Time Series Imputation by Conditional Diffusion Models

Zhenghui Chen
HKUST
zchenef@connect.ust.hk

Abstract

Missing values in time series data is a common problem that should be addressed in many practical applications. Various models have been designed for time series imputation with different missingness scenarios. Conditional diffusion models for time series imputation have recently achieved state-of-the-art performance on various benchmark datasets. Here, we focus on two conditional diffusion models including structured state space diffusion (SSSD) model and conditional score-based diffusion models for imputation (CSDI). We implemented the TensorFlow code of these two models based on the open-sourced PyTorch code in the original paper. We tried our best to reproduce the results using the same dataset and closest training config. We also applied two models on the stock dataset and compared the model performance.

1. Introduction

In real-world applications such as healthcare, geoscience and finance, time series data may contain many missing values for different reasons, which will affect the interpretation of the data in downstream analysis tasks. Various deep learning models and modern architectures such as recurrent neural networks (RNNs) [1], variational autoencoders (VAEs) [2], generative adversarial networks (GANs) [3] and Transformer [4] have been applied for time series imputation.

For RNN-based work, different variants such as GRU-D [5], M-RNN [6] and BRITS [7] use single or bidirectional RNNs to estimate missing values in the time series data. Due to the recurrent network structure, these models have many limitations including memory constraints, costly training time, and they cannot handle long-term dependency. For VAE-based and GAN-based work, the generative models such as GP-VAE [2] and GRUI-GAN [8] have unstable training and suffer from mode collapse. For self-attention-based work, different variants of Transformer architecture such as NRTSI [9] and SAITS [10] use self-attention layers to explore long-range dependencies in time series data.

Recently, the diffusion models have been developed for image generation [11] and audio synthesis [12], which achieved state-of-the-art performance among deep generative models. Variants of conditional diffusion model [13][14] with self-supervised training have been designed for time series imputation tasks and outperformed the conventional deep generative models. Conditional score-based diffusion models for imputation (CSDI) [13] and structured state space diffusion (SSSD) model [14] are both adopted the base architecture of DiffWave [12]. CSDI model replaces the convolution layer with temporal and feature Transformer encoder layer using self-attention mechanism [4] in each residual layer. SSSD model uses the structured state space sequence (S4) layers [15] to build the residual blocks, which can be denoted as SSSD^{S4}. The S4 layer combines linear state space models (SSMs) and the HiPPO framework to capture long-range dependencies for time-series imputation.

Here, we implemented TensorFlow code of CSDI and SSSD^{S4} model, validated the imputation performance on several benchmark datasets with various missingness scenarios. We also compared the performance of these two models with two different data-missing scenarios on three major stock datasets from the current Dow Jones 30, EuroStoxx50, and Hang Seng index, respectively. Finally, we utilized the SSSD^{S4} model to forecast the price change of two major stocks in the Hang Seng index using the historical market data from the Dow Jones stocks.

2. Background

2.1. Time series imputation

For benchmark datasets, we denote the time series data $x \in \mathbb{R}^{L \times K}$, where L represents the length of the time series, and K represents the number of features. We denote the observed masks as $m_{obs} \in [0,1]^{L \times K}$, which label the missing data with 0 and valid data with 1 in the raw time series. The values of observed masks for benchmark datasets are all one because no missing data exists. We denote the binary imputation masks or conditional masks as $m_{con} \in [0,1]^{L \times K}$, which label the missing values to be imputed with 0 and conditional values with 1 in the time

series data to simulate the missingness scenarios for model training and testing. We denote the loss masks as $m_{loss} \in [0,1]^{L\times K}$, which are one minus the conditional masks m_{con}. The loss masks are used to select the imputed value generated from the missing value by the model and calculate the imputation loss with the target value.

The different data-missing scenarios in the original SSSD paper include *random missing (RM), missing not at random (MNR),* and *blackout missing (BM). Random missing (RM)* sampled the missing index across the length randomly for each feature. *Missing not at random (MNR)* sampled the missing subset across the length for each feature. *Blackout missing (BM)* sampled the same missing subset across the length for all features.

For stock datasets, the raw time series data $x \in \mathbb{R}^{L\times K}$ already has the missing values. For the observed mask $m_{obs} \in [-1,0,1]^{L\times K}$, missing data includes holiday data and nan data are labeled with -1 and 0, valid data are labeled with 1. Two masking strategies are implemented for model training and testing, including *random missing with length* and *blackout missing with length*, which are only sampled from the index of the valid mask using RM and BM, respectively. The constructed masks label the missing indexes of the raw observed mask with 2 using the above masking strategies. The binary conditional masks m_{con} are generated from the constructed masks by setting label -1 and label 2 as label 0. The loss masks m_{loss} are generated from the constructed masks by setting label 2 as label 1 and other labels as label 0, because only the constructed missing indexes have the target value.

