UNIVERSAL TIME-SERIES GENERATION USING SCORE-BASED GENERATIVE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Score-based generative models (SGMs) have demonstrated unparalleled sampling quality and diversity in numerous fields, such as image generation, voice synthesis, and tabular data synthesis, etc. Inspired by those outstanding results, we apply SGMs to synthesize time-series by learning its conditional score function. To this end, we present a conditional score network for time-series synthesis, deriving a denoising score matching loss tailored for our purposes. In particular, our presented denoising score matching loss is the conditional denoising score matching loss for time-series synthesis. In addition, our framework is such flexible that both regular and irregular time-series can be synthesized with minimal changes to our model design. Finally, we obtain exceptional synthesis performance on various time-series datasets, achieving state-of-the-art sampling diversity and quality.

1 INTRODUCTION

Time-series frequently occurs in our daily lives, e.g., stock data, climate data, and so on. Especially, time-series forecasting and classification are popular research topics in the field of machine learning (Ahmed et al., 2010; Fu, 2011; Ismail Fawaz et al., 2019). In many cases, however, time-series samples are incomplete and/or the number of samples is insufficient, in which case training machine learning models cannot be fulfilled in a robust way. To overcome the limitation, time-series synthesis has been studied actively recently (Chen et al., 2018; Dash et al., 2020). These synthesis models have been designed in various ways, including variational autoencoders (VAEs) and generative adversarial networks (GANs) (Desai et al., 2021; Yoon et al., 2019; Jeon et al., 2022).

Moreover, real-world time series often inevitably contain missing values because of privacy reasons or medical events. It has been noted that the missing values involve important information, called *informative missingness* (Rubin, 1976). To remedy the problem, several works have been developed to deal with irregular time-series, i.e., the inter-arrival time between observations is not fixed and/or some observations can be missing (Schafer and Graham, 2002; Che et al., 2016; Kidger et al., 2020), in which case synthesizing irregular time series is challenging (Jeon et al., 2022).

Score-based generative models (SGMs) have shown good sampling quality and diversity in numerous fields, such as image generation, voice synthesis, and tabular data synthesis, etc (Yang et al., 2023). However, in time-series generation, SGMs should consider *autoregressiveness*, meaning each time-series observation is generated in consideration of its previously generated observations which makes time-series generation more difficult (Yoon et al., 2019; Jeon et al., 2022). To this end, we propose the method of **T**ime-series generation using conditional **S**core-based **G**enerative **M**odel (TSGM), which consists of three neural networks, i.e., an encoder, a score network, and a decoder (see Figure 2).

Score-based time-series synthesis SGMs have its potential to deal with autoregressiveness (Meng et al., 2020). We design our own autoregressive denoising score matching loss for time-series generation and prove its correctness (see Section 2.1.3). Along with its performances from image generation domain (cf. Fig. 1), here is our main motivation of using SGMs:

The mathematical structure of SGMs naturally aligns with the sequential nature of time-series data.

Besides, we design a conditional score network on time-series, which learns the gradient of the conditional log-likelihood.

Table 1: The table illustrates how many medals each method gets across all datasets and evaluation metrics, based on the generation evaluation scores presented in Table 2 and Table 12. Our method with the two specific types, TSGM-VP and TSGM-subVP, achieves superior generation performance compared to baselines.

Method	Olympic Rankings					
	Gold		Silver		Bronze	
	Regular	Irregular	R	I	R	I
TSGM-VP	4	11	4	11	0	1
TSGM-subVP	6	16	1	7	1	0
TimeGAN	1	0	0	0	1	0
TimeVAE	0	0	0	0	1	4
GT-GAN	0	1	1	1	2	16

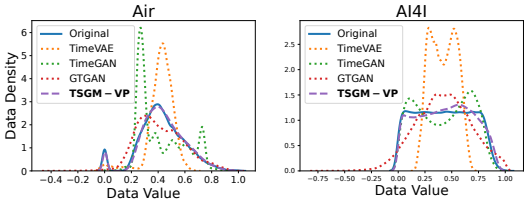

Figure 1: The KDE plots show the estimated distributions of original data and ones generated by several methods in the Air and AI4I datasets — we ignore time stamps for drawing these distributions. Unlike baseline methods, the distribution of TSGM-VP is almost identical to the original one. These figures provide an evidence of the excellent generation quality and diversity of our method. For TSGM-subVP, similar results are observed. Refer to Appendix L for additional visualizations

Regular vs. irregular time-series synthesis Time-series are prevalent throughout our daily lives. Although regular time-series are easy to be considered, it is common to have missing values because of privacy issue (Che et al., 2016; Kidger et al., 2020), which makes irregularly sampled time-series[1]. This hinders time-series processing and we consider the hardest problem condition: *providing complete time-series given not only regular time-series, but also irregular ones*. To this end, our method is considered 'universial' in that both both regular and irregular time-series samples can be treated with minimal changes to our model design. For synthesizing regular time series, we use a recurrent neural network-based encoder and decoder. Continuous-time methods, such as neural controlled differential equations (Kidger et al., 2020) and GRU-ODE (Brouwer et al., 2019), can be used as our encoder and decoder for synthesizing irregular time series (see Section 3.2 and Appendix I, J).

We conduct in-depth experiments with 4 real-world datasets under regular and irregular settings. To be specific, for the irregular settings, we randomly drop 30%, 50%, and 70% of observations from regular time-series on training, then generate complete time-series on evaluation. Therefore, we test with 16 different settings, i.e., 4 datasets for one regular and three irregular settings. Our specific choices of 9 baselines include almost all existing types of time-series generative paradigms, ranging from VAEs to GANs. In Table 1 and Figure 1, we compare our method to the baselines, ranking methods by their evaluation scores and estimating data distribution by kernel density estimation (KDE). We also visualize real and generated time-series samples onto a latent space using t-SNE (van der Maaten and Hinton, 2008) in Figure 3. Our proposed method shows the best generation quality in almost all cases. Furthermore, the t-SNE and KDE visualization results provide intuitive evidence that our method's generation diversity is also superior to that of the baselines. Our contributions are summarized as follows:

1. We, for the first time, propose an SGM-based universal time-series synthesis method.

2. We derive our own denoising score matching loss considering the autoregressive nature of sequential data, connecting SGMs to time-series generation domain.

3. We conduct comprehensive experiments with 4 real-world datasets and 9 baselines under one regular and three irregular settings since our method supports both regular and irregular time-series. Overall, our proposed method shows the best generation quality and diversity.

[1]For example, Physionet (Goldberger et al., 2000 (June 13), a famous dataset for time series classification, deliberately removed 90% of observations to protect the privacy of patients, posing challenges for learning and analysis. The synthesized time series can be used instead.

2 Related Work and Preliminaries

2.1 Score-based Generative Models

SGMs offer several advantages over other generative models, including their higher generation quality and diversity. SGMs follow a two-step process, wherein i) gaussian noises are continuously added to a sample and ii) then removed to recover a new sample. These processes are known as the forward and reverse processes, respectively. In this section, we provide a brief overview of the original SGMs in (Song et al., 2021), which will be adapted for the time-series generation tasks.

2.1.1 Forward and Reverse Process

At first, SGMs add noises with the following stochastic differential equation (SDE):

$$d\mathbf{x}^s = \mathbf{f}(s, \mathbf{x}^s)ds + g(s)d\mathbf{w}, s \in [0, 1], \tag{1}$$

where $\mathbf{w} \in \mathbb{R}^{\dim(\mathbf{x})}$ is a multi-dimensional Brownian motion, $\mathbf{f}(s, \cdot) : \mathbb{R}^{\dim(\mathbf{x})} \to \mathbb{R}^{\dim(\mathbf{x})}$ is a vector-valued drift term, and $g : [0, 1] \to \mathbb{R}$ is a scalar-valued diffusion function. Hereafter, we define $\mathbf{x}^s$ as a noisy sample diffused at time $s \in [0, 1]$ from an original sample $\mathbf{x} \in \mathbb{R}^{\dim(\mathbf{x})}$. Therefore, $\mathbf{x}^s$ can be understood as a stochastic process following the SDE.

There are several options for $\mathbf{f}$ and g: variance exploding(VE), variance preserving(VP), and subVP. Song et al. (2021) proved that VE and VP are continuous generalizations of the two discrete diffusion methods: one in Song and Ermon (2019) and the other in Sohl-Dickstein et al. (2015); Ho et al. (2020). The subVP method shows, in general, better negative log-likelihood (NLL) according to Song et al. (2021). We describe the exact form of each SDE in Table 14 with detailed explanation in Appendix N. Note that we only use the subVP-based TSGM in our main experiments and exclude the VE and VP-based one for its inferiority for time series synthesis in our experiments, but checked the VP-based method from ablation study for its better performances than that of the VE.

SGMs run the forward SDE with a sufficiently large number of steps to make sure that the diffused sample converges to a Gaussian distribution at the final step. The score network $M_\theta(s, \mathbf{x}^s)$ learns the gradient of the log-likelihood $\nabla_{\mathbf{x}^s} \log p(\mathbf{x}^s)$, which will be used in the reverse process.

For the forward SDE, there exists the following corresponding reverse SDE (Anderson, 1982):

$$d\mathbf{x}^s = [\mathbf{f}(s, \mathbf{x}^s) - g^2(s)\nabla_{\mathbf{x}^s} \log p(\mathbf{x}^s)]ds + g(s)d\bar{\mathbf{w}}. \tag{2}$$

The formula suggests that if knowing the score function, $\nabla_{\mathbf{x}^s} \log p(\mathbf{x}^s)$, we can recover real samples from the prior distribution $p_1(\mathbf{x}) \sim \mathcal{N}(\mu, \sigma^2)$, where μ, σ vary depending on the forward SDE type.

2.1.2 Training and Sampling

In order for the model M to learn the score function, the model has to optimize the following loss function:

$$L(\theta) = \mathbb{E}_s\{\lambda(s)\mathbb{E}_{\mathbf{x}^s}[\|M_\theta(s, \mathbf{x}^s) - \nabla_{\mathbf{x}^s} \log p(\mathbf{x}^s)\|_2^2]\}, \tag{3}$$

where s is uniformly sampled over $[0, 1]$ with an appropriate weight function $\lambda(s) : [0, 1] \to \mathbb{R}$. However, using the above formula is computationally prohibitive (Hyvärinen, 2005; Song et al., 2019). Thanks to Vincent (2011), the loss can be substituted with the following denoising score matching loss:

$$L^*(\theta) = \mathbb{E}_s\{\lambda(s)\mathbb{E}_{\mathbf{x}^0}\mathbb{E}_{\mathbf{x}^s|\mathbf{x}^0}[\left\|M_\theta(s, \mathbf{x}^s) - \nabla_{\mathbf{x}^s} \log p(\mathbf{x}^s|\mathbf{x}^0)\right\|_2^2]\}. \tag{4}$$

Since SGMs use an affine drift term, the transition kernel $\mathrm{p}(\mathbf{x}^s|\mathbf{x}^0)$ follows a certain Gaussian distribution (Särkkä and Solin, 2019) and therefore, $\nabla_{\mathbf{x}^s} \log p(\mathbf{x}^s|\mathbf{x}^0)$ can be analytically calculated.

