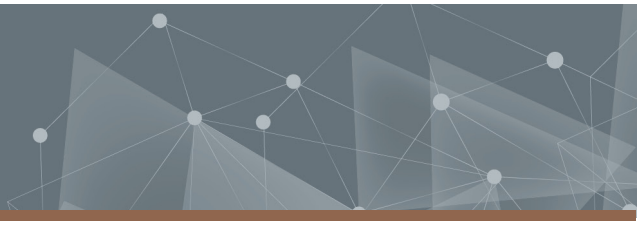
CHALMERS
UNIVERSITY OF TECHNOLOGY



Bridging the Resolution Gap: Conditional Generative Model for High-Fidelity Time-Series in Automotive Applications

Master's Thesis in Computer science and engineering

Tharinrath Jatupattrapiboorn
Fizza Farooq

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Degree project report 2025

Bridging the Resolution Gap: Conditional Generative Model for High-Fidelity Time Series in Automotive Applications

Tharinrath Jatupattrapiboorn
Fizza Farooq

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2025

Bridging the Resolution Gap: Conditional Generative Model for High-Fidelity Time Series in Automotive Applications
Tharinrath Jatupattrapiboorn
Fizza Farooq

Supervisor: Alexander Gower, Department of Computer Science and Engineering
Advisor: Björn Lindenberg, Volvo Group Trucks Technology
Examiner: Ahmed Ali-Eldin Hassan, Department of Computer Science and Engineering

Master's Thesis 2025
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2025

Bridging the Resolution Gap: Conditional Generative Model for High-Fidelity Time Series in Automotive Applications
Tharinrath Jatupattrapiboorn
Fizza Farooq
Department of Computer Science and Technology
Chalmers University of Technology

Abstract

The automotive sector increasingly relies on high-resolution vehicle data for advanced analytics, yet readily available customer vehicle data is often sparse and low-resolution due to hardware, cost, and privacy constraints. Existing time-series generative models frequently struggle with long sequences (exceeding 1000 timesteps) and lack robust mechanisms for controlled generation using continuous conditional inputs. This thesis aims to bridge this data resolution gap by developing and evaluating advanced machine learning models capable of generating high-fidelity, long-sequence time-series data from low-resolution conditional inputs.

We evaluated current state-of-the-art time-series generative models for their efficacy in handling long sequences, identifying Variational Autoencoders (VAEs), particularly TimeVAE, as the most promising base due to their fast training time and superior predictive performance. A novel conditional generative model was then developed by integrating TimeVAE with a Transformer encoder, adapting the PatchTST architecture to encode the conditional information. Key architectural enhancements, including Rotary Positional Embeddings, Layer Normalization, and specifically, a residual integration approach for incorporating conditional information, were explored. Further improvements like "free bits" and "conditional prior" were implemented to mitigate posterior collapse and enhance overall model performance.

Our findings indicate that state-of-the-art models generally underperform on long sequences , and naive conditional integration is ineffective due to complex gradient flows. However, the proposed residual integration improved the model's ability to leverage conditional information. The combined application of "free bits" and "conditional prior" alleviated posterior collapse, leading to a more robust model. Crucially, the purely data-driven model was able to generate physically plausible long-sequence time-series data, with three key physical metrics showing less than 16% deviation from real signals, without explicit physics-based training. This demonstrates a viable solution for controlled high-fidelity time-series generation in automotive applications

Keywords: Time-Series Generation, Conditional Generative Models, High-Fidelity Data, Automotive Applications, Variational Autoencoders (VAEs), Transformers, Machine Learning, Deep Learning, Data Synthesis, Automotive Data Analytics

Acknowledgements

We extend our sincere gratitude to our supervisor, Alexander Gower, for his invaluable guidance, support, and encouragement. We are also deeply thankful to our advisor, Björn Lindenberg of Volvo Group Trucks Technology, for his practical insights and profound domain knowledge throughout this master's thesis. Finally, we acknowledge Chalmers University of Technology and Volvo Group for providing the essential environment and resources for this research.

Tharinrath Jatupattrapiboorn, Fizza Farooq, Gothenburg, June 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

BERT	Bidirectional Encoder Representations from Transformer
CVAE	Conditional Variational Autoencoder
DPM	Diffusion Probabilistic Models
DS	Discriminative Score
DSL	Data Science Lab
ELBO	Evidence Lower Bound
GAN	Generative Adversarial Network
GPT	Generative Pre-trained Transformer
GRU	Gated Recurrent Unit
HF	High Frequency
KL	Kullback-Leibler
LF	Low Frequency
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
PS	Predictive Score
RNN	Recurrent Neural Network
RevIN	Reversible Instance Normalization
RoPE	Rotary Positional Embedding
STFT	Short-Time Fourier Transform
TTS-GAN	Transformer Time-Series Generative Adversarial Network
VAE	Variational Autoencoder
VQ-VAE	Vector Quantized Variational Autoencoder

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices

t	Index for time step
i	Index for samples in dataset
T_l	Number of time steps in low-resolution sequence
T_h	Number of time steps in high-resolution sequence

Sets

$\mathcal{X}$	Set of input sequences (low-resolution signals)
$\mathcal{Y}$	Set of target sequences (high-resolution signals)
$\hat{\mathcal{Y}}$	Set of generated sequences (high-resolution signals)
$\mathcal{D}_x$	Set of input features
$\mathcal{D}_y$	Set of target features
$\mathcal{D}_{\text{train}}$	Training dataset with (input, target) pairs
$\mathcal{D}_{\text{inference}}$	Inference dataset with (input, generated) pairs

Parameters

k	Downsampling factor between low-res and high-res signals
β	Weight for reconstruction loss in the ELBO objective
$\mu(x)$	Mean of approximate posterior distribution in VAE
$\sigma(x)$	Standard deviation of approximate posterior distribution in VAE
ρ	Air density (for physics-based plausibility)

C_{rr}	Coefficient of rolling resistance
C_dA	Drag coefficient times frontal area
g	Acceleration due to gravity
m	Vehicle mass

Variables

x_t	Conditional input at time step t
y_t	Target high-resolution signal at time step t
$\hat{y}_t$	Generated high-resolution signal at time step t
z	Latent variable in VAE latent space
a_t	Acceleration at time step t
v_t	Velocity at time step t
F_{rr}	Rolling resistance force
F_{aero}	Aerodynamic drag force
F_{mech}	Mechanical force on vehicle
W_{eng}	Work done by engine
W_{brake}	Work done by braking
W_{aero}	Work done against aerodynamic drag
$W_{rolling}$	Work done to overcome rolling resistance

Contents

List of Figures

List of Tables

1
Introduction

This thesis explores how advanced machine learning models can tackle conditional time-series generation task. This chapter outlines our research context, aims, core research questions and limitations.

1.1 Background

Time-series generative models play a vital role in various domains, driving innovation in areas such as synthetic data generation, data augmentation, audio synthesis and privacy protection [4, 20, 39, 12, 21, 33, 6, 38]. Despite their widespread applicability, research on these models remains less explored compared to the advancement in natural language processing [5, 37] and computer vision [29, 15, 35].

Prior research on synthetic time-series generation has largely focused on Generative Adversarial Network (GAN) based approaches [8, 39, 12, 21, 23]. While effective, they still often struggle with inherent GAN challenges like unstable training and mode collapse. Although Variational Autoencoder (VAE) based time-series generation models such as TimeVAE [4], Time-VQVAE [20] and KoVAE [26], have been introduced, they still have received comparatively less attention. This is despite VAE's known advantages such as more stable training, less prone to mode collapse, and consistently outperforming GANs in image generation tasks [36, 29, 31, 34, 35, 3].

However, most existing state-of-the-art models mentioned, whether GAN or VAE based, lack support for controlled generation through conditional inputs. While few of them provide limited mechanisms for conditional generation based on categorical labels [20, 23, 22], they tend to overlook the incorporation of meaningful continuous information, which is vital for many time-series applications. Addressing these issues is essential for advancing the practical applicability of these models in complex real-world scenarios.

This challenge is also evident in the automotive sector, where data-driven techniques are increasingly employed to enhance vehicle performance, analyze user behaviour, and generate operational insights. At Volvo GTT, two types of vehicle data are collected for these purposes: first, *high-resolution test vehicle data* obtained from various signals during extensive truck testing, and second, sparse *low-resolution customer vehicle data*, obtained from relatively less number of signals, constrained by hardware limitations, cost considerations, and regulatory requirements such as

the EU AI Act and GDPR. Due to these limitations, the low-resolution customer vehicle data restricts advanced analytics that depend on high-frequency data to accurately capture dynamic behaviours.

Beyond the challenge of conditional generation, a yet underexplored limitation in existing time-series generative models lies in their ability to handle long sequences. While many models demonstrate strong performance on short to medium-length time series, their performance and scalability on sequences exceeding 1000 timesteps, which are typical in real-world industrial applications like automotive data, remain largely untested. This gap in research presents significant hurdles for practical deployment, as accurately capturing long-range temporal dependencies is crucial for generating realistic and useful data in complex systems.

To enable effective time-series data generation in this context, controlled generation that leverages continuous conditional information is required, all while ensuring the quality of the generated data. In this thesis, we developed a novel architecture for conditional generation of time-series data based on continuous information. Our approach employs Transformer architectures with patch-based embeddings, a technique successfully demonstrated in time-series forecasting tasks [27, 40, 24]. Our primary aim is to explore how such architecture could be used to address the challenge of conditional generation, especially for long sequences, such as generating fuel consumption data based on continuous inputs like distance travelled and engine speed. We investigate whether this mechanism enables the effective integration of conditional information into existing models, potentially offering a more robust and controllable solution for time-series generation, and assess its capability to produce physically plausible data.

1.2 Aim

Our objective is to develop and evaluate advanced machine learning models for generating long-sequence time-series data which is not present in customer vehicle datasets, leveraging high-resolution test vehicle data. This involves evaluating current state-of-the-art models for their efficacy in generating sequences longer than typically explored in research (e.g., beyond 1000 timesteps), focusing on both training efficiency and generated data quality. We aim to establish robust evaluation metrics to assess data quality, ensuring the generated data remains plausible and accurately preserves crucial temporal relationships. Furthermore, we will investigate the effects of incorporating continuous conditional information into the most promising model to enhance controlled generation. Ultimately, the developed method should be versatile enough to adapt across different vehicle types and operating conditions, demonstrating robustness and reliability in real-world automotive applications, including the ability to generate physically plausible time-series data without explicit physics-based training

1.3 Research Questions

1. Can current state-of-the-art time-series generative models effectively generate long sequences (e.g., beyond 1000 timesteps)?
2. Among effective models, which one demonstrates superior performance in generating long sequences, considering both computational efficiency (training time) and the quality of the generated time-series data?
3. What are the effects of incorporating continuous conditional information into the selected best-performing model for long-sequence generation?
4. Can a purely data-driven generative model, without explicit physics-based training, learn to produce physically plausible long-sequence time-series data?

1.4 Limitations

There are some limitations that arise primarily from constraints in time and available resources. We discard the exploration of Diffusion Probabilistic Models even though they show strong performance in generative tasks. This is due to the complexity of the model and our constraints. We focus on generating only one feature that is of highest importance for our application, i.e. instantaneous velocity, thus limiting the scope of this thesis to only univariate target signal. Furthermore, we didn't perform extensive hyperparameter searching. While we could have experimented with more state-of-the-art models, we limited their number to four due to the constraints mentioned above.

2
Problem Formulation

In vehicle data analysis, a key challenge lies in generating high-resolution time-series data when only low-resolution measurements are available. As discussed in Chapter 1, while Volvo's test vehicles provide rich, high-frequency sensor data, customer vehicle data is limited to lower sampling rates due to hardware constraints, cost considerations, and privacy regulations. This disparity creates a resolution gap that restricts the applicability of advanced data-driven methods on customer data. Our goal is to bridge this gap using generative models capable of synthesizing high-resolution time series conditioned on low-resolution inputs.

