthetic samples from UCR data sets 49 to 64.

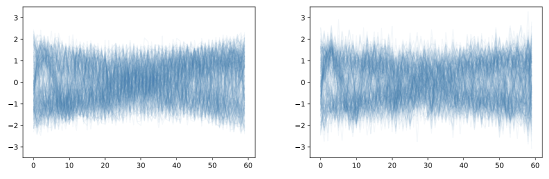

(a) SonyAIBORobotSurface2

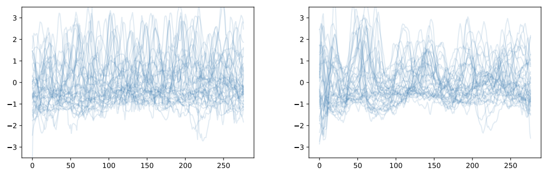

(b) StarLightCurves

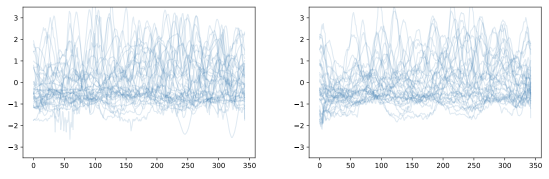

(c) Strawberry

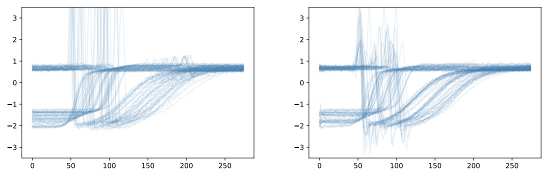

(d) SwedishLeaf

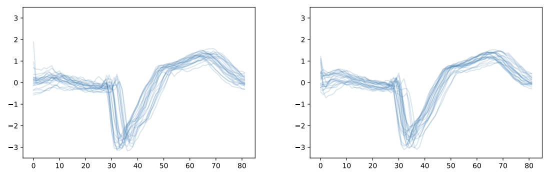

(e) Symbols

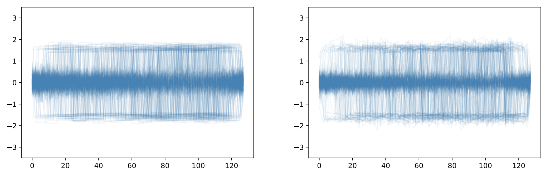

(f) SyntheticControl

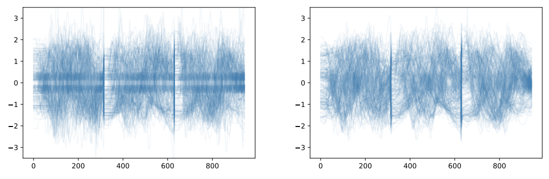

(g) ToeSegmentation1

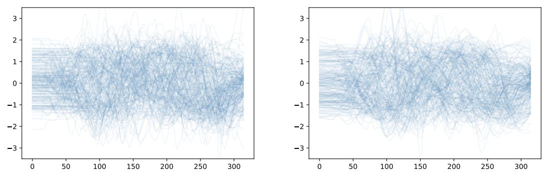

(h) ToeSegmentation2

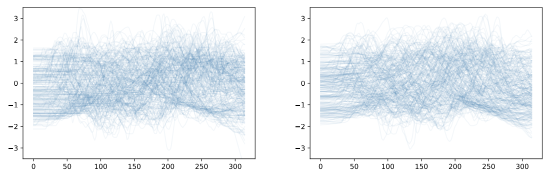

(i) Trace

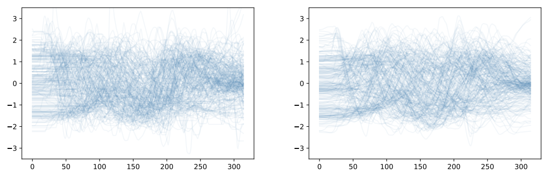

(j) TwoLeadECG


(k) TwoPatterns


(l) UWaveGestureLibraryAll


(m) UWaveGestureLibraryX


(n) UWaveGestureLibraryY


(o) UWaveGestureLibraryZ

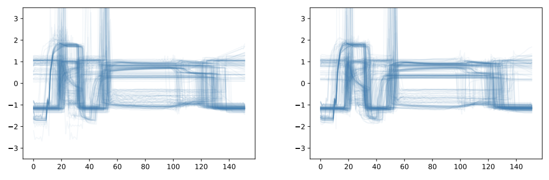
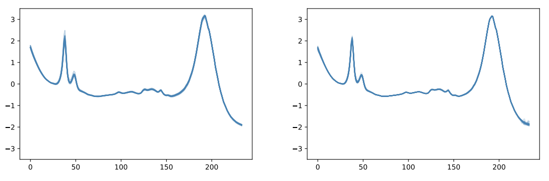
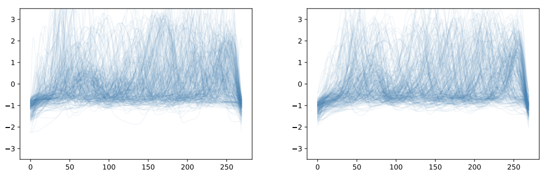

(p) Wafer

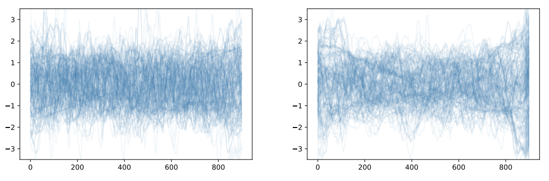
Figure 21: Real / synthetic samples from UCR data sets 65 to 80.



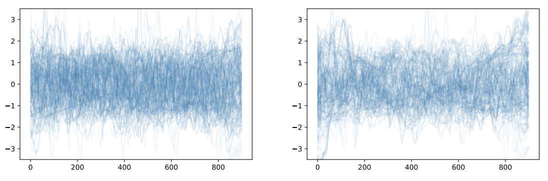
(a) Wine



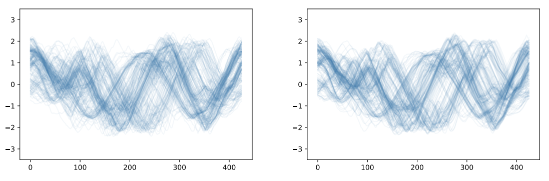
(b) WordSynonyms



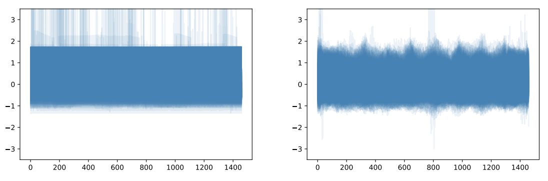
(c) Worms



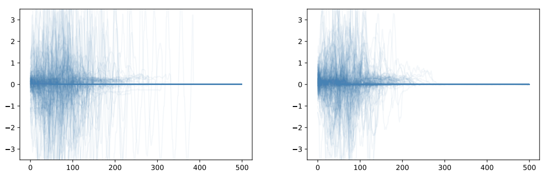
(d) WormsTwoClass



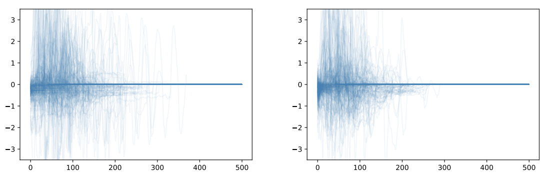
(e) Yoga



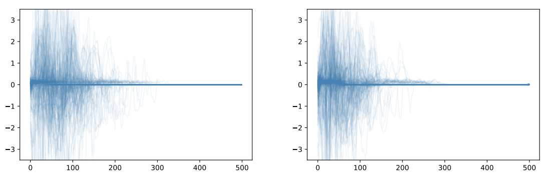
(f) ACSF1



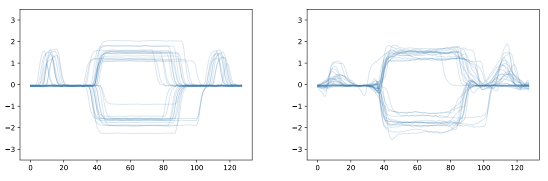
(g) AllGestureWiimoteX



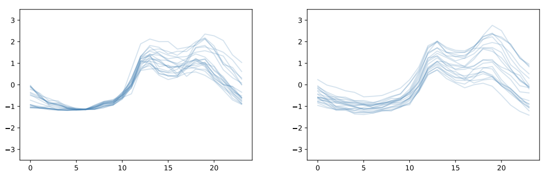
(h) AllGestureWiimoteY



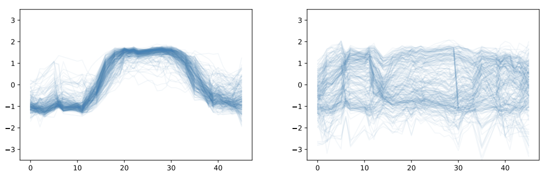
(i) AllGestureWiimoteZ



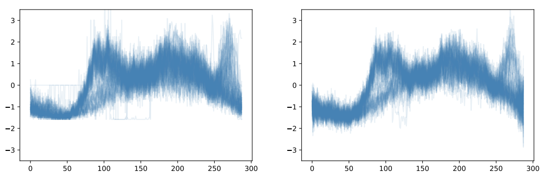
(j) BME



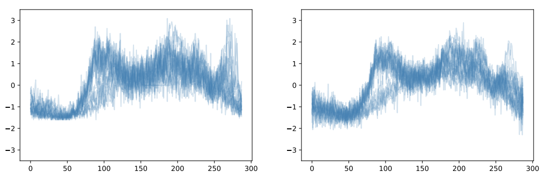
(k) Chinatown



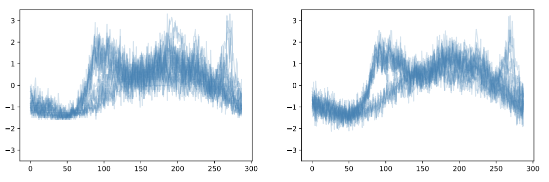
(l) Crop



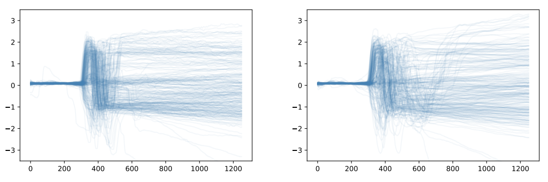
(m) DodgerLoopDay



(n) DodgerLoopGame

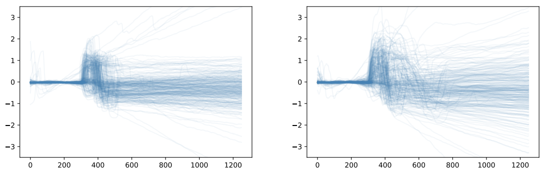


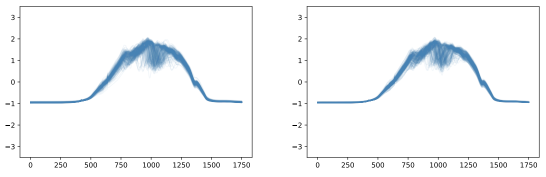
(o) DodgerLoopWeekend

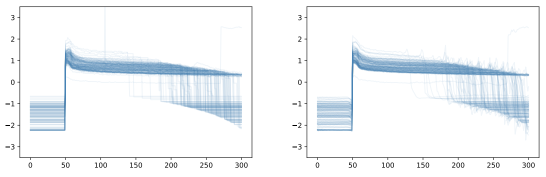


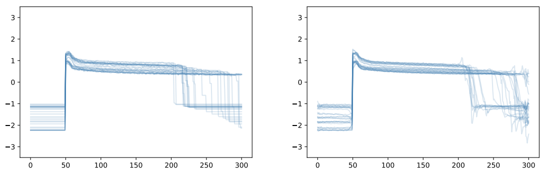
(p) EOGHorizontalSignal

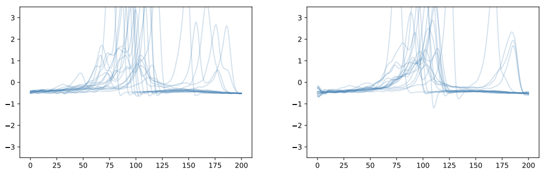
Figure 22: Real / synthetic samples from UCR data sets 81 to 96.

