# REPORT: DIFFUSION-BASED SSSD AND CSDI IMPUTATIONS, IMPLEMENTED WITH TENSORFLOW

**LU Yitao**
School of Data Science
City University of Hong Kong
Hong Kong SAR, China
Email: yitao.lu@my.cityu.edu.hk

**ABSTRACT**

The stock market is known for its volatility and unpredictability, even during holidays when markets are closed. In this study, we tackle two important challenges in the stock market: imputing missing stock prices during holidays and predicting the volatility of the Hong Kong market. To address these challenges, we analyze ten years of data from stocks listed in the Dow Jones 30, EuroStoxx 50 and Hang Seng 70 indices. Our approach involves evaluating the performance of four state-of-the-art methods: the Structured State Space Diffusion Model (SSSD), the Conditional Score-Based Diffusion Model (CSDI). And their variant based on the Simplified Structured State Space Sequence Model (S5), the Mega Moving Average Equipped Gated Attention (Mega). Our results suggest that the SSSD does not outperform CSDI in the stock price imputation task. Furthermore, we implement the Mega encoder-decoder approach for imputing the holiday missing data and forecasting stock prices, which can be seen as a specific case of missing data imputation. Results shown that the imputation algorithms are not suitable for predicting the volatility of Hong Kong market by using the US and EU market.

***K*eywords** Diffusion-based model · Structured State Space Model · Time Series · Generative Models ·

## 1 Introduction

The stock market is fast-moving and unpredictable, and that won't stop even when markets are closed and investors go home for the holidays. Market volatility sometimes increases during the holidays, which called festive or holiday effects [1]. In this report, we are going to complete a few tasks involving imputation of missing stocks data due to holidays and predicting the Hong Kong market by last night's American and European markets. Missing data in stocks is common due to the holiday, and also holidays in different places are not the same, resulting that the missing data in some of the dates are different.

This report aims to address the issue of forecasting the market volatility and the challenge of imputing missing stock data during holidays. The study focuses on the analysis of ten-year data of stocks listed in the Dow Jones 30, EuroStoxx 50, and Hang Seng 70 indices sourcing from yahoo finance. However, the large sequence length of over 2500 data points presents a potential overfitting problem with few stocks at hand (about 150). With this concern, we crop the data sequence into small pieces and explore the effect of sequence length to the algorithm performance. Some stocks were excluded from the analysis due to their insufficient ten-year sequence length. A more comprehensive description of the training data can be found in the Methods section 2.

We utilize three cutting-edge methods for imputing missing stock prices during holidays and two methods predicting the Hong Kong stock market's volatility based on the U.S. and E.U. markets. The first method is the Structured State Space Diffusion Model (SSSD-S4) [2], which utilizes the S4 model as its key component to capture long-term dependencies in the time series. The second method is the Conditional Score-Based Diffusion Model (CSDI) [3], utilizing score-based diffusion models conditioned on observed data. We also investigate the performance of Simplified Structured State Space Sequence Model (S5) proposed in [4] as a drop-in replacement for S4 model in SSSD or CSDI. This approach

simplifies the initialization and parameterization by using a multi-input and multi-output (MIMO) structure, which has been shown to achieve comparable performance with the state-of-the-art methods like S4.

The third method utilized in this study is the Mega Moving Average Equipped Gated Attention (MEGA) approach, as presented in [5]. The authors contend that the combination of gated attention mechanism and moving average inputs constitutes the optimal architecture for sequence processing. To address the limitations of plain transformers, such as weak inductive bias and quadratic computational complexity, MEGA incorporates inductive bias into the attention mechanism through the use of exponential moving average (EMA). A variant of MEGA, known as Mega-Chunk, has linear complexity and simply segments the input sequence into fixed chunks. In this study, we construct the Mega imputer by incorporating the Mega layer into a traditional transformer model in place of the multi-head attention layer. Like state-space models or attention models, the Mega layer captures long-term dependencies in the input sequence and can serve as a drop-in replacement in the SSSD or CSDI models. Further details about the model can be found in the Methods section 2.

To predict the volatility of the Hong Kong stock market, we aim to forecast the difference between the maximum and minimum prices of a stock achieved during the trading hours of the day by conditioning on the previous night's market data from the American and European markets. This prediction can be considered as a specific instance of missing data imputation where only the Hong Kong stock prices are absent in the last day of a sequence. To achieve this goal, we employ the Structured State Space Diffusion Model (SSSD-S4) and the Mega Moving Average Equipped Gated Attention (MEGA) model. The training data used for this task is the data imputed in the previous step.

Hence, in this project, we are going to investigate the performance of SSSD/CSDI, their variants and Mega Encode-Decoder model in the imputation task of stock market data. For the prediction task, we propose two methods based on SSSD-S4 and Mega. In the following sections we will first introduce the assumptions that we need, the details of the techniques that we use, and then we will present the results to compare the performance of these algorithms with comprehensive design of experiments.

## 2 Methods and Background

### 2.1 Assumptions and Stock Markets

Due to the holiday, the retail, food, and tourism industries tend to flourish before and after the holidays, which can have a significant impact on the stock market. This is known as the "holiday effect" or "pre-holiday effect" and it results in a slight increase in stock prices in the days leading up to a holiday.

There are various theories about the reason for this phenomenon. One possibility is that it is related to trading volumes, as more investors take time off and fewer remain active, the actions of persistent investors can lead to more pronounced market volatility. Another explanation could be the changing investor sentiment, where some investors become more risk-averse before the holidays, and opt to sell riskier stocks to avoid any potential negative developments that might occur during their absence.