Specifically, we aim to learn the conditional distribution $p(\mathbf{y} \mid \mathbf{x})$, which represents the likelihood of observing a specific high-resolution time series $\mathbf{y}$ given a corresponding low-resolution time series $\mathbf{x}$. Here, $\mathbf{x}$ denotes a segment of low-frequency signals, such that $\mathbf{x} = (x_1, x_2, \ldots, x_{T_l})$, where T_l is the number of data points in the low-resolution time-series and each $x_t \in \mathbb{R}^{D_x}$ is a vector of D_x features. Conversely, $\mathbf{y}$ represents a segment of the corresponding high-frequency signals, such that $\mathbf{y} = (y_1, y_2, \ldots, y_{T_h})$, where T_h is the number of data points in the high-resolution time-series and each $y_t \in \mathbb{R}^{D_y}$ is a vector of D_y features. Notably, T_h is typically a multiple of T_l by a scaling factor k (i.e., $T_h = kT_l$; for example, if $\mathbf{x}$ is sampled per minute and $\mathbf{y}$ is sampled per second, then $k = 60$). Our goal is to generate $\hat{\mathbf{y}}$, the synthetic high-frequency time series, which we expect to closely approximate the true $\mathbf{y}$.

To achieve this, we leverage **datasets** from two distinct sources:

- $\mathcal{D}_{\text{train}}$: Training data from test vehicles.
- $\mathcal{D}_{\text{inference}}$: Partially observed data from customer vehicles.

The test vehicle data offers complete $(\mathbf{x}, \mathbf{y})$ pairs from a training set $\mathcal{D}_{\text{train}} = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{N_{train}}$, providing a representation of the full vehicle population's behaviour. While the customer vehicle data only provide $\mathbf{x}$, thus the need to generate $\hat{\mathbf{y}}$ to complete $\mathcal{D}_{\text{inference}} = \{(\mathbf{x}^{(i)}, \hat{\mathbf{y}}^{(i)})\}_{i=1}^{N_{inference}}$, where N denotes number of samples in the dataset.

Our primary objective is to train a conditional generative model using the observed $(\mathbf{x}, \mathbf{y})$ pairs from $\mathcal{D}_{\text{train}}$ to accurately estimate $p(\mathbf{y} \mid \mathbf{x})$. Once trained, this model will be applied to the customer vehicle data. Using $\mathbf{x}$ from $\mathcal{D}_{\text{inference}}$, the model will generate the corresponding high-frequency signals, $\hat{\mathbf{y}}$, for these customer vehicles. This approach enables us to unlock advanced analytics and insights from the cus-

tomer vehicles that would otherwise be limited by uncaptured data and mismatch in resolution, directly addressing the need for controlled generation based on continuous inputs, as discussed in Chapter 1.

3

Theory

Following the problem definition in Chapter 2, this chapter establishes the essential theoretical foundation for our work. We will explore three main categories of generative models in this section, Variational Autoencoders (VAEs), Generative Adversarial Networks (GANs), and Transformer models, detailing their core principles and architectural designs, which would be used further in our methodology.

3.1 Variational Autoencoders (VAEs)

Generative models aim to learn the underlying distribution of data to generate new, similar samples. Variational Autoencoders (VAEs) [17] represent a powerful class of such models, distinguishing themselves by framing data generation within a probabilistic framework. Unlike traditional autoencoders that learn a direct, deterministic mapping, VAEs learn to model data by mapping inputs to a probabilistic latent space, enabling the generation of novel data points through sampling. Their core innovation lies in the integration of variational inference, which provides a principled approach to optimize this probabilistic mapping and regularize the latent space. VAEs are particularly appealing for our task due to their stable training and ability to model complex data distributions probabilistically, which is crucial for generating realistic high-resolution time series.

3.1.1 Probabilistic Formulation and Model Architecture

At the heart of the VAE framework is the assumption that each observed data point, $\mathbf{x}$, is generated from a lower-dimensional, unobserved (latent) variable, $\mathbf{z}$. This generative process is described by the joint probability $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x}|\mathbf{z})p(\mathbf{z})$. Typically, a simple prior distribution, $p(\mathbf{z})$, such as a multivariate standard normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$, is assumed for the latent variables. The conditional likelihood $p(\mathbf{x}|\mathbf{z})$ models how data is generated from these latent factors.
Since, directly computing the true posterior distribution $p(\mathbf{z}|\mathbf{x})$ is generally intractable, VAEs introduce an inference model, often called the **encoder**, to approximate this posterior. Parameterized by a neural network, the encoder learns an approximate posterior $q(\mathbf{z}|\mathbf{x})$. Commonly, $q(\mathbf{z}|\mathbf{x})$ is chosen as a Gaussian distribution, whose mean $\boldsymbol{\mu}(\mathbf{x})$ and variance $\boldsymbol{\sigma}^2(\mathbf{x})$ are outputs of the encoder network:

$$q(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}(\mathbf{x}), \text{diag}(\boldsymbol{\sigma}^2(\mathbf{x}))) \tag{3.1}$$

To enable backpropagation through the stochastic sampling process of $\mathbf{z}$, the **reparameterization trick** is employed. This technique re-expresses the sampling of $\mathbf{z}$ as a deterministic function of a simple random variable $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$:

$$\mathbf{z} = \boldsymbol{\mu}(\mathbf{x}) + \boldsymbol{\sigma}(\mathbf{x}) \odot \boldsymbol{\epsilon} \tag{3.2}$$

Once a latent code $\mathbf{z}$ is sampled, a second neural network, the **decoder** (or generative model), defines the likelihood $p(\mathbf{x}|\mathbf{z})$. Its role is to reconstruct the input $\mathbf{x}$ from its latent representation $\mathbf{z}$, effectively learning the mapping from the latent space back to the data space.

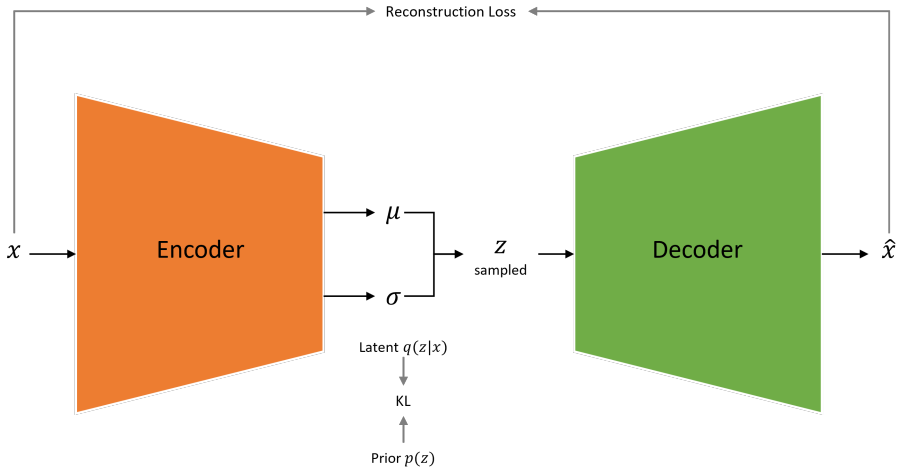


Figure 3.1: Conceptual diagram of a Variational Autoencoder (VAE). The encoder maps input data $\mathbf{x}$ to the parameters of a latent distribution $q(\mathbf{z}|\mathbf{x})$. A latent variable $\mathbf{z}$ is sampled using the reparameterization trick, which is then passed to the decoder to reconstruct the original input as $\hat{\mathbf{x}}$. The training objective balances reconstruction accuracy with regularization of the latent space.

3.1.2 The Evidence Lower Bound (ELBO) Objective

Training a VAE involves optimizing a carefully constructed objective function: the Evidence Lower BOund (ELBO) on the marginal likelihood $p(\mathbf{x})$. Maximizing the ELBO corresponds to maximizing the likelihood of the observed data under the generative model. The ELBO is formally expressed as:

$$\mathcal{L}(\mathbf{x}) = \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})] - \mathrm{KL}\left(q(\mathbf{z}|\mathbf{x}) \,\|\, p(\mathbf{z})\right) \tag{3.3}$$

This objective balances two critical aspects of VAE training:

1. **Reconstruction Term ($\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})]$):** This component encourages the decoder to accurately recreate the original input data $\mathbf{x}$ from its sampled latent representation $\mathbf{z}$. Intuitively, it ensures that the VAE can effectively compress and decompress data without losing too much information. The specific form of this term depends on the data type; for instance, a Gaussian likelihood is used for continuous data, while a Bernoulli likelihood is suitable for binary data.

2. **KL Divergence Term ($\mathrm{KL}\left(q(\mathbf{z}|\mathbf{x}) \,\|\, p(\mathbf{z})\right)$):** This regularization term plays a crucial role in shaping the latent space. It measures the dissimilarity between the approximate posterior $q(\mathbf{z}|\mathbf{x})$ and the chosen prior $p(\mathbf{z})$. By minimizing this divergence, the VAE is encouraged to learn a latent space where individual latent distributions $q(\mathbf{z}|\mathbf{x})$ are close to the prior. This prevents the encoder from collapsing into a trivial solution and ensures that the latent space is well-structured and continuous, allowing for meaningful sampling and interpolation for generation.

During training, the overall objective is to maximize the ELBO, which is equivalent to minimizing its negative form:

$$\mathcal{L}_{\mathrm{VAE}} = -\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})] + \mathrm{KL}\left(q(\mathbf{z}|\mathbf{x}) \,\|\, p(\mathbf{z})\right) \tag{3.4}$$

The reparameterization trick is fundamental here, as it allows for the gradients of both terms to be computed and backpropagated through the entire network, enabling efficient end-to-end training of both the encoder and decoder.

3.2 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) [10] revolutionized generative modeling by introducing an adversarial training paradigm. Instead of directly modeling data distributions, GANs employ a competitive framework where two neural networks—a generator and a discriminator—are trained simultaneously. The generator aims to produce synthetic data samples that are indistinguishable from real data, while the discriminator learns to differentiate between real and generated samples. This adversarial process drives both networks to continuously improve, ultimately leading to a generator capable of synthesizing highly realistic data. GANs offer a powerful framework for generating highly realistic synthetic time series, making them a strong candidate for bridging the data resolution gap.

3.2.1 The Adversarial Game: Generator and Discriminator

The core of a GAN lies in its two competing components:

1. **Generator (G):** This network takes a random noise vector $\mathbf{z}$, typically sampled from a simple prior distribution $p(\mathbf{z})$ (e.g., a multivariate standard normal distribution), and transforms it into a synthetic data sample $G(\mathbf{z})$ that attempts to mimic the true data distribution $p_{\mathrm{data}}(\mathbf{x})$.
2. **Discriminator (D):** This network acts as a binary classifier. It receives an input, which can be either a real data sample $\mathbf{x} \sim p_{\mathrm{data}}(\mathbf{x})$ or a generated sample $G(\mathbf{z})$, and outputs a probability $D(\mathbf{x})$ (or $D(G(\mathbf{z}))$) indicating how likely the input is to be real.

The training process is formulated as a minimax game, where the discriminator tries to maximize its ability to correctly classify real and fake data, while the generator tries to minimize the discriminator's ability to distinguish its outputs from real data. This dynamic struggle is captured by the value function $V(D, G)$:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\left[\log D(\mathbf{x})\right] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[\log\left(1 - D(G(\mathbf{z}))\right)\right] \tag{3.5}$$

At equilibrium, the generator is expected to produce samples that are perfectly indistinguishable from real data, meaning $D(G(\mathbf{z})) = 0.5$ for any generated sample.

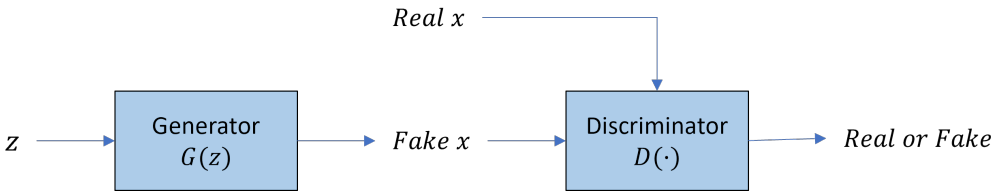


Figure 3.2: Conceptual diagram of a Generative Adversarial Network (GAN). The Generator (G) takes a random noise vector and produces fake data. The Discriminator (D) receives both real data and fake data, attempting to distinguish between them. This adversarial game drives the generator to create increasingly realistic samples and the discriminator to become more adept at detection.