2.2. Diffusion models

Diffusion probabilistic models [16] are latent variable models inspired by the non-equilibrium statistical physics. Diffusion models convert the data distribution by gradually adding the Gaussian noise in the forward process and restore the data by sequentially removing the noise in the reverse process.

The forward process is defined as the following Markov chain:

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

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t\mathbf{I}) \quad (2)$$

where and the variances β_t represent the noise level for timestep t, which usually are constant hyperparameters. The closed form of the sampling can be expressed as:

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\overline{\alpha_t}}x_0, (1-\overline{\alpha_t})\mathbf{I}) \quad (3)$$

where $\alpha_t = 1-\beta_t$ and $\overline{\alpha_t} = \prod_{i=1}^{t}\alpha_i$. Therefore, x_t can be written as $x_t = \sqrt{\overline{\alpha_t}}x_0 + (1-\overline{\alpha_t})\epsilon$ where $\epsilon \sim \mathcal{N}(0,\mathbf{I})$.

The reverse process is defined as the following Markov chain:

$$p_\theta(x_{0:T}) = p(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) \quad (4)$$

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t,t), \sigma_\theta(x_t,t)\mathbf{I}) \quad (5)$$

where $x_T \sim \mathcal{N}(0,\mathbf{I})$. Ho et al. [11] used the specific parameterization of $p_\theta(x_{t-1}|x_t)$ and trained the reverse process by optimizing the following objective:

$$\mathcal{L} = \min_\theta \mathbb{E}_{x_0\sim q(x_0),\epsilon\sim\mathcal{N}(0,\mathbf{I}),t\sim\mathcal{U}(1,T)} \|\epsilon - \epsilon_\theta(x_t,t)\|_2^2 \quad (6)$$

where $\epsilon_\theta(x_t,t)$ can be the trainable neural network for unconditional diffusion process, e.g., denoising diffusion probabilistic models (DDPM).

The conditional diffusion models such as CSDI [13] and SSSD^{S4} [14] are the extension of the above unconditional diffusion model, which can exploit useful conditioning information for accurate imputation. For CSDI model, $\epsilon_\theta = \epsilon_\theta(x_t^{tar}, t|x_0^{con})$, where x_0^{con} denotes conditional observations, x_0^{tar} denotes imputation targets, and noisy target $x_t^{tar} = \sqrt{\overline{\alpha_t}}x_0^{tar} + (1-\overline{\alpha_t})\epsilon$. For SSSD^{S4} model, $\epsilon_\theta = \epsilon_\theta(x_t, t, c)$, where $c = \mathrm{Concat}(x_0 \odot m_{con}, m_{con})$, $\odot$ denotes point-wise multiplication, m_{con} denotes the conditional masks (imputation masks). The loss function is only targeted for the constructed missing values in the imputation mask.

2.3. Evaluation metrics

We utilized different metrics to evaluate the imputation performance of the models, including mean absolute error (MAE), mean squared error (MSE), root mean square error (RMSE), and continuous ranked probability score (CRPS).

$$MAE = \frac{1}{n}\sum_{t=1}^{n}\sum_{k=1}^{K}|(y-\hat{y})\odot m_{loss}|_{t,k} \quad (7)$$

$$MSE = \frac{1}{n}\sum_{t=1}^{n}\sum_{k=1}^{K}((y-\hat{y})\odot m_{loss})_{t,k}^2 \quad (8)$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}\sum_{k=1}^{K}((y-\hat{y})\odot m_{loss})_{t,k}^2} \quad (9)$$

The total absolute errors and total squared errors are calculated between the target values y and imputed values $\hat{y}$ for MAE (7) and MSE (8), respectively, where only the constructed missing values in the loss masks m_{loss} are considered and divided by the total number of missing values n. RMSE (9) is the square root of the MSE (8).

The continuous ranked probability score (CRPS) measures the compatibility of a probability distribution F with an observation y, and can be defined as the integral of the quantile loss $\Lambda_\alpha(q,y) := (\alpha - \mathbb{I}_{y<q})(y-q)$ for all thresholds $\alpha \in [0,1]$:

$$CRPS(F^{-1}(\alpha), y) = \int_0^1 2\Lambda_\alpha(F^{-1}(\alpha), y)\, d\alpha \quad (10)$$

where $\mathbb{I}$ is the indicator function. The distribution F is approximated through several imputations for each test sample. The integral is approximated using the finite sum of discretized quantile values $\alpha \in A := \{0.05, 0.10, \ldots, 0.95\}$. The normalized average of the CRPS across all features and time steps with the reference to the loss masks is expressed as:

$$CRPS = \frac{\sum_{t=1}^{n}\sum_{k=1}^{K} m_{loss\,t,k}\frac{2}{|A|}\sum_{\alpha\in A}\Lambda_\alpha(F^{-1}(\alpha)_{t,k}, y_{t,k})}{\sum_{t=1}^{n}\sum_{k=1}^{K} m_{loss\,t,k}|y_{t,k}|} \quad (11)$$

3. Experiments

3.1. Experimental protocol

For the implemented TensorFlow code of the CSDI and SSSD^{S4} model, we followed the same structure as the open-sourced PyTorch code of two models. Most of the functions and model layers have the same definition of input and output because we used the TensorFlow, Keras, and NumPy API to reproduce all the PyTorch API inside the function. Therefore, the usage of TensorFlow code is same as the original PyTorch code. We followed the default hyper-parameters settings for CSDI and SSSD^{S4} model as shown in the appendix of the SSSD paper [14].