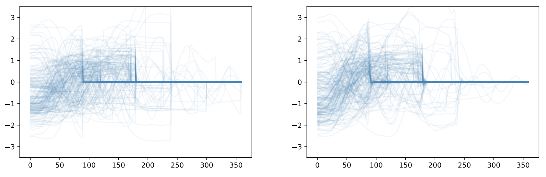

(a) EOGVerticalSignal

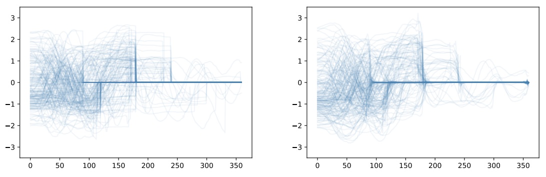

(b) EthanolLevel

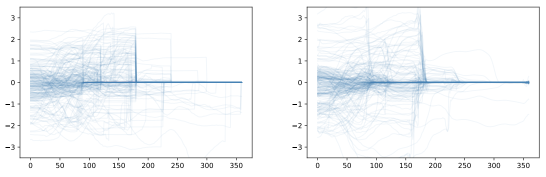

(c) FreezerRegularTrain

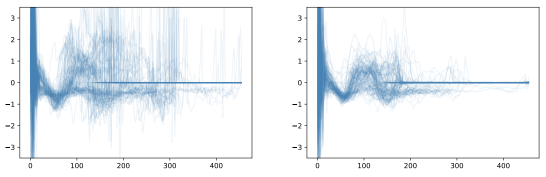

(d) FreezerSmallTrain

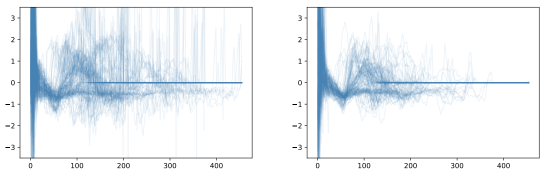

(e) Fungi

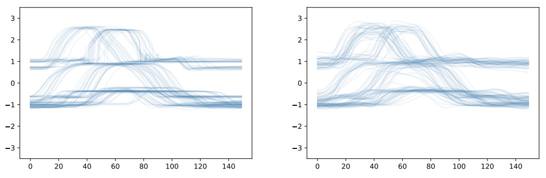

(f) GestureMidAirD1

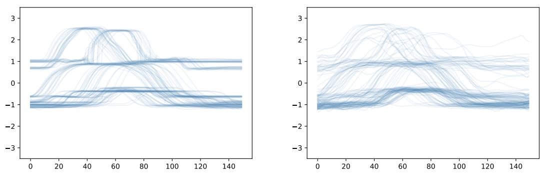

(g) GestureMidAirD2

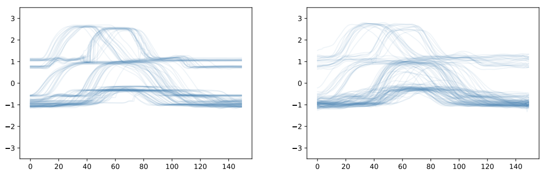

(h) GestureMidAirD3

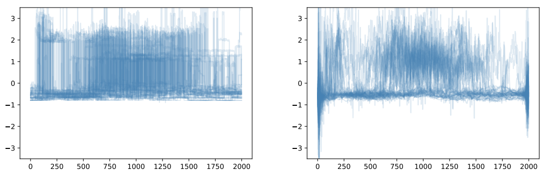

(i) GesturePebbleZ1

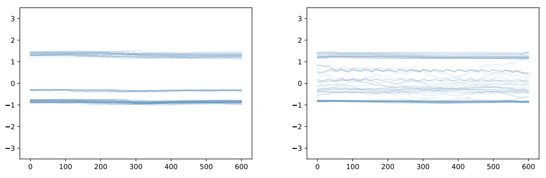

(j) GesturePebbleZ2


(k) GunPointAgeSpan


(l) GunPointMaleVersusFemale


(m) GunPointOldVersusYoung


(n) HouseTwenty


(o) InsectEPGRegularTrain

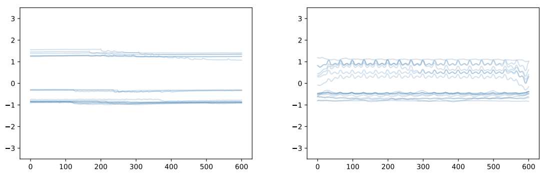
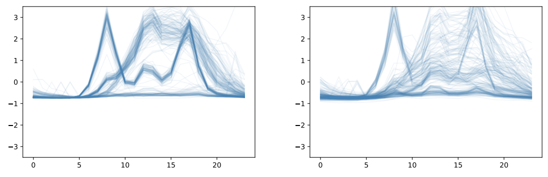
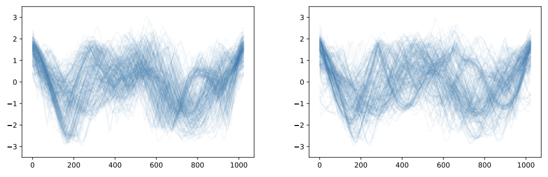
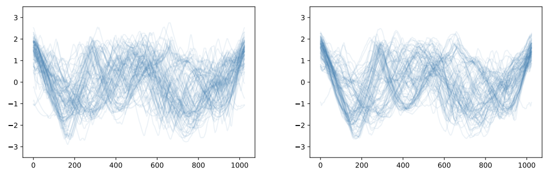
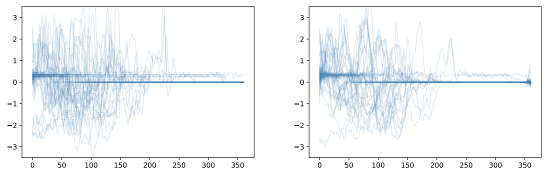

(p) InsectEPGSmallTrain

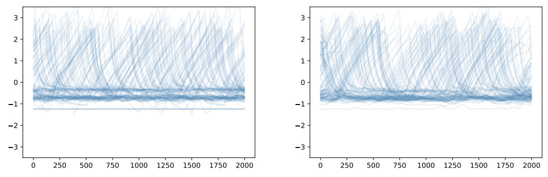
Figure 23: Real / synthetic samples from UCR data sets 97 to 112.

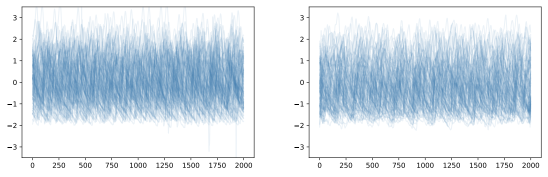

(a) MelbournePedestrian

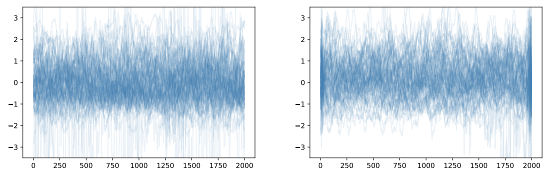

(b) MixedShapesRegularTrain

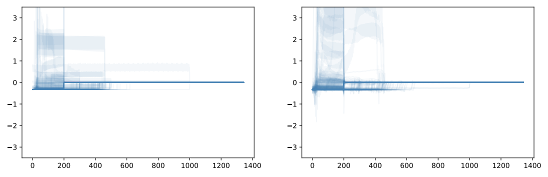

(c) MixedShapesSmallTrain

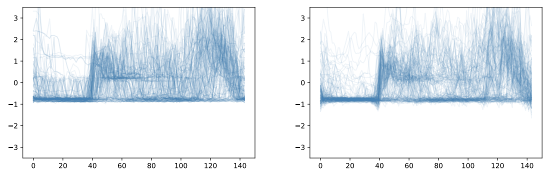

(d) PickupGestureWiimoteZ

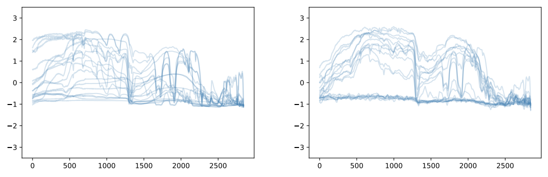

(e) PigAirwayPressure

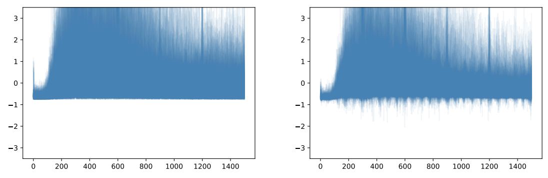

(f) PigArtPressure

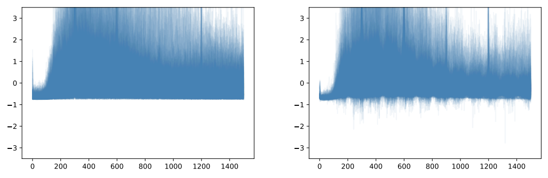

(g) PigCVP

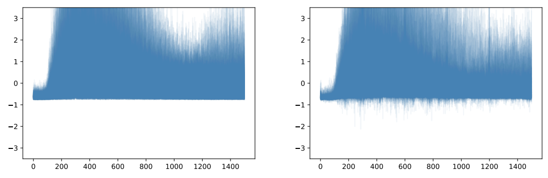

(h) PLAID

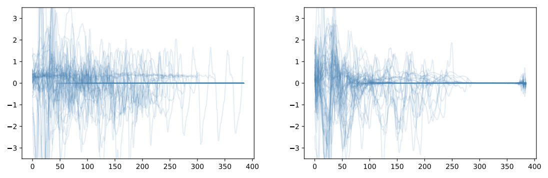

(i) PowerCons

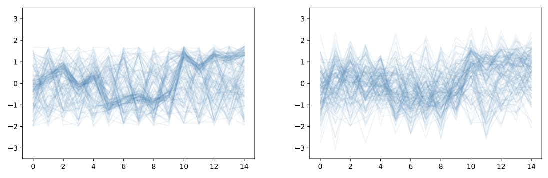

(j) Rock


(k) SemgHandGenderCh2


(l) SemgHandMovementCh2


(m) SemgHandSubjectCh2


(n) ShakeGestureWiimoteZ


(o) SmoothSubspace

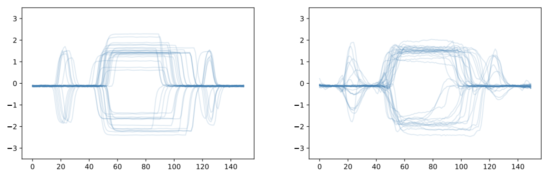

(p) UMD

Figure 24: Real / synthetic samples from UCR data sets 113 to 128.

5 Discussion

5.1 Qualitative assessment of generated samples

Continuous signals vs. digital-like signals

A visual inspection of generated time series from the UCR archive with real samples is displayed in Figure 17 to Figure 24. By comparing the run charts, TimeVQVDM seems to be able to learn the marginal distribution to some degree for most data sets in the UCR time series archive. The model seems to be good for learning the distribution of motion-derived data and continuously varying signals such as InsectWingBeatSiund, StarlightCurves, OSULeaf, WordSynonyms, InlineSkate, MedicalImages and the UWaveGestureLibrary-datasets. The model seems worse at learning the distribution of device-data and digital-like signals, such as EarthQuakes, ElectricalDevices, Computers, LargeKitchenAppliances, SmallKitchenAppliances and ACSF1. This trend is also observed for the generated samples in TimeVQVAE. One reason for this could be that the digital-like time series have little to no noise and the range of the time series is not continuous. In the theory section, it was assumed that the range of the latents for the diffusion models are continuous. However, many of the displayed training data sets have clear bands. For instance, the data set UWaveGestureLibraryAll have clear horizontal lines which are not so profound in the generated samples. The most clear example is in SmallKitchenAppliances. The training data have clear lower and upper bands for the time series which TimeVQVDM fails to capture.

Poor mode coverage for some data sets

For some data sets TimeVQVDM generates time series that doesn't resemble the modes of the distribution in the training data. For data sets such as Crop, Chinatown, EOGVerticalSignal and PLAID, the generated samples resembles the training data, but the variability of the distribution is higher. Other data sets such as GesturePebbleZ1 and GesturePebbleZ2, the variability of the generated time series are lower. And lastly, some data sets such as MelbournePedestrian, InsectEPGRegularTrain and InsectEPGSmallTrain, the generated time series seem to lie in-between modes in the training data. This is clearly visible in InsectEPGRegularTrain, as the training data has clearly separated modes while the generated data have modes in the middle. One reason for the poor mode coverage could be to a low number of training samples. For instance, Chinatown has only 20 training samples.

Trouble with representing end points

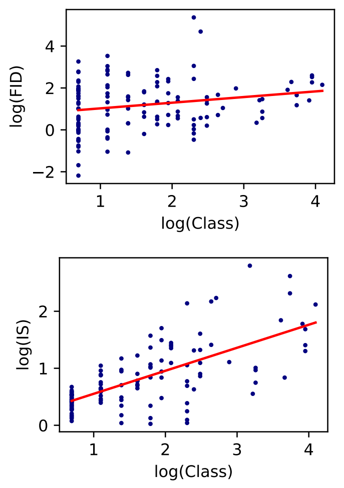
Some of the generated samples have shows bad representation at the endpoints. For instance, generated samples from HouseTwenty have high variability on both end points, in contrast to the training data, which are mostly flat at the two end points. The right-most end points seems to be the hardest, as some data sets only struggle with representing the it. For instance GestureMidAirD2, GestureMidAirD3, Lightning2 and Lightning7 all have some variability at the right end points that should not be there. The reason for this could be from how TimeVQVDM is implemented. Because of the down-sampling procedure using convolutions with `stride=2`, the right-most time step is removed if the time series has an odd length. Then, when up-sampling, this can lead to higher difficulty reconstructing the right-most time steps of the signals.