2.1.3 Autoregressiveness of Diffusion Model

Let $\mathbf{x}_{1:N}$ be a time-series sample which consists of N observations. In order to synthesize time-series $\mathbf{x}_{1:N}$, unlike other generation tasks, we must consider autoregressiveness: generating each observation $\mathbf{x}_n$ at sequential order $n \in \{2, ..., N\}$ considering its previous history $\mathbf{x}_{1:n-1}$. One

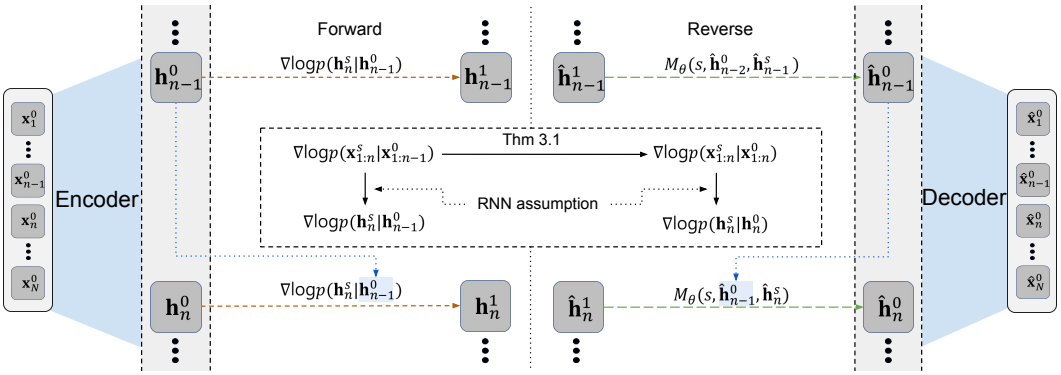

Figure 2: The overall workflow of TSGM (see Section 3.3). Our original learning objective is to approximate $\nabla \log p(\mathbf{x}^s_{1:n}|\mathbf{x}^0_{1:n-1})$, which is computationally prohibitive, with the conditional score network $M_\theta(s, \mathbf{x}^s_{1:n}, \mathbf{x}^0_{1:n-1})$ using an MSE loss. We then prove in Thm. 3.1 that learning $\nabla \log p(\mathbf{x}^s_{1:n}|\mathbf{x}^0_{1:n})$ is equivalent to $\nabla \log p(\mathbf{x}^s_{1:n}|\mathbf{x}^0_{1:n-1})$ for θ of M_θ in the MSE loss, i.e., their optimal model parameter θ is identical. At the end, our score network $M_\theta(s, \mathbf{h}^s_n, \mathbf{h}^0_{n-1})$ learns $\nabla \log p(\mathbf{h}^s_n|\mathbf{h}_n)$ since RNNs can encode $\mathbf{x}^0_{1:n}$ and $\mathbf{x}^0_{1:n-1}$ into their hidden states $\mathbf{h}^0_n$ and $\mathbf{h}^0_{n-1}$, respectively.

can train neural networks to learn the conditional likelihood $\mathrm{p}(\mathbf{x}_n|\mathbf{x}_{1:n-1})$ and generate each $\mathbf{x}_n$ recursively using it, depicting so-called autoregressive property of time-series domain.

Up to our survey, there's no paper about diffusion models considering autoregressiveness in time-series generation. Instead of it, Meng et al. (2020) dealt with a more generalized case, autoregressive conditional score matching to make its score model, $M_\theta(s, \cdot, \cdot)$, to be $M_\theta(s, \mathbf{x}^s_n, \mathbf{x}^0_{1:n-1}) \sim \nabla_{\mathbf{x}^s_n} \log p(\mathbf{x}^s_n|\mathbf{x}^0_{1:n-1})$. Inspired by the previous work, we enlighten a connection between SGMs and time-series generation by deriving an autoregressive denoising score matching (see Theorem 3.1). We emphasize that TSGM is fundamentally different in that (i) TSGM uses its own autoregressive denoising score matching loss, (ii) as a result, it optimizes not $\nabla_{\mathbf{x}^s_n} \log p(\mathbf{x}^s_n|\mathbf{x}^0_{1:n-1})$ but $\nabla_{\mathbf{x}^s_{1:n}} \log p(\mathbf{x}^s_{1:n}|\mathbf{x}^0_{1:n-1})$, which is fit for its RNN-based (markovian) encoder and decoder.

2.2 Time-series Generation

There are several time-series generation papers, and we introduce their ideas. TimeVAE (Desai et al., 2021) is a variational autoencoder to synthesize time-series data. This model can provide interpretable results by reflecting temporal structures such as trend and seasonality in the generation process. CTFP (Deng et al., 2020) is a well-known normalizing flow model. It can treat both regular and irreugular time-series data by a deformation of the standard Wiener process.

TimeGAN (Yoon et al., 2019) uses a GAN architecture to generate time-series. First, it trains an encoder and decoder, which transform a time-series sample $\mathbf{x}_{1:N}$ into latent vectors $\mathbf{h}_{1:N}$ and recover them by using a recurrent neural network (RNN). Next, it trains a generator and discriminator pair on latent space, by minimizing the discrepancy between an estimated and true distribution, i.e. $\hat{p}(\mathbf{x}_n|\mathbf{x}_{1:n-1})$ and $p(\mathbf{x}_n|\mathbf{x}_{1:n-1})$. Since it uses an RNN-based encoder, it can efficiently learn the conditional likelihood $p(\mathbf{x}_n|\mathbf{x}_{1:n-1})$ by treating it as $p(\mathbf{h}_n|\mathbf{h}_{n-1})$, since $\mathbf{h}_n \sim \mathbf{x}_{1:n}$ under the regime of RNNs. Therefore, it can generate each observation $\mathbf{x}_n$ considering its previous history $\mathbf{x}_{1:n-1}$. However, GAN-based generative models are vulnerable to the issue of mode collapse (Xiao et al., 2022) and unstable behavior problems during training (Chu et al., 2020). GT-GAN (Jeon et al., 2022) attempted to solve the problems by incorporating an invertible neural network-based generator into its framework. There also exist GAN-based methods to generate other types of sequential data, e.g., video, sound, etc (Esteban et al., 2017; Mogren, 2016; Xu et al., 2020; Donahue et al., 2019). In our experiments, we also use them as our baselines for thorough evaluations.

3 Proposed Method

Our proposed TSGM consists of three networks: an encoder, a decoder, and a conditional score network (cf. Fig. 2). Firstly, we train the encoder and the decoder to connect between time-series

samples and a latent space. Next, using the pre-trained encoder and decoder, we train the conditional score network on the latent space. The conditional score network will be used for sampling fake time-series on the latent space.

3.1 Problem Formulation

Let $\mathcal{X}$ and $\mathcal{H}$ denote a data space and a latent space, respectively. We define $\mathbf{x}_{1:N}$ as a time-series sample with a sequential length of N, and $\mathbf{x}_n$ is a multi-dimensional observation of $\mathbf{x}_{1:N}$ at sequential order n. Similarly, $\mathbf{h}_{1:N}$ (resp. $\mathbf{h}_n$) denotes an embedded time series (resp. an embedded observation).

Each observation $\mathbf{x}_n$ can be represented as a pair of time and features, i.e., $\mathbf{x}_n = (t_n, \mathbf{u}(t_n))$, where $t_n \in \mathbb{R}_{\geq 0}$ is a time stamp of feature $\mathbf{u}(t_n) \in \mathbb{R}^{\dim(\mathbf{u})}$, and $\dim(\mathbf{u})$ is a feature dimension. $\mathcal{X}$ can be classified into two types: regular time-series and irregular time-series. For irregular setting, we randomly remove 30%, 50% and 70% of samples from regular time-series. Therefore, the only difference between these types is whether time intervals, $\{t_{n+1} - t_n\}_{n=1}^{N-1}$, are the same or not.

Our irregular setting considers generating complete time-series given training data with missing values. Consequently, we generate total time-series on both regular and irregular settings and compare the generated ones with the original regular data on evaluation process.

3.2 Encoder and Decoder

The encoder and decoder have the task of mapping time-series data to a latent space and vice versa. We define e and d as an encoding function mapping $\mathcal{X}$ to $\mathcal{H}$ and a decoding function mapping $\mathcal{H}$ to $\mathcal{X}$, respectively. For simplicity but without loss of generality, we utilize an autoregressive autoencoder: RNN-based ones for regular time-series (Cho et al., 2014b) and Neural CDE, GRU-ODE for irregular setting (Kidger et al., 2020; Brouwer et al., 2019). In this section, we describe the regular time-series generation as representative one and leave the irregular case in Appendix I, depicting how continuous-time methods can be used for the encoder and the decoder to better synthesize irregular time-series.

The encoder e and the decoder d consist of recurrent neural networks, e.g., gated recurrent units (GRUs) (Cho et al., 2014b). Since we use RNNs, both e and d are defined recursively as follows:

$$\mathbf{h}_n = e(\mathbf{h}_{n-1}, \mathbf{x}_n), \qquad \hat{\mathbf{x}}_n = d(\mathbf{h}_n), \tag{5}$$

where $\hat{\mathbf{x}}_n$ denotes a reconstructed time-series sample at sequential order n. It is well-known that RNN was devised to efficiently handle variable sequences by summarizing past observations (Cho et al., 2014a). For example, Yoon et al. (2019) also used RNN-based encoder and decoder to provide a reversible mapping between features and latent representations, thereby reducing the high-dimensionality of the adversarial learning space, which is well supported by $\mathbf{h}_n \sim \mathbf{x}_{1:n}$.

After embedding real time-series data onto a latent space, we can train the conditional score network with its conditional log-likelihood, whose architecture is described in Appendix J.2. The encoder and decoder are pre-trained before our main training.

3.3 Training Objective Function

Loss for autoencoder We use two training objective functions. First, we train the encoder and the decoder using L_{ed}. Let $\mathbf{x}^0_{1:N} \sim p(\mathbf{x}^0_{1:N})$ and $\hat{\mathbf{x}}^0_{1:N}$ denote an real time-series sample and its reconstructed copy by the encoder-decoder process, respectively. Then, L_{ed} denotes the following MSE loss between $\mathbf{x}^0_{1:N}$ and its reconstructed copy $\hat{\mathbf{x}}^0_{1:N}$:

$$L_{ed} = \mathbb{E}_{\mathbf{x}^0_{1:N}}[\left\|\hat{\mathbf{x}}^0_{1:N} - \mathbf{x}^0_{1:N}\right\|_2^2]. \tag{6}$$

Loss for score network Next, we define another loss $L^{\mathcal{H}}_{score}$ in Eq. equation 11 to train the conditional score network M_θ, which is one of our main contributions. In order to derive the training loss $L^{\mathcal{H}}_{score}$ from the initial loss definition L_1, we describe its step-by-step derivation procedure. At sequential order n in $\{1, ..., N\}$, we diffuse $\mathbf{x}^0_{1:n}$ through a sufficiently large number of steps of the forward SDE to a Gaussian distribution. Let $\mathbf{x}^s_{1:n}$ denotes a diffused sample at step $s \in [0, 1]$ from

$\mathbf{x}^0_{1:n}$. Then, the conditional score network $M_\theta(s, \mathbf{x}^s_{1:n}, \mathbf{x}^0_{1:n-1})$ can be trained to learn the gradient of the conditional log-likelihood with the following L_1 loss:

$$L_1 = \mathbb{E}_s \mathbb{E}_{\mathbf{x}^0_{1:N}} \left[\sum_{n=1}^{N} \lambda(s) l_1(n, s) \right], \tag{7}$$

where

$$l_1(n, s) = \mathbb{E}_{\mathbf{x}^s_{1:n}} \left[\left\| M_\theta(s, \mathbf{x}^s_{1:n}, \mathbf{x}^0_{1:n-1}) - \nabla_{\mathbf{x}^s_{1:n}} \log p(\mathbf{x}^s_{1:n} | \mathbf{x}^0_{1:n-1}) \right\|_2^2 \right]. \tag{8}$$

In the above definition, $\nabla_{\mathbf{x}^s_{1:n}} \log p(\mathbf{x}^s_{1:n} | \mathbf{x}^0_{1:n-1})$, where $\mathbf{x}^0_i$ depends on $\mathbf{x}^0_{1:i-1}$ for each $i \in \{2, ..., n\}$, is designed specially for time-series generation. Note that for our training, $\mathbf{x}^s_{1:n}$ is sampled from $p(\mathbf{x}^s_{1:n} | \mathbf{x}^0_{1:n-1})$, and s is uniformly sampled from $[0, 1]$.

However, using the above formula, which is a naïve score matching on time-series, is computationally prohibitive (Hyvärinen, 2005; Song et al., 2019). Thanks to the following theorem, the more efficient denoising score loss L_{score} can be defined.