For the validation of TensorFlow code on benchmark datasets, we followed the identical missingness scenarios and ratios in the original paper for model training and evaluation, then compared with the paper results. We also found that the open-source CSDI PyTorch code has errors, which will be illustrated in section 3.3. For experiments on the stock datasets, we used the modified version of the TensorFlow code with some improvements, which will be illustrated in section 3.4.

3.2. Imputation on MuJoCo dataset

We validated the implemented TensorFlow code of the SSSD^{S4} model on MuJoCo dataset with highly sparse RM scenarios. The dataset has 10000 samples with a length of 100 and features of 14, which is further split for training dataset ([8000, 100, 14]) and testing dataset ([2000, 100, 14]) with 80% and 20%, respectively. The training batch is 50 and the testing batch is 500. Therefore, the training data ([160, 50, 100, 14]) and testing data ([4, 500, 100, 14]) can be obtained. The default training iterations are 232000 and 150000 under 70% RM and 90% RM scenarios in the original SSSD paper, respectively. Therefore, we utilized similar iterations of 240000 and 150000 for the TensorFlow model training under 70% RM and 90% RM scenarios, respectively.

Table I shows the imputation MSE results from the original SSSD paper and the implemented TensorFlow model. We successfully reproduced the MSE results under 70% RM and 90% RM, which are comparable with the original paper results. Note that our MSE results are averaged for the imputation on the testing dataset with 1 trial, while the paper provides the averaged MSE results with concise error notation over 3 trials.

Table I
Imputation MSE results for the MuJoCo dataset

SSSD^{S4}	70% RM	90% RM
Paper	0.59(8)e-3	1.90(3)e-3
TensorFlow	0.53e-3	1.67e-3

3.3. Imputation on PTB-XL dataset

We validated the modified CSDI PyTorch code and the implemented CSDI TensorFlow code on PTB-XL electrocardiogram (ECG) data. The dataset has 21837 samples with a length of 1000 and features of 12, which include 17441 training samples, 2193 validation samples and 2203 testing samples. We followed $L = 250$ time steps in the SSSD paper, cropped the data across the time step and merged into the sample size. Finally, we obtained the processed training data ([69764, 250, 12]), validation data ([8772, 250, 12]), and testing data ([8812, 250, 12]). The training batch is 32 and the training epoch is 200 for both PyTorch and TensorFlow code. For the model training of PyTorch code, the training time is 30 hours, the validation is 45 minutes, and the evaluation time with data imputation is 2.5 hours. For the model training of TensorFlow code, the training time is 50 hours, the validation is 76 minutes, and the evaluation time is 3 hours.

During the experiment, we found that the CSDI PyTorch code published by SSSD authors [14] has a major mistake. The default masking setting during the model training is not the 20% RM, 20% MNR, and 20% BM mentioned in the SSSD paper. Although the authors added the masking codes for 20% RM, 20% MNR, and 20% BM to generate the observed mask and the ground truth mask (imputation or conditional mask) in the dataset initialization. However, the ground truth mask is used as the conditional mask only during the model evaluation. We also found the RM code is applied on the flattened data, which is different from the RM code that is applied to the length of each feature in the original SSSD paper and code. For model training, the masking codes in the original CSDI paper [13] are still maintained, which will convert the observed mask into the conditional mask by random strategy (missing ratios between 0 and 1) or historical strategy. Therefore, we modified the original CSDI PyTorch code by adding the RM, MNR, and BM directly in the training module. These masking codes are referenced from the published SSSD code. The same masking strategies are applied for the implemented CSDI TensorFlow code.

Table II shows the MAE, RMSE, and CRPS results from the original paper, the modified PyTorch model and the implemented TensorFlow model. For MAE and RMSE calculation, the median value of 10 samples generated for each test sample was used to compare with the target value. Note that our results are averaged with 1 trial, while the paper results are averaged over 3 trials with concise error notation.

We found that MAE, RMSE, and CRPS results of the PyTorch and TensorFlow code are close under different missingness scenarios. Compared with the paper results on 20% RM, the PyTorch model has lower RMSE, and slightly higher MAE and CRPS, while the TensorFlow model has slightly lower MAE and CRPS, and much lower

RMSE. Compared with the paper results on 20% MNR, the PyTorch and TensorFlow model achieved a huge reduction in MAE and CRPS. However, the RMSE is larger than the paper results. Compared with the paper results on 20% BM, both the PyTorch and TensorFlow model achieved all the reduction in MAE and RMSE, and presented a similar cumulative distribution function (CDF) of the imputations against the targets CDF by smaller CRPS.