Hence, we have the following two assumptions if the holidays are declared non-holidays due to the nation laws:

- **Assumption 1**: We assume that the investors' behavior would remain the same on the days before the holidays;
- **Assumption 2**: We assume that the retail, food and tourism industries would remains flourish on the days before the holidays

For the purpose of testing these assumptions, we have defined a holiday window that spans from the onset of the holiday effect to when it subsides, denoted as $\mathcal{D}$. To test assumption 1, we increase the trading volume by $c$ percent within the holiday window and observe any changes in stock prices during the holiday period. To validate assumption 2, we decrease the stock prices by $p$ percent within the holiday window and assess the impact on stock prices during the holiday.

### 2.2 Diffusion Model

Diffusion models are a type of generative model based on non-equilibrium thermodynamics. This model has various forms including Diffusion Probabilistic Models (DPMs) [6], Noise-Conditioned Score Networks (NCSNs) [7], and Denoising Diffusion Probabilistic Models (DDPMs) [8]. These models all have a similar structure, consisting of a Markov chain of diffusion steps that add random noise to the data (Forward diffusion process) and a Reverse diffusion process that uses learned parameters to reconstruct the desired data from the noise.

### 2.2.1 Forward Diffusion Process

Given a data point sampled from a real data distribution $\boldsymbol{x}_0 \sim q(\boldsymbol{x})$, the forward diffusion process shown in Equation (1) is defined as adding small amount of Gaussian noise into the sample at $\boldsymbol{T}$ steps, producing a sequence of noisy sample data as $\boldsymbol{x}_1, \cdots, \boldsymbol{x_T}$. And the step sizes are controlled by a variance schedule $\beta_t \in (0,1)_{t=1}^{\boldsymbol{T}}$. Generally, we can afford a larger update step when the samples get noiser, so $\beta_1 < \beta_2 < \cdots < \beta_{\boldsymbol{T}}$.

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

, where $q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) = N(\boldsymbol{x}_t; \sqrt{1-\beta_t}\boldsymbol{x}_{t-1}, \beta_t I)$. The data sample $\boldsymbol{x}_0$ gradually loses its distinguishable features as the step $t$ becomes larger. Eventually when $\boldsymbol{T} \to \infty$, $\boldsymbol{x_T}$ is equivalent to an isotropic Gaussian distribution.

A nice property of the above process is that we can sample $x_t$ at any arbitrary time step $t$ in a closed form using reparameterization trick. Let $\alpha_t = 1 - \beta_t$ and $\bar{\alpha_t} = \prod_{i=1}^{t} \alpha_i$ :

$$\begin{aligned}
\boldsymbol{x}_t =& \sqrt{\alpha_t}\boldsymbol{x}_{t-1} + \sqrt{1-\alpha_t}\epsilon_{t-1} \\
=& \sqrt{\alpha_t\alpha_{t-1}}\boldsymbol{x}_{t-2} + \sqrt{1-\alpha_t\alpha t-1}\bar{\epsilon}_{t-2} \\
& \cdots \\
=& \sqrt{\bar{\alpha_t}}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha_t}}\boldsymbol{\epsilon}
\end{aligned} \tag{2}$$

, where $\bar{\epsilon}_{t-2}$ is the combination of $\epsilon_{t-1}$ and $\epsilon_{t-2}$, the mean and standard error can be obtained by the moment generation function. Normal is $M(t) = E(e^{tX}) = e^{\mu t + \frac{1}{2}\sigma^2 t^2}$. Hence, $q(\boldsymbol{x}_t|\boldsymbol{x}_0) = N(\boldsymbol{x}_t; \sqrt{\bar{\alpha_t}}\boldsymbol{x}_0, (1-\bar{\alpha_t})I)$

### 2.2.2 Reverse Diffusion Process

The forward process transform data $\boldsymbol{x}$ into noise step by step, if we could reverse the above procee and sample from $q(\boldsymbol{x_{t-1}}|\boldsymbol{x_t})$ we will be able to recreate the true data from a Gaussian noise input $\boldsymbol{x_T} \sim N(0, I)$. Unfortunately, we cannot easily estimate $q(\boldsymbol{x_{t-1}}|\boldsymbol{x_t})$ because it needs to use the entire data set and therefore we need to learn a model $p(\Theta)$ to approximate these conditional probabilities in order to run the reverse diffusion process, where

$$p_\Theta(\boldsymbol{x_{0:T}}) = p(\boldsymbol{x_T}) \prod_{t=0}^{\boldsymbol{T}} p_\Theta(\boldsymbol{x_{t-1}}|\boldsymbol{x_t}) \tag{3}$$

and $p_\Theta(\boldsymbol{x_{t-1}}|\boldsymbol{x_t}) = N(\boldsymbol{x_{t-1}}; \boldsymbol{\mu}_\Theta(\boldsymbol{x_t}), \Sigma_\Theta(\boldsymbol{x_t}))$.

