*Data-Centric Engineering* (2025), 6: e17
doi:10.1017/dce.2025.6

RESEARCH ARTICLE

# GenFormer: a deep-learning-based approach for generating multivariate stochastic processes

Haoran Zhao[1] and Wayne Uy[2]

[1]Department of Civil and Environmental Engineering, Cornell University, Ithaca, NY, USA
[2]Center for Applied Mathematics, Cornell University, Ithaca, NY, USA
**Corresponding author:** Haoran Zhao; Email: hz289@cornell.edu

**Received:** 24 June 2024; **Revised:** 14 January 2025; **Accepted:** 15 January 2025

**Keywords:** Markov processes; multivariate stochastic processes; stochastic generator; time series forecasting; Transformer-based deep learning model

## Abstract

Stochastic generators are essential to produce synthetic realizations that preserve target statistical properties. We propose GenFormer, a stochastic generator for spatio-temporal multivariate stochastic processes. It is constructed using a Transformer-based deep learning model that learns a mapping between a Markov state sequence and time series values. The synthetic data generated by the GenFormer model preserve the target marginal distributions and approximately capture other desired statistical properties even in challenging applications involving a large number of spatial locations and a long simulation horizon. The GenFormer model is applied to simulate synthetic wind speed data at various stations in Florida to calculate exceedance probabilities for risk management.

**Impact Statement**

This paper introduces a novel deep-learning-based stochastic generator for multivariate stochastic processes. Unlike previous methods that typically struggle with the curse of dimensionality, this approach performs satisfactorily in high-dimensional applications. The generator is applicable in various engineering fields requiring data augmentation for multivariate time series.

## 1. Introduction

Stochastic generators are tools to produce synthetic data which preserve desired statistical properties. They are crucial in situations wherein the number of available records is inadequate while a large amount of data are required. This is especially the case in reliability analysis and risk management. For example, performance-based engineering requires synthetic data that characterize excitations from natural hazards to provide accurate reliability estimates of building systems (Radu and Grigoriu, 2018; Ouyang and Spence, 2021). Robust risk assessments of parametric insurance products necessitate the generation of supplemental synthetic loss events for various perils such as hurricane and excess rainfall (Zhou et al., 2018; Ng et al., 2021). The underlying data in the aforementioned examples, for example, wind pressure

This research article was awarded Open Data and Open Materials badges for transparent practices. See the Data Availability Statement for details.

fields and precipitation time series, are typically spatio-temporal in nature and can be represented as multivariate stochastic processes.

While developing stochastic generators for multivariate Gaussian processes is a well-established research area, constructing models which can be applied to generate time series with consistent non-Gaussian features remains to be a challenge. One class of methods constitutes direct extensions of those used for Gaussian processes which target certain statistical properties beyond the second moment. The third-order spectral representation method introduced in (Vandanapu and Shields, 2021) appends additional terms to the traditional spectral representation which is typically employed to simulate Gaussian processes. This enables it to capture the third-moment properties. The polynomial chaos (Yang et al., 2013) and translation processes (Zhao et al., 2019) are nonlinear transformations of Gaussian processes. The former utilizes truncated Hermite polynomials while the latter applies a transformation based on marginal distributions to approximately match the finite dimensional distributions and exactly match the marginal distributions, respectively. Recently, a mix of the above two models has been proposed in (Xu et al., 2023). In addition, stochastic generators based on the Markov assumption have been formulated to produce synthetic time series approximating finite dimensional distributions. These incorporate a variety of techniques such as the resampling procedure (Breinl et al., 2013), $K$-nearest neighbor algorithm (Apipattanavis et al., 2007), and copula models (Zhao, 2017). A prevalent challenge in many of the aforementioned models is the curse of dimensionality in which increasing the number of variates, that is spatial locations, and the simulation time horizon leads to substantial computational demands or significant decline in model performance of approximating target statistical properties (Yao et al., 2023).

Deep learning models, a subset of machine learning algorithms, have gained prominence in recent years due to their capabilities in solving large-scale problems. These models excel in feature extraction and pattern recognition involving medium to large datasets, making them well-suited for complex tasks such as image and speech recognition (Deng et al., 2013; Islam et al., 2018), natural language processing (Vaswani et al., 2017; Yin et al., 2017), and time series forecasting (Li et al., 2019; Salinas et al., 2020). Time series forecasting is concerned with predicting future time series values based on the historical records. Recent deep forecasting models, particularly those based on the Transformer architecture, have achieved great progress in predicting multivariate processes with a large number of locations over a long time horizon (Lai et al., 2018; Liu et al., 2020). Due to the attention mechanism, Transformers are capable of modeling long-term dependencies and complex patterns in sequential data. This leads to markedly improved prediction accuracy over traditional methods like ARIMA and deep learning models based on recurrent neural networks (RNN) or multilayer perceptrons (MLP), across various applications in weather and environmental forecasting (Wu et al., 2021; Zhou et al., 2021).

Inspired by the use of deep learning models in forecasting applications, we propose GenFormer, a deep-learning-based stochastic generator for stationary and ergodic multivariate processes with continuous marginal distributions which aims to tackle the challenges of simulation in high dimensions. GenFormer is constructed under the Markov assumption and can be regarded as an extension of (Breinl et al., 2013). It is composed of two models. The first is a univariate discrete-time Markov process in which each type of spatial variation across locations is represented by a Markov state. The second is a Transformer-based deep learning model which establishes a mapping from the Markov states to the values of time series. The generation of synthetic realizations of multivariate processes based on the GenFormer begins with the simulation of a synthetic univariate Markov sequence which subsequently serves as input for the deep-learning-based mapping. In practice, the synthetic data generated by the deep learning model may not preserve essential statistical properties such as the spatial correlation and marginal distributions. As such, the GenFormer model incorporates a model post-processing step involving a transformation of the resulting samples based on the Cholesky decomposition as well as a sample reshuffling technique to correct for key statistical properties. The final synthetic data produced by GenFormer is able to exactly match the marginal distributions and approximately match other statistical properties, including higher-order moments or probabilities of quantities of interest. Our numerical examples involving numerous spatial locations and simulation over a long time horizon demonstrate that synthetic realizations produced by GenFormer can be reliably utilized in downstream applications due to the superior performance of deep learning models for complex and high-dimensional tasks.

The paper is organized as follows. Section 2 outlines preliminaries of the GenFormer model. We review the stochastic generator proposed in (Breinl et al., 2013) designed for precipitation data as well as deep learning models used for time series forecasting. The construction and simulation algorithm of the GenFormer model are presented in Section 3. The performance of GenFormer on approximating the target statistical properties of interest is examined in Section 4 via numerical examples involving a synthetic dataset generated from stochastic differential equations and a real dataset of wind speeds measured at stations in Florida. Concluding remarks are offered in Section 5.

## 2. Preliminaries

We present in Section 2.1 a stochastic generator tailored to precipitation data. Existing Transformer-based deep learning models used for time series forecasting are then summarized in Section 2.2. These provide the foundation for the proposed stochastic generator in this work.

### 2.1. Stochastic generator for $m$-station physical processes

Let $\boldsymbol{X}(t) = [X_1(t), \ldots, X_m(t)]^T, t \in [0, T]$, be a $m$-variate stationary and ergodic stochastic process with continuous marginal distributions where $X_i(t)$ is a univariate stochastic process modeling the temporal evolution of a quantity of interest at spatial location $i$, $i = 1, \ldots, m$. In practice, $\boldsymbol{X}(t)$ is measured at $n$ discrete time stamps $t_1, \ldots, t_n$, $0 = t_1 < \cdots < t_n = T$, that evenly partition the time interval $[0, T]$. Stochastic generators, fitted using the observed realizations $\boldsymbol{x}(t) = [x_1(t), \ldots, x_m(t)]^T$ of $\boldsymbol{X}(t)$, produce additional synthetic realizations that preserve the statistical properties of $\boldsymbol{X}(t)$. These synthetic realizations can then be employed in downstream applications such as estimating exceedance probabilities of quantities of interest derived from $\boldsymbol{X}(t)$.

The stochastic generator proposed in (Breinl et al., 2013) is of interest in this subsection. It is designed for multi-station precipitation data, that is, $X_i(t)$ corresponds to a rainfall process measured at station $i$ with $x_i(t)$ denoting the observed data. The construction of the stochastic generator involves three steps. First, a univariate Markov sequence $Y_1, \ldots, Y_n$ that relates to the spatial rainfall pattern across stations is constructed from $\boldsymbol{X}(t_1), \ldots, \boldsymbol{X}(t_n)$. Second, a synthetic realization $\tilde{y}_1, \ldots, \tilde{y}_n$ of the Markov state sequence is generated which guides the simulation of the preliminary synthetic realization $\tilde{\boldsymbol{x}}(t_1), \ldots, \tilde{\boldsymbol{x}}(t_n)$ of the rainfall sequence via resampling. Third, the final synthetic sequence $\widehat{\boldsymbol{x}}(t_1), \ldots, \widehat{\boldsymbol{x}}(t_n)$ is obtained using the reshuffling technique according to the ranks of the realization in the previous step. We discuss these steps further in the subsections below.

#### 2.1.1. Univariate Markov sequences for m-variate stochastic processes

The approach proceeds by fitting a univariate Markov process $Y_1, Y_2, \ldots, Y_n$ corresponding to $\boldsymbol{X}(t_j)$ at time stamps $t_j, j = 1, \ldots, n$. The stochastic process $Y_1, Y_2, \ldots, Y_n$ is a $p^{\text{th}}$-order discrete-time Markov process if the conditional random variables $(Y_j | Y_{j-1}, \ldots, Y_{j-p})$ and $(Y_{j-p-1}, \ldots, Y_1 | Y_{j-1}, \ldots, Y_{j-p})$ are independent for $j > p+1$ (Ibragimov, 2009). Under the assumption that $Y_j$ is discrete-valued, the above definition readily implies the well-known Markov property.

$$P(Y_j = y_j | Y_{j-1} = y_{j-1}, \ldots, Y_1 = y_1) = P(Y_j = y_j | Y_{j-1} = y_{j-1}, \ldots, Y_{j-p} = y_{j-p}), \qquad j \geq p+1. \tag{2.1}$$

The state $Y_j$ at time stamp $t = t_j$ thus depends only upon the states at the past $p$ time stamps $t_{j-1}, \ldots, t_{j-p}$. The probability $P(Y_j = y_j | Y_{j-1} = y_{j-1}, \ldots, Y_{j-p} = y_{j-p})$ is also known as the transition probability, denoted by $P(y_j | y_{j-1}, \ldots, y_{j-p})$ in the following context.

For rainfall processes, the countable state space of $Y_j$ consists of $2^m$ states, corresponding to all the combinations of the wet (rain) and dry (no rain) scenarios at $m$ stations. To illustrate the construction of a Markov state sequence, consider the rainfall data at $m = 2$ locations recorded in Table 1. Also shown is the Markov state to which the observations at each time stamp are mapped. There are a total of four states, that

***Table 1.*** *Illustrated mapping between the observed rainfall data and the corresponding Markov states for $m = 2$ locations. Observations are mapped to the Markov states depending on which location experiences rainfall*

| Time stamp | Markov state | $x_1(t)$ | $x_2(t)$ |
|---|---|---|---|
| $t_1$ | 0 | 0 | 0 |
| $t_2$ | 1 | 10.54 | 0 |
| $t_3$ | 2 | 0 | 1.32 |
| $t_4$ | 3 | 2.28 | 1.63 |
| $t_5$ | 0 | 0 | 0 |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| $t_n$ | 1 | 12.21 | 0 |

is, $Y_j \in \{0,1,2,3\}$. We set $Y_j = 0$ if no rain is observed at both locations, $Y_j = 1$ if only the first location experiences rainfall, $Y_j = 2$ if only the second location experiences rainfall, and $Y_j = 3$ if both locations receive rain.

The Markov state sequence $Y_j, j = 1,\dots,n$, can be fully characterized by the order $p$ and the transition probability $P(y_j|y_{j-1},\dots,y_{j-p}), j \geq p+1$. Likelihood measures such as the Akaike information criterion (AIC) (Katz, 1981) or the Bayesian information criteria (BIC) (Schwarz, 1978) can be employed to estimate $p$. The corresponding transition probability, represented in matrix form, can then be estimated given the realizations of $Y_1,\dots,Y_n$ (Brémaud, 1999). Based on (2.1), a sample sequence of $Y_j$ is simulated in a sequential manner for time stamps $t_{p+1},\dots,t_n$. Given a realization $y_1,\dots,y_p$ of $Y_1,\dots,Y_p$, we initialize $\tilde{y}_1,\dots,\tilde{y}_p$ by using $y_1,\dots,y_p$. Subsequently, we simulate a realization $\tilde{y}_{p+1}$ from the transition probability $P(y|y_1,\dots,y_p)$. This is then used to simulate a realization $\tilde{y}_{p+2}$ from the transition probability $P(y|y_2,\dots,y_p,\tilde{y}_{p+1})$. This procedure is repeated to produce $\tilde{y}_{p+1},\dots,\tilde{y}_n$.

### *2.1.2. Resampling m-variate stochastic processes based on univariate Markov sequences*

In the next step, preliminary synthetic realizations of the rainfall sequence $\tilde{\boldsymbol{x}}(t_1),\dots,\tilde{\boldsymbol{x}}(t_n)$ are generated by resampling subsequences of rainy sequences present in the series $\tilde{y}_1,\dots,\tilde{y}_n$. A rainy sequence is defined as the sequence of Markov states such that at least one station experiences rain for consecutive time stamps. For the case where $m = 2$, the rainy sequence takes values from set $\{1,2,3\}$ which excludes 0 because it is a dry scenario. In Table 1, the subsequence 1,2,3 at time stamps $t = t_2, t_3, t_4$ is a rainy sequence.

The resampling procedure is based on the bootstrap algorithm (Horowitz, 2001). Suppose we have a synthetic realization $\tilde{y}_1,\dots,\tilde{y}_n$ from Section 2.1.1. For each rainy sequence present in this series, we seek an identical rainy sequence among the Markov state sequences of the observed data. The corresponding rainfall measurements of the identical rainy sequence are then used as the synthetic rainfall values. For example, suppose we have a synthetic Markov state sequence of 1,2,3 at the consecutive time stamps $t_j, t_{j+1}, t_{j+2}, j \in \{1,\dots,n-2\}$. Because it matches the existing rainy sequence shown in Table 1, we have the synthetic realization $\tilde{\boldsymbol{x}}(t_j) = [\tilde{x}_1(t_j), \tilde{x}_2(t_j)]^T = [10.54, 0]^T$, $\tilde{\boldsymbol{x}}(t_{j+1}) = [0, 1.32]^T$, and $\tilde{\boldsymbol{x}}(t_{j+2}) = [2.28, 1.63]^T$. When there are multiple matches, we randomly choose one in a uniform manner. The resampling procedure is repeated for all the synthetic rainy sequences present in $\tilde{y}_1,\dots,\tilde{y}_n$. This results in the synthetic realization $\tilde{\boldsymbol{x}}(t_1),\dots,\tilde{\boldsymbol{x}}(t_n)$ of length $n$.