We also noticed that the TensorFlow code has a better training loss decrease through the gradient descent compared to the PyTorch code during the mode training. Therefore, most of the imputation errors of TensorFlow code are smaller than the results of PyTorch code under 20% RM and 20% MNR. Overall, the modified PyTorch code and implemented TensorFlow code has better performance than the paper results.

Table II
Imputation for RM, MNR and BM scenarios on the PTB-XL dataset using the modified PyTorch code and the implemented TensorFlow code of CSDI model

CSDI	MAE	RMSE	CRPS
20% RM on PTB-XL			
Paper	0.0038±2e-6	0.0189±5e-5	0.0265±6e-6
PyTorch	0.0041	0.0143	0.0280
TensorFlow	**0.0034**	**0.0115**	**0.0234**
20% MNR on PTB-XL			
Paper	0.0186±1e-5	**0.0435±2e-4**	0.1306±5e-5
PyTorch	0.0124	0.0542	0.0839
TensorFlow	**0.0114**	0.0555	**0.0772**
20% BM on PTB-XL			
Paper	0.1054±4e-5	0.2254±7e-5	0.7468±2e-4
PyTorch	**0.0627**	**0.1742**	**0.4229**
TensorFlow	0.0649	0.1787	0.4477

We also validated the original SSSD^{S4} PyTorch code and the implemented SSSD^{S4} TensorFlow code on PTB-XL dataset. For the training config of the original PyTorch code, we adopted the batch size of 32 and iterations of 150000 in the SSSD paper. The training time is 1 day. For the training config of the TensorFlow code, we didn't use the batch size of 32 as the GPU would be out of memory. To keep similar training conditions, we set the batch size as 16 and the iterations as 300000. The training time is 2 days. To make fair comparison with CSDI model, the imputation results of SSSD^{S4} model are also based on 10 generated samples for each test sample in one trial. Similarly, the median value of 10 generated samples for the same test sample are used to calculate MAE and RMSE. The total imputation time is about 32 hours and 36 hours for PyTorch code and TensorFlow code, respectively.

Table III shows the MAE, RMSE and CRPS results of the SSSD^{S4} model under different missingness scenarios. We observed that the MAE, RMSE, and CRPS results of the original PyTorch code and implemented TensorFlow code are very similar. Compared to the paper results under 20% RM, 20% MNR and 20% BM, we found that the imputation errors of the PyTorch and TensorFlow model are all larger in our experiments. Since the authors haven't released the detailed training config of the PTB-XL dataset in the open-sourced SSSD^{S4} code, it is possible that we have missed some important training configs and tricks to further improve the model performance. Overall, the results under different missingness scenarios are in our expectations, that is the results on 20% RM are much better than those results on 20% MNR, and the results on 20% MNR are much better than those results on 20% BM.

Table III
Imputation for RM, MNR and BM scenarios on the PTB-XL dataset using the original PyTorch code and the implemented TensorFlow code of SSSD^{S4} model

SSSD^{S4}	MAE	RMSE	CRPS
20% RM on PTB-XL			
Paper	**0.0034±4e-6**	**0.0119±1e-4**	**0.0282±1e-3**
PyTorch	0.0041	0.0192	0.0295
TensorFlow	0.0043	0.0179	0.0305
20% MNR on PTB-XL			
Paper	**0.0103±3e-3**	**0.0226±9e-4**	**0.0787±3e-3**
PyTorch	0.0166	0.0628	0.1163
TensorFlow	0.0173	0.0654	0.1200
20% BM on PTB-XL			
Paper	**0.0324±3e-3**	**0.0832±8e-3**	**0.2689±3e-3**
PyTorch	0.0522	0.1408	0.3577
TensorFlow	0.0515	0.1469	0.3545

We compared the imputation performance of the CSDI and SSSD^{S4} model using the TensorFlow results from Table II and Table III, which are shown in Table IV. We found that the SSSD^{S4} model performed better under 20% BM, while the CSDI model performed better under 20% RM and 20% MNR. On the 20% BM scenario, the SSSD^{S4} model performed indeed better than the CSDI model, but not like the impressive results in the SSSD paper, e.g., more than 50% reduction in MAE. Because the imputation results of the CSDI model under 20% BM are much better than the paper results and the SSSD^{S4} results are not as good as the paper results in our experiments.

Table IV
Comparison of the CSDI and SSSD^{S4} model for imputation on the PTB-XL dataset under RM, MNR and BM scenarios.