3.2.2 Training Objectives

The overall training objective involves iteratively optimizing the discriminator and generator.

1. **Discriminator Training:** The discriminator is trained to maximize its objective, which involves correctly identifying real data as real and generated data as fake. This is equivalent to maximizing the log-likelihood of correct classification:

$$\mathcal{L}_D = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\left[\log D(\mathbf{x})\right] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[\log\left(1 - D(G(\mathbf{z}))\right)\right] \tag{3.6}$$

   The discriminator's goal is to output a high value for real inputs and a low value for generated inputs.

2. **Generator Training:** The generator's goal is to trick the discriminator into believing its generated samples are real. While the original formulation has the generator minimizing $\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[\log\left(1 - D(G(\mathbf{z}))\right)\right]$, this objective can suffer from vanishing gradients early in training when the discriminator easily rejects generated samples. To provide stronger gradients and facilitate training, a common practical modification is to have the generator maximize the probability of the discriminator being fooled. This is equivalent to maximizing $\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[\log D(G(\mathbf{z}))\right]$, or minimizing its negative:

$$\mathcal{L}'_G = -\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[\log D(G(\mathbf{z}))\right] \tag{3.7}$$

   This modified loss encourages the generator to produce samples that receive high scores from the discriminator.

The adversarial training process continues until the generator produces samples that the discriminator can no longer confidently distinguish from real data, indicating that the generator has learned a compelling approximation of the data distribution.

3.3 Transformers

Transformers [37] have fundamentally reshaped the landscape of natural language processing and other sequence-based tasks. Unlike traditional recurrent neural networks (RNNs) that process sequences token-by-token, Transformers excel at processing entire sequences in parallel, significantly improving computational efficiency and enabling the capture of long-range dependencies. This is primarily achieved through their **self-attention** mechanism, which allows each element in a sequence to dynamically weigh the importance of all other elements when computing its own representation. The architecture is typically organized into an encoder-decoder framework, though many influential models (e.g., BERT [5] and GPT [28]) utilize either only the encoder or decoder component.

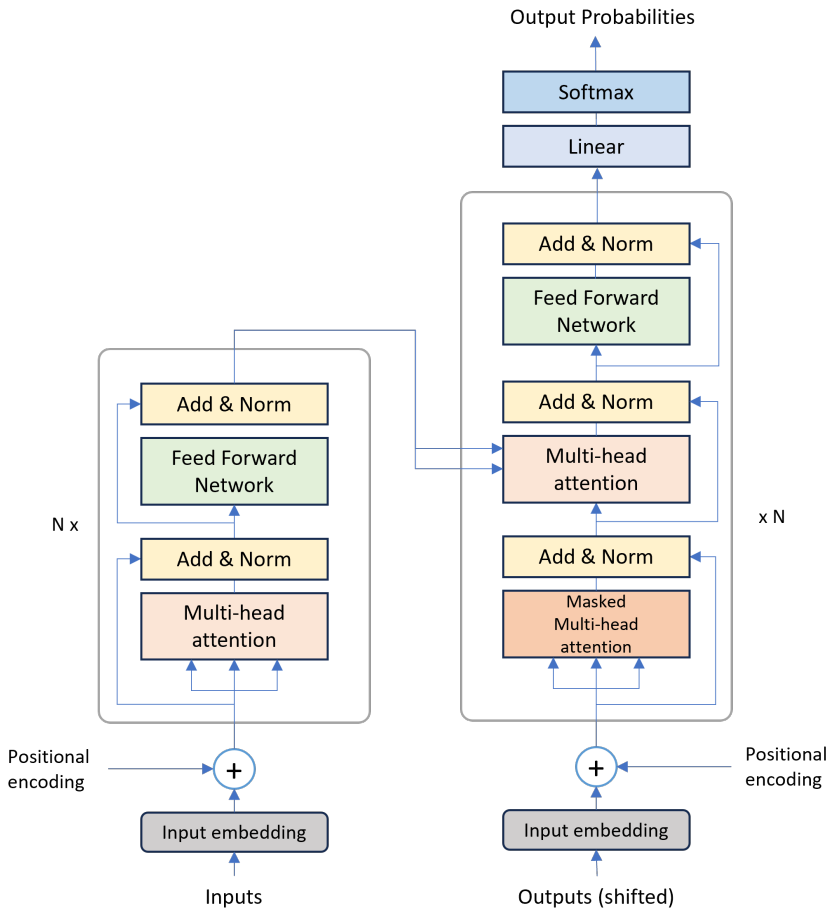


Figure 3.3: Overview of the Transformer architecture, adapted from [37]. It comprises an encoder stack and a decoder stack. Both utilize multi-head self-attention, feed-forward networks, residual connections, and layer normalization. The decoder additionally incorporates an encoder-decoder attention mechanism and masked self-attention.

3.3.1 Key Architectural Components

The power of the Transformer stems from several key components. We will briefly explain each component one by one.

3.3.1.1 Input Embeddings and Positional Encoding

Before being processed by the network, each token in the input sequence is converted into a continuous, high-dimensional vector using an embedding layer. Crucially, because the Transformer lacks recurrent or convolutional structures to inherently process sequence order, **positional encodings** are added to these embeddings. These encodings provide information about the absolute or relative position of each token within the sequence, ensuring that the model understands the order of each token. Positional encodings can be learned or, as in the original work, generated using sinusoidal functions.

3.3.1.2 Self-Attention Mechanism

At the core of the Transformer lies the self-attention mechanism. For each token in the sequence, self-attention computes a weighted sum of the representations of *all* tokens in the same sequence. The weights are dynamically determined by computing a compatibility score between a token's **query** (Q) vector and every other token's **key** (K) vector. These scores are then used to weight the **value** (V) vectors. The scaled dot-product attention function is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.8}$$

where d_k is the dimensionality of the key vectors, used to scale the dot products and prevent vanishing gradients. This mechanism allows the model to selectively focus on relevant parts of the input sequence when encoding a particular token, regardless of their position in the sequence.

3.3.1.3 Multi-Head Attention

To enrich the model's ability to capture diverse relationships, the Transformer employs **multi-head attention**. Instead of performing a single self-attention operation, the input is linearly projected multiple times into different "heads". Each head learns to focus on different aspects of the relationships between tokens (e.g., syntactic vs. semantic dependencies). The outputs from these independent attention heads are then concatenated and linearly transformed to form the final, more comprehensive representation.

3.3.1.4 Feed-Forward Networks

Following the attention layers, a position-wise feed-forward network is applied independently to each position in the sequence. These networks typically consist of two linear transformations with a non-linear activation (e.g., ReLU) in between. They

allow the model to process the information aggregated by the attention mechanism further.

3.3.1.5 Layer Normalization and Residual Connections

To facilitate training and enable the construction of very deep networks, each sub-layer (e.g., self-attention and feed-forward network) within the Transformer block is wrapped with a **residual connection** and followed by **layer normalization**. Residual connections help address the vanishing gradient problem by allowing gradients to flow directly through the network, while layer normalization stabilizes training by normalizing the activations across features for each sample.

3.3.2 Encoder and Decoder Structure

The complete Transformer model is typically structured as an encoder-decoder architecture:

3.3.2.1 Encoder

The encoder consists of a stack of identical layers. Each encoder layer contains two primary sub-layers: a multi-head self-attention mechanism and a position-wise feed-forward network. The encoder's role is to process the input sequence and produce a sequence of context-rich representations, capturing the global dependencies within the input.

3.3.2.2 Decoder

The decoder is also composed of a stack of identical layers, but each decoder layer has three main sub-layers: a masked multi-head self-attention mechanism, an encoder-decoder attention mechanism, and a position-wise feed-forward network.

- The **masked self-attention** in the decoder ensures that predictions for a given output position can only depend on known outputs at earlier positions. This masking is crucial for autoregressive generation, where the model generates output one token at a time.
- The **encoder-decoder attention** (or cross-attention) mechanism allows the decoder to attend to the context-rich output from the encoder stack, effectively enabling the decoder to focus on relevant parts of the input sequence when generating each output token.

This modular and parallelizable design, particularly the self-attention mechanism, enables Transformers to efficiently handle long sequences and has made them the backbone of many state-of-the-art models across various domains.

4
Methods

This chapter outlines the methodology developed to answer our research questions and construct the conditional generative model. We detail the data preparation, evaluation metrics, model architectures, and experimental design in this chapter.

4.1 Datasets and Preprocessing

- **Scope Identification:** We sourced data from Volvo's long-haul truck models which are mostly heavy duty trucks used for long distance drive. The reason was to get a sufficient number of data samples that have atleast 30 minutes long drive data.
- **Data Preparation:** We use test vehicle data as mentioned in chapter 2. The data is available as trips, where a trip contains data from igniting the engine until turning it off, recording many time-series features including speed, distance covered, elevation (from sea-level), engine power, fuel consumption, and engine speed. The feature speed is our target, while the rest of the five features are used as conditional information. The sampling rate of this data is 1 Hz.

  We cut trips into small segments of 30 minutes long and treat each segment as an independent one for simplicity. This makes the length of target speed feature to be 1800 timesteps, which we then normalize using min-max scaling. For conditional information, we downsample by taking one timestep value every 60 timesteps (a downsampling factor of 60). This creates sparse, minute-wise information, comprising 30 timesteps across all five conditional features. We do not pre-normalize the conditional information, as it undergoes instance normalization within the transformer encoder architecture.
- **Pre-Procesing:** We filter out non-representative trip segments from our data, for example, trips where the distance covered is less than 7 km, since we are interested in data where the truck is being driven for a significant amount of time and it is not in an idle position for long. For this, we filter out trip segments having an average speed below 40 kmh. Similarly, we apply a filter on elevation values using a realistic range (0-2000 meters) to avoid any wrong sensor values. For engine power, fuel consumption, and engine speed, we apply filters according to their typical value ranges that are expected of the trucks used in data.
- **Datasets:** We have two datasets, a small dataset for our development work to facilitate experimentation and lower training times, we call this one small DSL

dataset. For another dataset, we call it the large DSL dataset, which is used in our final step of development when comparing different methods, as well as reporting results. The small dataset consists of 2,644 trip segments in the training set and 882 in the validation set. The full dataset consists of 14,103 trip segments in the training set, with 1,764 samples each in the validation and test set. The test set was held out exclusively for final model evaluation as completely unseen data.

We used another dataset called Signet for our baseline model evaluation. This dataset was readily available to us from Volvo at the beginning, when the two datasets described earlier were yet to be available. It is similar to the other as it consists of the trips data from the same truck models, but it does not have all the features needed. This data includes only speed, fuel consumption, and road inclination, with 1024 timesteps in each data sample. This length of sequences was sufficient to test our baseline models for long sequence generation. The sampling rate was 1 Hz in this data as well.

4.2 Evaluation Metrics

4.2.1 Time-series Quality

- Discriminative Score (DS): We measure the Discriminative Score (DS) to evaluate how well the generated time series replicate the characteristics of the original data [39]. This involves training a post-hoc classification model, typically a 2-layer Gated Recurrent Unit (GRU) or Long Short-Term Memory (LSTM) network, to differentiate between original and synthetic series. Each original series is labeled as "real", while its generated counterpart is labeled as "synthetic". The generated data's ability to deceive the classifier is quantified by the classification accuracy on a test set, where a lower value indicates better fidelity of the generative model.
- Predictive Score (PS): The Predictive Score (PS) assesses the utility of the synthetic data by determining if models trained on it can accurately predict future values of real data [39]. For this, we train a post-hoc time series prediction model solely on the synthetic dataset. Using GRUs or LSTMs, this model is tasked with predicting either the subsequent temporal vectors of individual series [11, 39] or the complete series vector [13]. We then evaluate the model's predictive performance on the original, real dataset, typically using the Mean Absolute Error (MAE) or Mean Squared Error (MSE) to quantify its accuracy.