Alignment of low and high frequencies

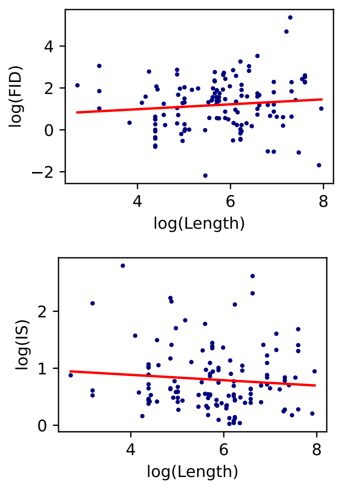
For many data sets, such as Wafer, TwoPatters and OSULeaf, the low and high frequencies seem to align well. For other data sets, such as Trace, FreezerREgularTRain and FreezerSmallTrain, the high-frequencies sometimes overshoots in sharp changes of modularity. This issue could potentially be fixed by implementing ways to identify miss-aligned frequencies in LF and HF and fix them accordingly.

5.2 Quantitative assessment of generated samples

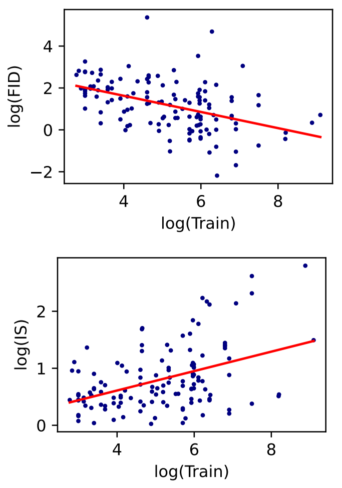
Table 4 and Table 5 gives the full results for both unconditional and conditional sampling on all the data sets in the UCR time series archive. To get a better feel for the relative performance of TimeVQVDM on the different data sets, the evaluation metrics are plotted together with the number of classes, the length of time series and the number of training samples. Figure 25 compares the scores for unconditional sampling on all data sets in the UCR time series archive. Similarly, 26 compares for conditional sampling while Figure 27 compares the classification accuracy for training a FCN model on synthetic data and testing it on real data.



(a) Number of classes.

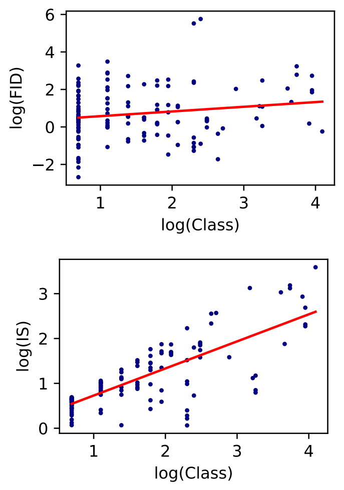


(b) Length of time series.

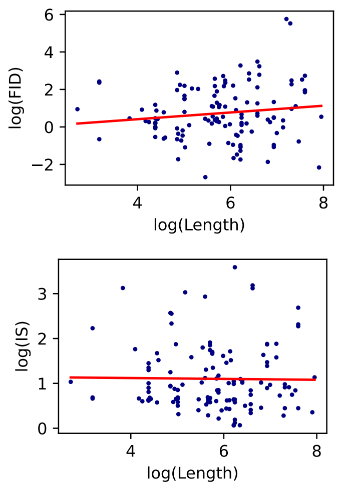


(c) Number of training samples.

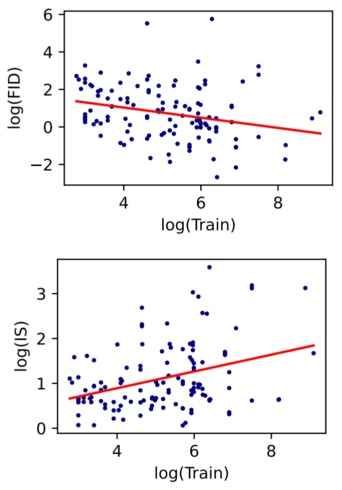
Figure 25: Log-log plots comparing FID and IS metrics for unconditional sampling relative to number of classes (25a), length of time series (25b) and number of training samples (25c). The plots include the best-fit regression line.



(a) Number of classes.

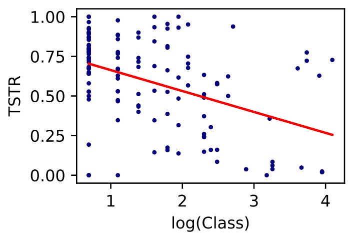


(b) Length of time series.

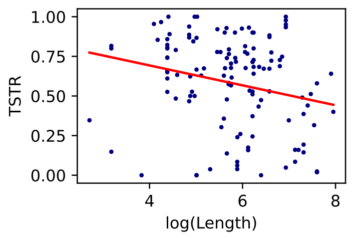


(c) Number of training samples.

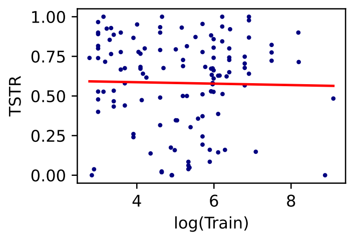
Figure 26: Log-log plots comparing FID and IS metrics for conditional sampling relative to number of classes (26a), length of time series (26b) and number of training samples (26c). The plots include the best-fit regression line.



(a) Number of classes.



(b) Length of time series.



(c) Number of training samples.

Figure 27: Log plots comparing TSTR relative to number of classes (27a), length of time series (27b) and number of training samples (27c). The plots include the best-fit regression line.

Bad performance for few training samples

From Figure 25, it seems like data sets with few training samples has higher FID scores and lower IS scores. This could be because the number of epochs is fixed at 1000, and the models don't have enough time to converge properly. We noticed that the TimeVQVAE model used 5000 maximum epochs for learning the bidirectional transformer. In this thesis, training was stopped at 1000 epochs for the diffusion models. Increasing the number of epochs could allow for learning the distributions better. However, increasing the number of training stages could lead to over-fitting to the data. This, in turn, could make the model very likely to produces almost exact replicas from the training data, creating a high bias towards it. Thus the generative models starts to approximate the empirical distribution, and as a result memorizes all the training data within the parameters of the model. To mitigate this, there should be an evaluation metric for measuring the ability to generalize from the training data, which is lacking in current evaluation protocols. Using the empirical distribution could be better is some regards when the number of training samples is very low.

Effect of halting the diffusion process

We found that halting the diffusion process at $t = 0.99$ lead to improvements in FID scores. However, (Chen, 2023) outlines that stopping the diffusion process before the terminal SNR becomes zero can lead to lower mode coverage. This is seen with a medium brightness problem in Stable Diffusion (Rombach et al., 2021). Stable diffusion has a terminal SNR of $\text{SNR}_{\text{min}} = 4.682 \cdot 10^{-3}$, which in terms of the variance conserving diffusion process corresponds to halting the diffusion process at $t = 0.956$. Thus using the halting procedure in this thesis is not too unreasonable. However, (Chen, 2023) always recommends picking the velocity-parameterization. With the velocity parameterization, the problem with low SNR vanishes. Future work could investigate using this parameterization instead of the nosie-parameterization.

6 Conclusion

Motivated by recent success in generative modelling using vector quantization and diffusion models, we propose TimeVQVDM for time series generation. Similar to TimeVQVAE (Lee et al., 2023), it uses VQ-VAEs to model time series in the time-frequency domain, which are further split into low and high frequencies. The continuous latent representations are then modelled using two variational diffusion models, one for low frequencies and one for high frequencies. TimeVQVDM generates time series in by first generating the low frequency components and then generating the higher frequencies. Trained using classifier-free guidance, it is capable of both unconditional as well as conditional sampling. For combining diffusion models with vector quantization for time series generation, five key implementation details were identified:

- Low dimensionality of tokens
- High temporal resolution in time-frequency domain
- Standardizing latents before the diffusion models
- Conditioning the HF diffusion model on LF latents
- Halting the diffusion process at $t = 0.99$ to avoid to low signal-to-noise ratio

By evaluating on all 128 data sets in the UCR time series archive, TimeVQVDM receives scores comparable to current state-of-the-art TSG models using transformer networks, such as TimeVQVAE. For unconditional sampling, TimeVQVDM achieves an average FID score of 7.76, beating the FID average of 11.44 by TimeVQVAE. This goes to show that switching to diffusion models for time series generation can lead to improvement gains. In particular, this the thesis identified that the diffusion architecture was especially useful for continuously varying time series, where the signals arise from continuous, rather than discrete, measurements. Additionally, it was able to generalize better for data sets with a high number of training samples.

Future work into TSG with diffusion model could explore the codebook utilizations, and investigate the number of tokens in the model. Additionally, the frequency bands for different data sets varies a lot. Some data sets have very little LF, while other data sets have very much LF. In other data sets, the HF dominates the LF. The implementation for the STFT uses a fixed-length window, so resolutions of the frequency-time domain is fixed. This can perhaps be fixed using a the Wavelet transform or multi-scale STFT, as in (Kumar et al., 2023).

Bibliography

Anderson, Brian D.O. (May 1982). 'Reverse-time diffusion equation models'. In: *Stochastic Processes and their Applications* 12 (3), pp. 313–326. ISSN: 0304-4149. DOI: 10.1016/0304-4149(82)90051-5.

Bank, Dor, Noam Koenigstein and Raja Giryes (Mar. 2020). 'Autoencoders'. In: URL: http://arxiv.org/abs/2003.05991.

Blei, David M., Alp Kucukelbir and Jon D. McAuliffe (Jan. 2016). 'Variational Inference: A Review for Statisticians'. In: *Journal of the American Statistical Association* 112 (518), pp. 859–877. DOI: 10.1080/01621459.2017.1285773. URL: http://arxiv.org/abs/1601.00670%20http://dx.doi.org/10.1080/01621459.2017.1285773.

Borji, Ali (Mar. 2021). 'Pros and Cons of GAN Evaluation Measures: New Developments'. In: URL: http://arxiv.org/abs/2103.09396.

Box, G. E. P., G. M. Jenkins and J. F. MacGregor (1974). 'Some Recent Advances in Forecasting and Control'. In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 23.2, pp. 158–179. ISSN: 00359254, 14679876. URL: http://www.jstor.org/stable/2346997 (visited on 15th July 2023).

Brophy, Eoin et al. (July 2021). 'Generative adversarial networks in time series: A survey and taxonomy'. In: URL: http://arxiv.org/abs/2107.11098.

Chang, Huiwen et al. (Feb. 2022). 'MaskGIT: Masked Generative Image Transformer'. In: DOI: 10.48550/arxiv.2202.04200. URL: https://arxiv.org/abs/2202.04200v1.

Chen, Ting (Jan. 2023). 'On the Importance of Noise Scheduling for Diffusion Models'. In: DOI: 10.48550/arxiv.2301.10972. URL: https://arxiv.org/abs/2301.10972.

Cohen, Max et al. (Feb. 2022). 'Diffusion bridges vector quantized Variational AutoEncoders'. In: URL: http://arxiv.org/abs/2202.04895.

Crowson, Katherine and Chainbreakers AI (2021). *v-diffusion-jax*. https://github.com/crowsonkb/v-diffusion-jax.

Dau, Hoang Anh et al. (Oct. 2018). 'The UCR Time Series Archive'. In: *IEEE/CAA Journal of Automatica Sinica* 6 (6), pp. 1293–1305. ISSN: 23299274. DOI: 10.48550/arxiv.1810.07758. URL: https://arxiv.org/abs/1810.07758v2.

Défossez, Alexandre et al. (Aug. 2022). 'Decoding speech from non-invasive brain recordings'. In: URL: http://arxiv.org/abs/2208.12266.

Dhariwal, Prafulla, Heewoo Jun et al. (Apr. 2020). 'Jukebox: A Generative Model for Music'. In: URL: http://arxiv.org/abs/2005.00341.

Dhariwal, Prafulla and Alex Nichol (May 2021). 'Diffusion Models Beat GANs on Image Synthesis'. In: *Advances in Neural Information Processing Systems* 11, pp. 8780–8794. ISSN: 10495258. DOI: 10.48550/arxiv.2105.05233. URL: https://arxiv.org/abs/2105.05233v4.