Model	MAE	RMSE	CRPS
20% RM on PTB-XL			
CSDI	**0.0034**	**0.0115**	**0.0234**
SSSD^{S4}	0.0043	0.0179	0.0305
20% MNR on PTB-XL			
CSDI	**0.0114**	**0.0555**	**0.0772**
SSSD^{S4}	0.0173	0.0654	0.1200
20% BM on PTB-XL			

CSDI	0.0649	0.1787	0.4477
SSSD^{S4}	**0.0515**	**0.1469**	**0.3545**

Figure 1 shows the imputation results on a 20% BM scenario by the CSDI and SSSD^{S4} model for the PTB-XL dataset. The CSDI model can only capture one peak of the two missing peaks and the peak is at the wrong time step with the early appearance. The SSSD^{S4} model can capture all essential signal features, and the two missing peaks are predicted at the right time steps.

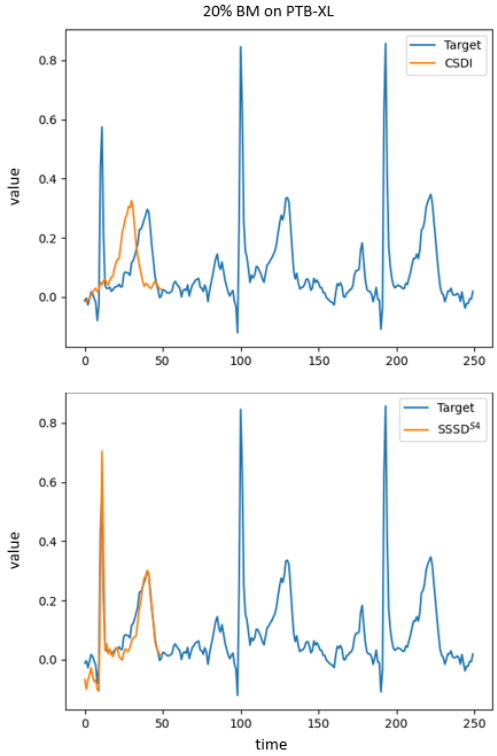


Figure 1. Visualization of PTB-XL BM imputations by CSDI and SSSD^{S4} model.

3.4. Imputation on stock dataset

We compared the performance of the CSDI and SSSD^{S4} model under *random missing with length* and *blackout missing with length* scenarios on three stock datasets. We downloaded the stock data of Hang Seng index, Dow Jones 30, and EuroStoxx50, and filtered the stock data with the trading days less than ten years. For data preprocessing, we labeled the data with the observed mask, normalized the data with the min-max scaler for each feature and discarded the batch with many nan data. The feature of stock data is 6, including opening price, high price, low price, closing price, adjusted closing price, and trading volume. Each dataset is split into training and testing dataset with 80% and 20%, respectively. Hang Seng dataset has 2737 samples with time steps of 103, including training dataset ([2190, 103, 6]) and testing dataset ([547, 103, 6]). Dow Jones dataset has 2604 samples with time steps of 137, including training dataset ([2083, 137, 6]) and testing dataset ([521, 137, 6]). EuroStoxx dataset has 3096 samples with time steps of 94, including training dataset ([2477, 94, 6]) and testing dataset ([619, 94, 6]). We set around 20% percent missing values across the length for different scenarios. The missing number k is 21, 28 and 19 for Hang Seng, Dow Jones and EuroStoxx dataset under *random missing with length* scenario, respectively. The segment k is 5 for all stock datasets under *blackout missing with length* scenario.

The implemented TensorFlow code for the stock dataset has some improvements or changes. For the CSDI model, all the useless masking functions and config codes were removed. The masking code for *random missing with length* and *blackout missing with length* was implemented and can be changed through the config. The training batch is 32 and training epoch is 200. The training time is less than 1.5 hours for all three datasets. For the SSSD^{S4} model, the same masking strategies were implemented and the masks for each batch in the same iteration were generated with different values for better randomness. The training batch is 30, 40, 45 for Hang Seng, Dow Jones and EuroStoxx dataset, respectively. The training iterations are 150000. The training time is about 20 hours for Hang Seng dataset, 23 hours for EuroStoxx dataset, and 55 hours for Dow Jones dataset. The longer training time on the Dow Jones dataset is mainly caused by the low GPU utilization rate, which may be related to the small batch size or CPU bottleneck.