4.2.2 Computational Efficiency

- Number of parameters: This metric represents the total count of trainable parameters in the model.
- Training time: We measure the wall-clock duration of the model's training.
- Inference time: This refers to the wall-clock time taken to generate each sequence.

4.2.3 Physical Plausibility

We evaluate physical plausibility in conditional generative models by comparing the energy function of generated sequences to that of real sequences, given their respective conditional information. This measure assesses how well the synthetic data adhere to the underlying physical principles dictated by its conditions. These domain-specific measures were proposed by our supervisor at Volvo.

- $W_{\text{rolling\_resist}}$ reflects the work done to overcome rolling resistance due to tire-ground interaction. This energy is linearly proportional to the distance travelled.

$$W_{\text{rolling\_resist}} = \sum_{t=1}^{T} (mgC_{\text{rr}} \cdot v_t) \tag{4.1}$$

- W_{aero} quantifies the work done against aerodynamic drag. It is proportional to both distance and the variance in velocity, since drag increases with the square of speed.

$$F_{\text{aero},t} = \frac{1}{2}\rho C_d A v_t^2 \tag{4.2}$$

$$W_{\text{aero}} = \sum_{t=1}^{T} F_{\text{aero},t} \cdot v_t = \sum_{t=1}^{T} \left(\frac{1}{2}\rho C_d A v_t^3\right) \tag{4.3}$$

- W_{eng} captures the energy supplied by the engine to accelerate the vehicle and overcome resistive forces. It is directly proportional to fuel consumption and indicates propulsion effort. The mechanical force $F_{\text{mech},t}$ represents the sum of all forces that must be overcome by the engine, including inertial, rolling resistance, aerodynamic drag, and gravitational forces.

$$W_{\text{eng}} = \sum_{t=1}^{T} [\mathbb{I}(F_{\text{mech},t} > 0) \cdot F_{\text{mech},t} \cdot v_t] \tag{4.4}$$

  where

$$F_{\text{mech},t} = ma_t + mgC_{\text{rr}} + \frac{1}{2}\rho C_d A v_t^2 + F_{\text{potential}} \tag{4.5}$$

$$F_{\text{potential}} = mg\Delta h \tag{4.6}$$

- W_{brake} measures the braking energy required to reduce speed. It accumulates the work done to counteract negative mechanical forces, representing energy lost due to deceleration. Higher values here suggest frequent or aggressive deceleration events.

$$W_{\text{brake}} = \sum_{t=1}^{T} [\mathbb{I}(F_{\text{mech},t} < 0) \cdot (-F_{\text{mech},t}) \cdot v_t] \tag{4.7}$$

where,

m : Vehicle mass [kg]
g : Acceleration due to gravity ($9.82\,\mathrm{m/s^2}$)
C_{rr} : Coefficient of rolling resistance
ρ : Air density ($1.2\,\mathrm{kg/m^3}$)
C_dA : Drag coefficient times frontal area
v_t : Velocity at time t
a_t : Acceleration at time t
Δh : change in altitude in a trip
$\mathbb{I}(\cdot)$: Indicator function (1 if condition is true, else 0)

4.3 Evaluation of State-of-the-Art Models

4.3.1 Selection of Candidate Models

For this evaluation, we selected four state-of-the-art models. TimeVAE [4] and Time-VQVAE [20] were chosen based on their consistent strong performance and recommendations from TSGBench [1], where they frequently outperformed other baselines across various metrics. We included TimeGAN [39] as a widely accepted and robust baseline. Our final selection was TTS-GAN [21], primarily due to its transformer-based architecture, which was reported to be effective for long sequence generation.

4.3.2 Architectural Details of Candidate Models

4.3.2.1 TimeVAE

TimeVAE is a straightforward yet efficient architecture for generating time-series data with reduced training times. In our evaluation, we use the Base TimeVAE architecture as described in [4]. The encoder is built using traditional 1-D convolutional layers followed by dense linear layers, which together form the latent space representation. In contrast, the decoder comprises a dense layer and transposed 1-D convolutional layers that work to reconstruct the data back to its original input shape.

The model accepts input data formatted as a three-dimensional array of size N x T x D, where N represents the batch size, T is the number of time steps, and D denotes the number of features or signals. The training is performed by minimizing a composite loss function as mentioned in section 3.1.2. This loss combines the reconstruction loss—with a weighted β term to emphasize accurate reconstruction in TimeVAE implementation—with the KL divergence loss, which regularizes the latent space. The weighted reconstruction term ensures that the model focuses more on reproducing the input data, while the KL divergence maintains a well-behaved latent distribution.

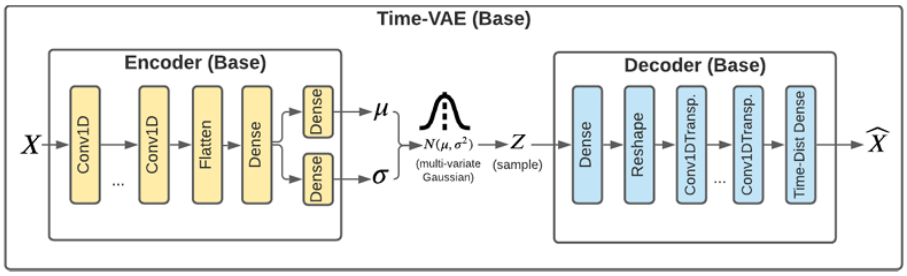


Figure 4.1: The Architecture of TimeVAE model from its original paper [4]. For further details on the architecture, refer to the source paper.

4.3.2.2 TimeVQVAE

TimeVQVAE is a two-stage generative model that introduces vector quantization and a bidirectional transformer-based prior to synthesize time-series data[20]. The first stage consists of encoding the time-series input into discrete latent tokens via a VQ-VAE architecture. The second stage uses two bidirectional transformers to learn the prior distributions over these latent tokens. The model separates the input into low-frequency (LF) and high-frequency (HF) components using a Short-Time Fourier Transform (STFT), which allows specialized encoding-decoding paths and enhances reconstruction quality for each frequency band.

The VQ encoders for LF and HF branches differ in downsampling rates, enabling the LF encoder to capture global structure and the HF encoder to retain detailed information. During training, reconstruction loss and vector quantization loss are minimized in both time and frequency domains. For sequence generation, LF tokens are sampled first from the LF transformer, and then HF tokens are sampled conditioned on the LF tokens using a second transformer. Finally, the combined tokens are decoded iteratively to generate better quality time-series.

The loss function of TimeVQVAE combines vector quantization and reconstruction objectives to ensure both efficient representation and faithful generation of time-series data. It is defined as:

$$L_{\text{VQ}} = L_{\text{codebook}} + L_{\text{recons}} \tag{4.8}$$

where L_{codebook} is the vector quantization loss which measures the distance between the continuous encoder outputs and the closest entries in the discrete codebook. and L_{recons} is the reconstruction loss defined over both time and frequency domains.

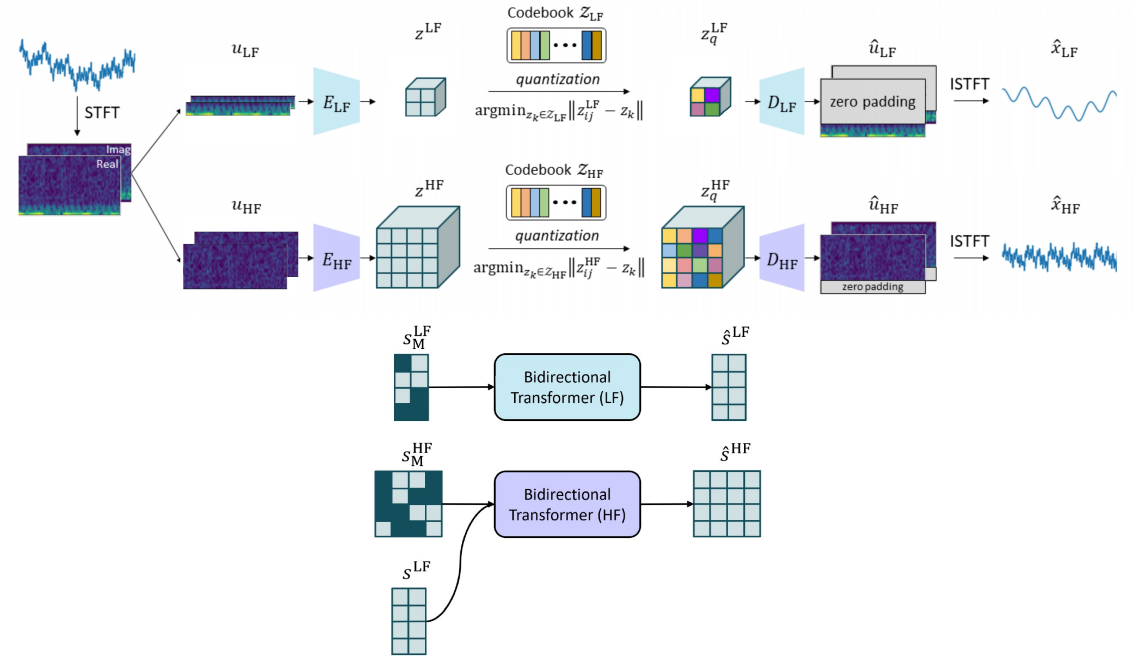


Figure 4.2: The Architecture of TimeVQVAE model from its original paper [20]. The top figure shows the Stage 1 (Codebook learning), and the bottom figure shows Stage 2 (Prior Learning). For further details on the architecture, refer to the source paper.

4.3.2.3 TimeGAN

Traditional GANs, when applied to sequential data fails to respect the temporal dynamics and relationship between variables across time. TimeGAN addresses this limitation by integrating unsupervised adversarial training with supervised learning signals that explicitly guide the model to to maintain the step-wise conditional distributions inherent in time-series data[39].

The key contribution is their hybrid architecture, which consists of four components: an embedding network, a recovery network, a generator, and a discriminator. The model learns a reversible mapping between original data and a latent space using the embedding and recovery networks. The generator then produces synthetic latent sequences, and the discriminator attempts to distinguish between real and generated sequences.

To better capture temporal dependencies, they introduces a novel supervised loss in addition to the standard adversarial loss. This supervised loss is based on minimizing the discrepancy between the latent representations of the real and generated sequences at each time step, effectively aligning their temporal dynamics. This dual-loss strategy (unsupervised and supervised) enhances the quality of generated data and predictive consistency. This makes TimeGAN a widely known baseline among GAN based models and our reason to choose this among candidate models.

However, they implemented their networks as a minimal example by using RNNs to isolate the source of gains in their experiments which could cause limitation in generating long sequences for our case.

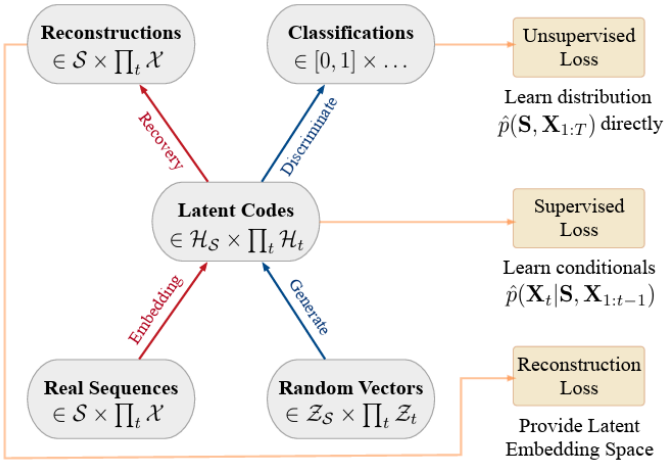


Figure 4.3: The architecture of TimeGAN model from its original paper [39]. For further details on the architecture, refer to the source paper.