Dowson, D. C. and B. V. Landau (Sept. 1982). 'The Fréchet distance between multivariate normal distributions'. In: *Journal of Multivariate Analysis* 12 (3), pp. 450–455. ISSN: 0047-259X. DOI: 10.1016/0047-259X(82)90077-X.

Engle, Robert F. (1982). 'Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation'. In: *Econometrica* 50.4, pp. 987–1007. ISSN: 00129682, 14680262. URL: http://www.jstor.org/stable/1912773 (visited on 15th July 2023).

Geman, Stuart and Donald Geman (1984). *Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images*, pp. 721–741.

Goodfellow, Ian J. et al. (June 2014). 'Generative Adversarial Networks'. In: DOI: 10.48550/arxiv.1406.2661. URL: http://arxiv.org/abs/1406.2661.

Google, Research (2022). *Variational Diffusion Models*. https://github.com/google-research/vdm.

Gray, Robert M (Apr. 1984). 'Vector Quantization'. In: *IEEE ASSP Magazine* 1 (2), pp. 4–29.

Hastings, W K (1970). *Monte Carlo Sampling Methods Using Markov Chains and Their Applications*, pp. 97–109.

He, Kaiming et al. (Dec. 2015). 'Deep Residual Learning for Image Recognition'. In: URL: http://arxiv.org/abs/1512.03385.

Heusel, Martin et al. (June 2017). 'GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium'. In: DOI: 10.48550/arxiv.1706.08500. URL: http://arxiv.org/abs/1706.08500.

Ho, Jonathan, William Chan et al. (Oct. 2022). 'Imagen Video: High Definition Video Generation with Diffusion Models'. In: DOI: 10.48550/arxiv.2210.02303. URL: https://arxiv.org/abs/2210.02303v1.

Ho, Jonathan, Ajay Jain and Pieter Abbeel (June 2020). 'Denoising Diffusion Probabilistic Models'. In: *Advances in Neural Information Processing Systems* 2020-December. ISSN: 10495258. DOI: 10.48550/arxiv.2006.11239. URL: https://arxiv.org/abs/2006.11239v2.

Ho, Jonathan, Chitwan Saharia et al. (May 2021). 'Cascaded Diffusion Models for High Fidelity Image Generation'. In: DOI: 10.48550/arxiv.2106.15282. URL: http://arxiv.org/abs/2106.15282.

Ho, Jonathan and Tim Salimans (July 2022). 'Classifier-Free Diffusion Guidance'. In: DOI: 10.48550/arxiv.2207.12598. URL: http://arxiv.org/abs/2207.12598.

Huang, Yu-Hsiang, Kan Huang and Pedro Diamel Marrero Fernández (2021). *CVAE and VQ-VAE*. https://github.com/crowsonkb/v-diffusion-jax.

Katharopoulos, Angelos et al. (June 2020). 'Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention'. In: URL: http://arxiv.org/abs/2006.16236.

Kingma, Diederik P and Max Welling (Dec. 2013). 'Auto-Encoding Variational Bayes'. In: DOI: 10.48550/arxiv.1312.6114. URL: http://arxiv.org/abs/1312.6114.

Kingma, Diederik P. et al. (July 2021). 'Variational Diffusion Models'. In: *Advances in Neural Information Processing Systems* 26, pp. 21696–21707. ISSN: 10495258. DOI: 10.48550/arxiv.2107.00630. URL: https://arxiv.org/abs/2107.00630v4.

Kumar, Rithesh et al. (2023). 'High-Fidelity Audio Compression with Improved RVQGAN — Papers With Code'. In: URL: https://paperswithcode.com/paper/high-fidelity-audio-compression-with-improved.

Lee, Daesoo, Sara Malacarne and Erlend Aune (2023). 'Vector Quantized Time Series Generation with a Bidirectional Prior Model'. In: *International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 7665–7693.

Li, Xiaomin, Anne Hee Hiong Ngu and Vangelis Metsis (June 2022). 'TTS-CGAN: A Transformer Time-Series Conditional GAN for Biosignal Data Augmentation'. In: URL: http://arxiv.org/abs/2206.13676.

Lin, Shanchuan et al. (May 2023). 'Common Diffusion Noise Schedules and Sample Steps are Flawed'. In: URL: http://arxiv.org/abs/2305.08891.

Liu, Yiheng et al. (Apr. 2023). 'Summary of ChatGPT/GPT-4 Research and Perspective Towards the Future of Large Language Models'. In: URL: http://arxiv.org/abs/2304.01852.

Loshchilov, Ilya and Frank Hutter (Nov. 2017). 'Decoupled Weight Decay Regularization'. In: URL: http://arxiv.org/abs/1711.05101.

Ni, Hao et al. (June 2020). 'Conditional Sig-Wasserstein GANs for Time Series Generation'. In: URL: http://arxiv.org/abs/2006.05421.

Nichol, Alex et al. (Dec. 2021). 'GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models'. In: DOI: 10.48550/arxiv.2112.10741. URL: https://arxiv.org/abs/2112.10741v3.

Nicholas, Metropolis et al. (1953). ‘Equation of State Calculations by Fast Computing Machines’. In: *The Journal of Chemical Physics*, pp. 1087–1092.

Oord, Aaron Van Den, Oriol Vinyals and Koray Kavukcuoglu (Nov. 2017). ‘Neural Discrete Representation Learning’. In: *Advances in Neural Information Processing Systems* 2017-December, pp. 6307–6316. ISSN: 10495258. DOI: 10.48550/arxiv.1711.00937. URL: https://arxiv.org/abs/1711.00937v2.

Oppenlaender, Jonas (Nov. 2022). ‘The Creativity of Text-to-Image Generation’. In: Association for Computing Machinery, pp. 192–202. ISBN: 9781450399555. DOI: 10.1145/3569219.3569352.

Qiao, Siyuan et al. (Mar. 2019). ‘Micro-Batch Training with Batch-Channel Normalization and Weight Standardization’. In: URL: http://arxiv.org/abs/1903.10520.

Ramesh, Aditya et al. (Apr. 2022). ‘Hierarchical Text-Conditional Image Generation with CLIP Latents’. In: URL: http://arxiv.org/abs/2204.06125.

Rombach, Robin et al. (Dec. 2021). ‘High-Resolution Image Synthesis with Latent Diffusion Models’. In: pp. 10674–10685. ISSN: 10636919. DOI: 10.48550/arxiv.2112.10752. URL: https://arxiv.org/abs/2112.10752v2.

Ronneberger, Olaf, Philipp Fischer and Thomas Brox (May 2015). ‘U-Net: Convolutional Networks for Biomedical Image Segmentation’. In: 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’. In: DOI: 10.48550/arxiv.2205.11487. URL: https://arxiv.org/abs/2205.11487v1.

Salimans, Tim, Ian Goodfellow et al. (June 2016). ‘Improved Techniques for Training GANs’. In: URL: http://arxiv.org/abs/1606.03498.

Salimans, Tim and Jonathan Ho (Feb. 2022). ‘Progressive Distillation for Fast Sampling of Diffusion Models’. In: DOI: 10.48550/arxiv.2202.00512. URL: https://arxiv.org/abs/2202.00512v2.

Smith, Kaleb E and Anthony O Smith (June 2020). ‘Conditional GAN for timeseries generation’. In: URL: https://arxiv.org/abs/2006.16477v1.

Sohl-Dickstein, Jascha et al. (Mar. 2015). ‘Deep Unsupervised Learning using Nonequilibrium Thermodynamics’. In: *32nd International Conference on Machine Learning, ICML 2015* 3, pp. 2246–2255. DOI: 10.48550/arxiv.1503.03585. URL: https://arxiv.org/abs/1503.03585v8.

Vaswani, Ashish et al. (June 2017). ‘Attention Is All You Need’. In: URL: http://arxiv.org/abs/1706.03762.

Wang, P., K. Olsen and W. Bouaziz (2022). *supervised-FCN*. https://github.com/lucidrains/vector-quantize-pytorch.

Wang, Phil (2022). *Denoising Diffusion Probabilistic Model, in Pytorch*. https://github.com/lucidrains/denoising-diffusion-pytorch.

Wang, Zhiguang, Weizhong Yan and Tim Oates (Nov. 2016). ‘Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline’. In: DOI: 10.48550/arxiv.1611.06455. URL: https://arxiv.org/abs/1611.06455.

Xiao, Zhisheng, Karsten Kreis and Arash Vahdat (Dec. 2021). ‘Tackling the Generative Learning Trilemma with Denoising Diffusion GANs’. In: DOI: 10.48550/arxiv.2112.07804. URL: https://arxiv.org/abs/2112.07804.

Yang, Ling et al. (Sept. 2022). ‘Diffusion Models: A Comprehensive Survey of Methods and Applications’. In: DOI: 10.48550/arxiv.2209.00796. URL: https://arxiv.org/abs/2209.00796v9.

Yoon, Jinsung, Daniel Jarrett and Mihaela Van Der Schaar (2019). *Time-series Generative Adversarial Networks*.

Zeghidour, Neil et al. (July 2021). ‘SoundStream: An End-to-End Neural Audio Codec’. In: URL: http://arxiv.org/abs/2107.03312.

Appendix

A Diffusion process as a SDE

The diffusion process given in Equation (13) is equivalent to the following SDE,

$$\mathrm{d}\boldsymbol{z}_t = f(t)\mathrm{d}t + g(t)\mathrm{d}\boldsymbol{W}_t, \tag{54}$$

where $f(t) : \mathbb{R}^l \to \mathbb{R}^l$, $g(t) \in \mathbb{R}$ and $\boldsymbol{W}_t \in \mathbb{R}^l$ are the drift, diffusion and Brownian motion represented using a standard Wiener process. Ho and Salimans, 2022 finds that the particular functions are

$$f(t) = \frac{\mathrm{d}}{\mathrm{d}t}\left(\log \alpha_t\right), \qquad g^2(t) = \frac{\mathrm{d}}{\mathrm{d}t}\left(\sigma_t^2\right) - 2\frac{\mathrm{d}}{\mathrm{d}t}\left(\log \alpha_t\right)\sigma_t^2. \tag{55}$$

As derived by Anderson, 1982, reversing Equation (54) gives another SDE traveling backwards in time,

$$\mathrm{d}\boldsymbol{z}_t = \left[f(t) - g^2(t)\nabla_{\boldsymbol{z}_t}\log q_\phi(\boldsymbol{z}_t)\right]\mathrm{d}t + g(t)\mathrm{d}\boldsymbol{W}_t. \tag{56}$$

Here the Wiener process runs backwards through time and the infinitesimal time step dt is negative. There also exists a deterministic process whose trajectories share the same marginal probability $q_\phi(\boldsymbol{z})$ for all $\boldsymbol{z}_t \in [0, 1]$. This process is governed by the following ordinary differential equation (ODE),

$$\mathrm{d}\boldsymbol{z}_t = \left[\boldsymbol{f}(t) - \frac{1}{2}g^2(t)\nabla_{\boldsymbol{z}_t}\log q_\phi(\boldsymbol{z}_t)\right]\mathrm{d}t. \tag{57}$$

B Derivation of $q_\phi(\boldsymbol{z}_s \mid \boldsymbol{z}_t, \boldsymbol{z}_0)$