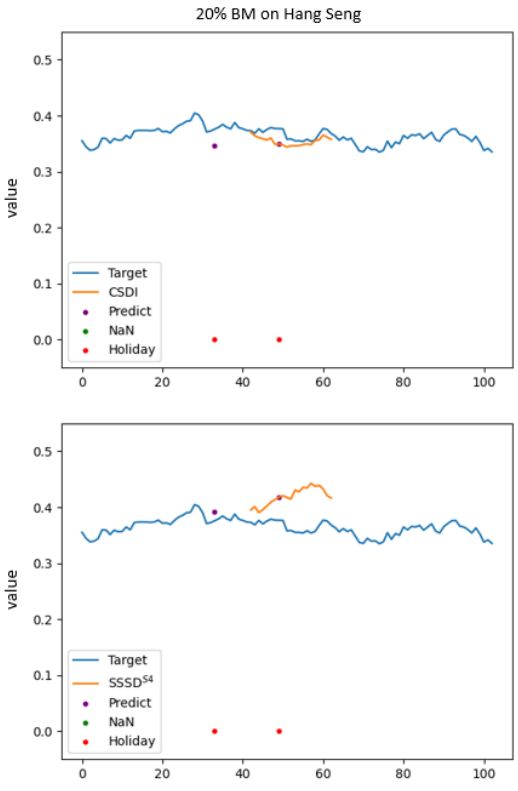
Table V shows the comparison results for the CSDI and SSSD^{S4} model on stock dataset. The imputation results are averaged MAE and RMSE for 3 samples generated for each test sample with one trial. We found that the CSDI model has better performance than the SSSD^{S4} model on all datasets with all missingness scenarios. On the 20% RM scenario, the CSDI model achieved the reduction in MAE of more than 50% and in RMSE of more than 30% on Hang Seng and Dow Jones dataset. For EuroStoxx dataset, the CSDI model achieved the reduction in MAE of more than 65%. On the 20% BM scenario, the CSDI model achieved the reduction in MAE of 40% and in RMSE of 45% on Hang Seng and EuroStoxx dataset. For Dow Jones dataset, the CSDI model achieved the reduction in MAE and RMSE of more than 65%.

Table V
Imputation for RM and BM scenarios on the stock dataset using CSDI and SSSD^{S4} model

Model	MAE	RMSE	MAE	RMSE
	20% RM on Hang Seng		20% BM on Hang Seng	
CSDI	**0.0079**	**0.0170**	**0.0226**	**0.0379**
SSSD^{S4}	0.0162	0.0249	0.0389	0.0607

	20% RM on Dow Jones		20% BM on Dow Jones	
CSDI	**0.0058**	**0.0169**	**0.0139**	**0.0283**
SSSD^{S4}	0.0144	0.0250	0.0441	0.0866
	20% RM on EuroStoxx		20% BM on EuroStoxx	
CSDI	**0.0092**	**0.0247**	**0.0225**	**0.0390**
SSSD^{S4}	0.0274	0.0377	0.0366	0.0597

Figure 2 shows the imputation examples for the stock opening price using CSDI and SSSD^{S4} model for different missingness scenarios and stock datasets. For 20% RM on EuroStoxx dataset, the target values are represented by the blue curve, the imputation values by the model are marked with yellow dots, while the nan data and holiday data are marked with green and red dots, respectively. Although the imputation mask of the CSDI and SSSD^{S4} model is not the same, we can still easily observe that the imputation values of the CSDI model are closer to the target values than those of the SSSD^{S4} model. For 20% BM on Hang Seng and Dow Jones dataset, the imputation values of the missing subset are represented by the orange curves. If the nan and holiday data exist, the corresponding imputation values are marked with purple dots. We noticed that the CSDI model can better capture the trend of the stock price than the SSSD^{S4} model under the BM scenarios.



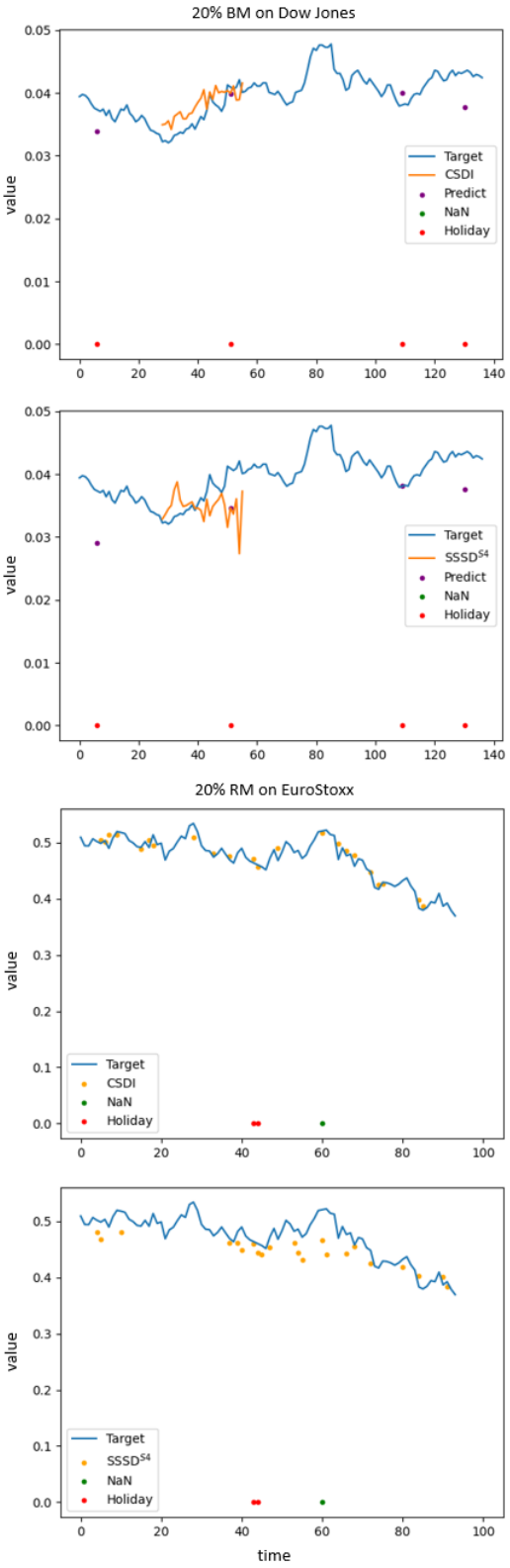


Figure 2. Visualization of the imputation results for the stock opening price by CSDI and SSSD^{S4} model.