Using Baye's rule, we have

$$\begin{aligned}
q(\boldsymbol{x_{t-1}}|\boldsymbol{x_t}, \boldsymbol{x_0}) =& q(\boldsymbol{x_t}|\boldsymbol{x_{t-1}}, \boldsymbol{x_0}) \frac{q(\boldsymbol{x_{t-1}}|\boldsymbol{x_0})}{q(\boldsymbol{x_t}|\boldsymbol{x_0})} \\
\propto& exp(-\frac{1}{2}(\frac{(\boldsymbol{x_t} - \sqrt{\alpha_t}\boldsymbol{x_{t-1}})^2}{\beta_t} + \frac{(\boldsymbol{x_{t-1}} - \sqrt{\bar{\alpha_{t-1}}}\boldsymbol{x_0})^2}{1-\bar{\alpha_{t-1}}} - \frac{(\boldsymbol{x_t} - \sqrt{\bar{\alpha}_t}\boldsymbol{x_0})^2}{1-\bar{\alpha}_t})) \\
=& exp(-\frac{1}{2}(\frac{\boldsymbol{x_t^2} - \sqrt{\alpha_t}\boldsymbol{x_t}\boldsymbol{x_{t-1}} + \alpha_t\boldsymbol{x_{t-1}^2}}{\beta_t} + \frac{\boldsymbol{x_{t-1}^2} - 2\sqrt{\bar{\alpha_{t-1}}}\boldsymbol{x_0}\boldsymbol{x_{t-1}} + \bar{\alpha_{t-1}}\boldsymbol{x_0^2}}{1-\bar{\alpha_{t-1}}} - \frac{(\boldsymbol{x_t} - \sqrt{\bar{\alpha}_t}\boldsymbol{x_0})^2}{1-\bar{\alpha}_t})) \\
=& exp(-\frac{1}{2}((\frac{\alpha_t}{\beta_t} + \frac{1}{1-\bar{\alpha_{t-1}}})\boldsymbol{x_{t-1}^2} - 2(\frac{\sqrt{\alpha_t}}{\beta_t}\boldsymbol{x_t} + \frac{\sqrt{\bar{\alpha_{t-1}}}}{1-\bar{\alpha_{t-1}}}\boldsymbol{x_0})\boldsymbol{x_{t-1}} + C(\boldsymbol{x_t}, \boldsymbol{x_0})))
\end{aligned} \tag{4}$$

Recall the definition of $\alpha_t$ and $\bar{\alpha_t}$, we can define :

$$\begin{aligned}
\tilde{\beta}_t =& 1/(\frac{\alpha_t}{\beta_t} + \frac{1}{1-\bar{\alpha}_{t-1}}) \\
=& 1/(\frac{\alpha_t - \bar{\alpha}_t + \beta_t}{\beta_t(1-\bar{\alpha}_{t-1})}) \\
=& \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha_t}}\beta_t
\end{aligned} \tag{5}$$

### 2.2.3 Parameterization for loss function

Then, the mean of $q(\boldsymbol{x_{t-1}}|\boldsymbol{x_t}, \boldsymbol{x_0})$ can be obtained as followed:

$$\begin{aligned}\tilde{\mu}_t(\boldsymbol{x_t}, \boldsymbol{x_0}) =&(\frac{\sqrt{\alpha_t}}{\beta_t}\boldsymbol{x_t} + \frac{\sqrt{\bar{\alpha_{t-1}}}}{1-\bar{\alpha_{t-1}}}\boldsymbol{x_0})/(\frac{\alpha_t}{\beta_t} + \frac{1}{1-\bar{\alpha}_{t-1}})\\ =&(\frac{\sqrt{\alpha_t}}{\beta_t}\boldsymbol{x_t} + \frac{\sqrt{\bar{\alpha_{t-1}}}}{1-\bar{\alpha_{t-1}}}\boldsymbol{x_0}) \cdot \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha_t}}\beta_t\\ =&\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\boldsymbol{x_t} + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha_t}}\boldsymbol{x_0}\end{aligned} \tag{6}$$

By Equation (2), we have $\boldsymbol{x_0} = \frac{1}{\sqrt{\bar{\alpha_t}}}(\boldsymbol{x_t} - \sqrt{1-\bar{\alpha_t}}\epsilon_t)$, substituting into Equation (6) and obtaining

$$\begin{aligned}\tilde{\mu}_t =&\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\boldsymbol{x_t} + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha_t}}\frac{1}{\sqrt{\bar{\alpha_t}}}(\boldsymbol{x_t} - \sqrt{1-\bar{\alpha_t}}\epsilon_t)\\ =&\frac{1}{\sqrt{\alpha_t}}(\boldsymbol{x_t} - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha_t}}}\epsilon_t)\end{aligned} \tag{7}$$

Recall that we need to learn a neural network to approximate the conditioned probability distributions in the reverse diffusion process, $p_\Theta(\boldsymbol{x_{t-1}}|\boldsymbol{x_t}) = N(\boldsymbol{x_{t-1}}; \boldsymbol{\mu_\Theta}(\boldsymbol{x_t}), \Sigma_\Theta(\boldsymbol{x_t}))$. Hence, we use $\mu_\Theta$ to approximate $\tilde{\mu}_t$, with loss function as

$$\mathcal{L}_t = E_{\boldsymbol{x}_0,\epsilon}[\frac{1}{2||\Sigma_\Theta(\boldsymbol{x}_t, t)||_2^2}||\tilde{\mu}_t - \mu_\Theta||^2] \tag{8}$$