4.3.2.4 TTS-GAN

We included a Transformer-based Time-series GAN as our final candidate model. Transformer-based Time-Series GAN (TTS-GAN) is a generative adversarial framework specifically designed to overcome the limitations of RNN-based GANs in modeling long and complex time-series sequences. TTS-GAN employs pure transformer encoder architectures for both its generator and discriminator, leveraging self-attention mechanisms to model long-range temporal dependencies more effectively.

TTS-GAN processes time-series data by treating sequences similarly to image patches, using positional encodings and transformer blocks to learn rich temporal representations. The generator takes a random noise vector, maps it to a sequence via transformer layers, and outputs synthetic time-series data. The discriminator then evaluates these sequences against real ones to determine authenticity.

To stabilize training in TTS-GAN, label heuristics such as soft labels (e.g., real label slightly less than 1, fake label slightly above 0) or label flipping (assigning 0 to real and 1 to fake samples occasionally) are used[21].

They define the discriminator and generator loss function as:

$$d\_real\_loss = \text{MSELoss}(D(\text{real}), \text{real\_label}) \tag{4.9}$$
$$d\_fake\_loss = \text{MSELoss}(D(G(z)), \text{fake\_label}) \tag{4.10}$$
$$d\_loss = d\_real\_loss + d\_fake\_loss \tag{4.11}$$
$$g\_loss = \text{MSELoss}(D(G(z)), \text{real\_label}) \tag{4.12}$$

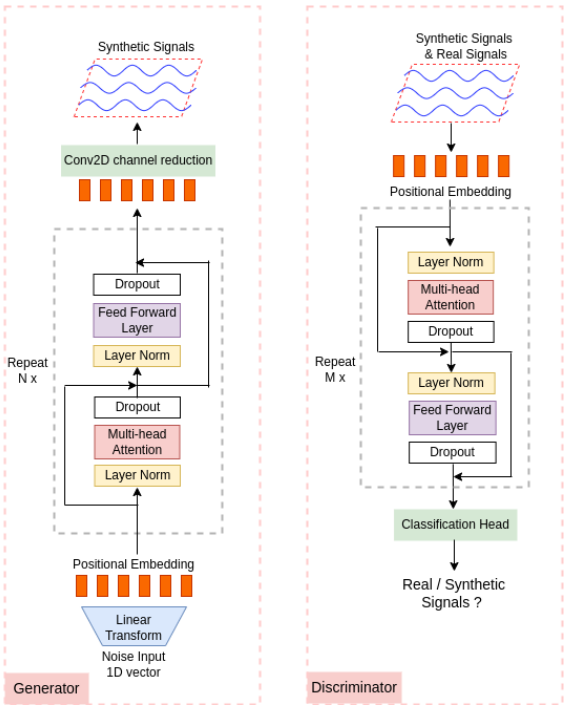


Figure 4.4: The architecture of TTS-GAN model from its original paper [21]. For further details on the architecture, refer to the source paper.

4.3.3 Training and Evaluation

For training and evaluation, all models were trained on the Signet dataset, featuring multivariate time series with a length of 1024 timesteps. To ensure a fair and consistent comparison across all models, we opted not to tune any model-specific hyperparameters, utilizing their default configurations as provided by their respective implementations. All models were trained using the Adam optimizer with a fixed learning rate of 10^{-3} for 100 epochs.

Following training, the models were evaluated based on their Discriminative Score (DS), Predictive Score (PS), and computational efficiency. The detailed results and analysis are presented in Chapter 5, Section 5.1.

4.4 Development of Conditional Model

4.4.1 Selection of Best-Performing Base Model

After evaluation of the candidate models, as detailed in Section 4.3, we identified TimeVAE as the best-performing base model for our specific tests. This finding aligns with recommendations from TSGBench [1], which also highlights TimeVAE's strong performance across various metrics. Consequently, TimeVAE was selected as the foundation for the development of our conditional generative model.

4.4.2 Conditional Model Architecture

We combine our foundation model with a transformer encoder to produce a conditional generative model, which in our case, resembles the architecture of Conditional Variational Auto Encoders (CVAEs). The foundation model learns how to generate the target time-series, while the transformer learns to encode the conditional information and provide it as a fixed-size vector to the foundation model. The high-level overview of the architecture is shown in Figure 4.5. For the transformer encoder, we primarily adapt the PatchTST architecture [27]. We've incorporated several known performance enhancements: we employ Rotary Positional Embeddings (RoPE) [32] instead of a learned positional embedding, and utilize Layer Normalization (LayerNorm) [2] rather than Batch Normalization, aligning with common practices in modern transformer designs. Figure 4.6 shows the overview of the transformer encoder architecture.

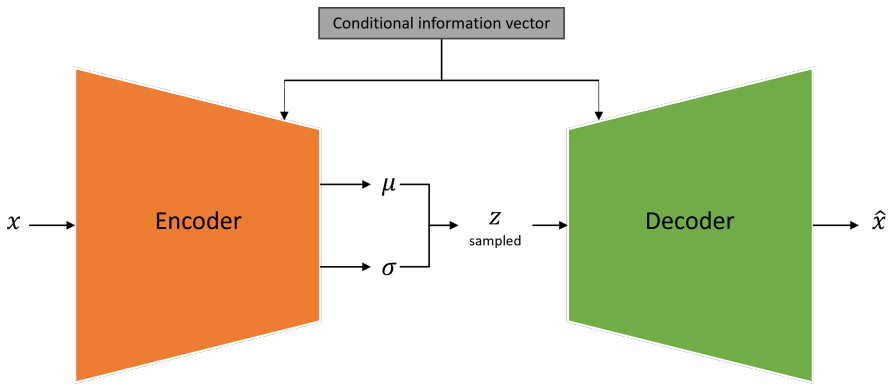


Figure 4.5: Architecture overview of the conditional model.

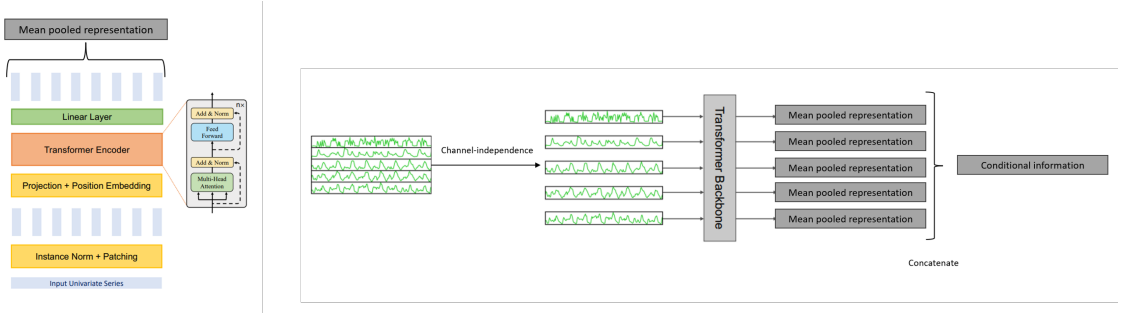


Figure 4.6: Overview of the Transformer Encoder Architecture. This figure is adapted from the original PatchTST architecture presented in Yu et al. (2023) [27]. The transformer encoder backbone is on the left and channel independence is on the right.

4.4.2.1 RevIN

Reversible Instance Normalization (RevIN) [16] is a technique employed in PatchTST [27]. It addresses the issue of distribution shifts in time series data, which can negatively impact model performance. By performing normalization and denormalization steps within the network in a reversible manner, RevIN allows the model to operate on a more stable data distribution while preserving the original data scale and meaning. This is particularly useful in our setup as it enhances the model's robustness to varying data characteristics and improves the generalizability of the learned representations across different time series instances.

4.4.2.2 Patch-Based Embedding

Inspired by their successful application in Vision Transformers (ViT) [7], patch-based embeddings were first introduced for time series by [27]. This approach involves dividing time series into fixed-size, non-overlapping segments, or "patches". Each patch is then transformed into a fixed-length vector through a Linear Patch-based Embedding layer. This method is crucial for several reasons: it enhances locality by grouping related timesteps, captures more semantic information that might be missed at the point-level by aggregating temporal data, and significantly reduces the computational complexity of the attention mechanism, particularly for long sequences [27].

4.4.2.3 Positional Encoding

We employ Rotary Positional Encoding (RoPE) [32] to incorporate positional information into the transformer. Unlike absolute positional embeddings, RoPE integrates positional information into the self-attention mechanism by rotating query and key vectors. This enables attention to explicitly capture relative position dependencies, which is particularly beneficial for long sequences as it avoids extrapolation issues seen with absolute encodings. Furthermore, RoPE offers benefits such as improved handling of variable sequence lengths and enhanced performance in tasks requiring a strong understanding of temporal order.

4.4.2.4 Channel Independence

Channel independence is a technique that treats each channel (i.e., each individual time series feature) of a multivariate time series separately. Instead of mixing information from all channels into a single input token for the Transformer (known as channel-mixing), each input token is designed to contain information from only one channel. This approach has proven effective in convolutional neural networks and linear models for time series, and PatchTST [27] was the first to successfully adapt it for Transformer-based models. This design choice is particularly beneficial for preserving the unique temporal dynamics within each channel and can prevent inter-channel noise from negatively impacting the learning process.

4.4.3 Objective Function

Since the model resembles a Conditional Variational Autoencoder (CVAE), we train it using a similar objective function to the foundation model. This objective function weights the reconstruction term, allowing for a stronger focus on reconstruction accuracy. The Conditional Model's objective function is:

$$\mathcal{L}_{\text{VAE}} = \beta \cdot MSE + \text{KL}\left(q(\mathbf{z}|\mathbf{x}) \,\|\, p(\mathbf{z})\right) \tag{4.13}$$

In this formulation, β represents a weighting coefficient for the reconstruction error. It is critical to understand that the reconstruction term, typically expressed as the negative expected log-likelihood $-\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})]$, reduces to the Mean Squared Error (MSE) loss under the common assumption of a Gaussian likelihood $p(\mathbf{x}|\mathbf{z})$ for continuous output data. β is a tunable hyperparameter.

4.4.4 Architectural Integration

Our conditional generative model integrates the transformer-encoded conditional information into the underlying Variational Autoencoder (VAE) framework by providing a fixed-size embedding of this information to both the VAE encoder and the decoder. We explore two primary methods for incorporating conditional information into the foundation model.

4.4.4.1 Concatenation

In this approach, the conditional information is first processed by a dedicated transformer encoder. The resulting representation from each channel is then mean-pooled and concatenated to the last layer of the main VAE encoder. This combined vector is subsequently transformed via a linear layer to parameterize the latent space of the VAE. For the decoder, the same conditional representation is concatenated with the vector sampled from the latent space before being passed to the TimeVAE decoder components, as shown in Figure 4.7.

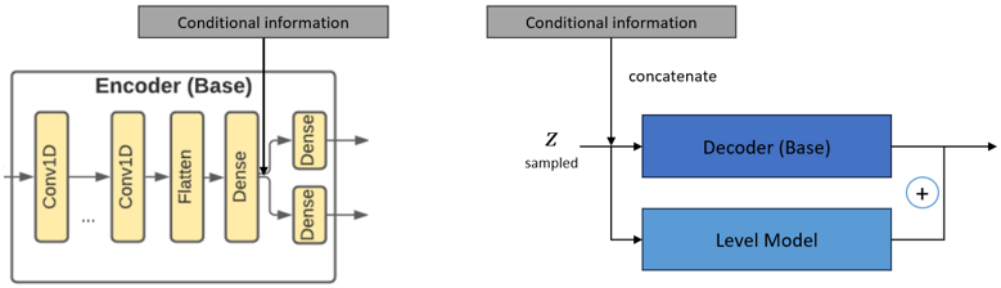


Figure 4.7: Concatenation approach for architectural integration. Part of the figure is from TimeVAE implementation [4]. The encoder part is on the left and the decoder part is on the right

4.4.4.2 Residual Integration

This method introduces residual connection in the decoder's reconstruction process. The conditional information is integrated into the VAE encoder in the same way as in the previous method. However, in the decoder, rather than passing the conditional information together with the vector sampled from the latent space, the conditional information is passed through a separate Multi-Layer Perceptron (MLP), that is designed to be small relative to the main decoder components. This MLP then transforms the conditional information vector to match the shape of the VAE's reconstructed sequence as shown in Figure 4.8. This allows for a more direct influence of the conditional information on the generated output, and also allows better gradient flow to the transformer encoder.