Theorem 3.1 (Autoregressive denoising score matching)**.** *$l_1(n, s)$ can be replaced with the following $l_2(n, s)$*

$$L_{score} = \mathbb{E}_s \mathbb{E}_{\mathbf{X}^0_{1:N}} \left[\sum_{n=1}^{N} \lambda(s) l_2(n, s) \right], \tag{9}$$

where

$$l_2(n, s) = \mathbb{E}_{\mathbf{X}^s_{1:n}} \left[\left\| M_\theta(s, \mathbf{X}^s_{1:n}, \mathbf{X}^0_{1:n-1}) - \nabla_{\mathbf{X}^s_{1:n}} \log p(\mathbf{X}^s_{1:n} | \mathbf{X}^0_{1:n}) \right\|_2^2 \right]. \tag{10}$$

Then, $L_1 = L_{score}$ is satisfied. □

Since we pre-train the encoder and decoder, the encoder can embed $\mathbf{x}^0_{1:n}$ into $\mathbf{h}^0_n \in \mathcal{H}$. Ideally, $\mathbf{h}^0_n$ involves the entire information of $\mathbf{x}^0_{1:n}$. Therefore, L_{score} can be re-written as follows with the embeddings in the latent space:

$$L^{\mathcal{H}}_{score} = \mathbb{E}_s \mathbb{E}_{\mathbf{h}^0_{1:N}} \sum_{n=1}^{N} \left[\lambda(s) l_3(n, s) \right], \tag{11}$$

with $l_3(n, s) = \mathbb{E}_{\mathbf{h}^s_n} \left[\left\| M_\theta(s, \mathbf{h}^s_n, \mathbf{h}^0_{n-1}) - \nabla_{\mathbf{h}^s_n} \log p(\mathbf{h}^s_n | \mathbf{h}^0_n) \right\|_2^2 \right]$. $L^{\mathcal{H}}_{score}$ is what we use for our experiments (instead of L_{score}). Until now, we introduced our target objective functions, L_{ed} and $L^{\mathcal{H}}_{score}$. We note that we use exactly the same weight $\lambda(s)$ as that in (Song et al., 2021). Related proofs are given in Appendix A.

3.4 Training and Sampling Procedures

Training method We explain details of our training method. At first, we pre-train both the encoder and decoder using L_{ed}. After pre-training them, we train the conditional score network. When training the latter one, we use the embedded hidden vectors produced by the encoder. After encoding an input $\mathbf{x}^0_{1:N}$, we obtain its latent vectors $\mathbf{h}^0_{1:N}$ — we note that each hidden vector $\mathbf{h}^0_n$ has all the previous information from 1 to n for the RNN-based encoder's autoregressive property as shown in the Equation 3.2. We use the following forward process (Song et al., 2021), where n means the sequence order of the input time-series, and s denotes the time (or step) of the diffusion step :

$$d\mathbf{h}^s_n = \mathbf{f}(s, \mathbf{h}^s_n) ds + g(s) d\mathbf{w}, \qquad s \in [0, 1].$$

Note that we only use the VP and subVP-based TSGM in our experiments and exclude the VE-based one for its inferiority for time series synthesis in our experiments. During the forward process, the conditional score network reads the pair (s, $\mathbf{h}^s_n$, $\mathbf{h}^0_{n-1}$) as input and thereby, it can learn the conditional score function $\nabla \log p(\mathbf{h}^s_n | \mathbf{h}^0_{n-1})$ by using $L^{\mathcal{H}}_{score}$, where $\mathbf{h}^0_0 = \mathbf{0}$.

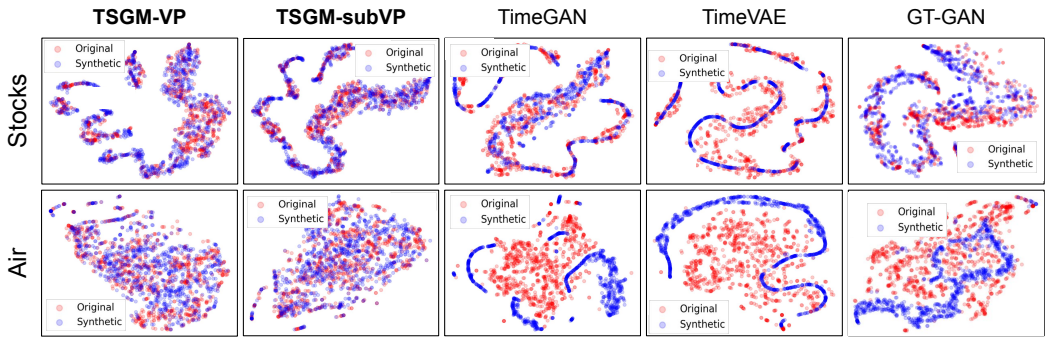

Figure 3: t-SNE plots for TSGM (1st and 2nd columns), TimeGAN (3rd columns), TimeVAE (4th columns), GT-GAN (5th columns) in Stocks and Air datasets. Red and blue dots mean original and synthesized samples, respectively. Refer to Appendix L for addition visualizations

Sampling method After the training procedure, we use the following conditional reverse process:

$$d\mathbf{h}_n^s = [\mathbf{f}(s, \mathbf{h}_n^s) - g^2(s)\nabla_{\mathbf{h}_n^s} \log p(\mathbf{h}_n^s | \mathbf{h}_{n-1}^0)]ds + g(s)d\bar{\mathbf{w}}, \tag{12}$$

where s is uniformly sampled over $[0, 1]$. The conditional score function in this process can be replaced with the trained score network $M_\theta(s, \mathbf{h}_n^s, \mathbf{h}_{n-1}^0)$. The detailed sampling method is as follows:

1. At first, we sample $\mathbf{z}_1$ from a Gaussian prior distribution and set $\mathbf{h}_1^1 = \mathbf{z}_1$ and $\mathbf{h}_0^0 = \mathbf{0}$. We then generates an initial observation $\hat{\mathbf{h}}_1^0$ by denoising $\mathbf{h}_1^1$ following the conditional reverse process with $M_\theta(s, \mathbf{h}_n^s, \mathbf{h}_0^0)$ via the *predictor-corrector* method (Song et al., 2021).
2. We repeat the following computation for every $2 \leq n \leq N$, i.e., recursive generation. We sample $\mathbf{z}_n$ from a Gaussian prior distribution and set $\mathbf{h}_n^1 = \mathbf{z}_n$ for $n \in \{2, ..., N\}$. After reading the previously generated samples $\hat{\mathbf{h}}_{n-1}^0$, we then denoise $\mathbf{h}_n^1$ following the conditional reverse process with $M_\theta(s, \mathbf{h}_n^s, \mathbf{h}_{n-1}^0)$ to generate $\hat{\mathbf{h}}_n^0$ via the *predictor-corrector* method.

Once the sampling procedure is finished, we can reconstruct $\hat{\mathbf{x}}_{1:N}^0$ from $\hat{\mathbf{h}}_{1:N}^0$ using the trained decoder at once.

4 Experiments

4.1 Experimental Environments

4.1.1 Baselines and Datasets

In the case of the regular time-series generation, we use 4 real-world datasets from various fields with 9 baselines. For the irregular time-series generation, we randomly remove some observations from each time-series sample with 30%, 50%, and 70% missing rates. Therefore, we totally treat 16 datasets, i.e., 4 datasets with one regular and three irregular settings, and 9 baselines.

Our collection of baselines covers almost all existing types of time-series synthesis methods, ranging from autoregressive generative models to VAEs and GANs. For the baselines, we reuse their released source codes in their official repositories and rely on their designed training and model selection procedures. If a baseline does not support irregular time-series synthesis, we replace its RNN encoder with GRU-D (Che et al., 2016) modified from GRUs to deal with irregular time-series. For those that do not use an RNN-based encoder, we add GRU-D in front of the encoder, such as TimeVAE and COT-GAN. Therefore, all baselines are tested for the regular and irregular environments. We refer to Appendix E for the detailed descriptions on our datasets, baselines, and Appendix G for other software/hardware environments.

4.1.2 Evaluation Metrics

In the image generation domain, researchers have evaluated the *fidelity* and the *diversity* of models by using the Fréchet inception distance (FID) and inception score (IS). On the other hand, to measure

Table 2: The left and right ones denote experimental results on regular time-series and irregular time-series with 30% missing rates, respectively. Results for higher missing rates are in Table 12. Note that except for our method (TSGM), CTFP, and GT-GAN, the other methods cannot deal with irregular time series, so we make it possible for them to operate on irregular settings by replacing RNN encoder with GRU-D.

	Method	Regular Settings				Irregular Settings (Missing Rate: 30%)			
		Stocks	Energy	Air	AI4I	Stocks	Energy	Air	AI4I
Disc. score	TSGM-VP	.022±.005	.221±.025	**.122±.014**	.147±.005	.062±.018	**.294±.007**	**.190±.042**	.142±.048
	TSGM-subVP	**.021±.008**	**.198±.025**	.127±.010	.150±.010	**.025±.009**	.326±.008	.240±.018	**.121±.082**
	T-Forcing	.226±.035	.483±.004	.404±.020	.435±.025	.409±.051	.347±.046	.458±.122	.493±.018
	P-Forcing	.257±.026	.412±.006	.484±.007	.443±.026	.480±.060	.491±.020	.494±.012	.430±.061
	TimeGAN	.102±.031	.236±.012	.447±.017	**.070±.009**	.411±.040	.479±.010	.500±.001	.500±.000
	RCGAN	.196±.027	.336±.017	.459±.104	.234±.015	.500±.000	.500±.000	.500±.000	.500±.000
	C-RNN-GAN	.399±.028	.499±.001	.499±.000	.499±.001	.500±.000	.500±.000	.500±.000	.450±.150
	TimeVAE	.175±.031	.498±.006	.381±.037	.446±.024	.423±.088	.382±.124	.373±.191	.384±.086
	COT-GAN	.285±.030	.498±.000	.423±.001	.411±.018	.499±.001	.500±.000	.500±.000	.500±.000
	CTFP	.499±.000	.500±.000	.499±.000	.499±.001	.500±.000	.500±.000	.500±.000	.499±.001
	GT-GAN	.077±.031	.221±.068	.413±.001	.394±.090	.251±.097	.333±.063	.454±.029	.435±.018
Pred. score	TSGM-VP	**.037±.000**	.257±.000	**.005±.000**	**.217±.000**	**.012±.002**	**.049±.001**	**.042±.002**	.067±.013
	TSGM-subVP	**.037±.000**	**.252±.000**	**.005±.000**	**.217±.000**	.012±.001	**.049±.001**	.044±.004	**.061±.001**
	T-Forcing	.038±.001	.315±.005	.008±.000	.242±.001	.027±.002	.090±.001	.112±.004	.147±.010
	P-Forcing	.043±.001	.303±.006	.021±.000	.220±.000	.079±.008	.147±.001	.101±.003	.134±.005
	TimeGAN	.038±.001	.273±.004	.017±.004	.253±.002	.105±.053	.248±.024	.325±.005	.251±.010
	RCGAN	.040±.001	.292±.005	.043±.000	.224±.001	.523±.020	.409±.020	.342±.018	.329±.037
	C-RNN-GAN	.038±.000	.483±.005	.111±.000	.340±.006	.345±.002	.440±.000	.354±.060	.400±.026
	TimeVAE	.042±.002	.268±.004	.013±.002	.233±.010	.207±.014	.139±.004	.105±.002	.144±.003
	COT-GAN	.044±.000	.260±.000	.024±.001	.220±.000	.274±.000	.427±.000	.451±.000	.570±.000
	CTFP	.084±.005	.469±.008	.476±.235	.412±.024	.070±.009	.499±.000	.060±.027	.424±.002
	GT-GAN	.040±.000	.312±.002	.007±.000	.239±.000	.077±.031	.221±.068	.064±.002	.087±.013
	Original	.036±.001	.250±.003	.004±.000	.217±.000	.011±.002	.045±.001	.044±.006	.059±.001

the fidelity and the diversity of synthesized time-series samples, we use the following predictive score and the discriminative score as in (Yoon et al., 2019; Jeon et al., 2022). We strictly follow the evaluation protocol agreed by the time-series research community (Yoon et al., 2019; Jeon et al., 2022). Both metrics are designed in a way that lower values are preferred. We run each generative method 10 times with different seeds, and report its mean and standard deviation of the following discriminative and predictive scores:

i) *Predictive Score*: We use the predictive score to evaluate whether a generative model can successfully reproduce the temporal properties of the original data. To do this, we first train a popular LSTM-based sequence model for time-series forecasting with synthesized samples. However, the existing predictive score only predicts a last sample of total sequence, so Jeon et al. (2022) suggests a comprehensive approach that considers the entire time series. For fair comparison, we reuse evaluation metrics of Yoon et al. (2019), Jeon et al. (2022) for regular and irregular settings, respectively.

ii) *Discriminative Score*: In order to assess how similar the original and generated samples are, we train a 2-layer LSTM model that classifies the real/fake samples into two classes, real or fake. We use the performance of the trained classifier on the test data as the discriminative score. Therefore, lower discriminator scores mean real and fake samples are similar.