There are two limitations of resampling. First, we may not find a rainy sequence from the existing realizations that matches the synthetic Markov state sequence. This is especially the case when $m$ is large which then implies that the Markov state space dimension is large. Special techniques such as divide-and-conquer need to be applied, where the synthetic rainy sequence is divided into subsequences to be matched. However, this results in inconsistencies in the statistical properties, for example, auto-correlation functions, of the resulting realizations. Second, resampling is unable to generate unobserved values of $\boldsymbol{X}(t)$ as it is performed via bootstrapping. This becomes a problem if interest is on extreme events, for example. The latter limitation is addressed by the reshuffling technique described in the following subsection. However, the former still remains a limitation that hinders the scalability of this stochastic generator for large $m$.

### *2.1.3. Reshuffling realizations of m-variate stochastic processes*

To ensure the inclusion of the unobserved values of $\boldsymbol{X}(t)$, especially the unprecedented extremes, we perform a reshuffling procedure for each station. First, new samples are simulated from the marginal distribution for each station. These are then reshuffled according to the ranks of the realizations resulting from the resampling step in Section 2.1.2.

In (Breinl et al., 2013), the authors suggest to characterize the marginal distributions of $\boldsymbol{X}(t)$ with parametric forms, for example, Weibull distribution for the positive parts of the distributions, calibrated to the existing rainfall observations. For each spatial location $i = 1,\ldots,m$, denote by $\tilde{\boldsymbol{x}}_i = [\tilde{x}_i(t_1),\ldots,\tilde{x}_i(t_n)]$ the sequence of a resulting realization from the resampling procedure described in Section 2.1.2 and let $\tilde{\boldsymbol{r}}_i = [\tilde{r}_i(t_1),\ldots,\tilde{r}_i(t_n)]$ be the corresponding ranks of the components of $\tilde{\boldsymbol{x}}_i$, sorted in descending order. Let $t_{j_1},\ldots,t_{j_n}$ be the time indices, $j_1,\ldots,j_n \in \{1,\ldots,n\}$, such that $\tilde{x}_i(t_{j_1}) \geq \ldots \geq \tilde{x}_i(t_{j_n})$. Thus, $\tilde{r}_i(t_{j_1}) = 1$ for the time stamp $t_{j_1}$ while $\tilde{r}_i(t_{j_n}) = n$ for the time stamp $t_{j_n}$. Denote by $\boldsymbol{z}_i = [z_i^1,\ldots,z_i^n]$, $z_i^1 \geq z_i^2 \geq \cdots \geq z_i^n$, a sorted sequence of $n$ new samples simulated from the marginal distribution of $X_i(t)$. The reshuffled sequence $\widehat{\boldsymbol{x}}_i = [\widehat{x}_i(t_1),\ldots,\widehat{x}_i(t_n)]$ is then defined by.

$$\widehat{x}_i(t_j) = z_i^{\tilde{r}_i(t_j)}, \qquad j = 1,\ldots,n. \tag{2.2}$$

To illustrate, consider a two-station rainfall process from the resampling step with a time horizon of five steps as shown in Table 2, where the rainfall sequences at the two stations are denoted by $\tilde{\boldsymbol{x}}_1$ and $\tilde{\boldsymbol{x}}_2$. The time sequence at the first location $\tilde{\boldsymbol{x}}_1 = [2.14,6.36,0.64,4.05,1.31]$ results in a rank sequence $\tilde{\boldsymbol{r}}_1 = [3,1,5,2,4]$ while the time sequence at the second location $\tilde{\boldsymbol{x}}_2 = [0.51,3.24,2.46,0.60,2.00]$ results in $\tilde{\boldsymbol{r}}_2 = [5,1,2,4,3]$. If the sorted simulated sequences for the locations are $\boldsymbol{z}_1 = [4.68,4.34,2.58,1.76,1.26]$ and $\boldsymbol{z}_2 = [5.53,5.27,4.34,2.75,1.52]$, respectively, the reshuffled sequences are then $\widehat{\boldsymbol{x}}_1 = [2.58,4.68,1.26,4.34,1.76]$ and $\widehat{\boldsymbol{x}}_2 = [1.52,5.53,5.27,2.75,4.34]$ following (2.2).

Although we introduce and illustrate the reshuffling above on a single realization of rainfall process of length $n$, it is suggested that the reshuffling procedure at each location be conducted over a relatively long time duration (Breinl et al., 2013), for example, over concatenated multiple synthetic realizations. This is because longer time series after reshuffling better resemble the original time series. The reshuffling approach matches the marginal distributions exactly and provides a satisfactory approximation for other statistical properties (Breinl et al., 2013).

## ***2.2. Deep learning models for time series forecasting***

Deep learning models for time series forecasting mirror models developed in the natural language processing (NLP) domain due to the sequential nature of both tasks. We focus on the Transformer architecture with an encoder–decoder structure, which has recently become prevalent in time series forecasting due to its superior performance over RNN- and MLP-based models (Bahdanau et al., 2014; Sutskever et al., 2014; Lai et al., 2018; Raffel et al., 2019; Zhou et al., 2021). The model architecture is shown in Figure 1. Within this structure, the encoder captures the dependencies and patterns inherent in the input sequence, subsequently conveying the extracted information to the decoder, which is tasked with generating predictions.

Let $\mathcal{T}^{\text{enc}} = \left[t_1,\ldots,t_{q_{\text{in}}^{\text{enc}}}\right]$ be a vector of increasing time stamps of length $q_{\text{in}}^{\text{enc}} < n$ and $\mathcal{X}^{\text{enc}} \in \mathbb{R}^{m \times q_{\text{in}}^{\text{enc}}}$ be the time series matrix where each row represents the stochastic process $X_i(t)$ for the $i^{\text{th}}$ location, $i = 1,\ldots,m$, at the time stamps indicated in $\mathcal{T}^{\text{enc}}$. Given $\mathcal{X}^{\text{enc}}$ and $\mathcal{T}^{\text{enc}}$, the deep learning model predicts the subsequent sequence $\mathcal{X}^{\text{out}} \in \mathbb{R}^{m \times q_{\text{out}}}$ at $q_{\text{out}}$ time stamps specified in $\mathcal{T}^{\text{out}} = \left[t_{q_{\text{in}}^{\text{enc}}+1},\ldots,t_{q_{\text{in}}^{\text{enc}}+q_{\text{out}}}\right]$. The

***Table 2.*** *Reshuffling of a hypothetical two-station example with five time stamps. (a) Samples of $\tilde{\boldsymbol{x}}(t)$ at $t_1,\ldots,t_5$ are generated from the resampling step described in Section 2.1.2, with $\tilde{\boldsymbol{r}}_1$ and $\tilde{\boldsymbol{r}}_2$ being the corresponding ranks; (b) Synthetic samples $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ are simulated from the marginal distribution at each location; (c) Samples are reshuffled according to the ranks $\tilde{\boldsymbol{r}}_1$ and $\tilde{\boldsymbol{r}}_2$*

(a) Samples from resampling

| Time stamp | $\tilde{\boldsymbol{x}}_1$ | $\tilde{\boldsymbol{x}}_2$ | $\tilde{\boldsymbol{r}}_1$ | $\tilde{\boldsymbol{r}}_2$ |
|---|---|---|---|---|
| $t_1$ | 2.14 | 0.51 | 3 | 5 |
| $t_2$ | 6.36 | 3.24 | 1 | 1 |
| $t_3$ | 0.64 | 2.46 | 5 | 2 |
| $t_4$ | 4.05 | 0.60 | 2 | 4 |
| $t_5$ | 1.31 | 2.00 | 4 | 3 |

(b) Sorted simulated samples

| $\boldsymbol{z}_1$ | $\boldsymbol{z}_2$ |
|---|---|
| 4.68 | 5.53 |
| 4.34 | 5.27 |
| 2.58 | 4.34 |
| 1.76 | 2.75 |
| 1.26 | 1.52 |

(c) Samples after reshuffling

| Time stamp | $\hat{\boldsymbol{x}}_1$ | $\hat{\boldsymbol{x}}_2$ | $\tilde{\boldsymbol{r}}_1$ | $\tilde{\boldsymbol{r}}_2$ |
|---|---|---|---|---|
| $t_1$ | 2.58 | 1.52 | 3 | 5 |
| $t_2$ | 4.68 | 5.53 | 1 | 1 |
| $t_3$ | 1.26 | 5.27 | 5 | 2 |
| $t_4$ | 4.34 | 2.75 | 2 | 4 |
| $t_5$ | 1.76 | 4.34 | 4 | 3 |

model proceeds by passing the inputs $\mathcal{X}^{\text{enc}}$, $\mathcal{T}^{\text{enc}}$ through the embedding layer (red block) to obtain the hidden representation $\mathcal{Z}^{\text{enc}}$. The hidden representation refers to the output matrices of the hidden layers, which all have dimension $d_{\text{model}}$, a hyperparameter determining the length of the hidden elements. The hidden representation is then updated through the layers of the encoder (gray block). Based on the resulting representation matrix $\mathcal{Z}^{\text{enc}}$ and $\mathcal{T}^{\text{out}}$, the decoder (green block) combined with a linear layer (purple block) performs generative inference to produce the output sequence $\mathcal{X}^{\text{out}}$ (highlighted in yellow).

In the following two subsections, we detail the embedding layer and the encoder-decoder structure, which are the two major components of this Transformer-based model.

### *2.2.1. Embedding layer*

The embedding layer serves to convert the inputs into a hidden representation which is then utilized by the encoder and decoder. It consists of two components, namely, the value embedding and time embedding, corresponding to the time series input $\mathcal{X} \in \mathbb{R}^{m \times q}$ and the time sequence $\mathcal{T}$ of length $q$, respectively.

The value embedding, inspired by dilated convolution networks (Li et al., 2024), is a 1D-convolutional layer with circular padding to map the spatial dimension $m$ to the hidden dimension $d_{\text{model}}$, preserving temporal resolution (Albawi et al., 2017). A kernel size of 3 is typically chosen to effectively capture short-term dependencies. We denote the operation of the value embedding applied to an input $\mathcal{X}$ by ValueEmbedding($\mathcal{X}$) which produces a hidden representation of dimension $d_{\text{model}} \times q$.

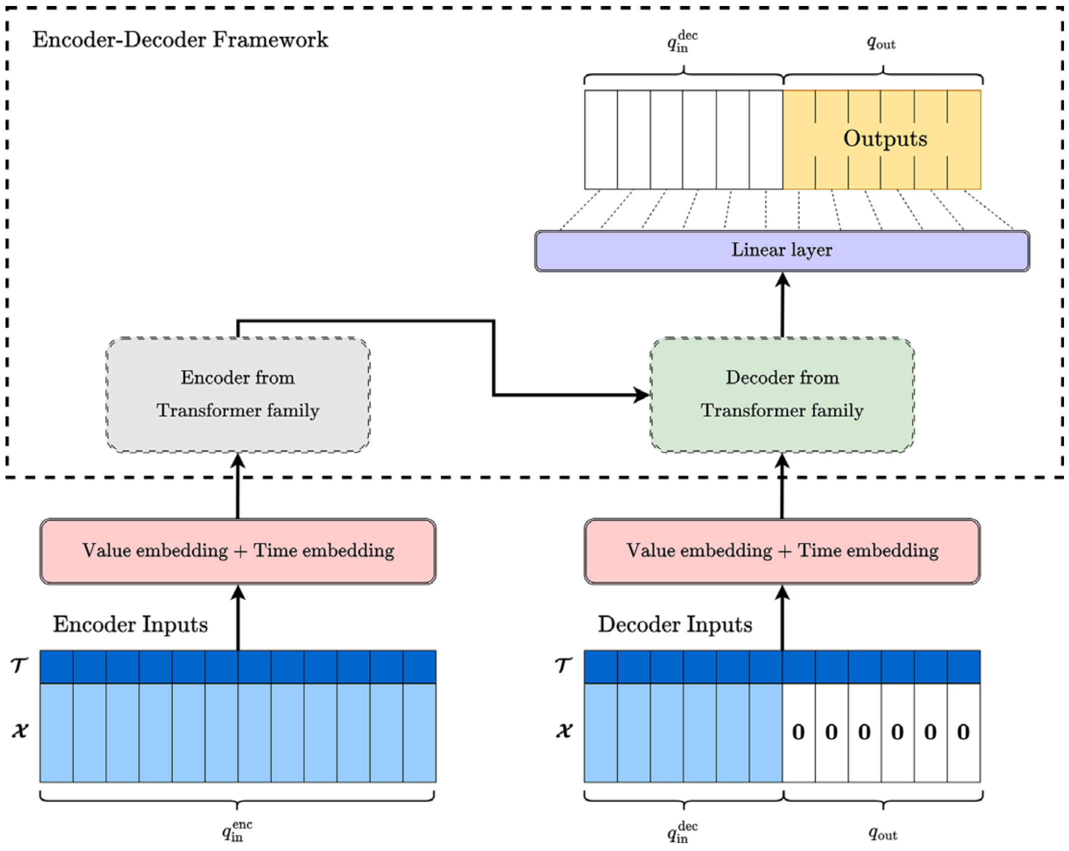

Encoder-Decoder Framework
$q_{\text{in}}^{\text{dec}}$
$q_{\text{out}}$
Outputs
Linear layer
Encoder from Transformer family
Decoder from Transformer family
Value embedding + Time embedding
Value embedding + Time embedding
Encoder Inputs
Decoder Inputs
$\mathcal{T}$
$\mathcal{X}$
0 0 0 0 0 0
$q_{\text{in}}^{\text{enc}}$
$q_{\text{in}}^{\text{dec}}$
$q_{\text{out}}$

***Figure 1.*** *Deep learning model architecture based on the encoder-decoder framework. The model processes inputs through an embedding layer (red block), generating the hidden representation which undergoes further updates in the encoder layers (gray block). The decoder (green block), in conjunction with a linear layer (purple block), utilizes the hidden representation from the encoder for generative inference, yielding the predicted sequence (highlighted in yellow).*

The time embedding, on the other hand, depends on whether or not the time argument is unitless. If the time argument is unitless, only the order of the sequence matters. Therefore, the positional embedding introduced in (Vaswani et al., 2017) is adopted as the time embedding. Given a time sequence of length $q$, the positional embedding returns a matrix of dimension $d_{\text{model}} \times q$, where the $(2k, j)$ and $(2k+1, j)$ entries, $k = 0, \ldots, \lfloor d_{\text{model}}/2 \rfloor - 1, j = 0, \ldots, q-1$, are defined as $\sin\left(j/\left(10000^{2k/d_{\text{model}}}\right)\right)$ and $\cos\left(j/\left(10000^{2k/d_{\text{model}}}\right)\right)$, respectively. In contrast, if the time argument contains unit information, that is, the data is in the format of year-month-day-hour-minute-second or a subset of any of these units, (Zhou et al., 2021) uses the time feature embedding instead to take the temporal information into consideration. To illustrate, if the time sequence is of length $q$ and data is recorded in the year-month-day-hour format, we reshape the $q$-dimensional time sequence into a $4 \times q$ matrix where each row records the standardized values for each unit of measure. A linear layer without bias is then applied to map the $4 \times q$ matrix to a $d_{\text{model}} \times q$ matrix. We denote the operation of the time embedding applied to an input $\mathcal{T}$ by TimeEmbedding($\mathcal{T}$) which produces a matrix of dimension $d_{\text{model}} \times q$.