Using Bayes' theorem, the conditional distribution $q_\phi(\boldsymbol{z}_s \mid \boldsymbol{z}_t, \boldsymbol{z}_0)$, where $0 \leq s < t \leq 1$, can be written as

$$q_\phi(\boldsymbol{z}_s \mid \boldsymbol{z}_t, \boldsymbol{z}_0) = \frac{q_\phi(\boldsymbol{z}_t \mid \boldsymbol{z}_s, \boldsymbol{z}_0)}{q_\phi(\boldsymbol{z}_t \mid \boldsymbol{z}_0)} q_\phi(\boldsymbol{z}_s \mid \boldsymbol{z}_0) = \frac{q_\phi(\boldsymbol{z}_t \mid \boldsymbol{z}_s)}{q_\phi(\boldsymbol{z}_t \mid \boldsymbol{z}_0)} q_\phi(\boldsymbol{z}_s \mid \boldsymbol{z}_0), \tag{58}$$

where the last equality follows from the Markov property. Using Equation (13), we can insert Gaussian distributions for all of the three conditional distributions. It follows that

$$\begin{aligned} q_\phi(\boldsymbol{z}_s \mid \boldsymbol{z}_t, \boldsymbol{z}_0) &\propto \exp\left\{-\frac{1}{2}\left(\left(\frac{\boldsymbol{z}_t - \alpha_{t|s}\boldsymbol{z}_s}{\sigma_{t|s}}\right)^2 + \left(\frac{\boldsymbol{z}_s - \alpha_s \boldsymbol{z}_0}{\sigma_s}\right)^2 - \left(\frac{\boldsymbol{z}_t - \alpha_t \boldsymbol{z}_0}{\sigma_t}\right)^2\right)\right\} \\ &\propto \exp\left\{-\frac{1}{2}\left(\frac{\boldsymbol{z}_t^2 - 2\alpha_{t|s}\boldsymbol{z}_t\boldsymbol{z}_s + \alpha_{t|s}^2\boldsymbol{z}_s^2}{\sigma_{t|s}^2} + \frac{\boldsymbol{z}_s^2 - 2\alpha_s\boldsymbol{z}_s\boldsymbol{z}_0 + \alpha_s^2\boldsymbol{z}_0^2}{\sigma_s^2} + C_1(\boldsymbol{z}_0, \boldsymbol{z}_t)\right)\right\} \\ &\propto \exp\left\{-\frac{1}{2}\left(\underbrace{\left(\frac{\alpha_{t|s}^2}{\sigma_{t|s}^2} + \frac{1}{\sigma_s^2}\right)}_{:=a}\boldsymbol{z}_s^2 - 2\underbrace{\left(\frac{\alpha_{t|s}\boldsymbol{z}_t}{\sigma_{t|s}^2} + \frac{\alpha_s\boldsymbol{z}_0}{\sigma_s^2}\right)}_{:=b}\boldsymbol{z}_s + C_2(\boldsymbol{z}_0, \boldsymbol{z}_t)\right)\right\}, \end{aligned} \tag{59}$$

where C_1 and C_2 are constants with respect to $\boldsymbol{z}_s$. We recognize (59) as a core of an isotropic Gaussian distribution. Define $q_\phi(\boldsymbol{z}_s \mid \boldsymbol{z}_t, \boldsymbol{z}_0)$ as a Gaussian distribution of the form

$$q_\phi(\boldsymbol{z}_s \mid \boldsymbol{z}_t, \boldsymbol{z}_0) = \mathcal{N}(\boldsymbol{z}_s; \boldsymbol{\mu}_{s|t,0},\ \sigma_{s|t,0}^2\mathbf{I}) \propto \exp\left\{-\frac{1}{2}\left(\frac{\boldsymbol{z}_s^2 - 2\boldsymbol{\mu}_{s|t,0}\boldsymbol{z}_s}{\sigma_{s|t,0}^2}\right)\right\}, \tag{60}$$

where the mean $\boldsymbol{\mu}_{s|t,0}$ and variance $\sigma_{s|t,0}^2$ can be matched with the constants a and b given in Equation (59). Solving for the variance gives

$$\begin{aligned} \sigma_{s|t,0}^2 = a^{-1} &= \left(\frac{\alpha_{t|s}^2}{\sigma_{t|s}^2} + \frac{1}{\sigma_s^2}\right)^{-1} = \left(\frac{\frac{\alpha_t^2}{\alpha_s^2}}{\left(1 - \frac{\alpha_t^2\sigma_s^2}{\sigma_t^2\alpha_s^2}\right)\sigma_t^2} + \frac{1}{\sigma_s^2}\right)^{-1} \\ &= \left(\frac{1}{\left(\frac{\alpha_s^2\sigma_t^2}{\sigma_s^2\alpha_t^2} - 1\right)\sigma_s^2} + \frac{1}{\sigma_s^2}\right)^{-1} = \left(1 - \frac{\alpha_t^2\sigma_s^2}{\sigma_t^2\alpha_s^2}\right)\sigma_s^2 = \frac{\sigma_s^2}{\sigma_t^2}\sigma_{t|s}^2, \end{aligned} \tag{61}$$

and solving for the mean gives

$$\boldsymbol{\mu}_{s|t,0} = a^{-1}b = \frac{\sigma_s^2}{\sigma_t^2}\sigma_{t|s}^2\left(\frac{\alpha_{t|s}\boldsymbol{z}_t}{\sigma_{t|s}^2} + \frac{\alpha_s\boldsymbol{z}_0}{\sigma_s^2}\right) = \frac{\sigma_s^2}{\sigma_t^2}\alpha_{t|s}\ \boldsymbol{z}_t + \frac{\alpha_s}{\sigma_t^2}\sigma_{t|s}^2\ \boldsymbol{z}_0. \tag{62}$$

To simplify further, the log signal-to-noise ratio, $\lambda_t = \log\frac{\alpha_t^2}{\sigma_t^2}$, is used. Rewriting the mean and variance found in (61) and (62) using the log signal to noise ratio then shows that the distribution $q_\phi(\boldsymbol{z}_s \mid \boldsymbol{z}_t, \boldsymbol{z}_0)$ can be written as

$$q_\phi(\boldsymbol{z}_s \mid \boldsymbol{z}_t, \boldsymbol{z}_0) = \mathcal{N}\left(\boldsymbol{z}_s;\ \boldsymbol{\mu}_{s|t,0},\ \sigma_{s|t,0}^2\mathbf{I}\right), \tag{63}$$

where

$$\boldsymbol{\mu}_{s|t,0} = e^{\lambda_t - \lambda_s}\frac{\alpha_s}{\alpha_t}\ \boldsymbol{z}_t + \left(1 - e^{\lambda_t - \lambda_s}\right)\alpha_s\ \boldsymbol{z}_0, \qquad \sigma_{s|t,0} = \left(1 - e^{\lambda_t - \lambda_s}\right)\sigma_s^2. \tag{64}$$

C Detailed derivation of the loss function

From section 2, the likelihood of TimeVQVDM is bounded by

$$\mathbb{E}_q[-\log p_\theta(\boldsymbol{x})] \leq \underbrace{\mathbb{E}_{q_\phi}\left[-\log \frac{p_\theta(\boldsymbol{z}_q^{\mathrm{LF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{LF}}, \boldsymbol{x}^{\mathrm{LF}})}{q_\phi(\boldsymbol{z}_q^{\mathrm{LF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{LF}} \mid \boldsymbol{x}^{\mathrm{LF}})}\right]}_{=:L_{\mathrm{ELBO}}^{LF}(\theta,\phi|\boldsymbol{x}^{\mathrm{LF}})} + \underbrace{\mathbb{E}_{q_\phi}\left[-\log \frac{p_\theta(\boldsymbol{z}_q^{\mathrm{HF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{HF}}, \boldsymbol{x}^{\mathrm{HF}} \mid \boldsymbol{z}_q^{\mathrm{LF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{LF}}, \boldsymbol{x}^{\mathrm{LF}})}{q_\phi(\boldsymbol{z}_q^{\mathrm{HF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{HF}} \mid \boldsymbol{x}^{\mathrm{HF}})}\right]}_{=:L_{\mathrm{ELBO}}^{HF}(\theta,\phi|\boldsymbol{x}^{\mathrm{HF}})}.$$

The derivation for the LF and HF ELBO terms are almost identical, so the derivation is only carried out for low frequencies.

ELBO for low frequencies

The loss function is derived assuming that the diffusion process is discretized as $\boldsymbol{\tau} = \{\tau_i\}_{t=0}^{T}$, where $\tau_i = i/T$. Afterwards, we let $T \to \infty$. The inference model for low frequencies can be written as

$$q_\phi(\boldsymbol{z}_q^{\mathrm{LF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{LF}} \mid \boldsymbol{x}^{\mathrm{LF}}) = q_\phi(\boldsymbol{z}_q^{\mathrm{LF}} \mid \boldsymbol{z}_0^{\mathrm{LF}})\ q_\phi(\boldsymbol{z}_0^{\mathrm{LF}} \mid \boldsymbol{x}^{\mathrm{LF}}) \prod_{i=1}^{T} q_\phi(\boldsymbol{z}_{\tau_i}^{\mathrm{LF}} \mid \boldsymbol{z}_{\tau_{i-1}}^{\mathrm{LF}}), \tag{65}$$

and the generative model can be written as

$$p_\theta(\boldsymbol{z}_q^{\mathrm{LF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{LF}}, \boldsymbol{x}^{\mathrm{LF}}) = p_\theta(\boldsymbol{x}^{\mathrm{LF}} \mid \boldsymbol{z}_q^{\mathrm{LF}})\ p_\theta(\boldsymbol{z}_q^{\mathrm{LF}} \mid \boldsymbol{z}_0^{\mathrm{LF}})\ p(\boldsymbol{z}_1^{\mathrm{LF}}) \prod_{i=1}^{T} p_\theta(\boldsymbol{z}_{\tau_{i-1}}^{\mathrm{LF}} \mid \boldsymbol{z}_{\tau_i}^{\mathrm{LF}}). \tag{66}$$

Using the inference model in (65) and the generative model in (66), the low frequency component in the evidence lower bound can be expressed as

$$\begin{aligned}
&L_{\mathrm{ELBO}}^{\mathrm{LF}}(\theta, \phi \mid \boldsymbol{x}) \\
&\overset{(i)}{=} \mathbb{E}_{q_\phi}\left[-\log p_\theta(\boldsymbol{x}^{\mathrm{LF}} \mid \boldsymbol{z}_q^{\mathrm{LF}}) - \log p(\boldsymbol{z}_1^{\mathrm{LF}})\ p_\theta(\boldsymbol{z}_q^{\mathrm{LF}} \mid \boldsymbol{z}_0^{\mathrm{LF}}) - \log \prod_{i=1}^{T} p_\theta(\boldsymbol{z}_{\tau_{i-1}}^{\mathrm{LF}} \mid \boldsymbol{z}_{\tau_i}^{\mathrm{LF}}) + \log q_\phi(\boldsymbol{z}_q^{\mathrm{LF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{LF}} \mid \boldsymbol{x}^{\mathrm{LF}})\right] \\
&\overset{(ii)}{=} \mathbb{E}_{q_\phi}\left[-\log \frac{p_\theta(\boldsymbol{x}^{\mathrm{LF}} \mid \boldsymbol{z}_q^{\mathrm{LF}})}{q_\phi(\boldsymbol{z}_0^{\mathrm{LF}} \mid \boldsymbol{x}^{\mathrm{LF}})} - \log \frac{p(\boldsymbol{z}_1^{\mathrm{LF}})\ p_\theta(\boldsymbol{z}_q^{\mathrm{LF}} \mid \boldsymbol{z}_0^{\mathrm{LF}})}{q_\phi(\boldsymbol{z}_q^{\mathrm{LF}} \mid \boldsymbol{z}_0^{\mathrm{LF}})} - \log \prod_{i=1}^{T} \frac{p_\theta(\boldsymbol{z}_{\tau_{i-1}}^{\mathrm{LF}} \mid \boldsymbol{z}_{\tau_i}^{\mathrm{LF}})}{q_\phi(\boldsymbol{z}_{\tau_i}^{\mathrm{LF}} \mid \boldsymbol{z}_{\tau_{i-1}}^{\mathrm{LF}})}\right] \\
&\overset{(iii)}{=} \mathbb{E}_{q_\phi}\left[-\log \frac{p_\theta(\boldsymbol{x}^{\mathrm{LF}} \mid \boldsymbol{z}_q^{\mathrm{LF}})}{q_\phi(\boldsymbol{z}_0^{\mathrm{LF}} \mid \boldsymbol{x}^{\mathrm{LF}})} - \log \frac{p(\boldsymbol{z}_1^{\mathrm{LF}})\ p_\theta(\boldsymbol{z}_q^{\mathrm{LF}} \mid \boldsymbol{z}_0^{\mathrm{LF}})}{q_\phi(\boldsymbol{z}_q^{\mathrm{LF}} \mid \boldsymbol{z}_0^{\mathrm{LF}})} - \sum_{i=1}^{T} \log \frac{p_\theta(\boldsymbol{z}_{\tau_{i-1}}^{\mathrm{LF}} \mid \boldsymbol{z}_{\tau_i}^{\mathrm{LF}})}{q_\phi(\boldsymbol{z}_{\tau_i}^{\mathrm{LF}} \mid \boldsymbol{z}_{\tau_{i-1}}^{\mathrm{LF}})}\right] \\
&\overset{(iv)}{=} \mathbb{E}_{q_\phi}\left[-\log \frac{p_\theta(\boldsymbol{x}^{\mathrm{LF}} \mid \boldsymbol{z}_q^{\mathrm{LF}})}{q_\phi(\boldsymbol{z}_0^{\mathrm{LF}} \mid \boldsymbol{x}^{\mathrm{LF}})} - \log \frac{p(\boldsymbol{z}_1^{\mathrm{LF}}) p_\theta(\boldsymbol{z}_q^{\mathrm{LF}} \mid \boldsymbol{z}_0^{\mathrm{LF}})}{q_\phi(\boldsymbol{z}_q^{\mathrm{LF}} \mid \boldsymbol{z}_0^{\mathrm{LF}})} - \sum_{i=1}^{T} \log \left(\frac{p_\theta(\boldsymbol{z}_{\tau_{i-1}}^{\mathrm{LF}} \mid \boldsymbol{z}_{\tau_i}^{\mathrm{LF}})}{q_\phi(\boldsymbol{z}_{\tau_{i-1}}^{\mathrm{LF}} \mid \boldsymbol{z}_{\tau_i}^{\mathrm{LF}}, \boldsymbol{x}^{\mathrm{LF}})} \cdot \frac{q_\phi(\boldsymbol{z}_{\tau_{i-1}}^{\mathrm{LF}} \mid \boldsymbol{x}^{\mathrm{LF}})}{q_\phi(\boldsymbol{z}_{\tau_i}^{\mathrm{LF}} \mid \boldsymbol{x}^{\mathrm{LF}})}\right)\right] \\
&\overset{(v)}{=} \mathbb{E}_{q_\phi}\left[-\log \frac{p_\theta(\boldsymbol{x}^{\mathrm{LF}} \mid \boldsymbol{z}_q^{\mathrm{LF}})}{q_\phi(\boldsymbol{z}_0^{\mathrm{LF}} \mid \boldsymbol{x}^{\mathrm{LF}})} - \log \frac{p(\boldsymbol{z}_1^{\mathrm{LF}}) p_\theta(\boldsymbol{z}_q^{\mathrm{LF}} \mid \boldsymbol{z}_0^{\mathrm{LF}})}{q_\phi(\boldsymbol{z}_q^{\mathrm{LF}} \mid \boldsymbol{z}_0^{\mathrm{LF}})} - \sum_{i=1}^{T} \log \frac{p_\theta(\boldsymbol{z}_{\tau_{i-1}}^{\mathrm{LF}} \mid \boldsymbol{z}_{\tau_i}^{\mathrm{LF}})}{q_\phi(\boldsymbol{z}_{\tau_{i-1}}^{\mathrm{LF}} \mid \boldsymbol{z}_{\tau_i}^{\mathrm{LF}}, \boldsymbol{x}^{\mathrm{LF}})} - \log \frac{q_\phi(\boldsymbol{z}_0^{\mathrm{LF}} \mid \boldsymbol{x}^{\mathrm{LF}})}{q_\phi(\boldsymbol{z}_1^{\mathrm{LF}} \mid \boldsymbol{x}^{\mathrm{LF}})}\right] \\
&\overset{(vi)}{=} \mathbb{E}_{q_\phi}\left[-\log \frac{p(\boldsymbol{z}_1^{\mathrm{LF}})}{q_\phi(\boldsymbol{z}_1^{\mathrm{LF}} \mid \boldsymbol{x}^{\mathrm{LF}})} - \sum_{i=1}^{T} \log \frac{p_\theta(\boldsymbol{z}_{\tau_{i-1}}^{\mathrm{LF}} \mid \boldsymbol{z}_{\tau_i}^{\mathrm{LF}})}{q_\phi(\boldsymbol{z}_{\tau_{i-1}}^{\mathrm{LF}} \mid \boldsymbol{z}_{\tau_i}^{\mathrm{LF}}, \boldsymbol{x}^{\mathrm{LF}})} - \log \frac{p_\theta(\boldsymbol{z}_q^{\mathrm{LF}} \mid \boldsymbol{z}_0^{\mathrm{LF}})}{q_\phi(\boldsymbol{z}_q^{\mathrm{LF}} \mid \boldsymbol{z}_0^{\mathrm{LF}})} - \log p_\theta(\boldsymbol{x}^{\mathrm{LF}} \mid \boldsymbol{z}_q^{\mathrm{LF}})\right],
\end{aligned}$$