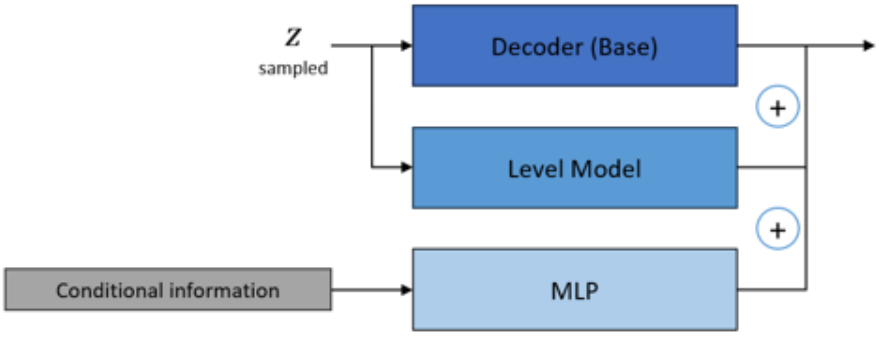


Figure 4.8: Residual integration approach for architectural integration. Only the decoder part is shown. Decoder (Base) refers to the TimeVAE decoder architecture shown in figure 4.1

4.4.5 Additional Improvements

We have experimented with two additional improvements to try and improve the performance of our conditional generative model further.

4.4.5.1 Free Bits

To further improve the utilization of the VAE's latent bottleneck and prevent posterior collapse (where the latent variable becomes uninformative, and the decoder learns to ignore it), we incorporate a free bits regularization term into the VAE's loss function. This modification ensures that the Kullback-Leibler (KL) divergence term in the VAE objective is only penalized if it exceeds a certain threshold (the "free bits" threshold), putting a constraint on the minimum amount of information per latent dimension [18]. By setting a minimum amount of information that the latent variable must encode, free bits encourage the model to learn more meaningful and diverse latent representations, thereby improving the quality and diversity of the conditional generations.

4.4.5.2 Conditional Prior

Beyond regularization of the latent space, we also investigated modifications to the latent prior itself to better incorporate conditional information. In a standard VAE, the latent space is typically constrained to follow a fixed prior distribution (e.g., a standard normal distribution). Here, taking inspiration from [19] and [14] we introduce a conditional prior, where the parameters (mean and variance) of the latent prior distribution are dynamically determined by the input conditional information. This is achieved by passing the transformer-encoded conditional embedding through a separate neural network (in our case, a small MLP) that outputs the mean (μ_z) and logarithmic variance ($\log \sigma_z^2$) for the prior of the latent variable z. The model is then trained jointly with the VAE objective through the modified KL divergence term.
The equation for the modified VAE objective is then:

$$\mathcal{L}_{\text{VAE}} = -\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})] + \text{KL}\left(q(\mathbf{z}|\mathbf{x}) \parallel p(\mathbf{z}|\mathbf{c})\right) \tag{4.14}$$

Where the reconstruction loss stays the same, while the Kullback-Leibler divergence is now between the approximate posterior $q(\mathbf{z}|\mathbf{x})$ and the conditional prior $p(\mathbf{z}|\mathbf{c})$ instead of the prior $p(\mathbf{z})$.
This allows the model to learn a more flexible and context-aware latent space, enabling the generation of outputs that are more accurately conditioned on the input.

4.5 Training and Evaluation of Conditional Model

4.5.1 Training protocol

We employed a two-phase training protocol for our conditional generative model.

4.5.1.1 Phase 1: Architectural Modification Evaluation

Initially, each architectural modification mentioned in Section 4.4.4 was trained using our small DSL dataset. For all architectural modifications in this phase, training was conducted using AdamW optimizer with a fixed learning rate of 10^{-3} and weight decay of 10^{-4} for 50 epochs.

4.5.1.2 Phase 2: Main Model Training and Improvement Exploration

Once the optimal architecture was identified from this initial training phase, we proceeded to train our main conditional generative model on the large DSL dataset. Subsequent experiments exploring additional performance improvements (i.e., Free Bits and Conditional Prior) were also conducted using this larger dataset. The training for this phase was performed using the same AdamW optimizer settings (fixed learning rate of 10^{-3} and weight decay of 10^{-4}), but for 100 epochs to allow for more extensive learning.

4.5.2 Hyperparameters

To ensure a fair and consistent comparison across all architectural modifications, a fixed set of hyperparameters was utilized throughout our experiments, with additional parameters introduced only as necessitated by specific modifications (detailed in Section 4.4.4). The selection of these hyperparameters was informed by initial exploratory trials conducted on a subset of the dataset. While a comprehensive, systematic hyperparameter search (e.g., via grid search or Bayesian optimization) was beyond the scope of this thesis due to computational and time constraints, these preliminary explorations allowed us to identify configurations that yielded stable training and reasonable performance for each model variant. The selected hyperparameters, as detailed below, represent a robust starting point for evaluation, and their consistent application ensures a fair comparison across all architectures. We acknowledge that more extensive hyperparameter optimization could potentially yield further performance improvements, a direction highlighted in our Future Work (Section 6.2).

For the experiments in both Phases 1 and 2, we set the latent dimension of VAE to 32, three convolutional layers of sizes 64, 128, 256 were used with reconstruction weight β of 10. In the transformer encoder, we used a patch length of 2 to encode conditional information. The number of heads for multi-head attention was set to 8, and we stacked four encoder blocks, while the output embedding vector size was fixed to 64. For all experiments, we used a batch size of 128.

4.5.3 Initialization and Regularization

4.5.3.1 Xavier Initialization

We employed Xavier initialization for the weights of every linear layer within our model. This technique, also known as Glorot initialization [9], initializes weights to maintain similar variance of activations across layers, preventing signals from exploding or vanishing. This approach generally contributes to more stable and efficient training in deep neural networks and usually the best practice for deep neural networks architecture in many cases.

4.5.3.2 Weight Decay

Weight decay, a form of L2 regularization, was applied during training through the AdamW optimizer [25]. Weight decay penalizes large weights in the model by adding a term to the loss function that is proportional to the square of the weights' magnitudes. This encourages the model to learn smaller, more generalized weights, effectively reducing overfitting to the training data and improving the model's ability to generalize to unseen sequences.

4.5.4 Evaluation

During the development of our model, when undergoing Phase 1 and 2, we evaluate each modification based on the reconstruction loss and KL loss on the validation sets (detailed results in chapter 5) as well as by visualising the quality of the reconstructed and the generated target time-series. The completion of this process resulted in finalizing the architecture of our conditional generative model.

To evaluate the performance of our conditional generative model, we primarily employed metrics focused on the physical plausibility of the generated sequences when compared to real sequences. While other metrics such as discriminative score or predictive score could provide broader insights into data fidelity, the requirement for industrial application put a strong emphasis on physical adherence. This specific focus on physical plausibility is directly aligned with the main objective set by Volvo for this task, emphasizing the functional utility and realistic behaviour of the generated data.

5
Results

In this section, we firstly present the evaluation of our candidate models for the selection of the foundation model. Later on, we evaluate Phase 1 for architectural integrations, and Phase 2 for additional model improvements. Finally, we present the results of our Final model on a holdout test set and discuss the physical plausibility of the generated signals.

5.1 Evaluation of Candidates for Foundation Model

To select a base model for our proposed architecture, we evaluated four candidate models, detailed in Section 4.3.1, using discriminative score (DS) and predictive score (PS).

For the DS, we trained a single-layer Long Short-Term Memory (LSTM) model as a discriminator on both real and synthetic data. The DS is reported as the deviation of the classification accuracy from 0.5. A lower DS indicates greater similarity between the synthetic and real data, thus signifying better performance. For the PS, we again used a single-layer LSTM as a predictor model. This model was trained on synthetic data to forecast the next timestep given all preceding timesteps and then tested on real data. We report the Mean Squared Error (MSE) loss from these predictions. To mitigate variations caused by random weight initializations, all experiments for both scores were run five times, and the averaged results are presented.
Table 5.1 shows that TimeVQVAE achieved the best DS, with TimeVAE closely following. However, for the downstream predictive task, TimeVAE excelled, yielding the best PS, with TimeVQVAE being the next closest option.

We chose TimeVAE as the base generative model for our proposed architecture due to its significantly faster training times and simpler architecture compared to TimeVQVAE. This choice allows us to benefit from faster training without a substantial compromise in performance. Given that our final proposed model will be an extension or adaptation of this base TimeVAE model and will likely be considerably larger, selecting a base model with efficient training was a critical factor in managing overall training times.

It is important to note that TimeGAN was excluded from this analysis because it failed to generate synthetic data with long time sequences (1024 timesteps) in our experiments. This limitation likely stems from its relatively simpler discriminator

and generator architectures (likely LSTMs), which are not designed to handle such long sequences effectively. While using more robust discriminator and generator models might have addressed this, our objective was to select from existing high-performing baselines rather than re-implementing them. Furthermore, the inclusion of TTSGAN covers the representation from GAN-based models, and its use of transformers for discriminator and generator models makes it more likely to be suitable for long sequences.

Table 5.1: Evaluation of Candidate models for choosing a base generative model using Discriminative Score (DS) (Lower the better), Predictive Score (PS) (Lower the better), Training Time per epoch and Total number of Model Parameters

Model	DS	PS	Training Time/epoch (sec)	Parameters
TimeVAE	0.4043	**0.0037**	24	10.2M
Time-VQVAE	**0.3780**	0.0045	228.8	2.65M
TTS-GAN	0.4999	0.0185	1080	1.14M

5.2 Evaluation of Phase 1

As shown in Table 5.2, the Residual Integration architecture greatly improved our model in Phase 1. It reduced the reconstruction loss by 80.4% compared to the naive concatenation-based integration of conditional information into the decoder. This substantial improvement was achieved with less than half the total number of model parameters.

The enhanced performance of the Residual Integration architecture likely stems from two key factors, detailed in Section 4.4.4.2: a more direct influence of conditional information on the decoder's output and an improved gradient flow to the transformer encoder. Consequently, we adopted the Residual Integration architecture for all subsequent stages of our conditional model's development.

Table 5.2: Results of Phase 1 experimentation. We evaluate two different architectures to incorporate conditional information into the foundation model using metrics like reconstruction loss (RL), KL divergence and their combined loss on validation data from Small DSL Dataset.

Architectural Integration	Parameters	RL	KL	Total Loss
Concatenation	27.67 M	88.04	15.24	1335.96
Residual Integration	13.05 M	17.26	16.14	188.77

5.3 Evaluation of Phase 2

In Phase 2, we took the best-performing model from Phase 1 and explore two further improvements: Free-bits and Conditional Prior, as detailed in Section 4.4.5.2.

As a first step, we trained the Phase 1 model on the Large DSL Dataset. We observed a significant issue of posterior collapse, where the KL loss dropped to 0.015. This collapse led to all generated signals being identical given specific conditional information, as illustrated in Figure 5.1. Although the reconstruction loss considerably improved compared to training on the smaller dataset (see Table 5.2 and Table 5.3 for comparison), this posterior collapse indicates that the conditional information became so context-rich that the decoder began to rely solely on it, disregarding the latent space to generate reconstructions. This behavior effectively minimizes the KL loss and causes the Variational Autoencoder (VAE) to behave like a deterministic autoencoder.

To address the problem of posterior collapse, we explored the utilization of Free-bits. This technique ensures that the KL loss does not drop below a certain threshold, thereby avoiding posterior collapse. As shown in Table 5.3, Free-bits successfully improved the KL loss and fulfilled its purpose. However, the visual quality of the generated signals remained unsatisfactory, as depicted in Figure 5.2.