4.2 Experimental Results

At first, on the regular time-series generation, Table 2 shows that our method achieves remarkable results, outperforming TimeGAN and GT-GAN except only for the discriminative score on AI4I. Especially, for Stock, Energy, and Air, TSGM exhibits overwhelming performance by large margins for the discriminative score. Moreover, for the predictive score, TSGM performs the best and obtains almost the same scores as that of the original data, which indicates that generated samples from TSGM preserve all the predictive characteristics of the original data.

Next, on the irregular time-series generation, we give the result with the 30% missing rate setting on Table 2 and other results in Appendix K. TSGM also defeats almost all baselines by large margins on both the discriminative and predictive scores. Interestingly, VP generates poorer data as the missing rate grows up, while subVP synthesizes better one.

We show t-SNE visualizations and KDE plots for the regular time-series generation in Figure 3 and Figure 1. TimeGAN, GT-GAN, and TimeVAE are representative GAN or VAE-based baselines. In the figures, unlike the baseline methods, the synthetic samples generated from TSGM consistently show successful recall from the original data.

Table 3: Sensitivity results on the depth of M_θ and the number of sampling steps. Our default TSGM has a depth of 4 and its number of sampling steps is 1,000. For other omitted datasets, we observe similar patterns.

Method		TSGM		Depth of 3		500 steps		250 steps		100 steps	
SDE		VP	subVP	VP	subVP	VP	subVP	VP	subVP	VP	subVP
Disc.	Stocks	.022±.005	.021±.008	.022±.004	**.020±.007**	.025±.005	.020±.004	.067±.009	.022±.009	.202±.013	.023±.005
	Energy	.221±.025	.198±.025	**.175±.009**	.182±.009	.259±.003	.248±.002	.250±.003	.247±.002	.325±.003	.237±.004
Pred.	Stocks	.037±.000	**.037±.000**	.037±.000	.037±.000	.037±.000	.037±.000	.037±.000	.037±.000	.039±.000	.037±.000
	Energy	.257±.000	**.252±.000**	.253±.000	.253±.000	.257±.000	.253±.000	.256±.000	.253±.000	.256±.000	.253±.000

Furthermore, TSGM generates more diverse synthetic samples compared to the three representative baselines across all cases. Notably, TSGM achieves significantly higher diversity on the Energy and Air dataset, which exhibits the most complex correlations (cf. Fig. 3 and Fig. 6). Diffusion models are known to produce more diverse data than GANs (Bayat, 2023; Yang et al., 2023). As demonstrated by our results, TSGM synthesizes diverse data, highlighting another advantage of using diffusion models as anticipated.

4.3 Sensitivity and Ablation Studies

We conduct two sensitivity studies on regular time-series: i) reducing the depth of our score network, ii) decreasing the sampling step numbers. The results are in Table 3. At first, we modify the depth of our score network from 4 to 3 to check the performance of the lighter conditional score network. Surprisingly, we achieve a better discriminative score with a slight loss on the predictive score. Next, we decrease the number of sampling steps for faster sampling from 1,000 steps to 500, 250, and 100 steps, respectively. For VP, the case of 500 steps achieves almost the same results as that of original TSGM. Surprisingly, in the case of subVP, we achieve good results until 100 steps.

As an ablation study, we simultaneously train the conditional score network, encoder, and decoder from scratch on regular time-series generation (i.e., without the pre-training process). The results are in Table 4. These ablation models are worse than the full model due to the increased training complexity, but they still outperform many baselines. This ablation study shows the efficacy of pre-training our autoencoder. Additionally, in Appendix M, we provide an additional ablation study about the efficacy of our recursive structures.

Table 4: Comparison between with and without pre-training the autoencoder

	Method	SDE	Stocks	Energy
Disc.	TSGM	VP	.022±.005	.221±.025
		subVP	**.021±.008**	**.198±.025**
	w/o pre-training	VP	.022±.004	.322±.003
		subVP	.059±.006	.284±.004
Pred.	TSGM	VP	.037±.000	.257±.000
		subVP	**.037±.000**	.252±.000
	w/o pre-training	VP	.037±.000	.252±.000
		subVP	.037±.000	**.251±.000**

5 Conclusions

We presented a score-based generative model framework for universal time-series generation. We combined an autoencoder and our score network into a single framework to accomplish the goal — our framework supports RNN-based or continuous-time method-based autoencoders. We also designed an appropriate denoising score matching loss for our generation task and achieved state-of-the-art results on various datasets in terms of the discriminative and predictive scores. In addition, we conducted rigorous ablation and sensitivity studies to prove the efficacy of our model design.

Limitations. Although our method achieves state-of-the-art sampling quality and diversity, there exists a fundamental problem that all SGMs have. That is, SGMs are slower than GANs for generating samples (see Appendix H). Since there are several accomplishments for faster sampling (Xiao et al., 2022; Jolicoeur-Martineau et al., 2021), however, one can apply them to our method and it would be much faster without any loss of sampling quality and diversity. Moreover, TSGM's generation is not too much slow since time-series data are much smaller than image data, which means TSGM still has its commercial value. We point out that TSGM generates a sample of the Energy dataset, the largest dataset among our baselines, using only 0.8 second (see Section H).

REFERENCES

Nesreen K. Ahmed, Amir F. Atiya, Neamat El Gayar, and Hisham El-Shishiny. An empirical comparison of machine learning models for time series forecasting. *Econometric Reviews*, 29(5-6): 594–621, 2010.

Brian D.O. Anderson. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 12(3):313–326, 1982.

Reza Bayat. A study on sample diversity in generative models: GANs vs. diffusion models, 2023. URL https://openreview.net/forum?id=BQpCuJoMykZ.

Edward De Brouwer, Jaak Simm, Adam Arany, and Yves Moreau. Gru-ode-bayes: Continuous modeling of sporadically-observed time series. *CoRR*, abs/1905.12374, 2019. URL http://arxiv.org/abs/1905.12374.

Luis M. Candanedo, Véronique Feldheim, and Dominique Deramaix. Data driven prediction models of energy use of appliances in a low-energy house. *Energy and Buildings*, 140:81–97, 2017. ISSN 0378-7788.

Zhengping Che, S. Purushotham, Kyunghyun Cho, David A. Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific Reports*, 8, 2016. URL https://api.semanticscholar.org/CorpusID:4900015.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations, 2018. URL https://arxiv.org/abs/1806.07366.

KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014a. URL http://arxiv.org/abs/1409.1259.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014b. URL http://arxiv.org/abs/1406.1078.

Casey Chu, Kentaro Minami, and Kenji Fukumizu. Smoothness and stability in GANs. In *International Conference on Learning Representations*, 2020.

Saloni Dash, Andrew Yale, Isabelle Guyon, and Kristin P. Bennett. Medical time-series data generation using generative adversarial networks. In *Artificial Intelligence in Medicine: 18th International Conference on Artificial Intelligence in Medicine, AIME 2020, Minneapolis, MN, USA, August 25–28, 2020, Proceedings*, page 382–391, Berlin, Heidelberg, 2020. Springer-Verlag. ISBN 978-3-030-59136-6. doi: 10.1007/978-3-030-59137-3_34. URL https://doi.org/10.1007/978-3-030-59137-3_34.

S. De Vito, E. Massera, M. Piga, L. Martinotto, and G. Di Francia. On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario. *Sensors and Actuators B: Chemical*, 129(2):750–757, 2008. ISSN 0925-4005.

Ruizhi Deng, Bo Chang, Marcus A Brubaker, Greg Mori, and Andreas Lehrmann. Modeling continuous stochastic processes with dynamic normalizing flows. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 7805–7815. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/58c54802a9fb9526cd0923353a34a7ae-Paper.pdf.

Abhyuday Desai, Cynthia Freeman, Zuhui Wang, and Ian Beaver. Timevae: A variational auto-encoder for multivariate time series generation. *arXiv:2111.08095*, 2021.

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

Cristóbal Esteban, Stephanie L. Hyland, and Gunnar Rätsch. Real-valued (medical) time series generation with recurrent conditional GANs. *arXiv:1706.02633*, 2017.

Tak-chung Fu. A review on time series data mining. *Engineering Applications of Artificial Intelligence*, 24(1):164–181, 2011.

A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. Ch. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley. PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220, 2000 (June 13). Circulation Electronic Pages: http://circ.ahajournals.org/content/101/23/e215.full PMID:1085218; doi: 10.1161/01.CIR.101.23.e215.

Anirudh Goyal, Alex Lamb, Ying Zhang, Saizheng Zhang, Aaron Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. In *Advances in Neural Information Processing Systems*, 2016.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv:1308.0850*, 2013.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, 2020.

Aapo Hyvärinen. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(24):695–709, 2005. URL `http://jmlr.org/papers/v6/hyvarinen05a.html`.

Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33:917–963, 2019.

Jinsung Jeon, Jeonghak Kim, Haryong Song, Seunghyeon Cho, and Noseong Park. Gt-gan: General purpose time series synthesis with generative adversarial networks, 2022. URL `https://arxiv.org/abs/2210.02040`.

Alexia Jolicoeur-Martineau, Ke Li, Rémi Piché-Taillefer, Tal Kachman, and Ioannis Mitliagkas. Gotta go fast when generating data with score-based models. *CoRR*, abs/2105.14080, 2021. URL `https://arxiv.org/abs/2105.14080`.

Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. *CoRR*, abs/1912.04958, 2019. URL `http://arxiv.org/abs/1912.04958`.

Patrick Kidger, James Morrill, James Foster, and Terry J. Lyons. Neural controlled differential equations for irregular time series. *CoRR*, abs/2005.08926, 2020. URL `https://arxiv.org/abs/2005.08926`.

Jayoung Kim, Chaejeong Lee, Yehjin Shin, Sewon Park, Minjung Kim, Noseong Park, and Jihoon Cho. Sos: Score-based oversampling for tabular data. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022.

Stephan Matzka. Ai4i 2020 predictive maintenance dataset. `https://archive.ics.uci.edu/ml/datasets/AI4I+2020+Predictive+Maintenance+Dataset`, 2020.

Chenlin Meng, Lantao Yu, Yang Song, Jiaming Song, and Stefano Ermon. Autoregressive score matching. *CoRR*, abs/2010.12810, 2020. URL `https://arxiv.org/abs/2010.12810`.

Olof Mogren. C-RNN-GAN: A continuous recurrent neural network with adversarial training. In *Constructive Machine Learning Workshop (CML) at NeurIPS 2016*, 2016.

Kashif Rasul, Calvin Seward, Ingmar Schuster, and Roland Vollgraf. Autoregressive denoising diffusion models for multivariate probabilistic time series forecasting. In *International Conference on Machine Learning*, 2021.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, pages 234–241, 2015.

Donald B. Rubin. Inference and missing data. *Biometrika*, 63(3):581–592, 1976. ISSN 00063444. URL http://www.jstor.org/stable/2335739.

Joseph L. Schafer and John W. Graham. Missing data: our view of the state of the art. *Psychological methods*, 7 2:147–77, 2002. URL https://api.semanticscholar.org/CorpusID:7745507.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, 2015.

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In *Advances in Neural Information Processing Systems*, 2019.