The output representation $\mathcal{Z}$ from the embedding layer is thus defined as.

$$\mathcal{Z} = \text{ValueEmbedding}(\mathcal{X}) + \text{TimeEmbedding}(\mathcal{T}). \tag{2.3}$$

### 2.2.2. Encoder–decoder framework

The encoder–decoder framework is commonly used for sequence-to-sequence tasks, for example, time series forecasting and machine translation in the NLP domain. The work (Vaswani et al., 2017) has shown the effectiveness of adopting the self-attention mechanism as the primary building block in the encoder-decoder framework. Self-attention processes a sequence by replacing each element by a weighted average of the rest of the sequence so that the dependencies of each element with respect to the others in the sequence are learned. The encoder encapsulates all the information about the input sequence through multiple layers of self-attention and its output is fed to the decoder for generating predictions.

Self-attention, first proposed in Vaswani et al. (2017)), maps a query and a set of key-value pairs to obtain an output representation where the query, key, and value matrices stem from the input representation $\mathcal{Z} \in \mathbb{R}^{d_{\text{model}} \times q}$. Mathematically, self-attention is defined as.

$$\text{Self-Attention}(\mathcal{Z}) = \boldsymbol{V}\text{Softmax}\left(\frac{\boldsymbol{Q}^T\boldsymbol{K}}{\sqrt{d_{\text{model}}}}\right), \tag{2.4}$$

$$\begin{aligned} \boldsymbol{Q} &= \boldsymbol{W}_Q\mathcal{Z}, \\ \boldsymbol{K} &= \boldsymbol{W}_K\mathcal{Z}, \\ \boldsymbol{V} &= \boldsymbol{W}_V\mathcal{Z}, \end{aligned}$$

where $\boldsymbol{W}_Q, \boldsymbol{W}_K, \boldsymbol{W}_V \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ are the query, key, and value conversion matrices, $\text{Softmax}(\cdot)$ denotes the softmax function defined in Goodfellow et al. (2016)), and $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{d_{\text{model}} \times q}$ are the resulting query, key, and value matrices. As seen in (2.4), the output representation of the self-attention layer can be regarded as the weighted sum of the input values. For simplicity, we set the first dimension of the query and key matrices to be $d_{\text{model}}$, but in general, they only need to be compatible with each other.

The attention mechanism introduced above is single-headed as it only performs the mapping from the $d_{\text{model}}$-dimensional query to the $d_{\text{model}}$-dimensional key-value pairs once. As suggested in Vaswani et al. (2017)), it is beneficial to adopt a multi-head mechanism, that is, we perform $n_{\text{head}}$ such mappings in parallel with $d_{\text{model}}/n_{\text{head}}$-dimensional query and key-value pairs. The resulting output representations from all independent mappings are concatenated and further linearly projected to match the output dimension $d_{\text{model}}$. More advanced attention mechanisms have been developed in the deep learning community to improve the performance. For example, the prob-sparse attention mechanism, developed for Informer (Zhou et al., 2021), enhances computational efficiency through distillation so that each key attends only to dominant queries, reducing the quadratic computational complexity $O(q^2)$ of standard attention to $O(q\log q)$. The Autoformer (Wu et al., 2021) utilizes auto-correlation-attention instead of the standard attention mechanism. It introduces the notion of sub-series similarity based on the series periodicity and aggregates similar sub-series from underlying periods.

The encoder is designed to learn and extract the dependencies and patterns of the input sequence $\mathcal{X} = \mathcal{X}^{\text{enc}}$ across the temporal and spatial dimensions. It is composed of a stack of $n_{\text{enc}}$ identical blocks, generally consisting of two sub-layers each. The first is a multi-head attention layer described above while the second is a fully-connected feed-forward network with relu activation function (Hendrycks and Gimpel, 2016) and can be mathematically expressed as.

$$\text{FFN}(\mathcal{Z}) = \boldsymbol{W}_2\max(0, \boldsymbol{W}_1\mathcal{Z} + \boldsymbol{b}_1) + \boldsymbol{b}_2, \tag{2.5}$$

where $\mathcal{Z}$ is the input hidden representation with $q = q_{\text{in}}^{\text{enc}}$, $\boldsymbol{W}_1 \in \mathbb{R}^{d_{\text{ff}} \times d_{\text{model}}}$ and $\boldsymbol{W}_2 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{ff}}}$ are weight matrices with $d_{\text{ff}}$ being the dimension of the feed-forward layer, and $\boldsymbol{b}_1 \in \mathbb{R}^{d_{\text{ff}} \times q}$ and $\boldsymbol{b}_2 \in \mathbb{R}^{d_{\text{model}} \times q}$ are the bias matrices. Each sub-layer is succeeded by a layer normalization (Ba et al., 2016). Special techniques such as distillment, decomposition, etc, may be applied in between sub-layers depending on the choice of the attention mechanism. The final hidden representation of the encoder is then fed to the decoder.

On the other hand, the aim of the decoder is to perform generative inference on the output sequence $\mathcal{X}^{\text{out}}$ based on the time sequences $\mathcal{T}^{\text{out}}$. Similar to NLP applications wherein we apply a start token for dynamic decoding (Devlin et al., 2018), we initiate the inference with $\mathcal{X}_{\text{start}}^{\text{dec}} \in \mathbb{R}^{m \times q_{\text{in}}^{\text{dec}}}$ and $\mathcal{T}_{\text{start}}^{\text{dec}} \in \mathbb{R}^{q_{\text{in}}^{\text{dec}}}$

which are subsets of $\mathcal{X}^{\text{enc}}$ and $\mathcal{T}^{\text{enc}}$, respectively, representing the last $q_{\text{in}}^{\text{dec}}$ columns of the aforementioned matrices. The input of the decoder is the embedding (2.3) applied to the following matrices:

$$\mathcal{X} = \mathcal{X}^{\text{dec}} = \text{Concat}\left(\mathcal{X}_{\text{start}}^{\text{dec}}, \mathbf{0}\right), \tag{2.6}$$

$$\mathcal{T} = \mathcal{T}^{\text{dec}} = \text{Concat}\left(\mathcal{T}_{\text{start}}^{\text{dec}}, \mathcal{T}^{\text{out}}\right),$$

where Concat$(\cdot)$ denotes the matrix concatenation operation, $\mathbf{0} \in \mathbb{R}^{m \times q_{\text{out}}}$ is a zero matrix which serves as the placeholder of the output sequence $\mathcal{X}^{\text{out}}$, $\mathcal{X}^{\text{dec}} \in \mathbb{R}^{m \times \left(q_{\text{in}}^{\text{dec}} + q_{\text{out}}\right)}$, and $\mathcal{T}^{\text{dec}} \in \mathbb{R}^{\left(q_{\text{in}}^{\text{dec}} + q_{\text{out}}\right)}$. The structure of the decoder is similar to that of the encoder, which is composed of a stack of $n_{\text{dec}}$ identical blocks. In addition to the two sub-layers in each block, namely the attention layer and the feed-forward layer, the decoder includes a third sub-layer which performs multi-head cross-attention on the output of the encoder stacks to incorporate the learned dependencies and patterns of the input sequence. The cross-attention mechanism is a variant of the self-attention mechanism described above. It has the same structure as self-attention, except that the input representation to the key and value matrices is the output from the encoder instead of the output from the previous block in the decoder. Therefore, we have $\boldsymbol{Q} = \boldsymbol{W}_Q \mathcal{Z}^{\text{dec}} \in \mathbb{R}^{d_{\text{model}} \times \left(q_{\text{in}}^{\text{dec}} + q_{\text{out}}\right)}$, $\boldsymbol{K} = \boldsymbol{W}_K \mathcal{Z}^{\text{enc}} \in \mathbb{R}^{d_{\text{model}} \times q_{\text{in}}^{\text{enc}}}$, and $\boldsymbol{V} = \boldsymbol{W}_V \mathcal{Z}^{\text{enc}} \in \mathbb{R}^{d_{\text{model}} \times q_{\text{in}}^{\text{enc}}}$. The output representation from the cross-attention layer and that of the decoder has dimension $d_{\text{model}} \times \left(q_{\text{in}}^{\text{dec}} + q_{\text{out}}\right)$. It is then transformed to have size $m \times \left(q_{\text{in}}^{\text{dec}} + q_{\text{out}}\right)$ via a linear layer with bias. Only the last $q_{\text{out}}$ positions of the output sequence are of interest. Note that the inference procedure under the decoder is conducted in a single forward step instead of in an auto-regressive manner.

### 2.3. *Problem formulation*

Given the realizations $\boldsymbol{x}(t)$ of the stationary and ergodic process $\boldsymbol{X}(t), t \in [0, T_{\text{obs}}]$, with continuous marginal distributions, we aim to construct a stochastic generator to generate additional samples $\widehat{\boldsymbol{x}}(t)$ of $\boldsymbol{X}(t)$ for $t \in [0, T_{\text{sim}}]$ that preserve statistical properties of the existing realizations $\boldsymbol{x}(t)$. These synthetic realizations can then be employed in various downstream applications.

In this work, we consider the case where the marginal distributions of the components of $\boldsymbol{X}(t)$ are Gaussian without loss of generality. If the observed data $\boldsymbol{x}(t)$ of $\boldsymbol{X}(t)$ have components with non-Gaussian marginals, a transformation can be applied to the data during pre-processing. For each location $i$ with marginal cumulative distribution function (CDF) $F_i$, the transformed observations $x_i(t)$ are given by the invertible mapping $\Phi^{-1}[F_i(x_i(t))]$, where $\Phi$ is the standard Gaussian CDF. The marginal distribution $F_i$ can be characterized by parametric distributions as in Breinl et al. (2013)), by its empirical distribution, or by a mixture of both (Zhao et al., 2019).

We propose a stochastic generator that combines and extends the two models introduced in Sections 2.1 and 2.2. In particular, the Transformer-based deep learning model for time series forecasting is adopted as a critical component of this generator, facilitating scalability in scenarios where the number of spatial locations is large and the simulation horizon is long.

## 3. Stochastic generators for *m*-variate stochastic processes using deep learning models

We present GenFormer, a stochastic generator for $m$-variate stochastic processes using deep learning models. It extends the stochastic generator described in Section 2.1 in the following three aspects. First, we provide a generalized approach to define the state space of the univariate Markov sequence $Y_j$ by using $K$-means clustering, thereby extending the applicability of the model in Section 2.1.1 beyond precipitation data. Second, we replace the resampling procedure in Section 2.1.2 by a deep learning model which constitutes the mapping from the Markov states to the inferred values of the m-variate process. The deep learning model has the same encoder-decoder framework used for time series forecasting in Section 2.2 but with an additional embedding for the Markov states in the embedding layer. The deep learning model serves to improve the scalability of the resampling procedure. However, the fidelity of the resulting mapping in approximating statistical properties of interest depends on the performance of the deep learning model. Thus, to preserve statistical properties such as the spatial correlation and marginal distributions, we include a model post-processing step based on Cholesky decomposition and the reshuffling technique in Section 2.1.3 as the third extension. It is also worth-noting that since simulating univariate Markov sequences can become challenging for high Markov orders, an additional light-weight deep learning model is proposed to address this limitation.

The resulting synthetic data produced by GenFormer are able to exactly match the target marginal distributions and approximately match the second-moment properties. Moreover, it can also capture the higher-order statistical properties of quantities of interest derived from $\boldsymbol{X}(t)$. Most importantly, GenFormer can be applied to stochastic processes in which the number of locations is large and the simulation horizon is long.

The proposed approach is composed of two stages, namely model construction and model simulation, described in Sections 3.1 and 3.2, respectively. In Section 3.1, we focus on model training and computational aspects of the univariate Markov sequence generator based on $K$-means clustering and the deep learning model with Markov state embedding. In Section 3.2, we discuss how synthetic realizations can be generated using the aforementioned models coupled with the post-processing procedure. Section 3.3 summarizes the proposed GenFormer algorithm.

## *3.1. Model construction*

### *3.1.1. Construction of univariate Markov sequence via clustering*

Suppose we have $n_{\text{obs}}$ time stamps $t_1,\ldots,t_{n_{\text{obs}}}$ that evenly partition the duration $[0,T_{\text{obs}}]$ of the observed data. The countable state space of the univariate Markov sequence $Y_1,\ldots,Y_{n_{\text{obs}}}$ of the stochastic generator proposed in (Breinl et al., 2013) contains all the combinations of the wet and dry scenarios of the $m$ stations, resulting in $2^m$ Markov states. In general, there is no established rule to define the state space. We thus propose to partition the realizations $\boldsymbol{x}(t)$ of $\boldsymbol{X}(t)$ into subsets of interest, each indexed by a positive integer, which represents certain spatial variation of $\boldsymbol{X}(t)$.

We achieve this through $K$-means clustering (Hartigan and Wong, 1979) which is a commonly-used unsupervised learning algorithm to efficiently partition a set of vectors into distinct and non-overlapping clusters based on inherent similarities. It can thus be employed to segregate the set $\{\boldsymbol{x}(t_1),\ldots,\boldsymbol{x}(t_{n_{\text{obs}}})\}$ of realizations at $n_{\text{obs}}$ time stamps into $n_{\text{clusters}}$ clusters, where $n_{\text{clusters}}$ is a prescribed hyperparameter. Each cluster is represented by a centroid. The clustering algorithm is an iterative process of sample reassignment and centroid recalculation with the goal of minimizing within-cluster variance while maximizing between-cluster distance. The algorithm is sensitive to the initial choice of centroids and has to be performed with multiple sets of starting points. It results in each observation being allocated to a centroid which then corresponds to a state of the univariate Markov sequence.

We illustrate the application of $K$-means clustering in Table 3. Consider a 3-variate stochastic process $\boldsymbol{X}(t)$ with 5 time stamps. The components $X_1(t)$, $X_2(t)$, and $X_3(t)$ are highly-correlated with each other and all follow a bimodal distribution centered at 12 and 2. We set $n_{\text{clusters}}=2$. By applying $K$-means clustering, the Markov state $y_j$ is assigned to 1 and 2 at time stamps $t_j$ when the realizations $\boldsymbol{x}(t_j)$ are near the modes 12 and 2, respectively.

Given the mapped Markov state sequence $y_1,\ldots,y_{n_{\text{obs}}}$, the estimation of the Markov order $p$ along with the transition matrix is akin to the approach outlined in Section 2.1.1. Utilizing these estimates, the synthetic realizations of the Markov state sequence can be subsequently generated.