Furthermore, instead of predicting the mean of $\tilde{\mu}_t$ we can reparameterize the Gaussian noise to predict $\epsilon_t$ from the input data $\boldsymbol{x_t}$ at time $t$. That's to say $\mu_\Theta(\boldsymbol{x_t}, t) = \frac{1}{\sqrt{\alpha_t}}(\boldsymbol{x_t} - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha_t}}}\epsilon_\Theta(\boldsymbol{x_t}, t))$. Finally, we can derive the explicit form of the loss function as

$$\begin{aligned}\mathcal{L}_t =&E_{\boldsymbol{x}_0,\epsilon}[\frac{(1-\alpha_t)^2}{2\alpha_t(1-\bar{\alpha_t})||\Sigma_\Theta||_2^2}||\epsilon_t - \epsilon_\Theta(\boldsymbol{x_t}, t)||^2]\\ =&E_{\boldsymbol{x}_0,\epsilon}[\frac{(1-\alpha_t)^2}{2\alpha_t(1-\bar{\alpha_t})||\Sigma_\Theta||_2^2}||\epsilon_t - \epsilon_\Theta(\sqrt{\bar{\alpha_t}}\boldsymbol{x_0} + \sqrt{1-\bar{\alpha_t}}\epsilon_t, t)||^2]\end{aligned} \tag{9}$$

Empirically, [8] found that training the diffusion model works better with a simplified objective that ignores the weighting term:

$$\mathcal{L}_t = E_{t\sim\mathcal{U}[1,T],\boldsymbol{x}_0,\epsilon}[||\epsilon_t - \epsilon_\Theta(\sqrt{\bar{\alpha_t}}\boldsymbol{x_0} + \sqrt{1-\bar{\alpha_t}}\epsilon_t, t)||^2] \tag{10}$$

## 2.3 State Space Model

State space model (SSM) is an efficient modeling techniques especially for long-term dependencies sequences. The continuous-time linear SSMs is defined by defferential functions:

$$\begin{aligned}\frac{d\boldsymbol{x}(t)}{dt} =&\boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t)\\ \boldsymbol{y}(t) =&\boldsymbol{C}\boldsymbol{x}(t) + \boldsymbol{D}\boldsymbol{u}(t)\end{aligned} \tag{11}$$

, where $\boldsymbol{u}(t) \in \Re^K$ is an one-dimensional input signal, $\boldsymbol{x}(t) \in \Re^P$ is a latent state and $\boldsymbol{y}(t) \in \Re^M$ is an one-dimensional output signal. And $\boldsymbol{A}, \boldsymbol{BCD}$ are transition matrices. Previous work introduced structured state space model (S4) [9] and simplified S4 (S5) [4] model based on SSM that solved the computational bottleneck.

### 2.3.1 Structured State Space Model (S4)

The key is the initialization and re-parameterization of matrix $\boldsymbol{A}$. [10] proved that a specific matrix called High-order Polynomial Projection Operators (HiPPO-LegS) matrix, which allows $\boldsymbol{x}(t)$ to memorize the history of input $\boldsymbol{u}(t)$, defiend as:

$$A_{nk} = \begin{cases}(2n+1)^{1/2}(2k+1)^{1/2} & n > k\\ n+1 & n = k\\ 0 & n < k\end{cases} \tag{12}$$

, where HiPPO-LegS matrix enjoys favorable theoretical properties: it is invariant to input timescale, is fast to compute, and has bounded gradients and approximation error[10].

To be applied on the discrete input signals $(u_1, \cdots, u_n)$, the ODEs 11 must be discretirized by a step size $\Delta$ that represents the resolution of the input $\boldsymbol{u}$. Following the bilinear technique [11], the SSM 11 can be converted into recurrence form as

$$\begin{aligned} x_k =& \bar{\boldsymbol{A}} x_{k-1} + \bar{\boldsymbol{B}} u_k, \quad \bar{\boldsymbol{A}} = (I - \Delta/2 \cdot \boldsymbol{A})^{-1}(I + \Delta/2 \cdot \boldsymbol{A}) \\ y_k =& \bar{\boldsymbol{C}} x_k, \qquad\qquad\quad \bar{\boldsymbol{B}} = (I - \Delta/2 \cdot \boldsymbol{A})^{-1}\Delta \boldsymbol{B}, \bar{\boldsymbol{C}} = \boldsymbol{C} \end{aligned} \tag{13}$$

The recurrence form is not practical for training due to the long-term sequence. However, Equation 13 can be written as a discrete convolution, which unrolling 13 we can get:

$$\begin{aligned} x_0 =& \overline{\boldsymbol{B}} u_0 \qquad x_1 = \overline{\boldsymbol{A}\boldsymbol{B}} u_0 + \overline{\boldsymbol{B}} u_1 x_2 = \overline{\boldsymbol{A}^2 \boldsymbol{B}} u_0 + \overline{\boldsymbol{A}\boldsymbol{B}} u_1 + \overline{\boldsymbol{B}} u_2 \qquad \cdots \\ y_0 =& \overline{\boldsymbol{C}\boldsymbol{B}} u_0 \quad y_1 = \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}} u_0 + \overline{\boldsymbol{C}\boldsymbol{B}} u_1 y_2 = \overline{\boldsymbol{C}\boldsymbol{A}^2\boldsymbol{C}\boldsymbol{B}} u_0 + \overline{\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}} u_1 + \overline{\boldsymbol{C}\boldsymbol{B}} u_2 \quad \cdots \end{aligned} \tag{14}$$

, this gives the convolution kernel as:

$$\begin{aligned} y_k =& \overline{\boldsymbol{C}\boldsymbol{A}^k \boldsymbol{B}} u_0 + \overline{\boldsymbol{C}\boldsymbol{A}^{k-1}\boldsymbol{B}} u_1 + \cdots + \overline{\boldsymbol{C}\boldsymbol{B}} u_k \\ y =& \overline{\boldsymbol{K}} * u \\ \overline{\boldsymbol{K}} \in \Re^L :=& \mathcal{K}_L(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \overline{\boldsymbol{C}}) = (\overline{\boldsymbol{C}\boldsymbol{B}}, \cdots, \overline{\boldsymbol{C}\boldsymbol{A}^{L-1}\boldsymbol{B}}) \end{aligned} \tag{15}$$

In other words, Equation (15) is a single (non-circular) convolution and can be computed very efficiently with FFTs, provided that $\overline{\boldsymbol{K}}$ is known. However, computing $\overline{\boldsymbol{K}}$ in (15) is non-trivial, which the computation bottleneck is the repeated matrix multiplication by $\overline{\boldsymbol{A}}$. This can be overcome by a structural results, which we could do transformation to diagonalize the HiPPO matrix $\bar{\boldsymbol{A}}$. Unfortunately, the naive application of diagonalization does not work due to the infeasibility. Hence, [9] proved that the matrix $\bar{\boldsymbol{A}}$ can be decomposed into a normal matrix plus a low-rank term (NPLR).