Yang Song, Sahaj Garg, Jiaxin Shi, and Stefano Ermon. Sliced score matching: A scalable approach to density and score estimation. *CoRR*, abs/1905.07088, 2019. URL http://arxiv.org/abs/1905.07088.

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021.

Ilya Sutskever, James Martens, and Geoffrey Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, page 1017–1024, 2011.

Simo Särkkä and Arno Solin, editors. *Applied stochastic differential equations*, volume 10. Cambridge University Press, 2019.

Yusuke Tashiro, Jiaming Song, Yang Song, and Stefano Ermon. Csdi: Conditional score-based diffusion models for probabilistic time series imputation. In *Advances in Neural Information Processing Systems*, 2021.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(86):2579–2605, 2008.

Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural Computation*, 23(7):1661–1674, 2011.

Zhisheng Xiao, Karsten Kreis, and Arash Vahdat. Tackling the generative learning trilemma with denoising diffusion GANs. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=JprM0p-q0Co.

Tianlin Xu, Li K. Wenliang, Michael Munn, and Beatrice Acciaio. Cot-gan: Generating sequential data via causal optimal transport. In *Advances in Neural Information Processing Systems*, 2020.

Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. Diffusion models: A comprehensive survey of methods and applications, 2023.

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

A PROOFS

We introduce an additional lemma to prove Theorem 3.1. In the following lemma, we state about the denoising score matching on time-series.

Lemma A.1. *In L_1 loss function, $l_1(n,s)$ can be replaced by the following $l_2^\star(n,s)$:*

$$l_2^\star(n,s) = \mathbb{E}_{\mathbf{x}_n^0}\mathbb{E}_{\mathbf{x}_{1:n}^s}\left[\left\|M_\theta(s,\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0) - \nabla_{\mathbf{x}_{1:n}^s}\log p(\mathbf{x}_{1:n}^s|\mathbf{x}_{1:n}^0)\right\|_2^2\right], \tag{13}$$

where $\mathbf{x}_n^0$ and $\mathbf{x}_{1:n}^s$ are sampled from $p(\mathbf{x}_n^0|\mathbf{x}_{1:n-1}^0)$ and $p(\mathbf{x}_{1:n}^s|\mathbf{x}_{1:n}^0)$. Therefore, we can use an alternative objective, $L_2 = \mathbb{E}_s\mathbb{E}_{\mathbf{x}_{1:N}}\left[\sum_{n=1}^N \lambda(s) l_2^\star(n,s)\right]$ instead of L_1.

Proof. At first, if $n=1$, it can be substituted with the naive denoising score loss by Vincent (2011) since $\mathbf{x}_0^0 = \mathbf{0}$.

Next, let us consider $n > 1$. $l_1(n,s)$ can be decomposed as follows:

$$\begin{aligned} l_1(n,s) = -2\cdot\mathbb{E}_{\mathbf{x}_{1:n}^s}\langle M_\theta(s,\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0), \nabla_{\mathbf{x}_{1:n}^s}\log p(\mathbf{x}_{1:n}^s|\mathbf{x}_{1:n-1}^0)\rangle \\ +\mathbb{E}_{\mathbf{x}_{1:n}^s}\left[\left\|M_\theta(s,\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0)\right\|_2^2\right] + C_1 \end{aligned} \tag{14}$$

Here, C_1 is a constant that does not depend on the parameter θ, and $\langle\cdot,\cdot\rangle$ means the inner product. Then, the first part's expectation of the right-hand side can be expressed as follows:

$$\begin{aligned} &\mathbb{E}_{\mathbf{x}_{1:n}^s}[\langle M_\theta(s,\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0), \nabla_{\mathbf{x}_{1:n}^s}\log p(\mathbf{x}_{1:n}^s|\mathbf{x}_{1:n-1}^0)\rangle] \\ &= \int_{\mathbf{x}_{1:n}^s}\langle M_\theta(s,\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0), \nabla_{\mathbf{x}_{1:n}^s}\log p(\mathbf{x}_{1:n}^s|\mathbf{x}_{1:n-1}^0)\rangle \mathrm{p}(\mathbf{x}_{1:n}^s|\mathbf{x}_{1:n-1}^0) d\mathbf{x}_{1:n}^s \\ &= \int_{\mathbf{x}_{1:n}^s}\langle M_\theta(s,\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0), \frac{1}{\mathrm{p}(\mathbf{x}_{1:n-1}^0)}\frac{\partial \mathrm{p}(\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0)}{\partial \mathbf{x}_{1:n}^s}\rangle d\mathbf{x}_{1:n}^s \\ &= \int_{\mathbf{x}_n^0}\int_{\mathbf{x}_{1:n}^s}\langle M_\theta(s,\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0), \frac{1}{\mathrm{p}(\mathbf{x}_{1:n-1}^0)}\frac{\partial \mathrm{p}(\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0,\mathbf{x}_n^0)}{\partial \mathbf{x}_{1:n}^s}\rangle d\mathbf{x}_{1:n}^s d\mathbf{x}_n^0 \\ &= \int_{\mathbf{x}_n^0}\int_{\mathbf{x}_{1:n}^s}\langle M_\theta(s,\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0), \frac{\partial \mathrm{p}(\mathbf{x}_{1:n}^s|\mathbf{x}_{1:n}^0))}{\partial \mathbf{x}_{1:n}^s}\rangle \frac{\mathrm{p}(\mathbf{x}_{1:n-1}^0,\mathbf{x}_n^0)}{\mathrm{p}(\mathbf{x}_{1:n-1}^0)} d\mathbf{x}_{1:n}^s d\mathbf{x}_n^0 \\ &= \int_{\mathbf{x}_n^0}\int_{\mathbf{x}_{1:n}^s}\langle M_\theta(s,\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0), \frac{\partial \mathrm{p}(\mathbf{x}_{1:n}^s|\mathbf{x}_{1:n}^0)}{\partial \mathbf{x}_{1:n}^s}\rangle \mathrm{p}(\mathbf{x}_n^0|\mathbf{x}_{1:n-1}^0) d\mathbf{x}_{1:n}^s d\mathbf{x}_n^0 \\ &= \mathbb{E}_{\mathbf{x}_n^0}\left[\int_{\mathbf{x}_{1:n}^s}\langle M_\theta(s,\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0), \frac{\partial \mathrm{p}(\mathbf{x}_{1:n}^s|\mathbf{x}_{1:n}^0)}{\partial \mathbf{x}_{1:n}^s}\rangle d\mathbf{x}_{1:n}^s\right] \\ &= \mathbb{E}_{\mathbf{x}_n^0}\left[\int_{\mathbf{x}_{1:n}^s}\langle M_\theta(s,\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0), \nabla_{\mathbf{x}_{1:n}^s}\log p(\mathbf{x}_{1:n}^s|\mathbf{x}_{1:n}^0)\rangle \mathrm{p}(\mathbf{x}_{1:n}^s|\mathbf{x}_{1:n}^0) d\mathbf{x}_{1:n}^s\right] \\ &= \mathbb{E}_{\mathbf{x}_n^0}\mathbb{E}_{\mathbf{x}_{1:n}^s}[\langle M_\theta(s,\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0), \nabla_{\mathbf{x}_{1:n}^s}\log p(\mathbf{x}_{1:n}^s|\mathbf{x}_{1:n}^0)\rangle] \end{aligned} \tag{15}$$

Similarly, the second part's expectation of the right-hand side can be rewritten as follows:

$$
\begin{aligned}
&\mathbb{E}_{\mathbf{x}^s_{1:n}}[\left\|M_\theta(s,\mathbf{x}^s_{1:n},\mathbf{x}^0_{1:n-1})\right\|_2^2] \\
&= \int_{\mathbf{x}^s_{1:n}} \left\|M_\theta(s,\mathbf{x}^s_{1:n},\mathbf{x}^0_{1:n-1})\right\|_2^2 \cdot \mathrm{p}(\mathbf{x}^s_{1:n}|\mathbf{x}^0_{1:n-1}) d\mathbf{x}^s_{1:n} \\
&= \int_{\mathbf{x}^0_n}\int_{\mathbf{x}^s_{1:n}} \left\|M_\theta(s,\mathbf{x}^s_{1:n},\mathbf{x}^0_{1:n-1})\right\|_2^2 \cdot \frac{\mathrm{p}(\mathbf{x}^s_{1:n},\mathbf{x}^0_{1:n-1},\mathbf{x}^0_n)}{\mathrm{p}(\mathbf{x}^0_{1:n-1})} d\mathbf{x}^s_{1:n} d\mathbf{x}^0_n \\
&= \int_{\mathbf{x}^0_n}\int_{\mathbf{x}^s_{1:n}} \left\|M_\theta(s,\mathbf{x}^s_{1:n},\mathbf{x}^0_{1:n-1})\right\|_2^2 \cdot \mathrm{p}(\mathbf{x}^s_{1:n}|\mathbf{x}^0_{1:n}) \frac{\mathrm{p}(\mathbf{x}^0_{1:n-1},\mathbf{x}^0_n)}{\mathrm{p}(\mathbf{x}^0_{1:n-1})} d\mathbf{x}^s_{1:n} d\mathbf{x}^0_n \\
&= \int_{\mathbf{x}^0_n}\int_{\mathbf{x}^s_{1:n}} \left\|M_\theta(s,\mathbf{x}^s_{1:n},\mathbf{x}^0_{1:n-1})\right\|_2^2 \cdot \mathrm{p}(\mathbf{x}^s_{1:n}|\mathbf{x}^0_{1:n}) \mathrm{p}(\mathbf{x}^0_n|\mathbf{x}^0_{1:n-1}) d\mathbf{x}^s_{1:n} d\mathbf{x}^0_n \\
&= \mathbb{E}_{\mathbf{x}^0_n}\mathbb{E}_{\mathbf{x}^s_{1:n}}[\left\|M_\theta(s,\mathbf{x}^s_{1:n},\mathbf{x}^0_{1:n-1})\right\|_2^2]
\end{aligned}
\tag{16}
$$

Finally, by using above results, we can derive following result:

$$
\begin{aligned}
l_1 &= \mathbb{E}_{\mathbf{x}^0_n}\mathbb{E}_{\mathbf{x}^s_{1:n}}\left[\left\|M_\theta(s,\mathbf{x}^s_{1:n},\mathbf{x}^0_{1:n-1})\right\|_2^2\right] + C_1 \\
&- 2\cdot \mathbb{E}_{\mathbf{x}^0_n}\mathbb{E}_{\mathbf{x}^s_{1:n}}\langle M_\theta(s,\mathbf{x}^s_{1:n},\mathbf{x}^0_{1:n-1}), \nabla_{\mathbf{x}^s_{1:n}} \log p(\mathbf{x}^s_{1:n}|\mathbf{x}^0_{1:n})\rangle \\
&= \mathbb{E}_{\mathbf{x}^0_n}\mathbb{E}_{\mathbf{x}^s_{1:n}}\left[\left\|M_\theta(s,\mathbf{x}^s_{1:n},\mathbf{x}^0_{1:n-1}) - \nabla_{\mathbf{x}^s_{1:n}} \log p(\mathbf{x}^s_{1:n}|\mathbf{x}^0_{1:n})\right\|_2^2\right] + C
\end{aligned}
\tag{17}
$$

C is a constant that does not depend on the parameter θ. □

Theorem A.1 (Autoregressive denoising score matching)**.** *$l_1(n,s)$ can be replaced with the following $l_2(n,s)$*

$$
L_{score} = \mathbb{E}_s\mathbb{E}_{\mathbf{x}^0_{1:N}}\left[\sum_{n=1}^{N}\lambda(s) l_2(n,s)\right], \tag{18}
$$

where

$$
l_2(n,s) = \mathbb{E}_{\mathbf{x}^s_{1:n}}\left[\left\|M_\theta(s,\mathbf{x}^s_{1:n},\mathbf{x}^0_{1:n-1}) - \nabla_{\mathbf{x}^s_{1:n}} \log p(\mathbf{x}^s_{1:n}|\mathbf{x}^0_{1:n})\right\|_2^2\right]. \tag{19}
$$

Then, $L_1 = L_{score}$ is satisfied.