### *3.1.2. Deep learning model with Markov state embedding*

The resampling procedure introduced in Breinl et al. (2013) is specifically designed for rainy sequences and suffers from the curse of dimensionality. We therefore propose to train a Transformer-based deep learning model with Markov state embedding that maps the Markov state sequence $y_1,\ldots,y_{n_{\text{obs}}}$ to the realization of the $m$-variate process $\boldsymbol{x}(t_1),\ldots,\boldsymbol{x}(t_{n_{\text{obs}}})$. Let $\mathcal{Y}^{\text{enc}}\in\mathbb{R}^{q_{\text{in}}^{\text{enc}}}$ be a vector of the Markov state sequence corresponding to the vector of increasing time stamps $\mathcal{T}^{\text{enc}}$ and the matrix of observations $\mathcal{X}^{\text{enc}}$ at the specified time stamps. Given $\mathcal{X}^{\text{enc}}$, $\mathcal{T}^{\text{enc}}$, and $\mathcal{Y}^{\text{enc}}$, the deep learning model aims to infer the

***Table 3.*** *Illustrated mapping of Markov states to a 3-variate process based on K-means clustering. We have $y_j = 1$ when $x_1(t_j)$, $x_2(t_j)$, $x_3(t_j)$ are near 12, and $y_j = 2$ when $x_1(t_j)$, $x_2(t_j)$, $x_3(t_j)$ are near 2*

| Time stamp | $y$ | $x_1(t)$ | $x_2(t)$ | $x_3(t)$ |
|---|---|---|---|---|
| $t_1$ | 1 | 11.32 | 10.12 | 12.56 |
| $t_2$ | 1 | 10.54 | 11.98 | 14.12 |
| $t_3$ | 2 | 0.5 | 1.32 | 2.63 |
| $t_4$ | 2 | 1.8 | 3.24 | 2.12 |
| $t_5$ | 1 | 12.21 | 12.34 | 11.79 |

subsequent sequence $\mathcal{X}^{\text{out}}$ based on the corresponding Markov state sequence $\mathcal{Y}^{\text{out}}$ specified at time stamps in $\mathcal{T}^{\text{out}}$. The inputs of the model to perform inference starting from time stamp $j+1$ are as follows:

- $\mathcal{T}^{\text{enc}} = \left[t_{j-q_{\text{in}}^{\text{enc}}+1}, \dots, t_j\right] \in \mathbb{R}^{q_{\text{in}}^{\text{enc}}}$, a sequence of time stamps with recorded observations;
- $\mathcal{X}^{\text{enc}} = \left[\boldsymbol{x}\left(t_{j-q_{\text{in}}^{\text{enc}}+1}\right), \dots, \boldsymbol{x}(t_j)\right] \in \mathbb{R}^{m \times q_{\text{in}}^{\text{enc}}}$, a matrix of observations of $\boldsymbol{X}(t)$ at time stamps in $\mathcal{T}^{\text{enc}}$;
- $\mathcal{Y}^{\text{enc}} = \left[y_{j-q_{\text{in}}^{\text{enc}}+1}, \dots, y_j\right] \in \mathbb{R}^{q_{\text{in}}^{\text{enc}}}$, a Markov state sequence corresponding to $\mathcal{X}^{\text{enc}}$;
- $\mathcal{T}^{\text{out}} = \left[t_{j+1}, \dots, t_{j+q_{\text{out}}}\right] \in \mathbb{R}^{q_{\text{out}}}$, a sequence of time stamps for the inference stage;
- $\mathcal{Y}^{\text{out}} = \left[y_{j+1}, \dots, y_{j+q_{\text{out}}}\right] \in \mathbb{R}^{q_{\text{out}}}$, a Markov state sequence for the inference stage.

The output of the model is $\mathcal{X}^{\text{out}} = \left[\boldsymbol{x}(t_{j+1}), \dots, \boldsymbol{x}(t_{j+q_{\text{out}}})\right] \in \mathbb{R}^{m \times q_{\text{out}}}$, a time series matrix corresponding to the Markov state sequence $\mathcal{Y}^{\text{out}}$ and time stamps in $\mathcal{T}^{\text{out}}$.

In contrast to the deep learning model described in Section 2.2, the embedding layer also needs to incorporate information from the Markov state sequence $\mathcal{Y}$ in the hidden representation. We propose to include a Markov state embedding, wherein each Markov state is represented by a embedding vector of size $d_{\text{model}}$ learned in the training stage. The operation MarkovStateEmbedding($\mathcal{Y}$) retrieves the respective embedding vectors in the same order as the Markov states in $\mathcal{Y}$ via a dictionary mapping, culminating in a matrix of $d_{\text{model}} \times q$. Figure 2 updates the architecture shown in Figure 1 to include the proposed embedding for the Markov states. Given the time series input $\mathcal{X}$, the Markov state sequence $\mathcal{Y}$, and the time sequence $\mathcal{T}$, the output representation $\mathcal{Z} \in \mathbb{R}^{d_{\text{model}} \times q}$ from the embedding layer then becomes

$$\mathcal{Z} = \text{ValueEmbedding}(\mathcal{X}) + \text{MarkovStateEmbedding}(\mathcal{Y}) + \text{TimeEmbedding}(\mathcal{T}). \tag{3.1}$$

As in Section 2.2, the deep learning model adopts the encoder–decoder framework with self-attention mechanism or more advanced variants such as those employed in the Informer and Autoformer. The encoder is designed to extract the spatial and temporal patterns of $\mathcal{X}^{\text{enc}}$ and its relationship to $\mathcal{Y}^{\text{enc}}$ and $\mathcal{T}^{\text{enc}}$ from the previous $q_{\text{in}}^{\text{enc}}$ time stamps. The decoder then uses the extracted representation from the encoder to perform inference on $\mathcal{X}^{\text{out}}$ at subsequent time stamps with the decoder inputs.

$$\begin{aligned}
\mathcal{X}^{\text{dec}} &= \text{Concat}\left(\mathcal{X}_{\text{start}}^{\text{dec}}, \boldsymbol{0}\right), \\
\mathcal{Y}^{\text{dec}} &= \text{Concat}\left(\mathcal{Y}_{\text{start}}^{\text{dec}}, \mathcal{Y}^{\text{out}}\right), \\
\mathcal{T}^{\text{dec}} &= \text{Concat}\left(\mathcal{T}_{\text{start}}^{\text{dec}}, \mathcal{T}^{\text{out}}\right),
\end{aligned} \tag{3.2}$$

where $\mathcal{X}_{\text{start}}^{\text{dec}}$, $\mathcal{Y}_{\text{start}}^{\text{dec}}$, and $\mathcal{T}_{\text{start}}^{\text{dec}}$ are sub-matrices pertaining to the last $q_{\text{in}}^{\text{dec}}$ columns of $\mathcal{X}^{\text{enc}}$, $\mathcal{Y}^{\text{enc}}$, and $\mathcal{T}^{\text{enc}}$, respectively. The $\boldsymbol{0}$ matrix is a placeholder for the output sequence $\mathcal{X}^{\text{out}}$ which is later replaced by the outputs from the decoder.

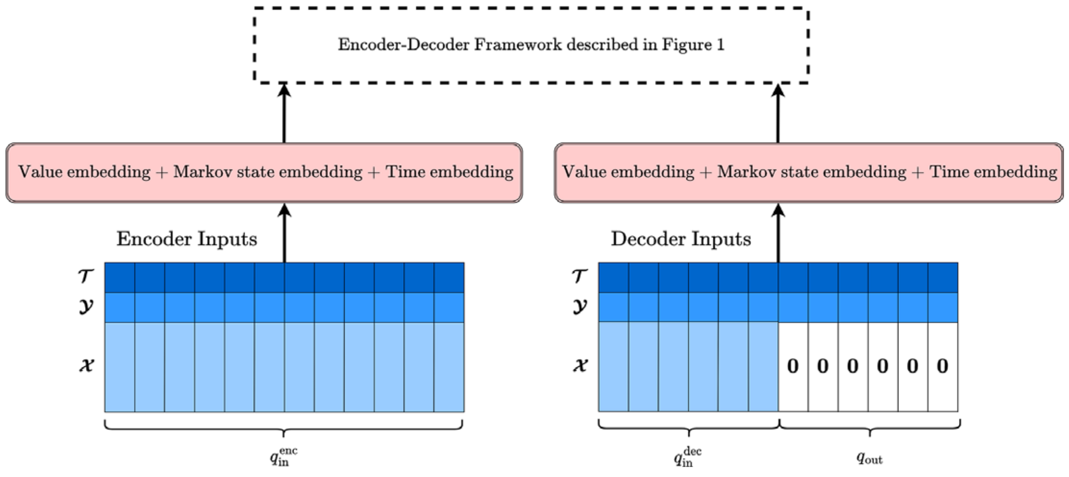

Encoder-Decoder Framework described in Figure 1
Value embedding + Markov state embedding + Time embedding
Value embedding + Markov state embedding + Time embedding
Encoder Inputs
Decoder Inputs
$\mathcal{T}$
$\mathcal{Y}$
$\boldsymbol{x}$
0 0 0 0 0 0
$q_{\text{in}}^{\text{enc}}$
$q_{\text{in}}^{\text{dec}}$
$q_{\text{out}}$

***Figure 2.*** *Transformer-based deep learning model with Markov state embedding. The proposed approach includes a Markov state embedding in addition to the value and time embedding present in the embedding layer. The remainder of the model architecture is the same as in Figure 1.*

Note that the hyperparameter $q_{\text{in}}^{\text{enc}}$ is not required to be the same as the Markov order $p$. We can determine $q_{\text{in}}^{\text{enc}}$ through hyperparameter tuning or by adhering to standard practice in time series analysis, where $q_{\text{in}}^{\text{enc}}$ is set as the empirical memory length of $\boldsymbol{X}(t)$ approximated by the time lag at which autocorrelations for all components of $\boldsymbol{X}(t)$ fall below a specified threshold. The resulting $q_{\text{in}}^{\text{enc}}$ is greater than $p$ in general. Similarly, setting $q_{\text{out}} = 1$ is not obligatory. A larger $q_{\text{out}}$ can significantly accelerate the inference by generating samples at $q_{\text{out}}$ time stamps at once instead of repeating the inference $q_{\text{out}}$ times. On the other hand, $q_{\text{out}}$ cannot be too large since this may potentially hinder the accuracy of the model, as will be discussed in Section 3.1.4. For simplicity, $q_{\text{out}}$ is typically set to $q_{\text{in}}^{\text{enc}}$ or $q_{\text{in}}^{\text{enc}}/2 > 1$ when not employing hyperparameter tuning. The proposed deep learning model is trained using the available realizations $\boldsymbol{x}(t)$ of $\boldsymbol{X}(t)$.

### *3.1.3. Deep learning model for Markov state sequence generation*

In order to infer the $m$-variate process, the decoder in Section 3.1.2 requires a synthetic realization of the Markov state sequence $\mathcal{Y}^{\text{out}}$. Synthetic realizations of the Markov state sequence can be obtained based on the specified transition matrix with Markov order $p$, which can be estimated from the available realizations of the mapped Markov state sequence $y_1, y_2, \ldots, y_{n_{\text{obs}}}$. While this is feasible for $p = 1$, the estimation of the transition matrix when $p \geq 2$ may be challenging due to the exponential growth of the transition matrix dimension given by $n_{\text{clusters}}^p \times n_{\text{clusters}}$. When $n_{\text{clusters}}$ and $p$ are large, an accurate estimation of the transition matrix can be computationally intensive, or even prohibitive, and requires a significant amount of data to avoid obtaining a sparse matrix which can restrict the generation of unprecedented sequences. We address this limitation by using a light-weight deep learning model, referred to as the deep learning model for Markov state sequence generation. Such model takes the Markov states at the previous $p$ time stamps as the input, calculates the probability of each of the $n_{\text{clusters}}$ states, and samples a realization of the Markov state according to these probabilities for the next time stamp.

The architecture of the light-weight model is shown in Figure 3. The model only adopts a decoder structure which takes as input the Markov states in the previous $p$ time stamps that is concatenated by a placeholder represented by a vector of length 1. This is followed by a Markov state embedding and time embedding (red block). The resulting hidden representation is then fed into $n_{\text{Markov}}$ blocks of the decoder (green block), wherein each block contains a multi-headed attention layer and a feed-forward layer without the cross attention. This process yields a vector of size $n_{\text{clusters}}$ (white block), with each component representing the weight of the respective Markov state in the state space. These weights are then

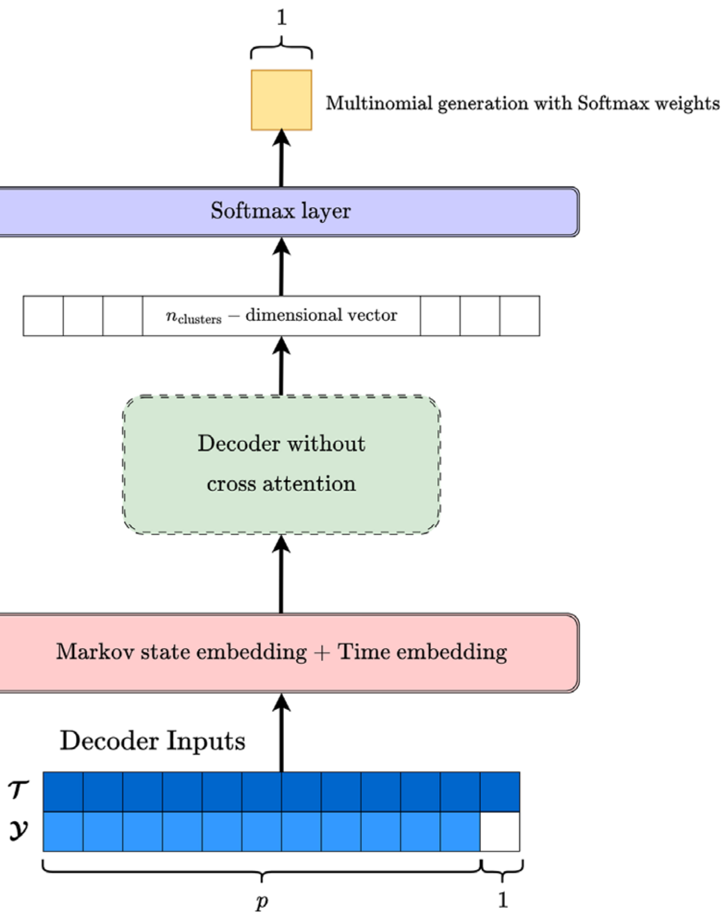

1
Multinomial generation with Softmax weights
Softmax layer
$n_{\text{clusters}}$ – dimensional vector
Decoder without cross attention
Markov state embedding + Time embedding
Decoder Inputs
$\mathcal{T}$
$\mathcal{Y}$
$p$
1

***Figure 3.*** *Deep learning model for Markov state sequence generation when Markov order p ≥ 2. We adopt a decoder-only structure without cross attention mechanism. The input of the model is the Markov states in the previous p time stamps concatenated by a vector of length 1. This is passed to an embedding layer and multiple decoder blocks. The Softmax layer normalizes the weights of Markov states to obtain probabilities which the multinomial random variable generator utilizes to generate synthetic Markov states.*

normalized using a softmax layer (purple block) to obtain the probability for each Markov state. Finally, a Markov state is simulated from a multinomial random variable generator based on these probabilities. The training of this model is based on the realizations of $y_1, \ldots, y_{n_{\text{obs}}}$.

### *3.1.4. Computational aspects of training deep learning models*

Sections 3.1.2 and 3.1.3 discussed the architectures of the deep learning models we utilize in this work. In this subsection, we focus on the computational aspects for the practical implementation of these models.