**Theorem 1**: All HiPPO matrix in form 12 have a NPLR representation with unitary matrix $\boldsymbol{V} \in \Im^{N \times N}$, diagonal $\boldsymbol{\Lambda}$ and low-rank factorization $\boldsymbol{P}, \boldsymbol{Q} \in \Re^{N \times N}$. For HiPPO-LegS, the low rank r=1.

$$\boldsymbol{A} = \boldsymbol{V}\boldsymbol{A}\boldsymbol{V}^* - \boldsymbol{P}\boldsymbol{Q}^{'} = \boldsymbol{V}(\boldsymbol{\Lambda} - (\boldsymbol{V}^*\boldsymbol{P})(\boldsymbol{V}\boldsymbol{P}^*)^*)\boldsymbol{V}^* \tag{16}$$

With the availability of NPLR of HiPPO matrix, the convolution kernel can be calculated with the **Algorithm 1** in [9].

#### 2.3.2 Simplified Structured State Space Model (S5)

The S5 layer utilizes a multi-input, multi-output state state model to replace the banks of single-input, single-output state space model in S4. The key of the parameterization of S5 Layers is the to diagonalize the system, writing the continuous-time state matrix as $\boldsymbol{A} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{-1}$, where $\boldsymbol{\Lambda} \in \Im^{P \times P}$ denotes the diagonal matrix containing the eigenvalyes and $\boldsymbol{V} \in \Im^{P \times P}$ corresponds to the eigenvectors. Therefore, Equation (11) can be re-written as

$$\frac{d\boldsymbol{V}^{-1}\boldsymbol{x}(t)}{dt} = \boldsymbol{\Lambda}\boldsymbol{V}^{-1}\boldsymbol{x}(t) + \boldsymbol{V}^{-1}\boldsymbol{B}\boldsymbol{u}(t) \tag{17}$$

Defining $\tilde{\boldsymbol{x}(t)} = \boldsymbol{V}^{-1}\boldsymbol{x}(t)$, $\tilde{\boldsymbol{B}} = \boldsymbol{V}^{-1}\boldsymbol{B}$ and $\tilde{\boldsymbol{C}} = \boldsymbol{C}\boldsymbol{V}$ gives the reparameterized system as

$$\begin{aligned} \frac{d\tilde{\boldsymbol{x}}(t)}{dt} =& \boldsymbol{\Lambda}\tilde{\boldsymbol{x}}(t) + \tilde{\boldsymbol{B}}\boldsymbol{u}(t) \\ \boldsymbol{y}(t) =& \tilde{\boldsymbol{C}}\tilde{\boldsymbol{x}}(t) + \boldsymbol{D}\boldsymbol{u}(t) \end{aligned} \tag{18}$$

Note that, generally the state matrix $\boldsymbol{A}$ cannot be diagonalized stably. [12, 13] showed empirically that removing the low-rank term and initilizing the state matrix with HiPPO-N matrix still perform well. Hence, in S5 the initialization of state matrix is not by HiPPO-LegS matrix, but the HiPPO-N matrix.

### 2.4 Moving Average Equipped Gated Attention Model (Mega)

The Mega model combined exponential moving average (EMA) with single-head gated attention, which enjoys strong inductive bias while maintaining the capacity to learn long-term dependency. To improve the expressiveness of EMA,

[5] introduced a multi-dimensional variant of EMA, which gives the multi-dimensional damped EMA as

$$\boldsymbol{Y} := EMA(\boldsymbol{X}) \begin{cases} \boldsymbol{u}(t) = \boldsymbol{\beta}\boldsymbol{x}(t) \\ \boldsymbol{h}(t) = \boldsymbol{\alpha} \odot \boldsymbol{u}(t) + (1 - \boldsymbol{\alpha} \odot \boldsymbol{\delta}) \odot \boldsymbol{h}(t-1) \\ \boldsymbol{y}(t) = \boldsymbol{\eta}\boldsymbol{h}(t) \end{cases} \tag{19}$$

, where $\boldsymbol{\beta} \in \Re^{d \times h}$ is the expansion matrix, $\boldsymbol{\alpha}, \boldsymbol{\delta} \in (0, 1)^{d \times h}$ denote the EMA coeficients representing the degree of weighting decreases and the damping factor, and $\boldsymbol{\eta} \in \Re^{d \times h}$ is the projection matrix to map the hidden state back to original dimension.