where equality (i) follows after inserting $p_\theta(\boldsymbol{z}_q^{\mathrm{LF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{LF}}, \boldsymbol{x}^{\mathrm{LF}})$ given in (66) and (ii) follows after inserting $q_\phi(\boldsymbol{z}_q^{\mathrm{LF}}, \boldsymbol{z}_{\boldsymbol{\tau}}^{\mathrm{LF}} \mid \boldsymbol{x}^{\mathrm{LF}})$ given in (65). (iii) switches the logarithm of a product with the sum of logarithms. (iv) uses Bayes theorem on $q_\phi(\boldsymbol{z}_{\tau_i}^{\mathrm{LF}} \mid \boldsymbol{z}_{\tau_{i-1}}^{\mathrm{LF}}) = q_\phi(\boldsymbol{z}_{\tau_i}^{\mathrm{LF}} \mid \boldsymbol{z}_{\tau_{i-1}}^{\mathrm{LF}}, \boldsymbol{x}^{\mathrm{LF}})$. (v) removes a telescoping product $\prod_{i=1}^{T} \frac{q_\phi(\boldsymbol{z}_{\tau_{i-1}}^{\mathrm{LF}}|\boldsymbol{x}^{\mathrm{LF}})}{q_\phi(\boldsymbol{z}_{\tau_i}^{\mathrm{LF}}|\boldsymbol{x}^{\mathrm{LF}})} = \frac{q_\phi(\boldsymbol{z}_0^{\mathrm{LF}}|\boldsymbol{x}^{\mathrm{LF}})}{q_\phi(\boldsymbol{z}_1^{\mathrm{LF}}|\boldsymbol{x}^{\mathrm{LF}})}$. (vi) lastly rewrites the evidence lower bound.

The low frequency terms of the evidence lower bound can therefore be decomposed into four parts

$$L^{\mathrm{LF}}_{\mathrm{ELBO}}(\theta, \phi \mid \boldsymbol{x}) = L^{\mathrm{LF}}_{\mathrm{prior}} + L^{\mathrm{LF}}_{\mathrm{diffusion}} + L^{\mathrm{LF}}_{\mathrm{codebook}} + L^{\mathrm{LF}}_{\mathrm{reconstruct}},$$

consisting of the different steps needed in sampling from the generative model. The first is drawing from a prior distribution, corresponding to the fully noisy latent distribution. The second is using the denoising diffusion model to denoise the latent sample. The third term is to quantize the denoised latent vector using the codebook. Finally, the forth is to reconstruct the sample from the quantized latent vector.

Prior loss

Using the KL divergence, the prior loss is

$$L^{\mathrm{LF}}_{\mathrm{prior}} = \mathbb{E}_{q_\phi}\left[-\log \frac{p(\boldsymbol{z}^{\mathrm{LF}}_1)}{q_\phi(\boldsymbol{z}^{\mathrm{LF}}_1 \mid \boldsymbol{x}^{\mathrm{LF}})}\right] = D_{\mathrm{KL}}(q_\phi(\boldsymbol{z}^{\mathrm{LF}}_1 \mid \boldsymbol{x}^{\mathrm{LF}}) \parallel p(\boldsymbol{z}^{\mathrm{LF}}_1)).$$

Since we have a fixed diffusion prior $p(\boldsymbol{z}^{\mathrm{LF}}_1) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ with no trainable parameters, and the inference model also has a fixed distribution for the fully diffused sample, $q_\phi(\boldsymbol{z}^{\mathrm{LF}}_1 \mid \boldsymbol{x}^{\mathrm{LF}}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$, the prior loss is constant. In fact, since the two distributions are equal, the analytical KL divergence is zero. This term can therefore be omitted during training.

Diffusion loss

This derivation of the diffusion loss is similar to the one given in Appendix E in Diederik P. Kingma et al., 2021. Using the KL divergence, we can write the diffusion loss as

$$L^{\mathrm{LF}}_{\mathrm{diffusion}} = \mathbb{E}_{q_\phi}\left[-\sum_{i=1}^{T} \log \frac{p_\theta(\boldsymbol{z}^{\mathrm{LF}}_{\tau_{i-1}} \mid \boldsymbol{z}^{\mathrm{LF}}_{\tau_i})}{q_\phi(\boldsymbol{z}^{\mathrm{LF}}_{\tau_{i-1}} \mid \boldsymbol{z}^{\mathrm{LF}}_{\tau_i}, \boldsymbol{x}^{\mathrm{LF}})}\right] = \sum_{i=1}^{T} \underbrace{D_{\mathrm{KL}}(q_\phi(\boldsymbol{z}^{\mathrm{LF}}_{\tau_{i-1}} \mid \boldsymbol{z}^{\mathrm{LF}}_{\tau_i}, \boldsymbol{x}^{\mathrm{LF}}) \parallel p_\theta(\boldsymbol{z}^{\mathrm{LF}}_{\tau_{i-1}} \mid \boldsymbol{z}^{\mathrm{LF}}_{\tau_i}))}_{=:L^{\mathrm{LF}}_{\mathrm{diffusion},\tau_i}}.$$

Both distributions in the KL divergence are Gaussian with the same variance. For two Gaussian distributions $p = \mathcal{N}(\boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p)$ and $q = \mathcal{N}(\boldsymbol{\mu}_q, \boldsymbol{\Sigma}_q)$ where the variances satisfy $\boldsymbol{\Sigma}_p = \boldsymbol{\Sigma}_q = \sigma^2\mathbf{I}$, the analytical KL divergence is

$$\begin{aligned} D_{\mathrm{KL}}(p \parallel q) &= \frac{1}{2}\left(\log\frac{|\boldsymbol{\Sigma}_p|}{|\boldsymbol{\Sigma}_q|} - d + (\boldsymbol{\mu}_p - \boldsymbol{\mu}_q)^\top \boldsymbol{\Sigma}_p^{-1}(\boldsymbol{\mu}_p - \boldsymbol{\mu}_q) + \mathrm{tr}\left(\boldsymbol{\Sigma}_q^{-1}\boldsymbol{\Sigma}_p\right)\right) \\ &= \frac{1}{2\sigma^2}\left\|\boldsymbol{\mu}_p - \boldsymbol{\mu}_q\right\|_2^2, \end{aligned}$$

where d is the dimensionality of the two distributions. The two distributions we are comparing can be written as

$$\begin{aligned} q_\phi(\boldsymbol{z}^{\mathrm{LF}}_{\tau_{i-1}} \mid \boldsymbol{z}^{\mathrm{LF}}_{\tau_i}, \boldsymbol{x}^{\mathrm{LF}}) &\sim \mathcal{N}\left(\boldsymbol{\mu}_{\tau_{i-1}|\tau_i,0},\ \sigma^2_{\tau_{i-1}|\tau_i,0}\mathbf{I}\right), \\ p_\theta(\boldsymbol{z}^{\mathrm{LF}}_{\tau_{i-1}} \mid \boldsymbol{z}^{\mathrm{LF}}_{\tau_i}) &\sim \mathcal{N}\left(\boldsymbol{\mu}_\theta(\boldsymbol{z}^{\mathrm{LF}}_{\tau_i}),\ \sigma^2_{\tau_{i-1}|\tau_i,0}\mathbf{I}\right), \end{aligned}$$

where the means and variances are

$$\begin{aligned} \boldsymbol{\mu}_{\tau_{i-1}|\tau_i,0} &= \frac{\sigma^2_{\tau_{i-1}}}{\sigma^2_{\tau_i}}\alpha_{\tau_i|\tau_{i-1}}\ \boldsymbol{z}^{\mathrm{LF}}_{\tau_i} + \frac{\alpha_{\tau_{i-1}}}{\sigma^2_{\tau_i}}\sigma^2_{\tau_i|\tau_{i-1}}\ \boldsymbol{z}^{\mathrm{LF}}_0, \\ \boldsymbol{\mu}_\theta(\boldsymbol{z}^{\mathrm{LF}}_{\tau_i}) &= \frac{\sigma^2_{\tau_{i-1}}}{\sigma^2_{\tau_i}}\alpha_{\tau_i|\tau_{i-1}}\ \boldsymbol{z}^{\mathrm{LF}}_{\tau_i} + \frac{\alpha_{\tau_{i-1}}}{\sigma^2_{\tau_i}}\sigma^2_{\tau_i|\tau_{i-1}}\ f_\theta(\boldsymbol{z}^{\mathrm{LF}}_0; \boldsymbol{z}^{\mathrm{LF}}_{\tau_i}) \\ \sigma^2_{\tau_{i-1}|\tau_i,0} &= \frac{\sigma^2_{\tau_{i-1}}}{\sigma^2_{\tau_i}}\sigma^2_{\tau_i|\tau_{i-1}}. \end{aligned}$$