**Training and validation datasets**. Given the realizations at $n_{\text{obs}}$ time stamps $\boldsymbol{x}(t_1), \ldots, \boldsymbol{x}(t_{n_{\text{obs}}})$, the training and validation datasets are partitioned as follows. Let $\eta$ be the proportion of data to be allocated to the training set. For each sequence of the given realizations, the values at the first $\lfloor \eta n_{\text{obs}} \rfloor$ time stamps will be assigned to the training set, with the remainder forming the validation set. To construct the input–output data pairs for each dataset, we consider a sliding window of length $q_{\text{in}}^{\text{enc}} + q_{\text{out}}$, applied to the time series matrix $\mathcal{X}$ as well as the vectors $\mathcal{Y}$ and $\mathcal{T}$ of the Markov state and time sequences, as shown in Figure 4. To illustrate, applying this procedure to $\mathcal{X}$, we obtain $\left[\boldsymbol{x}(t_1), \ldots, \boldsymbol{x}\left(t_{q_{\text{in}}^{\text{enc}}+q_{\text{out}}}\right)\right]$, $\left[\boldsymbol{x}(t_2), \ldots, \boldsymbol{x}\left(t_{q_{\text{in}}^{\text{enc}}+q_{\text{out}}+1}\right)\right]$, …,

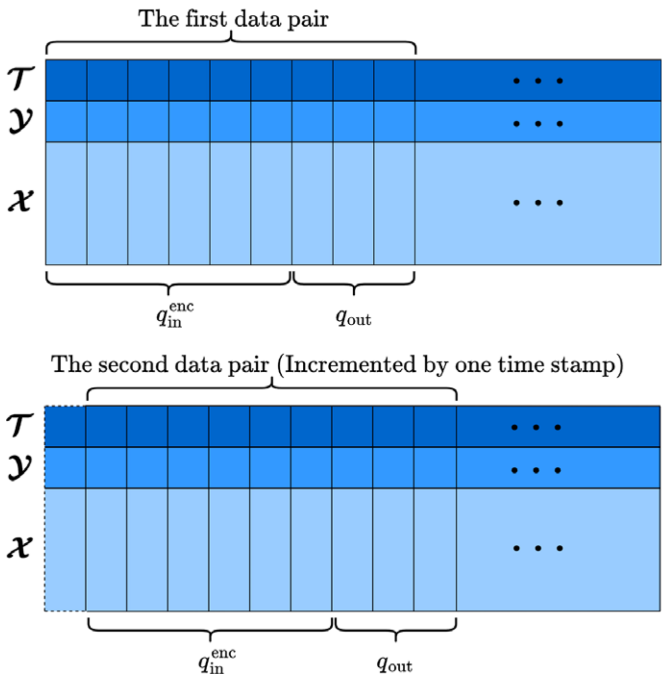


***Figure 4.*** *Construction of input–output data pairs. For each sequence of realizations, we apply a sliding window of length $q_{\text{in}}^{\text{enc}} + q_{\text{out}}$ to the time series matrix $\mathcal{X}$ and the vectors $\mathcal{Y}$ and $\mathcal{T}$ of Markov state and time sequences. The first $q_{\text{in}}^{\text{enc}}$ components of the window are inputs to the deep learning model while the subsequent $q_{\text{out}}$ components constitute the target output sequence for the model.*

$\left[\boldsymbol{x}\left(t_{\lfloor \eta n_{\text{obs}} \rfloor - q_{\text{in}}^{\text{enc}} - q_{\text{out}} + 1}\right), \dots, \boldsymbol{x}\left(t_{\lfloor \eta n_{\text{obs}} \rfloor}\right)\right]$ for training and $\left[\boldsymbol{x}\left(t_{\lfloor \eta n_{\text{obs}} \rfloor + 1}\right), \dots, \boldsymbol{x}\left(t_{\lfloor \eta n_{\text{obs}} \rfloor + q_{\text{in}}^{\text{enc}} + q_{\text{out}}}\right)\right]$, …, $\left[\boldsymbol{x}\left(t_{n_{\text{obs}} - q_{\text{in}}^{\text{enc}} - q_{\text{out}} + 1}\right), \dots, \boldsymbol{x}\left(t_{n_{\text{obs}}}\right)\right]$ for validation. The first $q_{\text{in}}^{\text{enc}}$ $m$-dimensional vectors are inputs to the deep learning model, while the subsequent $q_{\text{out}}$ vectors constitute the target output sequence for the model. For $\mathcal{Y}$ and $\mathcal{T}$, the $q_{\text{in}}^{\text{enc}} + q_{\text{out}}$ components all become the model inputs. There are in total $\lfloor \eta n_{\text{obs}} \rfloor - q_{\text{in}}^{\text{enc}} - q_{\text{out}} + 1$ and $(n_{\text{obs}} - \lfloor \eta n_{\text{obs}} \rfloor) - q_{\text{in}}^{\text{enc}} - q_{\text{out}} + 1$ input–output pairs obtained from each sequence of realizations for the training and validation datasets. Note that by splitting the data first and then constructing the input–output pairs, it is ensured that there is no data leakage between training and validation datasets. We also observe that the choice of $q_{\text{in}}^{\text{enc}}$ and $q_{\text{out}}$ leads to a trade-off between model accuracy and the computational efficiency of the inference procedure. Larger values of $q_{\text{in}}^{\text{enc}}$ and $q_{\text{out}}$ allow for reduced iterations in the auto-regressive inference of $\boldsymbol{x}(t)$, leading to less computational effort. However, there are fewer input–output pairs in the training dataset which may potentially hinder the model accuracy. Conversely, smaller values of these hyperparameters increase the number of iterations in inference, while providing a greater number of input–output pairs for model training. At every training iteration, a batch of data pairs are used to update the model weights. The training and validation datasets for the deep learning model for Markov state sequence generation in Section 3.1.3 are constructed in a similar manner, but they only consist of the data pairs of the Markov state and time sequences.

**Loss function**. For the model in Section 3.1.2, we use the $L_1$ or $L_2$ loss function, also known as the mean absolute error (MAE) or mean square error (MSE), to penalize the discrepancy between the target sequence and inference of the time series by the deep learning model. For the model in Section 3.1.3, we use the focal loss function (Lin et al., 2017), which is an extension of the cross entropy loss and applies a modulating term in order to focus learning on hard misclassified samples.

**Masking**. In the training stage, (Vaswani et al., 2017) suggests to apply triangular masks in the attention layers. This prevents each element of the sequence from attending to the elements at future times. The masking aims to mimic real scenarios in which information at future times is not available.

**Optimizer**. We use the ADAM optimizer (Kingma and Ba, 2015) because of its competitive performance in deep learning applications. The learning rate is set to decay after every few epochs.

**Regularization**. Regularization serves to prevent the deep learning model from overfitting. We utilize the dropout technique in training the model weights across all layers such that a proportion of these weights are not updated in every training batch. In addition, we adopt early stopping so that the training is automatically terminated if the validation loss is not decreasing over a certain number of epochs. This ensures that the training and validation losses are comparable which promotes the generalizability of the model to new data.

**Hyperparameter tuning**. The aforementioned architecture hyperparameters such as $n_{\text{enc}}$, $n_{\text{dec}}$ and the hyperparameters used in the training such as the dropout rate and learning rate can be selected by hyperparameter tuning based on a test dataset. This can be achieved by grid search, greedy search, or more advanced Bayesian algorithms (Frazier, 2018).

Table 4 lists the standard hyperparameters of the proposed GenFormer model that are subject to hyperparameter tunning.

### *3.2. Model simulation*

In the previous subsection, we discussed constructing a univariate Markov sequence via clustering and training a deep learning model with Markov state embedding based on the available observations. Here, we present the simulation methodology of GenFormer with the aim of generating new synthetic realizations of the $m$-variate process over $t \in [0, T_{\text{sim}}]$ for $n_{\text{sim}}$ time stamps. It consists of three steps. First, a synthetic realization $\tilde{y}_1, \ldots, \tilde{y}_{n_{\text{sim}}}$ of the univariate Markov sequence $Y_1, \ldots, Y_{n_{\text{sim}}}$ is produced. Subsequently, the realization $\tilde{y}_1, \ldots, \tilde{y}_{n_{\text{sim}}}$ is mapped to the preliminary synthetic realization $\tilde{\boldsymbol{x}}(t_1), \ldots, \tilde{\boldsymbol{x}}(t_{n_{\text{sim}}})$ of the $m$-variate stochastic process by utilizing the deep learning model with Markov state embedding. Since we want to preserve statistics of interest such as the spatial correlation matrix and marginal distributions, the final step involves model post-processing via Cholesky decomposition and the reshuffling technique.

Simulating from the univariate Markov sequence and the deep learning model requires initial data. For Markov state sequence generation, the Markov states at the first $p$ time stamps have to be specified. The deployment of the deep learning model requires the initial data $\mathcal{X}^{\text{enc}}$, $\mathcal{Y}^{\text{enc}}$, and $\mathcal{T}^{\text{enc}}$ measured at the first $q_{\text{in}}^{\text{enc}}$ time points. Set $q_{\max} = \max\left(p, q_{\text{in}}^{\text{enc}}\right)$. We therefore assume that data on the $m$-variate process $\tilde{\boldsymbol{x}}(t_1), \ldots, \tilde{\boldsymbol{x}}\left(t_{q_{\max}}\right)$ along with the corresponding Markov state sequence $\tilde{y}_1, \ldots, \tilde{y}_{q_{\max}}$ at time stamps $t_1, \ldots, t_{q_{\max}}$ are available. To obtain such data, we randomly select a subsequence $\boldsymbol{x}(t_1), \ldots, \boldsymbol{x}\left(t_{q_{\max}}\right)$ of length $q_{\max}$ and its corresponding Markov states $y_1, \ldots, y_{q_{\max}}$ from the given observations.

***Table 4.*** *Standard hyperparameters of the GenFormer model for tunning*

| Notation | Description |
|---|---|
| $d_{\text{model}}$ | Dimension of the hidden embedding and attention layers |
| $d_{\text{ff}}$ | Dimension of the hidden feed-forward network |
| $n_{\text{head}}$ | Number of heads in the attention mechanism |
| $n_{\text{enc}}$ | Number of encoder blocks in the deep learning model for inference of $m$-variate processes |
| $n_{\text{dec}}$ | Number of decoder blocks in the deep learning model for inference of $m$-variate processes |
| $n_{\text{Markov}}$ | Number of decoder blocks in the deep learning model for Markov state sequence generation |

### *3.2.1. Simulation of m-variate stochastic processes*

**Simulation of a univariate Markov sequence**. Given the data $\tilde{\boldsymbol{x}}(t_1),\dots,\tilde{\boldsymbol{x}}(t_{q_{\max}})$ with the corresponding Markov states $\tilde{y}_1,\dots,\tilde{y}_{q_{\max}}$ at time stamps $t_1,\dots,t_{q_{\max}}$, sampling the univariate Markov sequence is initialized using the last $p$ Markov states $\tilde{y}_{q_{\max}-p+1},\dots,\tilde{y}_{q_{\max}}$. For Markov order $p=1$, we utilize the estimated transition matrix to simulate a sample $\tilde{y}_{q_{\max}+1}$, which is repeated recursively to produce $\tilde{y}_{q_{\max}+2},\dots,\tilde{y}_{n_{\text{sim}}}$. For $p\geq 2$, we adopt the trained deep learning model for Markov state sequence generation in Section 3.1.3. In the first iteration, the model takes $\tilde{y}_{q_{\max}-p+1},\dots,\tilde{y}_{q_{\max}}$ and $t_{q_{\max}-p+1},\dots,t_{q_{\max}}$ as inputs to produce the sample $\tilde{y}_{q_{\max}+1}$ using the multinomial layer. At iteration $l$, the sequences $\tilde{y}_{q_{\max}-p+l},\dots,\tilde{y}_{q_{\max}+l-1}$ and $t_{q_{\max}-p+l},\dots,t_{q_{\max}+l-1}$ are then fed to the model to simulate $\tilde{y}_{q_{\max}+l}$. This process is repeated to generate $\tilde{y}_{q_{\max}+1},\tilde{y}_{q_{\max}+2},\dots,\tilde{y}_{n_{\text{sim}}}$.

**Inference from the deep learning model with Markov state embedding**. The second step is to infer the $m$-variate stochastic process $\tilde{\boldsymbol{x}}(t_{q_{\max}+1}),\dots,\tilde{\boldsymbol{x}}(t_{n_{\text{sim}}})$ corresponding to the synthetic realization $\tilde{y}_{q_{\max}+1},\dots,\tilde{y}_{n_{\text{sim}}}$ using the deep learning model in Section 3.1.2. Typically, we have $q_{\text{out}}\ll n_{\text{sim}}-q_{\max}$ which implies that the inference needs to be performed in an auto-regressive manner. In the $l^{\text{th}}$ iteration, we infer $\tilde{\boldsymbol{x}}(t_{q_{\max}+(l-1)q_{\text{out}}+1}),\dots,\tilde{\boldsymbol{x}}(t_{q_{\max}+lq_{\text{out}}})$. This auto-regressive procedure is repeated $\lfloor (n_{\text{sim}}-q_{\max})/q_{\text{out}}\rfloor$ times.

### *3.2.2. Model post-processing*

The deep learning model alone cannot fully preserve statistics of interest. We therefore introduce a model post-processing procedure. It is composed of a transformation based on Cholesky decomposition and the reshuffling technique to correct the spatial correlation matrix and marginal distributions, respectively, of the simulated realizations.

Let $\tilde{\mathcal{X}}=[\tilde{\boldsymbol{x}}(t_1),\dots,\tilde{\boldsymbol{x}}(t_{n_{\text{sim}}})]\in\mathbb{R}^{m\times n_{\text{sim}}}$ be the time series matrix of preliminary synthetic realizations of $\boldsymbol{X}(t)$ produced by the deep learning model during inference. The spatial correlation matrix $\tilde{\boldsymbol{C}}$ of the inference from the deep learning model can be estimated by $\tilde{\boldsymbol{C}}=\tilde{\mathcal{X}}\tilde{\mathcal{X}}^T/n_{\text{sim}}\in\mathbb{R}^{m\times m}$ due to stationarity and ergodicity. Similarly, let $\mathcal{X}=[\boldsymbol{x}(t_1),\dots,\boldsymbol{x}(t_{n_{\text{obs}}})]\in\mathbb{R}^{m\times n_{\text{obs}}}$ be the matrix of observations of $\boldsymbol{X}(t)$. The target spatial correlation $\boldsymbol{C}$ of $\boldsymbol{X}(t)$ is approximated via $\boldsymbol{C}=\mathcal{X}\mathcal{X}^T/n_{\text{obs}}$. The accuracy of the approximation $\tilde{\boldsymbol{C}}$ with respect to the target $\boldsymbol{C}$ is contingent upon the accuracy of the trained deep learning model. Consequently, we apply a transformation based on Cholesky decomposition to reduce the discrepancy between $\tilde{\boldsymbol{C}}$ and $\boldsymbol{C}$.