Adopted from Gated Recurrent Unit (GRU) [14] and Gated Attention Unit (GAU) [15], the moving average gated attention mechanism in Mega exploits the output of $EMA(\boldsymbol{X})$ to compute the queries, keys and values respectively in the following way

$$MAGE := \begin{cases} \boldsymbol{X}' = EMA(\boldsymbol{X}) \in \Re^{n \times d} \\ \boldsymbol{Z} = \phi_{silu}(\boldsymbol{X}' W_z + b_z) \in \Re^{n \times z} \\ \boldsymbol{Q} = \boldsymbol{\kappa}_q \odot \boldsymbol{Z} + \boldsymbol{\mu}_q \in \Re^{n \times z} \\ \boldsymbol{K} = \boldsymbol{\kappa}_k \odot \boldsymbol{Z} + \boldsymbol{\mu}_k \in \Re^{n \times z} \\ \boldsymbol{V} = \phi_{silu}(\boldsymbol{X} W_v + b_v) \in \Re^{n \times v} \end{cases} \tag{20}$$

Then the output of Mega attention is

$$\boldsymbol{O} = f(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\tau(\boldsymbol{X})} + \boldsymbol{b}_{rel}) + \boldsymbol{V} \tag{21}$$

, where $f$ can be softmax or laplace attention function introduced in the Mega paper.

The Multi-dimensional Damped EMA can be viewed as a simplified variant of a state space model and has similarities with S4 [9]. The EMA sub-layer in Mega performs diagonalization on the state matrix and limits the diagonal elements to the range of (0, 1), resulting in a convolution kernel that is a Vandermonde product and can be calculated efficiently and with numerical stability. In this perspective, we can use mega layer as a drop-in replacement in SSSD-S4.

To implement Mega for the stock price imputation task, we designed Mega-encoder and Mega-decoder, which are based on the Mega layer. The encoder and decoder both uses non-causal bi-directory attention, only differ in the way that decoder has a linear dense layer for output, which is similar to traditional Transformer models. The process of encoding and decoding involves the following steps: the input sequence $\boldsymbol{x}$ (batch, length, feature) is fed into the encoder, producing the encoder output $\boldsymbol{x_{en}}$. Then, $\boldsymbol{x_{en}}$ is fed into the decoder to produce the desired sequence.

## 3 Experiments and Results

We conducted an implementation of the SSSD and CSDI algorithms, with several variants (S4/S5/Mega/Transformer), using TensorFlow. The stock data were obtained from Yahoo Finance, covering the period from 2013-1-1 to 2023-1-1, and include the stocks listed in the Dow Jones 30, EuroStoxx 50, and Hang Seng 70 indices. The full sequence of ten years is of length 2609, which is too long to learn, given the 150 stocks at hand, and can easily lead to overfitting. Due to limited GPU resources (two RTX 3090 chips), full-sequence learning was also infeasible. Therefore, we designed our experiments to explore how the sequence length affects the learning results and to find the best sequence length empirically for this imputation task. Finally, we compared the performance of the variants (S4/S5/Mega) of SSSD and CSDI, trained with the best sequence length found in the aforementioned experiment.

We design two sequence-chunking setting. First is the simple cropping: the sequence are cropped into fixed length $L$ with no overlapping. Second is moving cropping: from the beginning of the sequence, which is cropped into fixed length $L$ by fixed moving window $\omega$ time steps. Our experiments showed that moving cropping had a lesser impact on the results, but incurred a significantly higher computational cost compared to simple cropping. Therefore, in the subsequent sections, we only present the results obtained from the simple cropping setting.

### 3.1 Imputation stock price

Since there are no actual stock prices available for holidays, it is not possible to validate the imputed stock prices with a ground truth. However, most holidays occur on the same day each year, with a fixed pattern, and account for only a small percentage of the year. This allows us to view the holiday missing scenario as a random missing scenario. To evaluate the performance of the algorithms, we randomly set a number of days with observed data as missing (denoted

as $\boldsymbol{x}_{fm}$) and compare it with the imputed data (denoted as $\boldsymbol{x}_g$) by computing metrics such as mean square error (MSE) and mean absolute error (MAE).

To evaluate the performance of SSSD-S4 and CSDI-S4 for imputing stock prices, we computed the mean squared error (MSE) and mean absolute error (MAE) for different sequence lengths: 100, 200, 400, and 800. The results are reported in Table (3.1). The evaluation was done using the simple cropping sequence-chunking setting. Due to limited GPU resources, the number of residual layers used for the SSSD-S4 model when the sequence length was 800 was 20, which was reduced from the original setting of 36 layers.

| Models | 100 | 200 | 400 | 800 [1] |
|---|---|---|---|---|
| | MSE ($\times 10^{-4}$) | | | |
| SSSD-S4 | 17.56 $\pm$1.61 | 13.4 $\pm$0.39 | 10.8 $\pm$0.36 | 9.67 $\pm$0.30 |
| CSDI-S4 | 4.7 $\pm$0.043 | 2.2 $\pm$0.015 | 2.77 $\pm$0.082 | 9.34 $\pm$0.38 |
| MEGA | 6.42 $\pm$0.093 | 6.29 $\pm$0.09 | 8.85 $\pm$0.12 | 22.8 $\pm$0.18 |
| | MAE ($\times 10^{-2}$) | | | |
| SSSD-S4 | 5.05 $\pm$0.025 | 1.53 $\pm$0.0067 | 1.45 $\pm$0.0061 | 1.56 $\pm$0.0056 |
| CSDI-S4 | 1.45 $\pm$0.0032 | 0.99 $\pm$0.0021 | 1.10 $\pm$0.0026 | 2.13 $\pm$0.0046 |
| MEGA | 1.68 $\pm$0.0038 | 1.75 $\pm$0.0036 | 2.07 $\pm$0.0044 | 3.05 $\pm$0.0077 |

Table 1: MSE and MAE for imputating stocks with SSSD-S4, CSDI-S4 and MEGA for different sequence length: 100, 200, 400 and 800. Note that the sequence-chunking setting here is simple cropping. Due to the limited GPU resource, the number of residual layers used in SSSD-S4 for length 800 is 20, while rest is 36 same as the original setting.