Writing out the KL divergence, a single term in the diffusion loss then can be expressed as

$$\begin{aligned}
L^{\mathrm{LF}}_{\text{diffusion},\tau_i} &= D_{\mathrm{KL}}\left(q(\boldsymbol{z}^{\mathrm{LF}}_{\tau_{i-1}} \mid \boldsymbol{z}^{\mathrm{LF}}_{\tau_i}, \boldsymbol{x}^{\mathrm{LF}}) \parallel p_\theta(\boldsymbol{z}^{\mathrm{LF}}_{\tau_{i-1}} \mid \boldsymbol{z}^{\mathrm{LF}}_{\tau_i})\right) \\
&= \frac{1}{2} \cdot \frac{1}{\sigma^2_{\tau_{i-1}|\tau_i,0}} \left\| \boldsymbol{\mu}_{\tau_{i-1}|\tau_i,0} - \boldsymbol{\mu}_\theta(\boldsymbol{z}^{\mathrm{LF}}_{\tau_i}) \right\|_2^2 \\
&= \frac{1}{2} \cdot \frac{\sigma^2_{\tau_i}}{\sigma^2_{\tau_{i-1}} \sigma^2_{\tau_i|\tau_{i-1}}} \cdot \frac{\alpha^2_{\tau_{i-1}}}{\sigma^4_{\tau_i}} \sigma^4_{\tau_i|\tau_{i-1}} \left\| \boldsymbol{z}^{\mathrm{LF}}_0 - f_\theta(\boldsymbol{z}^{\mathrm{LF}}_0; \boldsymbol{z}^{\mathrm{LF}}_{\tau_i}) \right\|_2^2 \\
&= \frac{1}{2} \cdot \left(\frac{\alpha^2_{\tau_{i-1}}}{\sigma^2_{\tau_{i-1}}} - \frac{\alpha^2_{\tau_i}}{\sigma^2_{\tau_i}} \right) \left\| \boldsymbol{z}^{\mathrm{LF}}_0 - f_\theta(\boldsymbol{z}^{\mathrm{LF}}_0; \boldsymbol{z}^{\mathrm{LF}}_{\tau_i}) \right\|_2^2 \\
&= \frac{1}{2} \cdot \left(e^{\lambda_{\tau_{i-1}}} - e^{\lambda_{\tau_i}} \right) \left\| \boldsymbol{z}^{\mathrm{LF}}_0 - f_\theta(\boldsymbol{z}^{\mathrm{LF}}_0; \boldsymbol{z}^{\mathrm{LF}}_{\tau_i}) \right\|_2^2,
\end{aligned}$$

where it was inserted that $\sigma^2_{\tau_i|\tau_{i-1}} = \left(1 - \frac{\alpha^2_{\tau_i} \sigma^2_{\tau_{i-1}}}{\sigma^2_{\tau_i} \alpha^2_{\tau_{i-1}}}\right) \sigma^2_{\tau_i}$ and simplified. Adding all terms in the discrete diffusion loss, the full diffusion loss is

$$L^{\mathrm{LF}}_{\text{diffusion}} = \frac{1}{2} \sum_{i=1}^{T} \left(e^{\lambda_{\tau_{i-1}}} - e^{\lambda_{\tau_i}} \right) \left\| \boldsymbol{z}^{\mathrm{LF}}_0 - f_\theta(\boldsymbol{z}^{\mathrm{LF}}_0; \boldsymbol{z}^{\mathrm{LF}}_{\tau_i}) \right\|_2^2. \tag{67}$$

Equation (67) shows that the diffusion loss is only dependent on the log-SNR λ_t, not on the magnitude of α_t and σ_t.

In the limit as the number of denoising steps tend to infinity, a discrete time step diffusion model approximates a continuous time step diffusion model. The resulting objective function becomes very similar as for the discrete case. When $T \to \infty$ it holds that

$$\lim_{T \to \infty} T \left(e^{\lambda_{t-1/T}} - e^{\lambda_t} \right) = -\frac{\mathrm{d}}{\mathrm{d}t} \left(e^{\lambda_t} \right), \tag{68}$$

by the definition of derivation. In the limit as the time dimension is discretized using an infinite number of steps, the discrete diffusion loss contains a Riemann sum approximating an integral, such that the continuous time step loss becomes

$$\begin{aligned}
\lim_{T\to\infty} L^{\mathrm{LF}}_{\text{diffusion}} &= \lim_{T\to\infty} \frac{1}{2} \sum_{i=1}^{T} \left(e^{\lambda_{\tau_{i-1}}} - e^{\lambda_{\tau_i}} \right) \left\| \boldsymbol{z}^{\mathrm{LF}}_0 - f_\theta(\boldsymbol{z}^{\mathrm{LF}}_0; \boldsymbol{z}^{\mathrm{LF}}_{\tau_i}) \right\|_2^2 \\
&= \lim_{T\to\infty} \frac{1}{2T} \sum_{i=1}^{T} T \left(e^{\lambda_{\tau_{i-1}}} - e^{\lambda_{\tau_i}} \right) \left\| \boldsymbol{z}^{\mathrm{LF}}_0 - f_\theta(\boldsymbol{z}^{\mathrm{LF}}_0; \boldsymbol{z}^{\mathrm{LF}}_{\tau_i}) \right\|_2^2 \\
&= -\frac{1}{2} \int_0^1 \frac{\mathrm{d}}{\mathrm{d}t} \left(e^{\lambda_t} \right) \left\| \boldsymbol{z}^{\mathrm{LF}}_0 - f_\theta(\boldsymbol{z}^{\mathrm{LF}}_0; \boldsymbol{z}^{\mathrm{LF}}_t) \right\|_2^2 \mathrm{d}t.
\end{aligned} \tag{69}$$

Computing (69) is computationally demanding. To avoid having too massive computations, an unbiased Monte Carlo estimator is constructed. An unbiased estimator of the continuous diffusion loss is given by

$$\hat{L}^{\mathrm{LF}}_{\text{diffusion}} = -\frac{1}{2} \mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),\ t \sim \mathrm{U}(0,1)} \left[\frac{\mathrm{d}}{\mathrm{d}t} \left(e^{\lambda_t} \right) \left\| \boldsymbol{z}^{\mathrm{LF}}_0 - f_\theta(\boldsymbol{z}^{\mathrm{LF}}_0; \boldsymbol{z}^{\mathrm{LF}}_t) \right\|_2^2 \right], \tag{70}$$

where $\boldsymbol{z}^{\mathrm{LF}}_t = \alpha_t \boldsymbol{z}^{\mathrm{LF}}_0 + \sigma_t \boldsymbol{\epsilon}$. This loss function is a weighted mean squared error for reconstructing the initial low frequency latent given a noisy latent $\boldsymbol{z}^{\mathrm{LF}}_t$, evaluated with a random noise at a random time $t \in [0, 1]$. It is worth noticing is that the weight $-\frac{1}{2}\frac{\mathrm{d}}{\mathrm{d}t}\left(e^{\lambda_t}\right) \geq 0$ for all time steps, since it is assumed that the SNR is monotonic and decreasing. The loss function can also be written in terms of the $\boldsymbol{\epsilon}$-prediction model. Using Equation (22) to switch to the noise-parameterization, the diffusion loss can be written as

$$\begin{aligned}
\hat{L}^{\mathrm{LF}}_{\text{diffusion}} &= -\frac{1}{2} \mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),\ t \sim \mathrm{U}(0,1)} \left[\frac{\mathrm{d}}{\mathrm{d}t} \left(e^{\lambda_t} \right) e^{-\lambda_t} \left\| \boldsymbol{\epsilon} - f_\theta(\boldsymbol{\epsilon}; \boldsymbol{z}^{\mathrm{LF}}_t) \right\|_2^2 \right] \\
&= -\frac{1}{2} \mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),\ t \sim \mathrm{U}(0,1)} \left[\frac{\mathrm{d}}{\mathrm{d}t} (\lambda_t) \left\| \boldsymbol{\epsilon} - f_\theta(\boldsymbol{\epsilon}; \boldsymbol{z}^{\mathrm{LF}}_t) \right\|_2^2 \right].
\end{aligned} \tag{71}$$

Codebook loss

The derivation for the codebook loss is similar to the one given in (Cohen et al., 2022). The codebook loss is

$$L_{\text{codebook}}^{\text{LF}} = \mathbb{E}_{q_\phi}\left[-\log \frac{p_\theta(\boldsymbol{z}_q^{\text{LF}} \mid \boldsymbol{z}_0^{\text{LF}})}{q_\phi(\boldsymbol{z}_q^{\text{LF}} \mid \boldsymbol{z}_0^{\text{LF}})}\right].$$

The generative distribution over the quantized low frequency latent is assumed to be the discrete distribution

$$p_\theta(\boldsymbol{z}_q^{\text{LF}} \mid \boldsymbol{z}_0^{\text{LF}}) = \text{Softmax}\left\{-\|\boldsymbol{z}_0^{\text{LF}} - \boldsymbol{e}_k\|_2^2\right\}_{1\leq k\leq K},$$

and the inference model is assumed as in (Oord et al., 2017) to be the Dirac mass distribution

$$q_\phi(\boldsymbol{z}_q^{\text{LF}} \mid \boldsymbol{z}_0^{\text{LF}}) = \delta_{\hat{\boldsymbol{z}}_q^{\text{LF}}}(\boldsymbol{z}_q^{\text{LF}}), \qquad (\hat{\boldsymbol{z}}_q^{\text{LF}})_i = \underset{\boldsymbol{e}_k \in \mathcal{Z}_{\text{LF}}}{\text{argmin}} \left\|(\boldsymbol{z}_0^{\text{LF}})_i - \boldsymbol{e}_k\right\|_2.$$

Then according to (Cohen et al., 2022), this yields the following Monte Carlo estimation of $L_{\text{codebook}}^{\text{LF}}$:

$$L_{\text{codebook}}^{\text{LF}} = \left\|\boldsymbol{z}_0^{\text{LF}} - \hat{\boldsymbol{z}}_q^{\text{LF}}\right\|_2 + \log\left(\sum_{k=1}^{K} \exp\left\{-\left\|\boldsymbol{z}_0^{\text{LF}} - \boldsymbol{e}_k\right\|_2\right\}\right). \tag{72}$$

The codebook loss has two terms, where the second is a normalizing term of $p_\theta(\boldsymbol{z}_q^{\text{LF}} \mid \boldsymbol{z}_0^{\text{LF}})$. The first term is non-differentiable, so standard back-propagation cannot be applied. (Oord et al., 2017) resolves this issue by copying gradients from the decoder to the encoder. The codebook loss is split into two parts

$$L_{\text{codebook}}^{\text{LF}} = \left\|\text{sg}\left[\boldsymbol{z}_0^{\text{LF}}\right] - \hat{\boldsymbol{z}}_q^{\text{LF}}\right\|_2^2 + \beta\left\|\boldsymbol{z}_0^{\text{LF}} - \text{sg}\left[\hat{\boldsymbol{z}}_q^{\text{LF}}\right]\right\|_2^2, \tag{73}$$

where sg[·] is the stop gradient-operator. The second term of (73) is the commitment loss, which is weighted by a parameter β.

Reconstruction loss

The reconstruction loss is

$$L_{\text{reconstruct}}^{\text{LF}} = \mathbb{E}_{q_\phi}\left[-\log p_\theta(\boldsymbol{x}^{\text{LF}} \mid \boldsymbol{z}_q^{\text{LF}})\right].$$

Following the work of (Lee et al., 2023) and (Défossez et al., 2022), the reconstruction loss is calculated for both the time-frequency domain and the time domain. So the loss is given as the sum of two mean squared error terms,

$$L_{\text{reconstruct}}^{\text{LF}} = \left\|\boldsymbol{u}^{\text{LF}} - \hat{\boldsymbol{u}}^{\text{LF}}\right\|_2^2 + \left\|\boldsymbol{x}^{\text{LF}} - \hat{\boldsymbol{x}}^{\text{LF}}\right\|_2^2, \tag{74}$$

where $\boldsymbol{u}^{\text{LF}} = \mathcal{P}_{\text{LF}}(\text{STFT}(\boldsymbol{x}))$ and $\hat{\boldsymbol{u}}^{\text{LF}} = \mathcal{P}_{\text{LF}}(D_{\text{LF}}(\boldsymbol{z}_q^{\text{LF}}))$ are in the time-frequency domain, and $\hat{\boldsymbol{x}}^{\text{LF}} = \text{ISTFT}(\hat{\boldsymbol{u}}^{\text{LF}})$ and $\boldsymbol{x}^{\text{LF}} = \text{ISTFT}(\boldsymbol{u}^{\text{LF}})$ are in the time domain.