Since the spatial correlation matrix $\tilde{\boldsymbol{C}}$ is positive semi-definite, the Cholesky decomposition of $\tilde{\boldsymbol{C}}$ is given by $\tilde{\boldsymbol{C}}=\tilde{\boldsymbol{L}}\tilde{\boldsymbol{L}}^T$, where $\tilde{\boldsymbol{L}}\in\mathbb{R}^{m\times m}$ is a unique lower triangular matrix (Nicholas, 1990). Likewise, $\boldsymbol{C}=\boldsymbol{L}\boldsymbol{L}^T$ for some lower triangular matrix $\boldsymbol{L}$. To correct the spatial correlation matrix, we apply the transformation $\tilde{\boldsymbol{U}}=\boldsymbol{L}\tilde{\boldsymbol{L}}^T\tilde{\mathcal{X}}$. The updated matrix $\tilde{\boldsymbol{U}}$ has the spatial correlation matrix $\boldsymbol{C}$ since.

$$\tilde{\boldsymbol{U}}\tilde{\boldsymbol{U}}^T/n_{\text{sim}}=\boldsymbol{L}\tilde{\boldsymbol{L}}^T\tilde{\mathcal{X}}\tilde{\mathcal{X}}^T\tilde{\boldsymbol{L}}\boldsymbol{L}^T/n_{\text{sim}}=\boldsymbol{L}\tilde{\boldsymbol{L}}^T\tilde{\boldsymbol{C}}\tilde{\boldsymbol{L}}\boldsymbol{L}^T=\boldsymbol{L}\boldsymbol{I}\boldsymbol{L}^T=\boldsymbol{C}$$

where $\boldsymbol{I}\in\mathbb{R}^{m\times m}$ is the identity matrix.

The reshuffling technique discussed in Section 2.1.3 is then employed to rectify the marginal distributions. It is applied to the matrix $\tilde{\boldsymbol{U}}\in\mathbb{R}^{m\times n_{\text{sim}}}$. We update each row $\tilde{\boldsymbol{u}}_i\in\mathbb{R}^{n_{\text{sim}}}, i=1,\dots,m,$ of $\tilde{\boldsymbol{U}}$ by first simulating $n_{\text{sim}}$ samples from the standard Gaussian distribution, then reshuffling these samples according to the ranks of $\tilde{\boldsymbol{u}}_i$, resulting in the final realization $\hat{\boldsymbol{x}}_i\in\mathbb{R}^{n_{\text{sim}}}$ for location $i$. It can be shown that the synthetic realization $\hat{\boldsymbol{x}}_i$ at location $i$ has the standard Gaussian distribution. However, this does not guarantee that the spatial correlation matrix resulting from the transformation based on Cholesky decomposition is preserved. Nevertheless, if $n_{\text{sim}}$ is sufficiently large, the reshuffled time series closely approximates the original, thereby minimizing the discrepancy in the spatial correlation matrix. The $j^{\text{th}}$ column of the matrix $\left[\hat{\boldsymbol{x}}_1^T,\dots,\hat{\boldsymbol{x}}_m^T\right]^T\in\mathbb{R}^{m\times n_{\text{sim}}}$ is the final realization $\hat{\boldsymbol{x}}(t_j)$.

**Algorithm 1** GenFormer model.

```
Model construction phase
1: Transform the data $\boldsymbol{x}(t_1),\ldots,\boldsymbol{x}(t_{n_{\text{obs}}})$ to have standard Gaussian marginal distributions
2: Partition the data into $n_{\text{clusters}}$ clusters via $K$-means clustering in Section 3.1.1 to obtain the Markov
   state sequence$y_1,\ldots,y_{n_{\text{obs}}}$
3: if Markov order $p = 1$ then
4:    Estimate the Markov transition matrix from the realizations of$y_1,\ldots,y_{n_{\text{obs}}}$
5: else if Markov order $p \geq 2$ then
6:    Train the deep learning model for Markov state sequence generation in Section 3.1.3
7: Train the deep learning model with Markov state embedding in Section 3.1.2 using the data
   $\boldsymbol{x}(t_1),\ldots,\boldsymbol{x}(t_{n_{\text{obs}}})$ and$y_1,\ldots,y_{n_{\text{obs}}}$
Model simulation phase
8: Set $q_{\max} = \max(p, q_{\text{in}}^{\text{enc}})$. Initialize the simulation by setting $\tilde{\boldsymbol{x}}(t_1),\ldots,\tilde{\boldsymbol{x}}(t_{q_{\max}})$ and $\tilde{y}_1,\ldots,\tilde{y}_{q_{\max}}$ to a
   randomly chosen subsequence from the given data and its respective Markov state sequence
9: if Markov order $p = 1$ then
10:   Simulate $\tilde{y}_{q_{\max}+1},\ldots,\tilde{y}_{n_{\text{sim}}}$ using the estimated transition matrix
11: else if Markov order $p \geq 2$ then
12:   Simulate $\tilde{y}_{q_{\max}+1},\ldots,\tilde{y}_{n_{\text{sim}}}$ using the trained deep learning model for Markov state sequence generation
      according to Section 3.2.1
13: Apply the trained deep learning model with Markov state embedding on $\tilde{y}_{q_{\max}+1},\ldots,\tilde{y}_{n_{\text{sim}}}$ to infer-
    $\tilde{\boldsymbol{x}}(t_{q_{\max}+1}),\ldots,\tilde{\boldsymbol{x}}(t_{n_{\text{sim}}})$
14: Correct spatial correlation by applying transformation based on Cholesky decomposition in
    Section 3.2.2 to the synthetic realizations
15: Correct the marginal distributions using the reshuffling technique in Section 2.1.3
```

### 3.3. Summary of the proposed GenFormer algorithm

We summarize the proposed GenFormer algorithm for generating synthetic realizations of a multivariate stochastic process in Algorithm 1. It is composed of the training stage and the simulation procedure. To construct the GenFormer model, the data $\boldsymbol{x}(t_1),\ldots,\boldsymbol{x}(t_{n_{\text{obs}}})$ is first transformed to have standard Gaussian marginal distributions. The $K$-means clustering algorithm in Section 3.1.1 is then employed to partition the data into $n_{\text{clusters}}$ clusters from which we deduce the Markov state sequence $y_1,\ldots,y_{n_{\text{obs}}}$. To fit a Markov process to the resulting state sequence, the Markov order is determined. If the order is $p = 1$, we estimate the transition matrix whereas if $p \geq 2$, we train the deep learning model for Markov state sequence generation in Section 3.1.3. Finally, the deep learning model with Markov state embedding is trained using $\boldsymbol{x}(t_1),\ldots,\boldsymbol{x}(t_{n_{\text{obs}}})$ and $y_1,\ldots,y_{n_{\text{obs}}}$ following Section 3.1.2.

To produce synthetic realizations using GenFormer, we initialize the simulation procedure by setting $\tilde{\boldsymbol{x}}(t_1),\ldots,\tilde{\boldsymbol{x}}(t_{q_{\max}})$ and $\tilde{y}_1,\ldots,\tilde{y}_{q_{\max}}$ to a randomly-chosen subsequence from the given observations. We then simulate the Markov state sequence at future times $\tilde{y}_{q_{\max}+1},\ldots,\tilde{y}_{n_{\text{sim}}}$ using the estimated Markov transition matrix if $p = 1$ or using the trained deep learning model for Markov state sequence generation if $p \geq 2$, following Section 3.2.1. Preliminary synthetic realizations $\tilde{\boldsymbol{x}}(t_{q_{\max}+1}),\ldots,\tilde{\boldsymbol{x}}(t_{n_{\text{sim}}})$ of the stochastic process are then obtained by applying the trained deep learning model with Markov state embedding on $\tilde{y}_{q_{\max}+1},\ldots,\tilde{y}_{n_{\text{sim}}}$. Synthetic realizations of $\boldsymbol{X}(t)$ then result from post-processing the preliminary synthetic realizations to correct the spatial correlation and the marginal distributions as discussed in Section 3.2.2.

The synthetic realizations generated by GenFormer match exactly the target marginal distributions and match approximately the second-moment properties. Furthermore, the estimates based on these realizations provide satisfactory approximations to the higher-order statistical properties derived from $\boldsymbol{X}(t)$, even when $m$ and $n_{\text{sim}}$ are large. We demonstrate these properties in Section 4.

## 4. Numerical examples

We apply the proposed GenFormer model to two examples. We first consider a synthetic dataset describing the dynamics of a 3-variate process generated from stochastic differential equations (SDE) in Section 4.1. Analytical expressions for the statistical properties of the solutions to the SDE can be derived. We then consider a real dataset on wind speeds at 6 stations in Florida in Section 4.2. The numerical results illustrate the following points:

- The GenFormer model is scalable as the produced synthetic data can be used to obtain satisfactory approximations of the statistical properties of interest such as exceedance probabilities of functionals of $\boldsymbol{X}(t)$, even when the numbers of locations and time stamps are large.
- The post-processing procedure of the GenFormer results in a model that exactly matches the target marginal distributions and reasonably approximates the spatial correlation matrix.
- For large Markov order $p$, the deep learning model for Markov state sequence generation is able to simulate synthetic Markov state sequences with comparable frequencies to the observed ones which is a challenge using traditional methods.
- The Markov state embedding incorporated in the deep learning model produces an accurate mapping to infer $m$-variate processes from Markov state sequences.

In this work, we conduct a limited hyperparameter search for the hyperparameters listed in Table 4. We set candidate values of $512, 1024, 2048$ for $d_{\text{model}}$ and $d_{\text{ff}}$, and $3, 4$ for $n_{\text{enc}}$, $n_{\text{dec}}$, and $n_{\text{Markov}}$. The configurations yielding the lowest validation losses are selected. Other hyperparameters such as $p$, $q_{\text{in}}^{\text{enc}}$ are set based on empirical judgments or standard practices detailed in Section 3.1.2. This approach is adopted in response to the satisfactory performance already observed with the trained model. However, if adequate computational resources are available, more extensive hyperparameter tuning is recommended to achieve better results. The deep learning models for Markov state sequence generation and inference of the $m$-variate processes are trained based on the focal and $L_1$ losses, respectively, with batch size 128. The ADAM optimizer (Kingma and Ba, 2015) is used with an initial learning rate of $10^{-4}$ which is set to decay during training. The maximum number of training epochs is 20, where the learning rate is set to be $1e^{-5}$, $5e^{-6}$, $1e^{-6}$, $5e^{-7}$ at epochs 6, 8, 10, 12, respectively. The training process is stopped early if the validation loss does not decrease over three consecutive iterations. We also employ a 5% dropout to prevent overfitting. A Python[1] implementation of the code using PyTorch (Paszke et al., 2019) is available. Both experiments were run on a single A100 GPU. The run time for model construction is approximately 30 minutes and the simulation of synthetic realizations can be completed within 1 minute.

To evaluate the efficacy of the Transformer-based architecture for mapping Markov state sequences to time series values, as detailed in Section 3.1.2, we also train an MLP and an RNN-based model, that is, Long Short-Term Memory (LSTM), to benchmark the performance of the Transformer architecture. For a fair comparison and ensuring no information loss, the inputs of the MLP comprise the concatenated time sequence $\text{Concat}(\mathcal{T}^{\text{enc}}, \mathcal{T}^{\text{out}})$, Markov state sequence $\text{Concat}\left(\mathcal{Y}^{\text{enc}}, \mathcal{Y}^{\text{out}}\right)$, and time series matrix $\text{Concat}(\mathcal{X}^{\text{enc}}, \mathbf{0})$, where $\mathbf{0} \in \mathbb{R}^{m \times q_{\text{out}}}$ is a zero matrix. The MLP architecture comprises an embedding layer followed by fully-connected linear layers, which are flattened across time and hidden neuron dimensions, with GeLU activations. We employ this MLP architecture to transform the zero matrix in $\text{Concat}(\mathcal{X}^{\text{enc}}, \mathbf{0})$ into final outputs. Hyperparameter tuning for MLP is performed for the number of hidden neurons given by 32, 64, 128, 256, the number of hidden layers given by 2, 3, 4, and dropout rates given by $5\%, 10\%, 20\%$, selecting hypeparameters with the best $L_1$ loss. For the LSTM model, we adopt the

[1] https://github.com/Zhaohr1990/GenFormer.

architecture described in (Bahdanau et al., 2014), consisting of an encoder and a decoder with the same inputs as the Transformer architecture. Both the encoder and decoder employ embedding layers followed by a number of LSTM layers, with a feed-forward layer following the decoder to produce the outputs. We tune the hyperparameters of LSTM by experimenting with different configurations of the number of hidden neurons given by 256, 512, 1024, the number of LSTM layers given by 1, 2, 3, and dropout rates given by 5%, 20%.

## 4.1. Synthetic data generated from stochastic differential equations

### 4.1.1. Problem setup

Consider the system of stochastic differential equations driven by Brownian motion with drift $a(x) = \theta(\alpha/\beta - x)$ and diffusion term $b(x) = \sqrt{2\theta x/\beta}, x \in \mathbb{R}$, given by.

$$dQ_i(t) = \theta\left(\frac{\alpha}{\beta} - Q_i(t)\right)dt + \sqrt{\frac{2\theta Q_i(t)}{\beta}}dB_i(t), \quad i = 0,\ldots,m, \tag{4.1}$$

where $t \in [0, T_{\text{obs}})$, $Q_i(t) \in \mathbb{R}$, $\theta > 0, \alpha \geq 1, \beta > 0$ are prescribed coefficients, and $B_i(t), i = 0,\ldots,m$, are mutually independent copies of Brownian motion. It can be shown based on Itô's formula that the second-moment properties of the stationary component of $Q_i(t), i = 0,\ldots,m$, depend only upon the coefficient $\theta$ [14, 435]. More specifically, the mean and variance of the stationary process $Q_i(t)$ are $\alpha/\beta$ and $\alpha/\beta^2$, respectively. The auto-correlation function of $Q_i(t)$ has exponential decay with rate $\theta$ and is given by $\exp(-\theta\tau)$, where $\tau$ is the time lag.

The marginal distribution of $Q_i(t)$ follows the Gamma distribution with shape parameter $\alpha$ and rate parameter $\beta$ (Bibby et al., 2005), which can be derived through the stationary Fokker-Planck equation defined in [14, 482].

Set $V_i(t) = Q_0(t) + Q_i(t), i = 1,\ldots,m$. It can be shown that $V_i(t) \sim \text{Gamma}(2\alpha, \beta)$, that is, $V_i(t)$ has a Gamma marginal distribution with mean $\gamma_1 = 2\alpha/\beta$ and variance $\gamma_2 = 2\alpha/\beta^2$. The cross correlation function between $V_k(t)$ and $V_i(t)$, $1 \leq k, i \leq m$, is given by.

$$r_{ki}(\tau) = E[(V_k(t+\tau) - \gamma_1)(V_i(t) - \gamma_1)]/\gamma_2 = \left(1 - \frac{1}{2}\mathbb{1}(k \neq i)\right)\exp(-\theta\tau), \tag{4.2}$$

where $\mathbb{1}(k \neq i)$ is the indicator function which is equal to 1 if $k \neq i$ and 0 otherwise. The normalized spatial correlation matrix of $\boldsymbol{V}(t)$ is given by $(r_{ki}(0))_{1 \leq k,i \leq m}$.