The results from the table 3.1 show that SSSD-S4 does not perform better than CSDI-S4 in terms of MAE or MSE in all the experiments conducted. However, it is observed that the performance of SSSD-S4 improves with longer sequences, with the best results being obtained for a sequence length of 800. On the other hand, for CSDI-S4, the MAE and MSE decrease and then increase at a sequence length of 800. This result can be attributed to SSSD-S4's ability to capture long-term dependencies, and a longer sequence providing more historical information, leading to better performance. Conversely, CSDI-S4 is better suited for random missing patterns, such imputation of holiday missing data can be viewed as one, and has weaker capabilities in capturing long-term dependencies. Thus, its performance increases and decreases with the increase in sequence length.

For Mega Encoder-Decoder, the results shown in Table 3.1 presents a competitive performance compared to the SSSD-S4 and CSDI-S4. And we notice that the MAE or MSE increases by sequence length, this indicates that Mega's ability in capturing long term dependencies decreases by the sequence length.

| Metrics | Model | S4 | S5 | Mega | Tranformer |
|---|---|---|---|---|---|
| MSE($\times 10^{-4}$) | SSSD | 13.4 $\pm$0.39 | – | – | – |
| | CSDI | 2.2 $\pm$0.015 | – | – | 6.55 $\pm$0.035 |
| MAE($\times 10^{-2}$) | SSSD | 1.53 $\pm$0.0067 | – | – | – |
| | CSDI | 0.99 $\pm$0.0021 | – | – | 1.79 $\pm$0.0041 |

Table 2: MSE and MAE for imputing stocks prices with SSSD and CSDi variants with sequence length set as 200. Note that the results of SSSD or CSDI variants: Mega/S5 are not obtained due to the low efficiency of the implementation.

In table (3.1), we reported the metrics of MSE and MAE for different variants of SSSD and CSDI. Due to the implementation, the training of S5 and Mega model is relatively slow and also memory comsuming even we reduce the number of residual layers into 18 for SSSD. Till the deadline of the report, we are not able to fetch meaningful results for this two variants.

### 3.2 Predict Hong Kong Market's Volatility by U.S. and E.U. Stock Market

Before the Hong Kong stock market opens, we have access to the previous night's market data from Europe and the USA. Our goal is to examine whether this data can be used to predict the difference between the maximum and minimum stock price that will be achieved during the day's trading hours. This problem can be viewed as a form of missing data imputation, where only the next day's price data in the Hong Kong market is missing. To address this problem, we conduct experiments using two predictors: SSSD-S4 and Mega Encoder-Decoder.

The SSSD-S4 model is used for imputing missing data by conditioning on the observed history of HK stock prices as well as US and EU stock prices. To apply both SSSD-S4 and Mega Encoder-Decoder to the prediction of Hong Kong

market volatility, we mask the day that needs to be predicted. The training data used in this section is obtained from imputation performed by the CSDI-S4 model with a sequence length of 200. The results of the prediction are presented in Table 3.2.

| Algorithm | MAE | MSE |
|---|---|---|
| SSSD-S4 | $2.93 \pm 0.0043 \times 10^{-2}$ | $2.52 \pm 0.29 \times 10^{-3}$ |
| Mega | $2.34 \pm 0.0039 \times 10^{-2}$ | $9.28 \pm 0.45 \times 10^{-4}$ |

From Table (3.2), it's evident that the accuracy of the prediction task is not as high as the imputation task previously applied. This may be due to two reasons: first, the imputed data may not be accurate enough for the prediction task, and second is a system designing problem, which is limited by using only one time stamp data per day to predict the difference between the maximum and minimum in the HK stock market. With more precise data from the US and EU stock market, a significant improvement in the prediction could be achieved.

## 4 Conclusion

In this study, the SSSD and CSDI algorithms with their variants were implemented and evaluated for imputing stock prices using TensorFlow. The performance was evaluated by computing the mean squared error (MSE) and mean absolute error (MAE) for different sequence lengths.

The results of the stock price imputation task using the SSSD-S4, CSDI-S4 and Mega Encoder-Decoder models show that the performance of SSSD-S4 improves with longer sequences, with the best results being obtained for a sequence length of 800. On the other hand, CSDI-S4 performs better for shorter sequences and has weaker capabilities in capturing long-term dependencies. Mega Encoder-Decoder presents competitive performance compared to SSSD-S4 and CSDI-S4, but its performance decreases with increasing sequence length. This suggests that each of the models has its strengths and weaknesses, and the choice of the model should depend on the nature of the missing data and the desired performance metric. Further research is needed to determine the best model for stock price imputation and to explore other approaches to improve performance.

For the prediction of the difference between the maximum and minimum prices of a Hong Kong stock achieved during the trading hours of the next day, two models, SSSD-S4 and Mega Encoder-Decoder, were used with previous American and European market data. Both models showed results in terms of mean absolute error (MAE) and mean squared error (MSE) are not as good as the performance shown in imputation task.

## 5 Discussion

In this project, we investigated the performance of SSSD, CSDI and Mega models in the applications of imputing and predicting stock prices, with the models implemented in TensorFlow.