The final improvement we explored was replacing the fixed prior with a Conditional Prior. This modification simultaneously improved both the reconstruction loss and the KL loss. With a Conditional Prior, the latent representation becomes more contextually rich with the conditional information, leading to better regularization of the latent space and a healthier KL divergence. The speed signals generated by this model appear much more promising and closely resemble real speed signals, as shown in Figure 5.3.

We adopted this Conditional Prior model as our final Conditional Model. Its performance was further evaluated on a holdout test set, with the losses reported in Table 5.4 and two generation examples provided in Figure 5.4. The final analysis of the physical plausibility of these generated signals is discussed in the next section.

Table 5.3: Results of Phase 2 experimentation. We evaluate the best performing model architecture from Phase 1 and further explore it with two improvements (free bits and conditional prior) based on the Reconstruction loss (RL), KL divergence and their combined loss on validation data of Large DSL Dataset

Model	Parameters	RL	KL	Total Loss
Conditional model (from Phase 1)	13.05 M	2.70	0.015	27.04
With Free bits	13.05 M	2.78	6.405	34.26
With Free bits & Conditional Prior	13.08 M	**1.50**	**8.62**	**23.70**

Table 5.4: Results of our Final Model (Conditional Model with Free-bits & Conditional Prior) on a holdout Test set

Model	Parameters	RL	KL	Total Loss
Best model (from Phase 2)	13.08 M	**1.60**	**8.89**	**24.96**

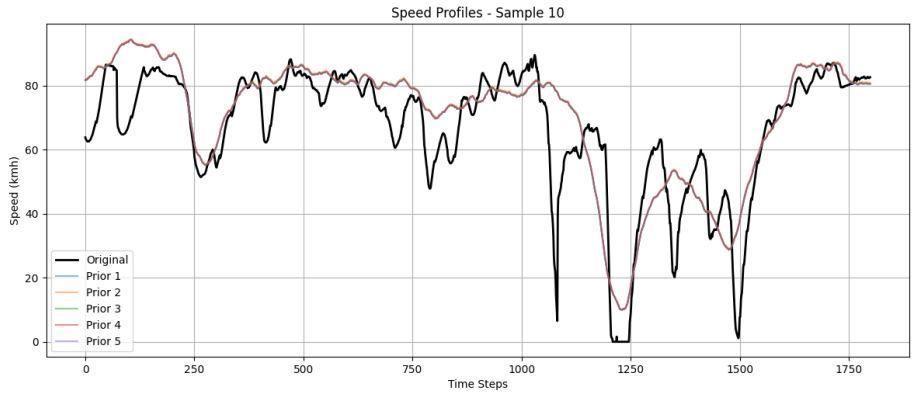


Figure 5.1: Generated samples by Conditional Model (from Phase 1)

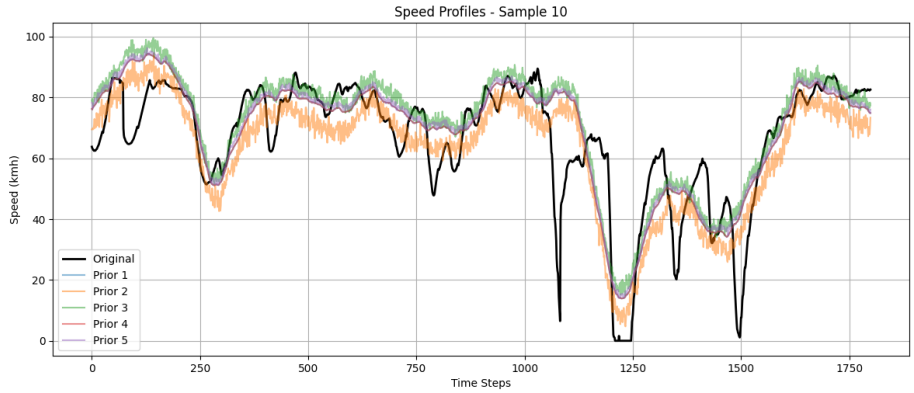


Figure 5.2: Generated samples by Conditional Model with Free-bits

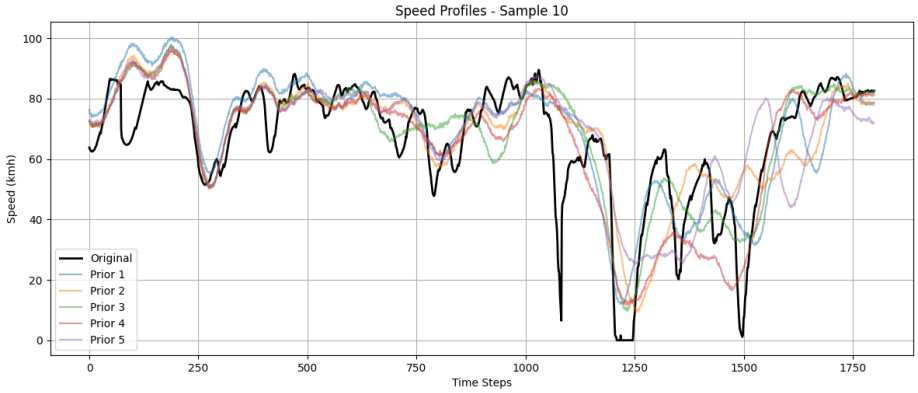


Figure 5.3: Generated samples by Conditional Model with Free-bits & Conditional Prior

5.4 Physical Plausibility of Generated Data

To evaluate the physical plausibility of our final conditional model, we compared the energy values computed from both real and generated speed signals. These energy values were derived using the formulations described in Section 4.2.3. The primary metric for comparison is the relative percentage difference in energy values between the generated and their corresponding real speed signals. We present the results using two evaluation approaches: Single-shot generation and Multiple-shot generation against a single conditional input.

For Single-shot generation, we assess the average energy deviation of generated signals from real ones across the entire test set. This is computed by generating a single synthetic speed signal for each conditional information in the test set and then averaging the relative differences across all samples.

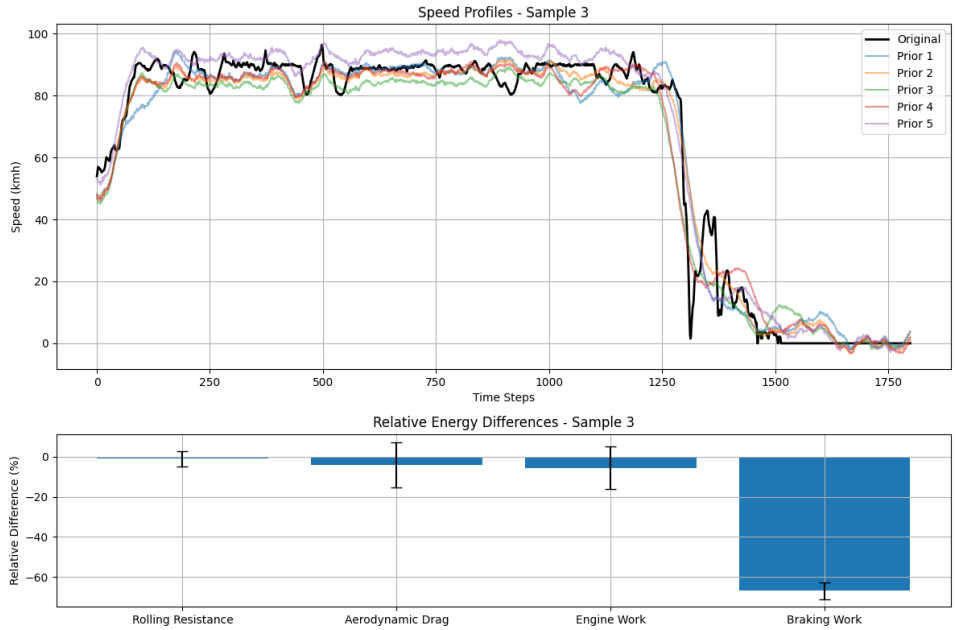
Table 5.5 presents the average relative difference in energies between synthetic and real speed signals. The results indicate that, despite the model not being explicitly trained with physics-based metrics, it implicitly learns some underlying physical characteristics of the real speed signals, particularly for the first three energy functions.

Specifically, we observe a notable negative difference for braking work. This suggests that the synthetic speed signals generally exhibit fewer braking events, appearing smoother compared to their real counterparts, a characteristic also evident in the generated signal plots in Figure 5.4. Conversely, positive differences for rolling resistance and work against aerodynamic drag indicate that synthetic speeds are, on average, higher than real ones. This aligns with their direct proportionality to velocity and the cube of velocity, respectively. Finally, the engine work for synthetic signals is generally slightly less than that for real ones, implying reduced fuel consumption. This can also be attributed to the less frequent fluctuations observed in the synthetic speed signals.

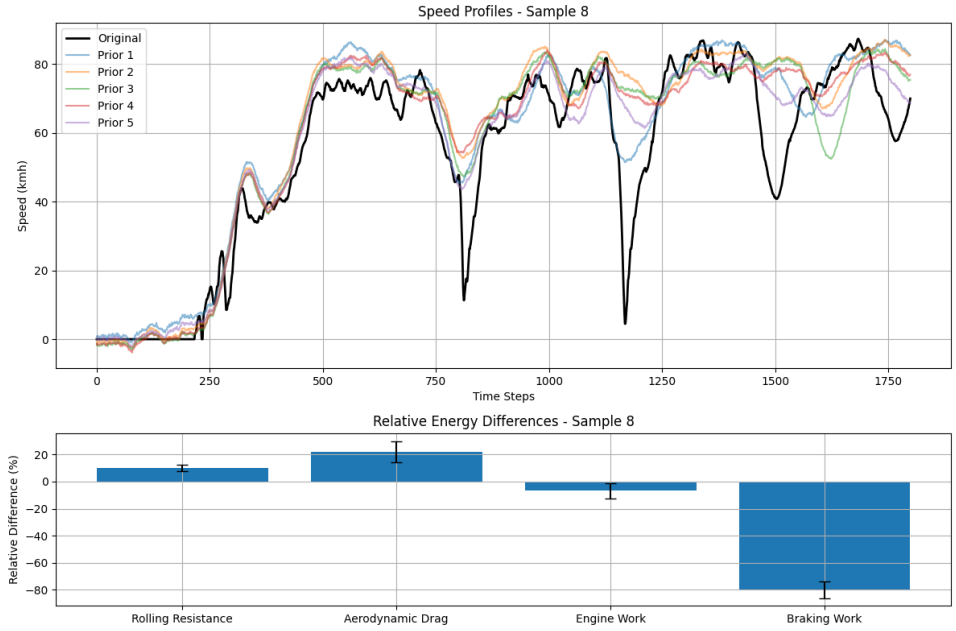
The Multiple-shot generation evaluation aims to visualize the variation in the relative energy differences between synthetic and real signals across multiple generations for a given conditional input. Figure 5.4 illustrates this with two examples from the test set, each corresponding to unique conditional information. The bar plots within this figure represent the average relative difference of multiple synthetic signals compared to their real counterpart for a single sample. The bars themselves indicate the average value, while the whiskers denote the variation observed across multiple generations. For example, in Figure 5.4a, the first three energy types show values very close to zero. The associated variations suggest that while some generated signals can be highly accurate in terms of energy, others may exhibit greater deviations.

Table 5.5: Average Relative Difference in Energy values of the Generated (Single Shot) & Original Speed signals by our Final Conditional Model. Negative sign indicates that the Generated speed signals exhibit lower energy values compared to their original counter-part. The values are averaged across all samples from the Test set.

Model	$W_{\text{rolling\_resist}}(\%)$	$W_{\text{aero}}(\%)$	$W_{\text{eng}}(\%)$	$W_{\text{brake}}(\%)$
Final Conditional Model	3.42	15.74	-9.71	-67.72



(a)



(b)

Figure 5.4: Two examples of the multiple-shot generation case. Each example corresponds to a unique conditional information from the test set to generate five synthetic speed signals (in color) against the real speed signal (in black) in the top subplot. The bottom bar subplot shows the average relative difference between the energies of multiple synthetic speed signals from its real counter-part. Black whiskers on the bars indicate the deviation from the mean across multiple synthetic speed signals. Negative value indicates energies of synthetic data are lower than that of real data.