D Training Details

Stage 1: VQ-VAE

- **Trainable parameters**: $\sim$ 550k for LF and $\sim$ 350k for HF, depending on data set
- **Hyperparameters**: Batch size: 128, max epochs: 2000
- **Additional parameters**: codebook dim: 4, codebook size: 512, encoder/decoder dim: 64, STFT: $\{n_{\text{fft}} = 4$, h=1$\}$, codebook weight decay: 0.8, downsampling width: 16 for LF and 64 for HF, perceptual loss weight: 0.0, commintment weight: $\beta = 1$, ResNet blocks: 4
- **Optimizer**: AdamW, cosine scheduler, initial learning rate: 10^{-3}, weight decay: 10^{-6}.
- **Loss function**: mean squared error of reconstructing LF and HF time representation and time-frequency representation

Stage 2: Diffusion models

- **Trainable parameters**: $\sim$ 4 million per diffusion model, depending on data set
- **Hyperparameters**: Batch size: 128, max epochs: 1000
- **Additional parameters**: parameterization: noise, time embedding: 256, class embedding: 256, class-dropout probability: 0.1, down-sampling steps: 3 for LF, 4 for HF, channels: (64, 128, 256) for LF, (32, 64, 128, 256) for HF, ResNet block groups: 4
- **Optimizer**: AdamW, cosine scheduler, initial learning rate: 10^{-3}, weight decay: 10^{-6}.
- **Loss function**: Mean squared error

Fully Convolutional Network

- **Trainable parameters**: 265000
- **Hyperparameters**: Batch size: 256, max epochs: 1000
- **Optimizer**: AdamW, cosine scheduler, initial learning rate: 10^{-3}, weight decay: 10^{-5}.
- **Loss function**: Cross-entropy

EMA updates of codebooks

Similar to (Oord et al., 2017), the codebook tokens are updates using *exponential moving average* (EMA). During training, let $\{\boldsymbol{z}^{\text{LF}}_{k,1}, \boldsymbol{z}^{\text{LF}}_{k,2}, \cdots, \boldsymbol{z}^{\text{LF}}_{k,n_k}\}$ be the output from the passing a mini-batch through the encoder that is closest to the token $\boldsymbol{e}_k \in \mathcal{Z}_{\text{LF}}$. The first term of Equation (73) can then be written as

$$\left\|\text{sg}\left[\boldsymbol{z}^{\text{LF}}_0\right] - \hat{\boldsymbol{z}}^{\text{LF}}_q\right\|_2 = \sum_{i=1}^{n_k} \|\boldsymbol{z}^{\text{LF}}_{k,i} - \boldsymbol{e}_k\|_2^2$$

The optimal value of $\boldsymbol{e}_k$ is the average of the elements in the set. Instead of setting it to the average of the current mini-batch, an EMA update with decay parameter γ is used. Let N_k denote the EMA of the number of outputs from the encoder closets to token $\boldsymbol{e}_k$ and m_k denote the EMA mass center of the latents closets to $\boldsymbol{e}_k$. Then an update of the token is given as

$$\begin{aligned} N_k^{\text{next}} &:= N_k^{\text{prev}}\gamma + n_k^{\text{next}}(1-\gamma), \\ m_k^{\text{next}} &:= m_k^{\text{prev}}\gamma + \sum_i \boldsymbol{z}^{\text{next}}_{k,i}(1-\gamma), \\ \boldsymbol{e}_k^{\text{next}} &= \frac{m_k^{\text{next}}}{N_k^{\text{next}}}. \end{aligned}$$

Simplified diffusion loss

The diffusion loss in Equation (53) is given as

$$\begin{aligned}L_{\text{diffusion}} =& -\frac{1}{2}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I}),\ t\sim\text{U}(0,1)}\left[\frac{\mathrm{d}}{\mathrm{d}t}(\lambda_t)\left\|\boldsymbol{\epsilon} - f_\theta(\boldsymbol{\epsilon}; \boldsymbol{z}_t^{\text{LF}})\right\|_2^2\right]\\ & -\frac{1}{2}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I}),\ t\sim\text{U}(0,1)}\left[\frac{\mathrm{d}}{\mathrm{d}t}(\lambda_t)\left\|\boldsymbol{\epsilon} - f_\theta(\boldsymbol{\epsilon}; \boldsymbol{z}_t^{\text{HF}}, \boldsymbol{z}_0^{\text{LF}})\right\|_2^2\right].\end{aligned}$$

It is a weighted mean squared error, and the weight function is equal to

$$-\frac{\mathrm{d}}{\mathrm{d}t}(\lambda_t) = \frac{\pi}{\cos\beta_t \sin\beta_t},$$

which becomes infinite at the two endpoints, $t = 0$ and $t = 1$. This loss function could potentially cause instabilities during training. Following the practice of (Ho, Jain et al., 2020), the loss function is simplified such that the weighting term is removed completely. The simplified diffusion loss is

$$\begin{aligned}L_{\text{diffusion}} =& \frac{1}{2}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I}),\ t\sim\text{U}(0,1)}\left[\left\|\boldsymbol{\epsilon} - f_\theta(\boldsymbol{\epsilon}; \boldsymbol{z}_t^{\text{LF}})\right\|_2^2\right]\\ & \frac{1}{2}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I}),\ t\sim\text{U}(0,1)}\left[\left\|\boldsymbol{\epsilon} - f_\theta(\boldsymbol{\epsilon}; \boldsymbol{z}_t^{\text{HF}}, \boldsymbol{z}_0^{\text{LF}})\right\|_2^2\right].\end{aligned}$$

Quasi-uniform diffusion time steps

Similar to (Diederik P. Kingma et al., 2021) we utilize a quasi-random approach for training the diffusion models. The authors showed that implementing a quasi-uniform time distribution lowers the variance of the diffusion loss during training. For a mini-batch consisting of n samples, the diffusion time steps are drawn from the quasi-uniform distribution

$$\boldsymbol{t} = \begin{bmatrix} t + \frac{0}{n} & t + \frac{1}{n} & \cdots & t + \frac{n-1}{n}\end{bmatrix}^\top \mod 1, \qquad t \sim \text{U}(0,1).$$

E Data set summary

Table 6 summarises the different data sets in the UCR archive. The table shows the data sets name, type, number of training and test samples, number of classes and the length of the time series.

Table 6: Summary of the data sets in the UCR archive.

ID	Name	Type	Train	Test	Class	Length
1	Adiac	Image	390	391	37	176
2	ArrowHead	Image	36	175	3	251
3	Beef	Spectro	30	30	5	470
4	BeetleFly	Image	20	20	2	512
5	BirdChicken	Image	20	20	2	512
6	Car	Sensor	60	60	4	577
7	CBF	Simulated	30	900	3	128
8	ChlorineConcentration	Sensor	467	3840	3	166
9	CinCECGTorso	Sensor	40	1380	4	1639
10	Coffee	Spectro	28	28	2	286
11	Computers	Device	250	250	2	720
12	CricketX	Motion	390	390	12	300
13	CricketY	Motion	390	390	12	300
14	CricketZ	Motion	390	390	12	300
15	DiatomSizeReduction	Image	16	306	4	345
16	DistalPhalanxOutlineAgeGroup	Image	400	139	3	80
17	DistalPhalanxOutlineCorrect	Image	600	276	2	80
18	DistalPhalanxTW	Image	400	139	6	80
19	Earthquakes	Sensor	322	139	2	512
20	ECG200	ECG	100	100	2	96
21	ECG5000	ECG	500	4500	5	140
22	ECGFiveDays	ECG	23	861	2	136
23	ElectricDevices	Device	8926	7711	7	96
24	FaceAll	Image	560	1690	14	131
25	FaceFour	Image	24	88	4	350
26	FacesUCR	Image	200	2050	14	131
27	FiftyWords	Image	450	455	50	270
28	Fish	Image	175	175	7	463
29	FordA	Sensor	3601	1320	2	500
30	FordB	Sensor	3636	810	2	500
31	GunPoint	Motion	50	150	2	150
32	Ham	Spectro	109	105	2	431
33	HandOutlines	Image	1000	370	2	2709
34	Haptics	Motion	155	308	5	1092
35	Herring	Image	64	64	2	512
36	InlineSkate	Motion	100	550	7	1882
37	InsectWingbeatSound	Sensor	220	1980	11	256
38	ItalyPowerDemand	Sensor	67	1029	2	24
39	LargeKitchenAppliances	Device	375	375	3	720
40	Lightning2	Sensor	60	61	2	637
41	Lightning7	Sensor	70	73	7	319
42	Mallat	Simulated	55	2345	8	1024
43	Meat	Spectro	60	60	3	448
44	MedicalImages	Image	381	760	10	99
45	MiddlePhalanxOutlineAgeGroup	Image	400	154	3	80
46	MiddlePhalanxOutlineCorrect	Image	600	291	2	80
47	MiddlePhalanxTW	Image	399	154	6	80
48	MoteStrain	Sensor	20	1252	2	84
49	NonInvasiveFetalECGThorax1	ECG	1800	1965	42	750

50	NonInvasiveFetalECGThorax2	ECG	1800	1965	42	750
51	OliveOil	Spectro	30	30	4	570
52	OSULeaf	Image	200	242	6	427
53	PhalangesOutlinesCorrect	Image	1800	858	2	80
54	Phoneme	Sensor	214	1896	39	1024
55	Plane	Sensor	105	105	7	144
56	ProximalPhalanxOutlineAgeGroup	Image	400	205	3	80
57	ProximalPhalanxOutlineCorrect	Image	600	291	2	80
58	ProximalPhalanxTW	Image	400	205	6	80
59	RefrigerationDevices	Device	375	375	3	720
60	ScreenType	Device	375	375	3	720
61	ShapeletSim	Simulated	20	180	2	500
62	ShapesAll	Image	600	600	60	512
63	SmallKitchenAppliances	Device	375	375	3	720
64	SonyAIBORobotSurface1	Sensor	20	601	2	70
65	SonyAIBORobotSurface2	Sensor	27	953	2	65
66	StarLightCurves	Sensor	1000	8236	3	1024
67	Strawberry	Spectro	613	370	2	235
68	SwedishLeaf	Image	500	625	15	128
69	Symbols	Image	25	995	6	398
70	SyntheticControl	Simulated	300	300	6	60
71	ToeSegmentation1	Motion	40	228	2	277
72	ToeSegmentation2	Motion	36	130	2	343
73	Trace	Sensor	100	100	4	275
74	TwoLeadECG	ECG	23	1139	2	82
75	TwoPatterns	Simulated	1000	4000	4	128
76	UWaveGestureLibraryAll	Motion	896	3582	8	945
77	UWaveGestureLibraryX	Motion	896	3582	8	315
78	UWaveGestureLibraryY	Motion	896	3582	8	315
79	UWaveGestureLibraryZ	Motion	896	3582	8	315
80	Wafer	Sensor	1000	6164	2	152
81	Wine	Spectro	57	54	2	234
82	WordSynonyms	Image	267	638	25	270
83	Worms	Motion	181	77	5	900
84	WormsTwoClass	Motion	181	77	2	900
85	Yoga	Image	300	3000	2	426
86	ACSF1	Device	100	100	10	1460
87	AllGestureWiimoteX	Sensor	300	700	10	Vary
88	AllGestureWiimoteY	Sensor	300	700	10	Vary
89	AllGestureWiimoteZ	Sensor	300	700	10	Vary
90	BME	Simulated	30	150	3	128
91	Chinatown	Traffic	20	343	2	24
92	Crop	Image	7200	16800	24	46
93	DodgerLoopDay	Sensor	78	80	7	288
94	DodgerLoopGame	Sensor	20	138	2	288
95	DodgerLoopWeekend	Sensor	20	138	2	288
96	EOGHorizontalSignal	EOG	362	362	12	1250
97	EOGVerticalSignal	EOG	362	362	12	1250
98	EthanolLevel	Spectro	504	500	4	1751
99	FreezerRegularTrain	Sensor	150	2850	2	301
100	FreezerSmallTrain	Sensor	28	2850	2	301
101	Fungi	HRM	18	186	18	201
102	GestureMidAirD1	Trajectory	208	130	26	Vary
103	GestureMidAirD2	Trajectory	208	130	26	Vary
104	GestureMidAirD3	Trajectory	208	130	26	Vary
105	GesturePebbleZ1	Sensor	132	172	6	Vary
106	GesturePebbleZ2	Sensor	146	158	6	Vary

107	GunPointAgeSpan	Motion	135	316	2	150
108	GunPointMaleVersusFemale	Motion	135	316	2	150
109	GunPointOldVersusYoung	Motion	136	315	2	150
110	HouseTwenty	Device	40	119	2	2000
111	InsectEPGRegularTrain	EPG	62	249	3	601
112	InsectEPGSmallTrain	EPG	17	249	3	601
113	MelbournePedestrian	Traffic	1194	2439	10	24
114	MixedShapesRegularTrain	Image	500	2425	5	1024
115	MixedShapesSmallTrain	Image	100	2425	5	1024
116	PickupGestureWiimoteZ	Sensor	50	50	10	Vary
117	PigAirwayPressure	Hemodynamics	104	208	52	2000
118	PigArtPressure	Hemodynamics	104	208	52	2000
119	PigCVP	Hemodynamics	104	208	52	2000
120	PLAID	Device	537	537	11	Vary
121	PowerCons	Power	180	180	2	144
122	Rock	Spectrum	20	50	4	2844
123	SemgHandGenderCh2	Spectrum	300	600	2	1500
124	SemgHandMovementCh2	Spectrum	450	450	6	1500
125	SemgHandSubjectCh2	Spectrum	450	450	5	1500
126	ShakeGestureWiimoteZ	Sensor	50	50	10	Vary
127	SmoothSubspace	Simulated	150	150	3	15
128	UMD	Simulated	36	144	3	150