proof. By Lemma A.1, it suffices to show that $L_2 = L_{score}$. Whereas one can use the law of total expectation, which means $E[X] = E[E[X|Y]]$ *if X,Y are on an identical probability space* to show the above formula, we calculate directly. At first, let us simplify the expectation of the inner part with a symbol $f(\mathbf{x}^0_{1:n})$ for our computational convenience, i.e., $f(\mathbf{x}^0_{1:n}) = \mathbb{E}_s\mathbb{E}_{\mathbf{x}^s_{1:n}}\left[\lambda(s)\left\|M_\theta(s,\mathbf{x}^s_{1:n},\mathbf{x}^0_{1:n-1}) - \nabla_{\mathbf{x}^s_{1:n}} \log p(\mathbf{x}^s_{1:n}|\mathbf{x}^0_{1:n})\right\|_2^2\right]$. Then we have the following definition:

$$
L_2 = \mathbb{E}_s\mathbb{E}_{\mathbf{x}^0_{1:N}}[l_2^\star] = \mathbb{E}_{\mathbf{x}^0_{1:N}}\left[\sum_{n=1}^{N}\mathbb{E}_{\mathbf{x}^0_n}[f(\mathbf{x}^0_{1:n})]\right] = \sum_{n=1}^{N}\mathbb{E}_{\mathbf{x}^0_{1:N}}\mathbb{E}_{\mathbf{x}^0_n}[f(\mathbf{x}^0_{1:n})] \tag{20}
$$

At last, the expectation part can be further simplified as follows:

$$
\begin{aligned}
&\mathbb{E}_{\mathbf{x}^0_{1:N}}\mathbb{E}_{\mathbf{x}^0_n}[f(\mathbf{x}^0_{1:n})] \\
&= \int_{\mathbf{x}^0_{1:N}}\int_{\mathbf{x}^0_n} f(\mathbf{x}^0_{1:n})p(\mathbf{x}^0_n|\mathbf{x}^0_{1:n-1})d\mathbf{x}^0_n \cdot p(\mathbf{x}^0_{1:n-1})p(\mathbf{x}^0_{n:N}|\mathbf{x}^0_{1:n-1})d\mathbf{x}^0_{1:N} \\
&= \int_{\mathbf{x}^0_{1:N}}\int_{\mathbf{x}^0_n} f(\mathbf{x}^0_{1:n})p(\mathbf{x}^0_{1:n})d\mathbf{x}^0_n \cdot p(\mathbf{x}^0_{n:N}|\mathbf{x}^0_{1:n-1})d\mathbf{x}^0_{1:N} \\
&= \int_{\mathbf{x}^0_{n:N}}\left(\int_{\mathbf{x}^0_{1:n}} f(\mathbf{x}^0_{1:n})p(\mathbf{x}^0_{1:n})d\mathbf{x}^0_{1:n}\right) p(\mathbf{x}^0_{n:N}|\mathbf{x}^0_{1:n-1})d\mathbf{x}^0_{n:N} \\
&= \int_{\mathbf{x}^0_{1:n}} f(\mathbf{x}^0_{1:n})p(\mathbf{x}^0_{1:n})d\mathbf{x}^0_{1:n} \\
&= \int_{\mathbf{x}^0_{1:n}}\left(\int_{\mathbf{x}^0_{n+1:N}} p(\mathbf{x}^0_{n+1:N}|\mathbf{x}^0_{1:n})d\mathbf{x}^0_{n+1:N}\right) f(\mathbf{x}^0_{1:n})p(\mathbf{x}^0_{1:n})d\mathbf{x}^0_{1:n} \\
&= \int_{\mathbf{x}^0_{1:N}} f(\mathbf{x}^0_{1:n})p(\mathbf{x}^0_{1:N})d\mathbf{x}^0_{1:N} \\
&= \mathbb{E}_{\mathbf{x}^0_{1:N}}[f(\mathbf{x}^0_{1:n})]
\end{aligned}
\tag{21}
$$

Since $\sum_{n=1}^{N}\mathbb{E}_{\mathbf{x}^0_{1:N}}[f(\mathbf{x}^0_{1:n})] = \mathbb{E}_{\mathbf{x}^0_{1:N}}[\sum_{n=1}^{N} f(\mathbf{x}^0_{1:n})] = L_{score}$, we prove the theorem. □

B Existing Time-Series Diffusion Models

There exist time-series diffusion models for forecasting and imputation (Rasul et al., 2021; Tashiro et al., 2021). However, our approach to time-series synthesis is technically distinct. While these models aim to generate fulfilled samples given partially known time-series, TSGM focuses on producing diverse samples. Furthermore, TSGM is designed to provide diverse regular time-series given regular or irregular data unlike time-series forecasting, which predicts future observations based on past data, and time-series imputation, which infers missing elements within a given time-series sample. This intuition is reflected in the experimental results in Section C.

We refer to a detailed analysis of these experiments in Section C.3.

B.1 Diffusion Models for Time-series Forecasting and Imputation

TimeGrad (Rasul et al., 2021) is a diffusion-based method for time-series forecasting, and CSDI (Tashiro et al., 2021) is for time-series imputation.

In TimeGrad (Rasul et al., 2021), they used a diffusion model for forecasting future observations given past observations. On each sequential order $n \in \{2, ..., N\}$ and diffusion step $s \in \{1, ..., T\}$, they train a neural network $\boldsymbol{\epsilon}_\theta(\cdot, \mathbf{x}_{1:n-1}, s)$ with a time-dependent diffusion coefficient $\bar{\alpha}_s$ by minimizing the following objective function:

$$
\mathbb{E}_{\mathbf{x}^0_n,\boldsymbol{\epsilon},s}[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_s}\mathbf{x}^0_n + \sqrt{1-\bar{\alpha}_s}\boldsymbol{\epsilon}, \mathbf{x}_{1:n-1}, s)\right\|_2^2], \tag{22}
$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. The above formula assumes that we already know $\mathbf{x}_{1:n-1}$, and by using an RNN encoder, $\mathbf{x}_{1:n-1}$ can be encoded into $\mathbf{h}_{n-1}$. After training, the model forecasts future observations recursively. More precisely speaking, $\mathbf{x}_{1:n-1}$ is encoded into $\mathbf{h}_{n-1}$ and the next observation $\mathbf{x}_n$ is forecast from the previous condition $\mathbf{h}_{n-1}$.

CSDI (Tashiro et al., 2021) proposed a general diffusion framework which can be applied mainly to time-series imputation. CSDI reconstructs an entire sequence at once, not recursively. Let $\mathbf{x}^0 \in \mathbb{R}^{\dim(\mathbf{X})\times N}$ be an entire time-series sequence with N observations in a matrix form. They define $\mathbf{x}^0_{co}$ and $\mathbf{x}^0_{ta}$ as conditions and imputation targets which are derived from $\mathbf{x}^0$, respectively. They then

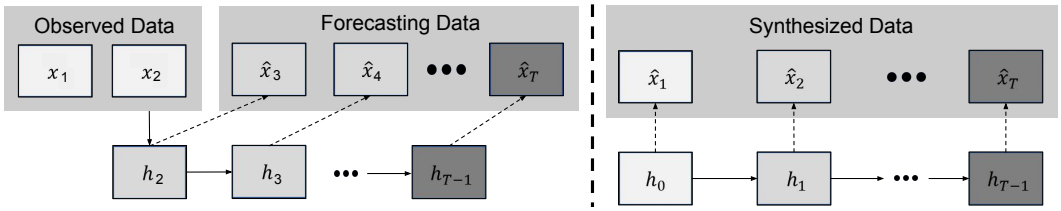

Figure 4: Graphical representation of TimeGrad (left) and TSGM (right). We adapt TimeGrad to our generation task but its results are not comparable even to other baselines' results (see Appendix C.1).

train a neural network $\epsilon_\theta(\cdot, \mathbf{x}^0_{co}, s)$ with a corresponding diffusion coefficient $\bar{\alpha}_s$ and a diffusion step $s \in \{1, ..., T\}$ by minimizing the following objective function:

$$\mathbb{E}_{\mathbf{x}^0,\epsilon,s}[\left\|\epsilon - \epsilon_\theta(s, \mathbf{x}^s_{ta}, \mathbf{x}^0_{co})\right\|^2_2], \tag{23}$$

where $\mathbf{x}^s_{ta} = \sqrt{\bar{\alpha}_s}\mathbf{x}^0_{ta} + (1-\bar{\alpha}_s)\epsilon$. By training the network using the above loss, it generates missing elements from the partially filled matrix $\mathbf{x}^0_{co}$.

B.2 Difference between Existing and Our Works

Although they have earned state-of-the-art results for forecasting and imputation, we found that they are not suitable for our generative task due to the fundamental mismatch between their model designs and our task (cf. Table 5 and Fig. 4).

Table 5: Comparison among various recent GAN, diffusion, and SGM-based methods for time-series. $\mathbf{x}_t$ (resp. $\hat{\mathbf{x}}_t$) means a raw (resp. synthesized) observation at time t. For CSDI, $\mathbf{x}_{co}$ means a set of known values and $\mathbf{x}_{ta}$ means a set of target missing values — it is not necessary that $\mathbf{x}_{co}$ precedes $\mathbf{x}_{ta}$ in time in CSDI.

Method	Type	Task Description
TimeGrad	Diffusion	From $\mathbf{x}_{1:N-K}$, infer $\hat{\mathbf{x}}_{N-K+1:N}$.
CSDI	Diffusion	Given known values $\mathbf{x}_{co}$, infer missing values $\hat{\mathbf{x}}_{ta}$.
TimeGAN	GAN	Synthesize $\hat{\mathbf{x}}_{1:N}$ from scratch.
GT-GAN	GAN	Synthesize $\hat{\mathbf{x}}_{1:N}$ from scratch.
TSGM	SGM	Synthesize $\hat{\mathbf{x}}_{1:N}$ from scratch.

TimeGrad generates future observations given the hidden representation of past observations $\mathbf{h}_{n-1}$, i.e., a typical forecasting problem. Since our task is to synthesize from scratch, past known observations are not available. Thus, TimeGrad cannot be directly applied to our task.

In CSDI, there are no fixed temporal dependencies between $\mathbf{x}^0_{co}$ and $\mathbf{x}^0_{ta}$ since its task is to impute missing values, i.e., $\mathbf{x}^0_{ta}$, from known values, i.e., $\mathbf{x}^0_{co}$, in the matrix $\mathbf{x}^0$. It is not necessary that $\mathbf{x}^0_{co}$ precedes $\mathbf{x}^0_{ta}$ in time, according to the CSDI's method design. Our synthesis task can be considered as $\mathbf{x}^0_{co} = \emptyset$, which is the most extreme case of the CSDI's task. Therefore, it is not suitable to be used for our task.

To our knowledge, we are the first proposing an SGM-based time-series synthesis method. We propose to train a conditional score network by using the denoising score matching loss proposed by us, which is denoted as $L^{\mathcal{H}}_{score}$. Unlike other methods (Rasul et al., 2021; Tashiro et al., 2021) that resort to existing known proofs, we design our denoising score matching loss in Eq. equation 11 and prove its correctness. Meanwhile, TimeGrad and CSDI can be somehow modified for time-series synthesis but their generation quality is mediocre (see Appendix C).

C Experimental Results for Inapplicability of Existing Time-Series Diffusion Models to Our Work

In this section, we provide experimental results to show inapplicability of the existing time-series diffusion models, TimeGrad and CSDI, to the time-series generation task.