In this example, we set $m = 3, \theta = 40, \alpha = \beta = 1$. A total of 1000 realizations of $V_i(t)$ are obtained by simulating sequences of $Q_i(t)$ under the Milstein scheme (Bayram et al., 2018) with duration $T_{\text{obs}} = 0.2$ and time increment $\Delta t = 0.001$, resulting in $n_{\text{obs}} = 200$ time steps. Since it is known that $Q_i(t) \sim \text{Gamma}(1,1), i = 0,1,2,3$, the numerical simulation of $Q_i(t)$ is initialized using samples from $\text{Gamma}(1,1)$ to ensure the stationarity of $Q_i(t)$. To pre-process the observations of $V_i(t), i = 1,2,3$, we first transform $V_i(t)$ into the standard Gaussian space via $X_i(t) = \Phi^{-1}[F(V_i(t))]$, where $F$ is the CDF of $\text{Gamma}(2,1)$ and $\Phi^{-1}$ denotes the inverse CDF of the standard Gaussian distribution.

### 4.1.2. Model considerations

The $K$-means clustering with $n_{\text{clusters}} = 300$ is applied to partition the realizations $\boldsymbol{x}(t) = [x_1(t), x_2(t), x_3(t)]^T$ of $\boldsymbol{X}(t) = [X_1(t), X_2(t), X_3(t)]^T$ across time in order to construct the state space for the discrete-time Markov process. Denote by $\{(x_1,x_2,x_3) : -\infty < x_1,x_2,x_3 < \infty\}$ the sample space of $\boldsymbol{X}(t)$ at a fixed time $t$. We consider the tail region of $\boldsymbol{X}(t)$ to be $\{(x_1,x_2,x_3) : \exists i, i \in \{1,2,3\}, \text{s.t.} x_i > q_{0.96}\}$, where $q_{0.96} \approx 1.75$ is the 96%-quantile of the standard Gaussian distribution. To ensure that there is a sufficient number of cluster centroids concentrated in the tail region, we segregate the realizations into those that lie within and outside the tail region. The $K$-means clustering is then performed on each region separately, with $n_{\text{clusters}} = 100$ in the tail region and $n_{\text{clusters}} = 200$ clusters outside the tail region. We repeat the clustering process 20 times with different starting centroids and select the clustering with the lowest within-cluster distance.

We set the Markov order to be $p = 10$ and adopt a deep learning model for Markov state sequence generation with $n_{\text{Markov}} = 3$ decoder blocks. The attention mechanism of the model follows the Informer (Zhou et al., 2021). The dimension of the hidden attention layers is $d_{\text{model}} = 1024$ which is split among $n_{\text{head}} = 8$ heads while the dimension of the feed-forward layers is $d_{\text{ff}} = 2048$. Because the time argument is unitless in this case, only the positional embedding is adopted as the time embedding in the embedding layer. Because the number of Markov states outside the tail region outnumbers that inside the tail region, we assign a weight of 1.3 to the loss values induced by states in the minority class to mitigate the class imbalance issue in focal loss calculation.

For the deep learning model for inferring the $m$-variate process in Section 3.1.2, we investigate two types of architectures. Our proposed GenFormer model utilizes the Transformer architecture with the attention mechanism of the Informer. The lengths of the input sequence of the encoder and the start sequence of the decoder are set to $q_{\text{in}}^{\text{enc}} = 40$ and $q_{\text{in}}^{\text{dec}} = 20$, respectively. The inference length is set to be $q_{\text{out}} = 20$. Both the encoder and decoder have $n_{\text{enc}} = n_{\text{dec}} = 3$ blocks while the other hyperparameters, for example, $n_{\text{head}}$, $d_{\text{model}}$, are the same as above. We use $\eta = 0.9$ to partition the training and validation datasets. However, the approach described in Section 3.1.4 yields insufficient data to construct the validation set. Consequently, we apply a train-validation split across realizations of sequences rather than time stamps, allocating the first 900 realizations for training and the subsequent 100 realizations for validation. From each sequence, $n_{\text{obs}} - q_{\text{in}}^{\text{enc}} - q_{\text{out}} + 1 = 200 - 40 - 20 + 1 = 141$ data pairs can be formed, resulting in $126900 = 900 \times 141$ pairs in the training set and $14100 = 100 \times 141$ pairs in the validation set. In addition to the Transformer architecture, we also investigate using a MLP to construct the mapping from the Markov state $y_j$ to the time series $\boldsymbol{x}(t_j)$, for comparison.

The trained GenFormer model is employed to simulate synthetic realizations on the duration of the observed data such that $T_{\text{sim}} = T_{\text{obs}} = 0.2$ and $n_{\text{sim}} = n_{\text{obs}} = 200$. For each of the 1000 observed time series that comprise the training and validation sets, we utilize the data in the first 40 time stamps, that is, $t \in [0, 0.04)$, and their corresponding Markov state sequences, to initialize the simulation of five synthetic sequences, resulting in synthetic dataset of 5000 sequences. The trained Transformer-based deep learning model is then applied to generate the sequence $\tilde{\boldsymbol{x}}(t_j), j = 41, \ldots, 200,$ from the simulated Markov states $\tilde{y}_{41}, \ldots, \tilde{y}_{200}$. Subsequently, we stack all the synthetic realizations over the time dimension and obtain a time series matrix of dimension $3 \times 10^6$. The model post processing procedures in Section 3.2.2 are then applied to this concatenated matrix to obtain the final synthetic realizations $\widehat{\boldsymbol{x}}(t)$.

### *4.1.3. Results*

We first examine the performance of the deep learning model for Markov state sequence generation. Using the observed Markov state sequences, we compute the normalized frequency of each of the 300 Markov states in the state space. We repeat the same procedure using the simulated Markov state sequences. In Figure 5, we show a scatter plot of the normalized frequency for each Markov state obtained from the observed sequences ($x$-axis) and the simulated sequences ($y$-axis). The approximate alignment of the scatter points along the diagonal line (red line) indicates that the frequencies of the Markov states in the observed and simulated sequences are similar.

We then compare the performance of the deep learning architectures we considered for constructing the mapping from the Markov state $y_j$ to the time series $\boldsymbol{x}(t_j)$. The MLP architecture consists of 3 hidden layers with 128 hidden neurons each while the LSTM architecture consists of 3 LSTM layers with hidden size 512, based on hyperparameter search. The training and validation $L_1$ losses and parameter counts for the selected Transformer, MLP, and LSTM architectures are shown in Table 5. The Transformer-based model achieves 40% and 8% lower $L_1$ loss than the MLP and LSTM models. We now visually assess the accuracy of the trained GenFormer model by comparing a single observation trajectory $\boldsymbol{x}(t_1), \ldots, \boldsymbol{x}(t_{200})$ with its inferred one $\tilde{\boldsymbol{x}}(t_1), \ldots, \tilde{\boldsymbol{x}}(t_{200})$ based on the same Markov state sequence $y_1, \ldots, y_{200}$. The target trajectory is randomly selected from the validation set while the synthetic time series is produced with an

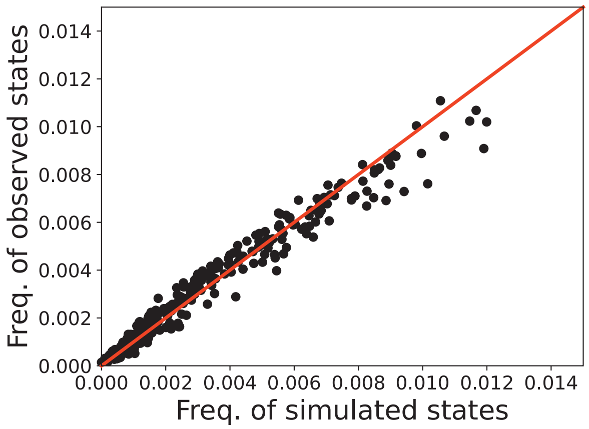


***Figure 5.*** *Scatter plot of the normalized frequencies of Markov states in the observed and simulated sequences. Generating Markov state sequences by estimating the transition matrix from data is computationally challenging for large Markov order p. This example shows that for large p, the trained deep learning model for Markov state sequence generation can closely reproduce the frequencies of Markov states in the observed Markov state sequence data.*

***Table 5.*** *Comparison among Transformer-based deep learning model, LSTM, and MLP for Section 3.1.2. The Transformer-based model achieves the lowest loss value in this example*

| Architecture | Training $L_1$ loss | Validation $L_1$ loss | Parameter count |
|---|---|---|---|
| Transformer | 0.1145 | 0.1199 | $64 \times 10^6$ |
| LSTM | 0.1247 | 0.1297 | $15 \times 10^6$ |
| MLP | 0.1415 | 0.1668 | $177 \times 10^6$ |

initialization using the data $\boldsymbol{x}(t_1),\ldots,\boldsymbol{x}(t_{40})$ and $y_1,\ldots,y_{40}$ since $q_{\text{in}}^{\text{enc}} = 40$. As $q_{\text{out}} = 20$, the inference is performed in an autoregressive manner comprised of 8 iterations based on the remaining sequence $y_{41},\ldots,y_{200}$. Figure 6 compares the target $\boldsymbol{x}(t_j), j = 1,\ldots,200$, and the synthetic time series $\tilde{\boldsymbol{x}}(t_j), j = 1,\ldots,200$. Data to the left of the dotted red line were used to initialize the model. It can be seen that the synthetic time series accurately approximates the target.

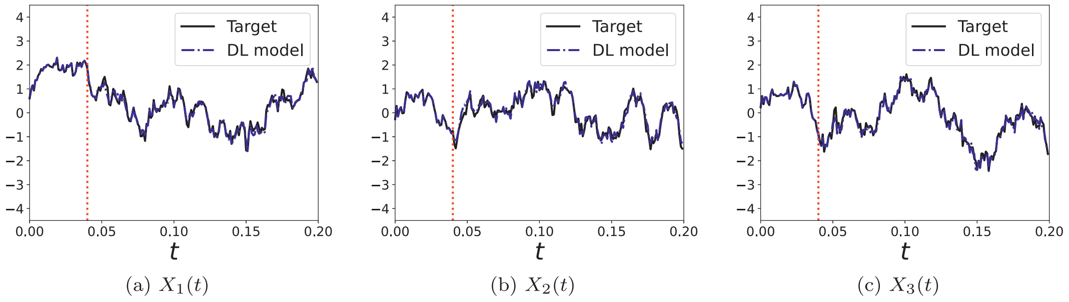


***Figure 6.*** *Target versus synthetic time series produced by the deep learning model for inference of m-variate processes. The Transformer-based model produces accurate inference of the target based on the same Markov state sequence.*

$$\begin{bmatrix} 1 & 0.47 & 0.48 \\ 0.47 & 1 & 0.45 \\ 0.48 & 0.45 & 1 \end{bmatrix}$$

(a) Target matrix from observations $\boldsymbol{x}(t)$

$$\begin{bmatrix} 0.98 & 0.46 & 0.50 \\ 0.46 & 0.94 & 0.44 \\ 0.50 & 0.44 & 0.96 \end{bmatrix}$$

(b) deep learning model estimate

$$\begin{bmatrix} 1 & 0.47 & 0.48 \\ 0.47 & 1 & 0.45 \\ 0.48 & 0.45 & 1 \end{bmatrix}$$

(c) Cholesky-based transformation estimate

$$\begin{bmatrix} 1 & 0.47 & 0.47 \\ 0.47 & 1 & 0.45 \\ 0.47 & 0.45 & 1 \end{bmatrix}$$

(d) GenFormer model estimate

$$\begin{bmatrix} 1 & 0.5 & 0.5 \\ 0.5 & 1 & 0.5 \\ 0.5 & 0.5 & 1 \end{bmatrix}$$

(e) Analytical correlation matrix of $\boldsymbol{V}(t)$

***Figure 7.*** *Target spatial correlation matrix of $\boldsymbol{X}(t)$ (a), various approximations (b), (c), (d), and analytical spatial correlation matrix of $\boldsymbol{V}(t)$ (e). The estimate produced by the GenFormer model has relative error that is 9 times more accurate than the estimate obtained by the deep learning model alone without the post-processing steps in this example. This highlights the need for the post-processing procedure as a supplement to the deep learning model in order to capture key statistical properties such as the spatial correlation matrix.*

We now examine the performance of the proposed GenFormer model in capturing desired statistical properties of the observed time series data. Figure 7 displays various approximations to the target spatial correlation matrix $\boldsymbol{C}$ in Figure 7(a), estimated from the 1000 given realizations of $\boldsymbol{x}(t)$ in the training and validation set. The estimate in Figure 7(b) is obtained using the 5000 synthetic realizations produced by the trained deep learning model in Section 3.1.2. Figure 7(c) and Figure 7(d) shows estimates resulting from samples obtained with the model post-processing steps discussed in Section 3.2.2. More specifically, Figure 7(c) is based on the samples obtained after applying the transformation based on Cholesky decomposition while Figure 7(d) is the approximation by the GenFormer model which subsequently applies the reshuffling procedure. For reference, Figure 7(e) provides the analytical correlation matrix of $\boldsymbol{V}(t)$, derived from (4.2). According to (Grigoriu, 2013), $\boldsymbol{X}(t)$ and $\boldsymbol{V}(t)$ possess roughly the same spatial correlations.

We compute the relative error between the target $\boldsymbol{C}$ and an approximation $\tilde{\boldsymbol{C}}$ via.

$$\frac{\|\boldsymbol{C}-\tilde{\boldsymbol{C}}\|_F}{\|\boldsymbol{C}\|_F}, \tag{4.3}$$

where $\|\cdot\|_F$ is the Frobenius norm. The relative errors between the matrices in (a) and (b), (a) and (c), (a) and (d) are given by 0.0411, 0.0008, 0.0045. Notice that the estimate using the samples obtained by applying a transformation based on Cholesky decomposition is able to match the target while the estimate using the samples obtained from the reshuffling procedure only induces minimal deviation. Consequently, the proposed GenFormer model produces an estimate of the spatial correlation that closely approximates the Monte Carlo estimate computed from the given realizations.

In Figure 8, we examine the auto-correlation functions of the components of $\boldsymbol{X}(t)$ and its various approximations. In each of the panels, the target is represented by the black solid line which is the estimate from the 1000 given realizations $\boldsymbol{x}(t)$. The blue dashdotted and red dashed lines are the estimates based on 5000 synthetic realizations $\tilde{\boldsymbol{x}}(t)$ and $\widehat{\boldsymbol{x}}(t)$ generated from the deep learning model in Section 3.1.2 and the proposed GenFormer model, respectively. For comparison, the green dotted line is the analytical auto-correlation function of $\boldsymbol{V}(t)$ which closely resembles the target following (Grigoriu, 2013). The resulting estimate from the GenFormer model provides a satisfactory approximation to the target. It can also be seen that the model post-processing procedure has negligible impact on the auto-correlation functions.