3.5. Forecasting on stock dataset

We applied the SSSD^{S4} TensorFlow model to forecast the price change of two Hang Seng stocks including Tencent (0700.HK) and AIA (1299.HK). Tencent dataset has 4359 samples with the time steps of 100 and features of 28, where the features are price difference between high price and low price from 27 Dow Jones stocks and Tencent. The dataset is split for training dataset ([3488, 100, 28]) and testing dataset ([871, 100, 28]) with 80% and 20%, respectively. AIA dataset has 2822 samples with the time steps of 100 and features of 30, where the features are price difference between high price and low price from 29 Dow Jones stocks and AIA. The dataset is split for training dataset ([2258, 100, 30]) and testing dataset ([564, 100, 30]) with 80% and 20%, respectively. Note that the last time step of the Dow Jones stocks is one day before the last time step of the forecast stock. The conditional mask is all one except the last time point of the forecast stock is zero. We processed the dataset with two methods. For the scale dataset, each stock feature is divided by the highest price of that stock. The training iteration is 100000. The training time is about 11 hours for Tencent scale dataset and 12.5 hours for AIA scale dataset, respectively. For the raw dataset, the original price change has been directly used. The training iteration is 150000. The training time is about 16.5 hours for Tencent raw dataset and 18.5 hours for AIA raw dataset, respectively.

Table VI shows the forecasting errors including MAE, RMSE and relative error on Tencent and AIA datasets. The relative error is calculated as the absolute error between the imputation value and target value, further divided by the target value for each individual test sample, and averaged on all test samples. The raw dataset is labeled with R in the bracket and the scale dataset is labeled with S in the bracket, respectively. For the forecasting on the Tencent and AIA raw dataset, the model checkpoints of 100000 and 150000 iterations were used. We can observe that the model performance slightly improved for Tencent raw dataset and slightly dropped for AIA raw dataset with additional 50000 iterations.

As a technology stock, Tencent has large price fluctuation during the historical period, ranging from 0.7 HKD to 775.5 HKD. The scale data is much smaller than the raw data after divided by the highest price. We found that the model has achieved much lower relative error on Tencent raw dataset than corresponding scale dataset. We also noticed that there are negative predictions (e.g., −7.5e-5) for testing data with small values (e.g., 5.8e-5) in Tencent scale dataset. The model may not good at predicting the small value accurately. As the insurance stock, AIA has a more stable stock price range from 19.8 HKD to 109.3 HKD. The model also achieved the lower relative error on AIA raw dataset. However, the model performance on AIA raw dataset and AIA scale dataset didn't have the huge gap as Tencent stock datasets. The relative error of AIA raw dataset is lower than that of Tencent raw dataset, which indicates a higher relative accuracy in forecasting price change of AIA stock than Tencent stock. Overall, the SSSD^{S4} model didn't perform well with the large relative error on this forecasting task. We believe that the model performance can be further improved using more additional data, e.g., applying other stocks in Hang Seng index as the features of the data.

Table VI
Forecasting on stock dataset using the SSSD^{S4} model.

Dataset	Checkpoint	MAE	RMSE	Relative error
Tencent(S)	100,000	2.32e-3	4.75e-3	95.2%
Tencent(R)	100,000	1.77	3.94	38.3%
	150,000	**1.73**	**3.86**	**36.7%**
AIA(S)	100,000	4.25e-3	6.30e-3	42.6%
AIA(R)	100,000	**0.371**	0.611	**30.0%**
	150,000	0.374	**0.610**	30.5%

4. Discussion

For the imputation on the large PTB-XL dataset, the SSSD^{S4} model has better performance than the CSDI model only on 20% BM scenario. For the imputation on the small stock dataset, the SSSD^{S4} model performed much worse than the CSDI model on both RM and BM scenarios. Compared to the PTB-XL ECG data with repeated signals and feature patterns, the irregular stock data usually have many unexpected fluctuations caused by different reasons such as the investors, market, and emergency. We believed the large SSSD^{S4} model with 36 residual layers may not learn the stock features well using the small stock dataset for model training, while the small CSDI model with only 4 residual layers can be trained and generalized well for the stock dataset. Due to the time limitation, we admitted that there are still lots of ablation study can be investigated for a fair comparison of the CSDI and SSSD^{S4} model. Many hyper-parameters of these two models are different in the original SSSD paper, for example, numbers of residual layers, training iterations, learning rate, diffusion steps and schedule.