Table 6: Comparison between TSGM and modified TimeGrad in Energy for its regular time-series setting

Method	Disc.	Pred.
TSGM-VP	.221±.025	.257±.000
TSGM-subVP	.198±.025	.252±.000
Modified TimeGrad	.500±.000	.287±.003

Table 7: Comparison between TSGM and modified CSDI in Stock, Air, Energy and AI4I for its regular time-series setting

Method	Stock		Air		Energy		AI4I	
	Disc.	Pred.	Disc.	Pred.	Disc.	Pred.	Disc.	Pred.
TSGM-VP	.022±.005	**.037±.000**	**.122±.014**	**.005±.000**	**.147±.005**	**.217±.000**	.221±.025	.257±.000
TSGM-subVP	**.021±.008**	**.037±.000**	.127±.010	**.005±.000**	.150±.010	**.217±.000**	**.198±.025**	.252±.000
Modified CSDI	.379±.008	.045±.001	.437±.144	.040±.001	.427±.081	.217±.000	.500±.000	**.251±.000**

C.1 Adapting TimeGrad toward Generation Task

In this section, TimeGrad (Rasul et al., 2021) is modified for the generation task. We simply add an artificial zero vector $\mathbf{0}$ in front of the all time-series samples of Energy. Therefore, TimeGrad's task becomes given a zero vector, forecasting (or generating) all other remaining observations. For the stochastic nature of its forecasting process, it can somehow generate various next observations given the sample input $\mathbf{0}$. Table 6 shows the experimental comparison between modified TimeGrad and TSGM in Energy for its regular time-series setting. TSGM gives outstanding performance, compared to modified TimeGrad. When checked in Table 2, modified TimeGrad is even worse than some baselines. Therefore, unlike TSGM, TimeGrad is not appropriate for the generation task.

C.2 Adapting CSDI toward Generation Task

In this section, we apply CSDI to the unconditional time-series generation task by regarding all observations as missing values (i.e., $\mathbf{x}_{co}^0 = \mathbf{0}$) varying i) its kernel size from 1 to 7 and ii) the number of diffusion steps from 50 to 250. However, as demonstrated in Table 7, CSDI fails to generate reliable time series samples in the datasets for its regular time series setting. In particular, TSGM with 250 steps in the ablation study section significantly outperforms it. Hence, we conclude that CSDI's unconditional generation is unsuitable for the time-series generation task.

C.3 Analysis on Experimental Results

Until now, we have demonstrated the inferior results of forecasting and imputation on time-series generation. As shown by the results, these two models achieve relatively good predictive score but poor discriminative score, indicating a lack of diversity. This is because the primary goal of imputation and forecasting is to generate precise values that closely match the ground truth, as we discussed in Section B. For example, CSDI takes its imputed time-series by averaging synthesized samples 100 times. Therefore, as the quality of forecasting and imputation improves, the diversity of the generated samples decreases, which means they are not suitable for time-series generation.

D Detailed Training Procedure

We train the conditional score network and the encoder-decoder pair alternately after the pre-training step. For some datasets, we found that training only the conditional score network achieves better results after pre-training the autoencoder. Therefore, $use_{alt} = \{True, False\}$ is a hyperparameter to set whether we use the alternating training method. We give the detailed training procedure in Algorithm 1.

Algorithm 1: Training algorithm

Input: $\mathbf{x}^0_{1:N}$; use_{alt} is a Boolean parameter to set whether to use the alternating training method; $iter_{pre}$ is the number of iterations for pre-training; $iter_{main}$ is the number of iterations for training.

```
for iter ∈ {1, ..., iter_pre} do
    Train Encoder and Decoder by using L_ed
end
for iter ∈ {1, ..., iter_main} do
    Train M_θ by using L^H_score
    if use_alt then
        Train the Encoder and Decoder by using L_ed
    end
end
return Encoder, Decoder, M_θ
```

E Datasets and Baselines

We use 4 datasets from various fields as follows. We summarize their data dimensions, the number of training samples, and their time-series lengths (window sizes) in Table 8.

- *Stock* (Yoon et al., 2019): The Google stock dataset was collected irregularly from 2004 to 2019. Each observation has (volume, high, low, opening, closing, adjusted closing prices), and these features are correlated.
- *Energy* (Candanedo et al., 2017): This dataset is from the UCI machine learning repository for predicting the energy use of appliances from highly correlated variables such as house temperature and humidity conditions.
- *Air* (De Vito et al., 2008): The UCI Air Quality dataset was collected from 2004 to 2005. Hourly averaged air quality records are gathered using gas sensor devices in an Italian city.
- *AI4I* (Matzka, 2020): AI4I means the UCI AI4I 2020 Predictive Maintenance dataset. This data reflects the industrial predictive maintenance scenario with correlated features including several physical quantities.

We use several types of generative methods for time-series as baselines. At first, we consider autoregressive generative methods: T-Forcing (teacher forcing) (Graves, 2013; Sutskever et al., 2011) and P-Forcing (professor forcing) (Goyal et al., 2016). Next, we use GAN-based methods: TimeGAN (Yoon et al., 2019), RCGAN (Esteban et al., 2017), C-RNN-GAN (Mogren, 2016), COT-GAN (Xu et al., 2020), GT-GAN (Jeon et al., 2022). We also test VAE-based methods into our baselines: TimeVAE (Desai et al., 2021). Finally, we treat flow-based methods. Among the array of flow-based models designed for time series generation, we have chosen to compare our TSGM against CTFP (Deng et al., 2020). This choice is informed by the fact that CTFP possesses the capability to handle both regular and irregular time series samples, aligning well with the nature of our task which involves generating both regular and irregular time series data.

Table 8: Characteristics of the datasets we use for our experiments

Dataset	Dimension	#Samples	Length
Stocks	6	3685	24
Energy	28	19735	
Air	13	9357	
AI4I	5	10000	

F Hyperparameters and its Search Space

Table 9 shows the best hyperparameters for our conditional score network M_θ on regular and irregular time-series, and we explain its neural network architecture in Appendix J.2. M_θ has various hyperparameters and for key hyperparameters, we set them as listed in Table 9. For other common

Table 9: The best hyperparameter setting for TSGM on regular and irregular time-series . D_{hidden} denotes the hidden dimension of GRU-ODE-decoder.

Dataset	Regular Settings				Irregular Settings				
	$\dim(\mathbf{h})$	use_{alt}	$iter_{pre}$	$iter_{main}$	D_{hidden}	$\dim(\mathbf{h})$	use_{alt}	$iter_{pre}$	$iter_{main}$
Stocks	24	True	50000	40000	48	24	True	50000	40000
Energy	56	False	100000		112	56	False		
Air	40	True	50000		40	40	True		
AI4I	24	True	50000		48	24	True		

hyperparameters with baselines, we reuse the default configurations of TimeGAN (Yoon et al., 2019) and VPSDE (Song et al., 2021) to conduct the regular time-series generation.

We give our search space for the hyperparameters of TSGM. $iter_{pre}$ is in {50000,100000}. The dimension of hidden features, d_{hidden}, ranges from 2 times to 5 times the dimension of input features. On regular time-series generation, we follow the default values in TimeGAN (Yoon et al., 2019) and VPSDE (Song et al., 2021). For irregular time-series tasks, we search the hidden dimension of decoder from 2 times to 4 times the dimension of input dimension, and follow GTGAN (Jeon et al., 2022) for other settings of NCDE-encoder and GRU-ODE-decoder.

For baselines, we check their hyperparameters as follow:

- T-forcing (Graves, 2013): We control batch size among {256, 512, 1024}.
- P-forcing (Goyal et al., 2016): We control batch size among {256, 512, 1024}.
- TimeGAN (Yoon et al., 2019): The dimension of hidden features range from 2 times to 4 times the dimension of input features.
- RCGAN (Esteban et al., 2017): We control learning rate of generator's optimizer and discriminator's optimizer from {1e-4, 2e-4} and {1e-3, 5e-3}, respectively.
- C-RNN-GAN (Mogren, 2016): We control learning rate of generator's optimizer and discriminator's optimizer from {1e-4, 2e-4} and {3e-4, 4e-4}, respectively. We also use label smoothing which is stated in the paper.
- TimeVAE (Desai et al., 2021): We control its latent dimension among {5, 10, 20}.
- COT-GAN (Xu et al., 2020): We calculate score every 250 epoch during 1000 epochs and get the best experimental results.
- GT-GAN (Jeon et al., 2022): For encoder-decoder pair, we test from exactly the same search space as TSGM. We calculate score every 5000 iteration during 40000 iterations and get the best score.

Especially for COT-GAN, since it is on video generation, modifying the architecture to one dimensional form was difficult. So, we augment our time-series data into two dimensional ones by stacking them. After generating two-dimensional data, we extract the first row of the synthesized one and calculate the score. We search every hyperparameter from {0.5, 1, 2} times of default value. Through the experiment, we acquire compatible result but lower than TimeGAN in several datasets.

We follow default values for miscellaneous settings which are not explained on the above. Additionally, to deal with irregular time-series, we search the hyperparameters of GRU-D, which substitutes for RNN or are added to the head of baselines. We test the hidden dimension of GRU-D from 2 times to 4 times the dimension of input features.

G Miscellaneous Experimental Environments

We give detailed experimental environments. The following software and hardware environments were used for all experiments: Ubuntu 18.04 LTS, Python 3.9.12, CUDA 9.1, NVIDIA Driver 470.141, i9 CPU, and GeForce RTX 2080 Ti.

In the experiments, we report only the VP and subVP-based TSGM and exclude the VE-based one for its lower performance. For baselines, we reuse their released source codes in their official repositories and rely on their designed training and model selection procedures. For our method, we

Table 10: The memory usage of TimeGAN, GTGAN, and TSGM for training and the sampling time of them for generating 100 samples on each dataset. Note that The original score-based model (Song et al., 2021) requires 3,214 seconds for sampling 1000 CIFAR-10 images while StyleGAN (Karras et al., 2019) needs 0.4 seconds, which is similar to the case between TSGM and TimeGAN.

Method	Memory Usage (GiB)		Inference Time (s)	
	Stock	Energy	Stock	Energy
TimeGAN	1.1	1.6	0.43	0.47
GTGAN	2.3	2.3	0.43	0.47
TSGM	1.9	1.9	86.32	85.89

select the best model for every 5000 iterations. For this, we synthesize samples and calculate the mean and standard deviation scores of the discriminative and predictive scores. Furthermore, TSGM requires 100 sampling steps, which is relatively lower compared to other diffusion-based models—CSDI (Tashiro et al., 2021) requires 50 steps but averages samples over 100 iterations, effectively requiring 5000 steps.

H Empirical Space and Time Complexity Analyses

We report the memory usage during training and the wall-clock time for generating 100 time-series samples in Table 10. We compare TSGM to TimeGAN (Yoon et al., 2019) and GTGAN (Jeon et al., 2022). Our method is relatively slower than TimeGAN and GTGAN, which is a fundamental drawback of all SGMs. For example, the original score-based model (Song et al., 2021) requires 3,214 seconds for sampling 1,000 CIFAR-10 images while StyleGAN (Karras et al., 2019) needs 0.4 seconds. However, we also emphasize that this problem can be relieved by using the techniques suggested in (Xiao et al., 2022; Jolicoeur-Martineau et al., 2021) as we mentioned in the conclusion section.

I Encoder and Decoder for Irregular Time-series

To process irregular time-series, one can use continuous-time methods for constructing the encoder and the decoder. In our case, we use neural controlled differential equations (NCDEs) for designing the encoder and GRU-ODEs for designing the decoder, respectively (Kidger et al., 2020; Brouwer et al., 2019). Our encoder based on NCDEs can be defined as follows:

$$\mathbf{h}(t_n) = \mathbf{h}(t_{n-1}) + \int_{t_{n-1}}^{t_n} f(t, \mathbf{h}(t); \theta_f) \frac{dX(t)}{dt} dt, \tag{24}$$

where $X(t)$ is an interpolated continuous path from $\mathbf{x}_{1:N}$ — NCDEs typically use the natural cubic spline algorithm to define $X(t)$, which is twice differentiable and therefore, there is not any problem to be used for forward inference and backward training. In other words, NCDEs evolve the hidden state $\mathbf{h}(t)$ by solving the above Riemann-Stieltjes integral.

For the decoder, one can use the following GRU-ODE-based definition:

$$\overline{\mathbf{d}}(t_n) = \mathbf{d}(t_{n-1}) + \int_{t_{n-1}}^{t_n} g(t, \mathbf{d}(t); \theta_g) dt, \qquad \mathbf{d}(t_n) = \text{GRU}(\mathbf{h}(t_n), \overline{\mathbf{d}}(t_n)), \qquad \hat{\mathbf{x}}_n = FC(\mathbf{d}(t_n)), \tag{25}$$

where FC denotes a fully-connected layer-based output layer. The intermediate hidden representation $\overline{\mathbf{d}}(t_n)$ is jumped into the hidden representation $\mathbf{d}(t_n)$ by the GRU-based jump layer. At the end, there is an output layer.