The key feature of the S4 model is the use of a black-box Cauchy kernel in place of the convolution kernel. The products of the state matrix $\boldsymbol{A}$ are computed using an inverse Fourier transform. The original implementation by PyTorch optimizes the Cauchy kernel calculation using two methods: PyKeOps or C++ optimization. However, these optimizations cannot be applied to the TensorFlow framework, leading to increased computational load, memory usage, and time complexity. To improve performance in the calculation of the Cauchy kernel, I propose the use of Numba-based GPU parallel computing.

To compute the following equation for all $\omega_i$ in space $\Omega$:

$$\sum_j \frac{q_j^* p_j}{\omega_i - \lambda_j} \tag{22}$$

where $P$, $Q$, and $\Lambda$ are vectors with length N, and $\Omega$ has length M, we can launch $M \times N$ threads with M blocks, each with N threads. The vectors $P$, $Q$, and $\Lambda$ are mapped to each thread's local memory for each block, and $\Omega$ is mapped to the shared memory of each block. Each thread performs a single calculation of $\frac{q_j^* p_j}{\omega_i - \lambda_j}$. Finally, the calculated $M \times N$ matrix is mapped back to the host device, and the desired result is obtained by summing up the values.

In general, a good memory allocation and management strategy can offset the cost of transferring data between the device and host by the increased speed of parallel computing. However, due to limited time, the implementation details are not provided in this project.

We view the holiday missing scenario as a case of random missing data, as the missing ratio is low and holidays are sparsely distributed throughout the year. Although this approximation is close to randomness, the fixed dates of most holidays and the varying number of days off for different holidays may cause the use of random missing masking strategies to result in a decrease in performance for both the CSDI and SSSD algorithms. If a more accurate masking strategy could be optimized, the results of imputing missing stock data would be improved.

Furthermore, our implementation of the S5 and Mega Encoder layers as a drop-in replacement in the SSSD or CSDI models with TensorFlow was slow and resulted in the failure to retrieve results. According to [4], the S5 model should have the same complexity as long as the size of the latent state $P$ is $O(H)$. In the S4 model, the dimension was set to 64, and we also set $P$ equal to 64 for the S5 model. However, the computing time for the same number of residual layers in the SSSD model was ten times that of the S4 variant. This could be due to the inferior vectorized mapping in TensorFlow compared to jax optimization. Upon profiling, we found that the major cost in computing the S5 model was the binary operator in the parallel scan of linear recurrence. We can also use GPU parallel computing to optimize the computation like we described for cauchy kernel computation.

The Mega Encoder layer, which was used as a drop-in replacement for the S4 in the SSSD or CSDI models, had moderate training speed compared to the S5 model but was still slower than the S4 version. However, the evaluation of imputing missing data took significantly more time, and I was unable to obtain the imputed data at last.

And in the prediction of the difference between the maximum and minimum of Hong Kong stock price by the last night's US and EU market data, the results are not bad. However, to improve the accuracy of the prediction, more detailed information could be obtained. For example, the use of hourly or half-hourly data from the US and EU markets could lead to better results in predicting the difference between the maximum and minimum prices.

Reflecting on the entire project, I realized that the period I enjoyed the most was the coding part, and how I made the codes work as expected. Additionally, I gained valuable experience by learning a new technique for implementing deep learning models by TensorFlow. Furthermore, my research into diffusion models and related papers was highly engaging and fascinating.

## References


[1] Marko Milošević, Goran Anđelić, Slobodan Vidaković, and Vladimir Đaković. The influence of holiday effect on the rate of return of emerging markets: a case study of slovenia, croatia and hungary. *Economic research-Ekonomska istraživanja*, 32(1):2354–2376, 2019.

[2] Juan Miguel Lopez Alcaraz and Nils Strodthoff. Diffusion-based time series imputation and forecasting with structured state space models. *arXiv preprint arXiv:2208.09399*, 2022.

[3] Yusuke Tashiro, Jiaming Song, Yang Song, and Stefano Ermon. Csdi: Conditional score-based diffusion models for probabilistic time series imputation. *Advances in Neural Information Processing Systems*, 34:24804–24816, 2021.

[4] Jimmy TH Smith, Andrew Warrington, and Scott W Linderman. Simplified state space layers for sequence modeling. *arXiv preprint arXiv:2208.04933*, 2022.

[5] Xuezhe Ma, Chunting Zhou, Xiang Kong, Junxian He, Liangke Gui, Graham Neubig, Jonathan May, and Luke Zettlemoyer. Mega: moving average equipped gated attention. *arXiv preprint arXiv:2209.10655*, 2022.

[6] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International Conference on Machine Learning*, pages 2256–2265. PMLR, 2015.

[7] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in neural information processing systems*, 32, 2019.

[8] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[9] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

[10] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33:1474–1487, 2020.

[11] Arnold Tustin. A method of analysing the behaviour of linear systems in terms of time series. *Journal of the Institution of Electrical Engineers-Part IIA: Automatic Regulators and Servo Mechanisms*, 94(1):130–142, 1947.

[12] Albert Gu, Ankit Gupta, Karan Goel, and Christopher Ré. On the parameterization and initialization of diagonal state space models. *arXiv preprint arXiv:2206.11893*, 2022.

[13] Ankit Gupta. Diagonal state spaces are as effective as structured state spaces. *arXiv preprint arXiv:2203.14343*, 2022.

[14] Rahul Dey and Fathi M Salem. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pages 1597–1600. IEEE, 2017.

[15] Devendra Singh Chaplot, Kanthashree Mysore Sathyendra, Rama Kumar Pasumarthi, Dheeraj Rajagopal, and Ruslan Salakhutdinov. Gated-attention architectures for task-oriented language grounding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.