Since we have utilized the TensorFlow and PyTorch code during the experiment, we would like to share some thoughts about these two deep learning frameworks. The dynamic graph mechanism of TensorFlow 2 is similar as the PyTorch, which make it easier to build and debug models. But there are still some features that we cannot get used to, e.g., assigning the value to tensors or changing the values of the certain index in the tensors are not convenient in TensorFlow 2, while the PyTorch is more NumPy style and supports in-place operations for tensors. The APIs of

the TensorFlow 2 are kind of messy for add-on libraries and different versions, unlike PyTorch APIs are stable and close to the NumPy APIs.

Although we achieved similar imputation results using the PyTorch and TensorFlow code through the above massive experiments, we noticed that the PyTorch code has higher training efficiency with less GPU memory consumption and shorter training time compared to the TensorFlow code with the same training config and dataset. The major reason of the larger GPU memory occupied by the SSSD^{S4} TensorFlow code on PTB-XL dataset is that TensorFlow directly loads the whole dataset on GPU memory before the model training, while PyTorch loads the dataset on the CPU and only transfers the data batch to GPU memory for model training. Many kinds of warnings will appear when the TensorFlow model starts to train. The retracing warnings for CSDI TensorFlow code seem to slow down the training speed with the low GPU utilization rate.

5. Conclusion

In this work, we implemented the CDSI and SSSD^{S4} TensorFlow model and validated on several datasets. We found that the CSDI model performed better than paper results on PTB-XL dataset, and also outperformed the SSSD^{S4} model on three stock datasets. In particular, we have demonstrated some visualization examples with distinguishable imputation quality difference for two models. We believe that the SSSD^{S4} model still has the potential to improve the performance with some training tricks and larger dataset.

References

[1] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[2] V. Fortuin, D. Baranchuk, G. Rätsch, and S. Mandt, "GP-VAE: Deep Probabilistic Time Series Imputation," in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, 2019.

[3] I. J. Goodfellow *et al.*, "Generative Adversarial Nets," in *Neural Information Processing Systems*, 2014, pp. 2672–2680.

[4] A. Vaswani *et al.*, "Attention Is All You Need," *Adv. Neural Inf. Process. Syst.*, vol. 2017-Decem, no. Nips, pp. 5999–6009, Jun. 2017.

[5] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu, "Recurrent Neural Networks for Multivariate Time Series with Missing Values," *Sci. Rep.*, vol. 8, no. 1, pp. 1–12, 2018.

[6] J. Yoon, W. R. Zame, and M. Van Der Schaar, "Estimating Missing Data in Temporal Data Streams Using Multi-Directional Recurrent Neural Networks," *IEEE Trans. Biomed. Eng.*, vol. 66, no. 5, pp. 1477–1490, 2019.

[7] W. Cao, H. Zhou, D. Wang, Y. Li, J. Li, and L. Li, "BRITS: Bidirectional recurrent imputation for time series," *Adv. Neural Inf. Process. Syst.*, vol. 2018-Decem, no. NeurIPS, pp. 6775–6785, 2018.

[8] Y. Luo, X. Cai, Y. Zhang, J. Xu, and X. Yuan, "Multivariate time series imputation with generative adversarial networks," *Adv. Neural Inf. Process. Syst.*, vol. 2018-Decem, no. NeurIPS, pp. 1596–1607, 2018.

[9] S. Shan, Y. Li, and J. B. Oliva, "NRTSI: Non-Recurrent Time Series Imputation," *arXiv preprint 2102.03340*, 2021.

[10] W. Du, D. Cote, and Y. Liu, "SAITS: Self-Attention-based Imputation for Time Series," *arXiv preprint 2202.08516*, 2022.

[11] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Adv. Neural Inf. Process. Syst.*, vol. 2020-Decem, no. NeurIPS 2020, pp. 1–25, 2020.

[12] Z. Kong, W. Ping, J. Huang, K. Zhao, and B. Catanzaro, "DiffWave: A Versatile Diffusion Model for Audio Synthesis," I*n 9th International Conference on Learning Representations, ICLR 2021*, 2021.

[13] Y. Tashiro, J. Song, Y. Song, and S. Ermon, "CSDI: Conditional Score-based Diffusion Models for Probabilistic Time Series Imputation," *Adv. Neural Inf. Process. Syst.*, vol. 30, no. NeurIPS, pp. 24804–24816, 2021.

[14] J. M. L. Alcaraz and N. Strodthoff, "Diffusion-based Time Series Imputation and Forecasting with Structured State Space Models," *arXiv preprint 2208.09399*, 2022.

[15] A. Gu, K. Goel, and C. Ré, "Efficiently Modeling Long Sequences with Structured State Spaces," In *International Conference on Learning Representations*, 2022.

[16] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," *32nd Int. Conf. Mach. Learn. ICML 2015*, vol. 3, pp. 2246–2255, 2015.