For our irregular time-series experiments, i.e, dropping 30%, 50%, and 70% of observations from regular time-series, we use the above encoder and decoder definitions and have good results.

J NEURAL NETWORK ARCHITECTURE

J.1 ARCHITECTURAL DETAILS OF NCDES AND GRU-ODES

As mentioned in Appendix I, we take the following architecture for functions f, g of (24) and (25) in Table 11.

Table 11: Architecture of functions f(upper) and g(lower). Each layer of encoder and gate of decoder takes $(\sigma \circ \text{Linear})$ form where σ denotes activation function. We describe which activation and Linear function are used.

Layer	Activation function	Linear
1	ReLU	dim(**x**)$\rightarrow$ 4 dim(**x**)
2	ReLU	4 dim(**x**)$\rightarrow$ 4 dim(**x**)
3	ReLU	4 dim(**x**)$\rightarrow$ 4 dim(**x**)
4	Tanh	4 dim(**x**)$\rightarrow$ dim(**x**)

Layer	Gate	Activation function	Linear
1	r_t	ReLU	dim(**h**)$\rightarrow$ dim(**h**)
	z_t	ReLU	
	u_t	Tanh	

J.2 CONDITIONAL SCORE NETWORK

Unlike other generation tasks, e.g., image generation (Song et al., 2021) and tabular data synthesis (Kim et al., 2022), where each sample is independent, time-series observations are dependent to their past observations. Therefore, the score network for time-series generation must be designed to learn the conditional log-likelihood given past generated observations, which is more complicated than that in image generation.

In order to learn the conditional log-likelihood, we modify the popular U-net (Ronneberger et al., 2015) architecture for our purposes. Since U-net has achieved various excellent results for other generative tasks (Song and Ermon, 2019; Song et al., 2021), we modify its 2-dimensional convolution layers to 1-dimensional ones for handling time-series observations. The modified U-net, denoted M_θ, is trained to learn our conditional score function (cf. Eq. equation 11). More details on training and sampling with M_θ are in Sec. 3.4.

K ADDITIONAL EXPERIMENTAL RESULTS

We give additional experimental results for irregular time-series generation with 50% and 70% missing rates in Table 12.

L ADDITIONAL VISUALIZATIONS

In this section, we provide additional visualization results in each dataset. Figure 5 illustrates the density function of each feature estimated by KDE in original and generated data. Figure 6 shows original and generated data points projected onto a latent space using t-SNE (van der Maaten and Hinton, 2008)

M EFFICACY OF OUR RECURSIVE GENERATION

In this section, we investigate the efficacy of our proposed recursive design. We compare TSGM to an method using *one-shot generation*. we call *one-shot generation* when a generation method generates all time-series observations at once, not recursively. In other words, $D \times N$ matrices, where D means the number of features and N means the sequence length, are synthesized at once. CSDI (Tashiro et al., 2021) is one of the most famous one-shot imputation model for time-series.

Table 12: The left and right ones denote experimental results on irregular time-series with 30% and 50% missing rates, respectively. Note that except for our method (TSGM), CTFP, and GT-GAN, the other methods cannot deal with irregular time series, so we make it possible for them to operate on irregular settings by replacing RNN encoder with GRU-D.

	Method	Irregular Settings (Missing Rate: 50%)				Irregular Settings (Missing Rate: 70%)			
		Stocks	Energy	Air	AI4I	Stocks	Energy	Air	AI4I
Disc. score	TSGM-VP	.051±.014	.398±.003	.272±.012	.156±.106	.065±.010	.482±.003	.337±.025	.327±.104
	TSGM-subVP	**.031±.012**	.421±.008	**.213±.025**	**.137±.102**	**.035±.009**	**.213±.025**	**.329±.027**	**.235±.123**
	T-Forcing	.407±.034	.376±.046	.499±.001	.473±.045	.404±.068	.336±.032	.499±.001	.493±.010
	P-Forcing	.500±.000	.500±.000	.494±.012	.437±.079	.449±.150	.494±.011	.498±.002	.440±.125
	TimeGAN	.477±.021	.473±.015	.500±.001	.500±.000	.485±.022	.500±.000	.500±.000	.500±.000
	RCGAN	.500±.000	.500±.000	.500±.000	.500±.000	.500±.000	.500±.000	.500±.000	.500±.000
	C-RNN-GAN	.500±.000	.500±.000	.500±.000	.450±.150	.500±.000	.500±.000	.500±.000	.500±.000
	TimeVAE	.411±.110	.436±.088	.423±.153	.389±.113	.444±.148	.498±.003	.426±.148	.371±.092
	COT-GAN	.499±.001	.500±.000	.500±.000	.500±.000	.498±.001	.500±.000	.500±.000	.500±.000
	CTFP	.499±.000	.500±.000	.500±.000	.499±.001	.500±.000	.500±.000	.500±.000	.499±.000
	GT-GAN	.265±.073	**.317±.010**	.434±.035	.276±.033	.230±.053	.325±.047	.444±.019	.362±.043
Pred. score	TSGM-VP	**.011±.000**	**.051±.001**	**.041±.001**	**.060±.001**	**.011±.000**	.053±.001	.043±.000	.092±.024
	TSGM-subVP	**.011±.000**	**.051±.001**	.042±.002	.065±.013	.012±.000	**.042±.002**	**.042±.001**	**.097±.020**
	T-Forcing	.038±.003	.090±.000	.121±.003	.143±.005	.031±.002	.091±.000	.116±.003	.144±.004
	P-Forcing	.089±.010	.198±.005	.101±.003	.116±.007	.107±.009	.193±.006	.107±.002	.125±.007
	TimeGAN	.254±.047	.339±.029	.325±.005	.251±.010	.228±.000	.443±.000	.425±.008	.323±.011
	RCGAN	.333±.044	.250±.010	.335±.023	.276±.066	.441±.045	.349±.027	.359±.008	.346±.029
	C-RNN-GAN	.273±.000	.438±.000	.289±.033	.373±.037	.281±.019	.436±.000	.306±.040	.262±.053
	TimeVAE	.195±.012	.143±.007	.103±.002	.144±.004	.199±.009	.134±.004	.108±.004	.142±.008
	COT-GAN	.246±.000	.475±.000	.557±.000	.449±.000	.278±.000	.456±.000	.556±.000	.435±.000
	CTFP	.084±.005	.469±.008	.476±.235	.412±.024	.084±.005	.469±.008	.476±.235	.412±.024
	GT-GAN	.018±.002	.064±.001	.061±.003	.113±.024	.020±.005	.076±.001	.059±.004	.124±.003
	Original	.011±.002	.045±.001	.044±.006	.059±.001	.011±.002	.045±.001	.044±.006	.059±.001

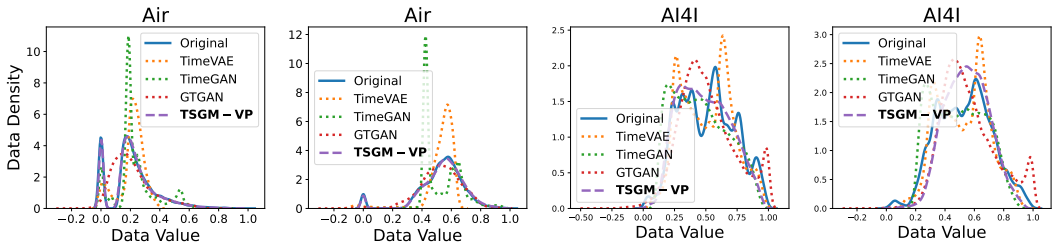

Figure 5: Additional KDE plots for each feature in Air and AI4I datasets.

We convert our TSGM for the one-shot generation by removing the RNN-based encoder. In Table 13, TSGM-oneshot shows poor generation quality in Stock and Energy. TSGM-oneshot achieves comparable predictive scores but its discriminative score gets worse a lot. From these results, we can support the efficacy of our recursive structures, compared to one-shot generation. One can also check the one-shot generation result by CSDI in Appendix C.2.

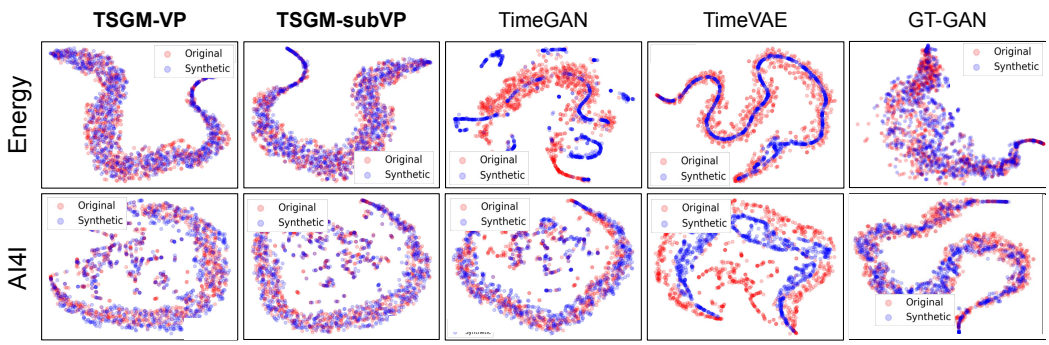

Figure 6: Additional t-SNE plots in Energy and AI4I datasets.

Table 13: Comparison between TSGM and one-shot generations. We give representative results. For other datasets, the results are similar or worse than the table.

Method	Stock		Energy	
	Disc.	Pred.	Disc.	Pred.
TSGM-VP	.022±.005	.037±.000	.221±.025	.257±.000
TSGM-subVP	.021±.008	.037±.000	.198±.025	.252±.000
TSGM-oneshot	.029±.018	.037±.000	.494±.001	.258±.000

Table 14: Comparison of drift and diffusion terms. $\sigma(s)$ means positive noise values which are increasing, and $\beta(s)$ denotes noise values in [0,1], which are used in Song and Ermon (2019); Ho et al. (2020).

SDE	drift ($\mathbf{f}$)	diffusion (g)
VE	0	$\sqrt{\frac{d\sigma^2(s)}{ds}}$
VP	$-\frac{1}{2}\beta(s)\mathbf{x}^s$	$\sqrt{\beta(s)}$
subVP	$-\frac{1}{2}\beta(s)\mathbf{x}^s$	$\sqrt{\beta(s)(1-e^{-2\int_0^s \beta(t)dt})}$

N Drift and diffusion terms in VE, VP and subVP SDE

In this section, we describe the detailed form of each SDE. In (Song et al., 2021), the authors investigated that SMLD (Song and Ermon, 2019) and DDPM (Ho et al., 2020) can be extended to continuous forms and as a result, suggested VE and VP SDEs. Furthermore, the author proposed an additional SDE form, called subVP SDE, which has a smaller variance than VP SDE but the same expectation. The exact calculation is not a main subject of this paper, so we only explain the form of these terms in Table 14. Please refer to (Song et al., 2021) for the detailed computation.

Along with already mentioned notations in Section 2.1, we define noise scales. $\sigma(s)$ means positive noise values which are increasing, and $\beta(s)$ denotes noise values in [0,1] which are used in SMLD and DDPM. Although we give the exact form of the three SDEs, we report only the VP and subVP-based TSGM in our experiments and exclude the VE-based one for its lower performance.

O Detailed Description of GRU-D

GRU-D (Che et al., 2016) is a modified GRU model which is for learning time-series data with missing values. This concept is similar with our problem statement, so we apply it to our baseline for irregular case. GRU-D needs to learn decaying rates along with the values of GRU. First, GRU-D learns decay rates which depict vagueness of data as time passed. After calculating the decay rates, each value is composed of decay rate, mask, latest observed data, and predicted empirical mean that of GRU. The code can be utilized in the following link: https://github.com/zhiyongc/GRU-D