6
Conclusion

This chapter aims to consolidate the insights gained throughout this work. We begin by revisiting our research questions, discussing how our findings address them, and concluding with recommendations for future research.

6.1 Discussion

To guide this discussion, we revisit the research questions formulated in Section 1.3 and interpret our findings concerning each question.

Can current state-of-the-art time-series generative models effectively generate long sequences (e.g., beyond 1000 timesteps)?

From the evaluation of Discriminative Score (DS) and Predictive Score (PS), none of the models achieved good performance when tasked with generating long sequences. This strongly suggests that current state-of-the-art models, primarily developed and evaluated on sequences shorter than 500 timesteps, considerably struggle with the increased complexity of longer sequences. Consequently, a specific architecture developed for long sequences is needed in order to achieve better quality in the generated sequence.

Among effective models, which one demonstrates superior performance in generating long sequences, considering both computational efficiency (training time) and the quality of the generated time-series data?

While all models showed limitations in long sequence generation, TimeVAE was selected for further development due to its overall superior performance profile. Although its Discriminative Score (DS) was marginally higher compared to TimeVQVAE (0.400 vs 0.3780, respectively, where lower is better), TimeVAE demonstrated better Predictive Score (PS) (0.0037 vs 0.0045, where lower is better) and offered comparatively efficient training time. This strong performance in predictive utility, coupled with its computational efficiency, made it the optimal choice for further development.

What are the effects of incorporating continuous conditional information into the selected best-performing model for long-sequence generation?

Our findings reveal that naively integrating continuous conditional information without architectural modification is ineffective. When a zero vector of conditional information was supplied, the model's output remained largely unchanged, indicating a failure to utilize the conditional input. We hypothesize this is due to the complex gradient flow required for the conditional information to pass through the VAE bottleneck and multiple deep layers of the decoder, hindering effective learning by the transformer encoder.

Conversely, the implementation of our residual integration architecture yielded major improvements in the model's ability to utilize conditional information. The model now learns to generate sequences based on conditional information, further supporting our hypothesis from section 4.4.3 that using residual integration would promote more direct influence on the generated output and better gradient flow to the transformer encoder. However, the model showed signs of posterior collapse, which is inherent in Conditional Variational Autoencoders (CVAEs), where the strong conditional information leads to the model completely ignoring the information from the latent space.

Additional improvements, such as "free bits" and "conditional prior", were beneficial in mitigating posterior collapse and improving model performance. Our results demonstrate that with their combined application, the model achieved a better reconstruction loss, healthier KL divergence, and improved generated sequence quality based on our evaluation.

Can a purely data-driven generative model, without explicit physics-based training, learn to produce physically plausible long-sequence time-series data?

Yes, to a notable extent. It can learn to generate long-sequence time-series data that is physically plausible, even without explicit physics-based training.

From our results on physical plausibility, three of the physical metrics showed less than 16% deviation from real signals. This suggests that the model implicitly learned the underlying physical patterns such as inertia, drag, and energy conservation. However, the model still struggled in certain areas, particularly with generating sharp braking events and precise constant speed intervals. We hypothesize that this limitation arises because the synthetic speed signals generally exhibit fewer braking events and appear smoother compared to their real counterparts, a phenomenon detailed in Section 5.4.

6.2 Future Work

Building on the present work, several areas for future research can further advance the model's capabilities and expand its application.

A key next step is to better integrate physical constraints or prior knowledge into the generative process. This could involve using specific architectural designs or adding regularization to the loss functions, guiding the model to produce more physically realistic outputs as well as creating a more suitable evaluation metrics to evaluate the physical plausibility.

Another direction involves investigating Diffusion Probabilistic Models (DPMs). While complex, these models have shown strong performance in other generative tasks. Studying their use for conditional time-series generation could lead to significant improvements in data quality and variety.

To enhance the model's performance and stability, more extensive hyperparameter optimization is needed. Since a thorough search wasn't done, future work should include a process like Bayesian optimization [30] or grid search to fine-tune the model's architecture, training settings (such as learning rates), and regularization techniques (like "free bits" and weight decay). This detailed tuning could boost performance and reliability.

Furthermore, benchmarking against a wider range of current models would provide a clearer picture of the proposed model's standing. As time-series generative models evolve quickly, evaluating against newer architectures or models specifically designed for very long sequences could reveal additional strengths or areas for improvement.

For practical use, it is also important to make the model more robust to data issues like anomalies and distribution shifts. Real-world time-series data often has noise or changes over time. Future research should focus on ensuring the conditional generative model reliably produces data even with imperfect or unexpected conditional inputs. This might involve robust training methods or pre-processing steps to detect anomalies.

Finally, while residual integration helped with conditioning, exploring other ways to integrate conditional information could lead to further improvements. Additionally, investigating long-term temporal dependencies and combining the model with forecasting capabilities would add more practical value, allowing the generated data to directly improve future predictions.

Bibliography

[1] Yihao Ang, Qiang Huang, Yifan Bao, Anthony KH Tung, and Zhiyong Huang. Tsgbench: Time series generation benchmark. *Proc. VLDB Endow.*, 17(3):305–318, 2023.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[3] Rewon Child. Very deep {vae}s generalize autoregressive models and can outperform them on images. In *International Conference on Learning Representations*, 2021.

[4] Abhyuday Desai, Cynthia Freeman, Zuhui Wang, and Ian Beaver. TimeVAE: A variational auto-encoder for multivariate time series generation.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[6] Chris Donahue, Julian McAuley, and Miller Puckette. Adversarial audio synthesis. In *International Conference on Learning Representations*, 2019.

[7] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[8] Cristóbal Esteban, Stephanie L. Hyland, and Gunnar Rätsch. Real-valued (Medical) Time Series Generation with Recurrent Conditional GANs, December 2017. arXiv:1706.02633 [stat].

[9] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Commun. ACM*, 63(11):139–144, October 2020.

[11] Daniel Jarrett, Ioana Bica, and Mihaela van der Schaar. Time-series generation by contrastive imitation. In *Proceedings of the 35th International Conference*

on Neural Information Processing Systems, NIPS '21, Red Hook, NY, USA, 2021. Curran Associates Inc.

[12] Paul Jeha, Michael Bohlke-Schneider, Pedro Mercado, Shubham Kapoor, Rajbir Singh Nirwan, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. PSA-GAN: Progressive self attention GANs for synthetic time series. In *International Conference on Learning Representations*, 2022.

[13] Jinsung Jeon, JEONGHAK KIM, Haryong Song, Seunghyeon Cho, and Noseong Park. GT-GAN: General purpose time series synthesis with generative adversarial networks. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[14] Penny Karanasou, Sri Karlapati, Alexis Moinet, Arnaud Joly, Ammar Abbas, Simon Slangen, Jaime Lorenzo Trueba, and Thomas Drugman. A learned conditional prior for the vae acoustic space of a tts system. *arXiv preprint arXiv:2106.10229*, 2021.

[15] Tero Karras, Samuli Laine, and Timo Aila. A Style-Based Generator Architecture for Generative Adversarial Networks . *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 43(12):4217–4228, December 2021.

[16] Taesung Kim, Jinhee Kim, Yunwon Tae, Cheonbok Park, Jang-Ho Choi, and Jaegul Choo. Reversible instance normalization for accurate time-series forecasting against distribution shift. In *International Conference on Learning Representations*, 2021.

[17] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[18] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems*, 29, 2016.

[19] Frantzeska Lavda, Magda Gregorová, and Alexandros Kalousis. Improving vae generations of multimodal data through data-dependent conditional priors. *arXiv preprint arXiv:1911.10885*, 2019.

[20] Daesoo Lee, Sara Malacarne, and Erlend Aune. Vector quantized time series generation with a bidirectional prior model. In Francisco Ruiz, Jennifer Dy, and Jan-Willem van de Meent, editors, *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, volume 206 of *Proceedings of Machine Learning Research*, pages 7665–7693. PMLR, 25–27 Apr 2023.

[21] Xiaomin Li, Vangelis Metsis, Huangyingrui Wang, and Anne Hee Hiong Ngu. Tts-gan: A transformer-based time-series generative adversarial network. In *Artificial Intelligence in Medicine: 20th International Conference on Artificial Intelligence in Medicine, AIME 2022, Halifax, NS, Canada, June 14–17, 2022, Proceedings*, page 133–143, Berlin, Heidelberg, 2022. Springer-Verlag.

[22] Xiaomin Li, Anne Hee Hiong Ngu, and Vangelis Metsis. TTS-CGAN: A Transformer Time-Series Conditional GAN for Biosignal Data Augmentation, June 2022. arXiv:2206.13676 [cs].

[23] Shujian Liao, Hao Ni, Marc Sabate-Vidales, Lukasz Szpruch, Magnus Wiese, and Baoren Xiao. Sig-wasserstein gans for conditional time series generation. *Mathematical Finance*, 34(2):622–670, 2024.

[24] Yong Liu, Guo Qin, Xiangdong Huang, Jianmin Wang, and Mingsheng Long. Timer-XL: Long-context transformers for unified time series forecasting. In *The Thirteenth International Conference on Learning Representations*, 2025.

[25] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2017.

[26] Ilan Naiman, N. Benjamin Erichson, Pu Ren, Michael W. Mahoney, and Omri Azencot. Generative modeling of regular and irregular time series data via koopman VAEs. In *The Twelfth International Conference on Learning Representations*, 2024.

[27] Yuqi Nie, Nam H. Nguyen, Phanwadee Sinthong, and Jayant Kalagnanam. A time series is worth 64 words: Long-term forecasting with transformers. In *International Conference on Learning Representations*, 2023.

[28] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018. Publisher: San Francisco, CA, USA.

[29] Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating Diverse High-Fidelity Images with VQ-VAE-2. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[30] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

[31] Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 3745–3753, Red Hook, NY, USA, 2016. Curran Associates Inc.

[32] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomput.*, 568(C), February 2024.

[33] Vajira Thambawita, Jonas L. Isaksen, Steven A. Hicks, Jonas Ghouse, Gustav Ahlberg, Allan Linneberg, Niels Grarup, Christina Ellervik, Morten Salling Olesen, Torben Hansen, Claus Graff, Niels-Henrik Holstein-Rathlou, Inga Strümke, Hugo L. Hammer, Mary M. Maleckar, Pål Halvorsen, Michael A. Riegler, and Jørgen K. Kanters. Deepfake electrocardiograms using generative adversarial networks are the beginning of the end for privacy issues in medicine. *Scientific Reports*, 11(1):21896, 2021.

[34] Quoc-Tuan Truong, Aghiles Salah, and Hady W. Lauw. Bilateral Variational Autoencoder for Collaborative Filtering. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, WSDM '21, pages 292–300, New York, NY, USA, March 2021. Association for Computing Machinery.

[35] Arash Vahdat and Jan Kautz. Nvae: a deep hierarchical variational autoencoder. In *Proceedings of the 34th International Conference on Neural Informa-*

tion Processing Systems, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[36] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6309–6318, Red Hook, NY, USA, 2017. Curran Associates Inc.

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.

[38] Jinsung Yoon, Lydia N. Drumright, and Mihaela van der Schaar. Anonymization Through Data Synthesis Using Generative Adversarial Networks (ADS-GAN). *IEEE journal of biomedical and health informatics*, 24(8):2378–2388, August 2020.

[39] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. Time-series generative adversarial networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[40] Yitian Zhang, Liheng Ma, Soumyasundar Pal, Yingxue Zhang, and Mark Coates. Multi-resolution time-series transformer for long-term forecasting. In *International conference on artificial intelligence and statistics*, pages 4222–4230. PMLR, 2024.

DEPARTMENT OF COMPUTER SCIENCE AND TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se