In Figure 9, we study the advantages of applying the reshuffling technique in the model post-processing procedure by visualizing estimate of the density function for each component of $\boldsymbol{V}(t)$. The blue dashdotted line in each panel corresponds to the inferred values of $\tilde{\boldsymbol{x}}(t)$ from the deep learning model without the model post-processing steps applied, mapped to the original Gamma space. The red dashed

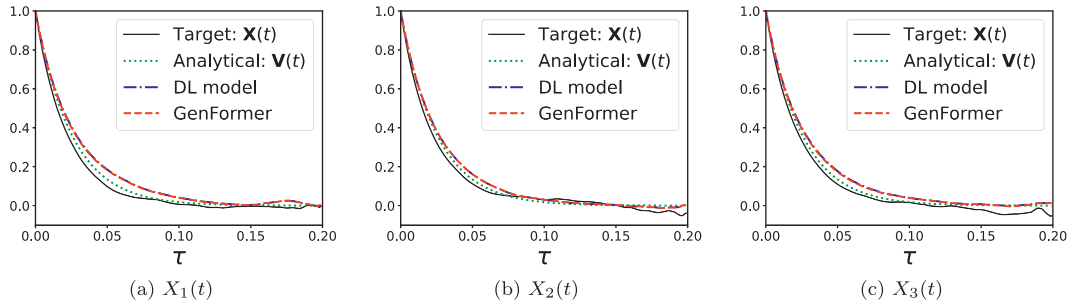


***Figure 8.*** *Auto-correlation functions of $\boldsymbol{X}(t)$ and various approximations. The proposed GenFormer model adequately preserves the second-moment properties of the given realizations.*

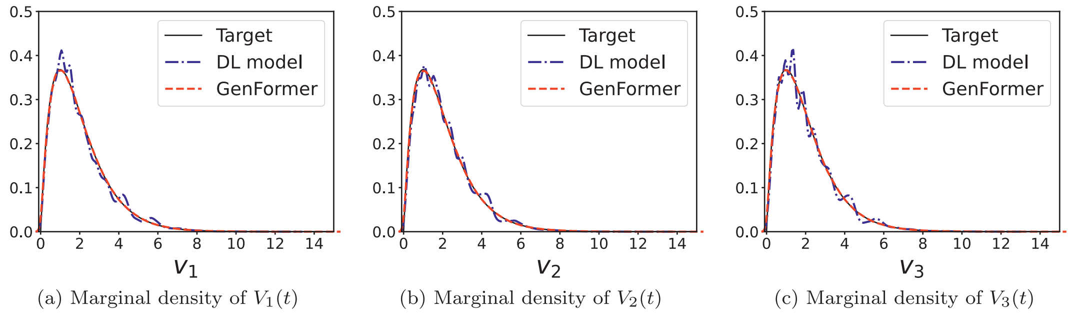


***Figure 9.*** *Marginal densities of $\boldsymbol{V}(t)$ and various approximations. The reshuffling technique in the GenFormer model reduces the $L_1$ relative error by 1 order of magnitude in this example. This is because the target marginal distributions are directly sampled from in the reshuffling procedure.*

line is computed from samples simulated from the GenFormer model which are subsequently transformed to have Gamma marginal distributions. The black solid line is the density of Gamma$(2,1)$ which is the target. We calculate the $L_1$ relative errors of the density estimates with respect to the target, averaged across the 3 dimensions. The error of the density estimate computed from samples without post-processing applied is 0.1173. In contrast, the error of the density estimate resulting from samples simulated from the GenFormer model is 0.0194. This demonstrates that the GenFormer model decreases the error in the approximation of the marginal distributions by 1 order of magnitude in this example.

Finally, we consider a downstream application of the GenFormer model in risk management. A metric of interest is the exceedance probability at a specified time $t$ defined as $p(s) = P(S(t) > s)$, where $S(t) = \sum_{i=1}^{m} V_i(t)$. A commonly-used model for computing such probability is the translation process (Zhao et al., 2019), which is computationally feasible in high dimensions and serves as our baseline model for comparison. A translation process $\boldsymbol{V}_T(t) = [V_{T,1}(t), \ldots, V_{T,m}(t)]^T$ is a nonlinear memoryless transformation of a standard Gaussian process whose $i^{\text{th}}$ component is expressed as $V_{T,i}(t) = F_i^{-1}\left[\Phi\left(X_i^*(t)\right)\right]$, where $F_i$ is the marginal distribution of $V_i(t)$ and $\boldsymbol{X}^*(t) = \left[X_1^*(t), \ldots, X_m^*(t)\right]^T$ is the $m$-variate Gaussian process which has the same second-moment properties (i.e., spatial correlations, auto-correlation functions, etc.) as $\boldsymbol{X}(t)$. However, in general, the statistical properties of $\boldsymbol{X}^*(t)$ and $\boldsymbol{X}(t)$ differ beyond the second moment. To generate synthetic realizations of $\boldsymbol{V}_T(t)$, the second-moment properties of $\boldsymbol{X}(t)$ are first estimated from the Gaussian-transformed observations of $\boldsymbol{V}(t)$. Subsequently, samples of $\boldsymbol{X}^*(t)$ are generated from a multivariate Gaussian distribution with the aforementioned second-moment properties. The mapping $F_i^{-1}\left[\Phi\left(X_i^*(t)\right)\right]$ is then applied to each component of these samples to obtain samples of $\boldsymbol{V}_T(t)$. In addition to the translation process, we consider two other models, substituting the Transformer

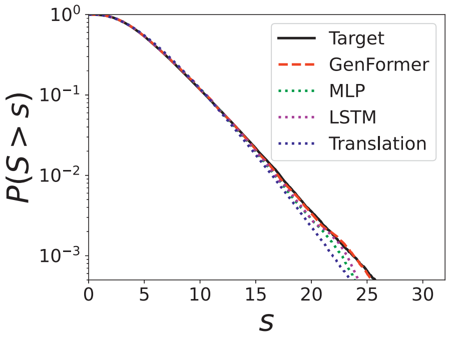


***Figure 10.*** *Exceedance probability of $S(t)$. The relative error in the return period attained by the proposed GenFormer model is approximately an order of magnitude lower than those of the translation model, the MLP, and the LSTM. The GenFormer model can capture higher-order statistical properties of $\boldsymbol{X}(t)$ beyond the second moment in this example.*

architecture in the GenFormer model with either MLP or LSTM. These baseline models are denoted by MLP and LSTM, respectively.

In this example, $F_i \sim \text{Gamma}(2,1), i = 1,2,3$. In Figure 10, we plot the exceedance probability $p(s)$ for $s \in [0,32]$ estimated using the given observations (black solid line), the synthetic realizations produced by the translation model (blue dotted line), the MLP (green dotted line), the LSTM (magenta dotted line), and the GenFormer model (red dashed line). In risk management, the inverse of the exceedance probability is the return period which quantifies the average time interval between events $\{S(t) > s\}$. The $L_1$ relative errors based on the return periods obtained from the translation model, the MLP, the LSTM, and the GenFormer model are 0.3825, 0.2746, 0.1971, and 0.0680, respectively. We notice that the exceedance probability curve due to the proposed GenFormer closely follows the target while the curves produced by the other three models deviate from the target as $s$ increases, resulting in underestimation of the exceedance probability and a significant error in estimating the return period for large values of $s$. The discrepancy between the approximations to the target is due to the fact that the proposed GenFormer model is able to capture the higher-order statistical properties of $\boldsymbol{X}(t)$ beyond the second moment.

## *4.2. Simulation of station-wise wind speeds in Florida*

### *4.2.1. Problem setup*

We apply the proposed GenFormer model to the hourly station-wise wind speed data[2] from the National Oceanic and Atmospheric Administration (NOAA). We select six weather stations around Coral Gables, Florida, an area that is frequently affected by hurricanes and high wind speeds. A detailed list of station information can be found in Table. 6. The hourly wind speed data ranging from January 1, 2006 to December 31, 2021 is collected which amounts to $n_{\text{obs}} = 140256$ data points per station over $T_{\text{obs}} = 5844$ days (equivalent to 16 years).

The wind speed data are preprocessed to remove the trend and any periodicities, rendering it stationary. Missing wind speeds are set to 0. Station-wise hourly periodicity in a day is removed by subtracting the hourly average, computed across all days. This is followed by applying a moving average to the resulting wind speed data per station with circular padding and kernel size 720 that is equivalent to a monthly average. This moving average is then subtracted from the data obtained from the previous step, resulting in stationary wind speed data. The marginal distribution per station is estimated empirically and the data are transformed to the Gaussian space.

[2] https://observablehq.com/@observablehq/noaa-weather-data-by-major-u-s-city.

***Table 6.*** *Weather stations in Florida selected in this work*

| Station ID | Station name | State | Latitude | Longitude |
|---|---|---|---|---|
| 747,830 | FT LAUD/HOLLYWOOD INTL APT | FL, US | 26.079 | −80.162 |
| 722,037 | NORTH PERRY AIRPORT | FL, US | 26.000 | −80.241 |
| 722,024 | OPA LOCKA AIRPORT | FL, US | 25.910 | −80.283 |
| 722,020 | MIAMI INTERNATIONAL AIRPORT | FL, US | 25.788 | −80.317 |
| 722,029 | KENDALL-TAMIAMI EXEC ARPT | FL, US | 25.642 | −80.435 |
| 722,026 | HOMESTEAD AFB AIRPORT | FL, US | 25.483 | −80.383 |

### *4.2.2. Model considerations*

As in the previous example, we designate a tail region that is defined identically as before. We then perform $K$-means clustering with $n_{\text{clusters}} = 100$ in the tail region and $n_{\text{clusters}} = 200$ outside the tail region to capture the patterns of the extremes.

We set the Markov order to be $p = 36$, and use a deep learning model for Markov state sequence generation with $n_{\text{Markov}} = 3$ decoder blocks. The attention mechanism utilized in this example is the Informer. The attention layers in the model have dimension $d_{\text{model}} = 512$ which is divided into $n_{\text{head}} = 8$ heads while the feed-forward layers have dimension $d_{\text{ff}} = 2048$. Given the hourly granularity of the time argument which is provided in year-month-day-hour format, the time feature embedding described in Section 2.2 is incorporated in the embedding layer. For the focal loss computation, a weight of 1.2 is applied to the Markov states in the tail region.

In this example, we aim to infer hourly wind speed data. In the Transformer architecture of the deep learning model for inference of the $m$-variate process using the GenFormer, we set $q_{\text{in}}^{\text{enc}} = 48$, $q_{\text{in}}^{\text{dec}} = 48$, and $q_{\text{out}} = 48$. This implies that we infer wind speeds for the next 2 days based on the observed wind speeds in the previous 2 days. Both the encoder and decoder consists of four blocks each, that is, $n_{\text{enc}} = n_{\text{dec}} = 4$, while $d_{\text{model}}$, $n_{\text{head}}$, and $d_{\text{ff}}$ are identical as above. In training the deep learning models for Markov state sequence generation and the inference of the $m$-variate process, the training and validation datasets are constructed following the approach in Section 3.1.4 with $\eta = 0.9$. We also investigate the use of an MLP architecture instead of the Transformer architecture.

We then simulate hourly wind speeds for 1 month using the convention that a month consists of 28 days. This means that the simulation period is $T_{\text{sim}} = 28$ days which implies that the number of simulation time stamps is $n_{\text{sim}} = 672 = 28 \times 24$. The simulation of synthetic realizations begins by extracting the observed data at $q_{\text{max}} = 48$ time stamps from each of the $192 = 16 \times 12$ sequences of monthly data over 16 years. Each initialization sequence from the observed data is used 10 times to generate 10 sequences, resulting in 1920 synthetic realizations of sequences of length 28 days.

### *4.2.3. Results*

We first evaluate the performance of the deep learning model for Markov state sequence generation in Figure 11. A scatter plot showing the normalized frequency of each Markov state computed from the observed Markov state sequences versus the simulated Markov state sequences from the deep learning model is presented. Despite a large $p$ in this example, the majority of the points in the scatter plot are aligned with the red diagonal line with only a few outliers. This indicates that the trained deep learning model can closely replicate the distribution of Markov state occurrences in the observed data, even when $p$ is large.

Among the hyperparameters, the MLP model with four hidden layers and 128 hidden neurons is selected, while the LSTM configuration with two LSTM layers, each containing 512 hidden neurons, achieves the lowest $L_1$ loss. A comparison of the Transformer architecture with the MLP and LSTM is shown in Table 7. We see that the Transformer architecture achieves a 60% and 10% lower $L_1$ loss in

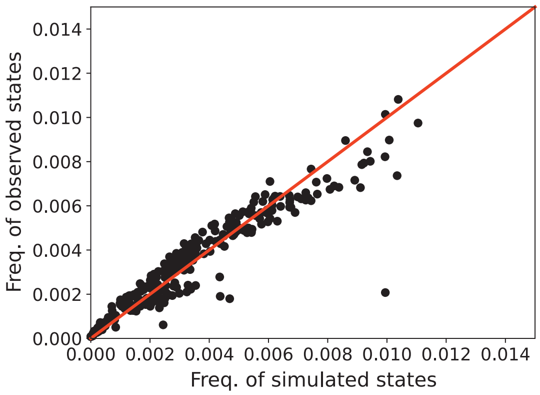


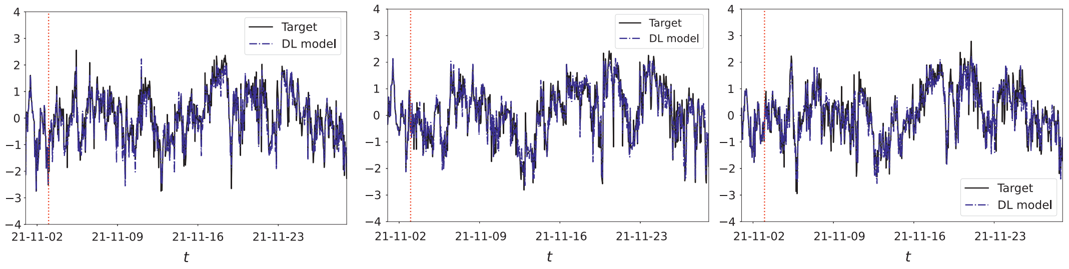
***Figure 11.*** *Scatter plot of the normalized frequencies of Markov states in the observed and simulated sequences. Estimating the transition matrix for large p is prohibitive since the transition matrix would have dimension* $300^{36}\times 300$*. In this example, the trained deep learning model for Markov state sequence generation offers a computationally feasible alternative for producing synthetic Markov state sequences with the occurrence frequency of each Markov state being similar to the observed one.*

***Table 7.*** *Comparison among Transformer-based deep learning model, LSTM, and MLP for Section 3.1.2. The Transformer architecture most effectively captures the temporal patterns in the time series data in this example*

| Architecture | Training $L_1$ loss | Validation $L_1$ loss | Parameter count |
|---|---|---|---|
| Transformer | 0.2296 | 0.2504 | $30\times 10^6$ |
| LSTM | 0.2443 | 0.2758 | $9\times 10^6$ |
| MLP | 0.3846 | 0.4015 | $604\times 10^6$ |

inferring the 6-variate wind speed time series compared to the MLP and LSTM. We then visually inspect the accuracy of the trained model by utilizing it to reproduce a randomly-chosen time series in the validation set provided that the exact Markov state sequence is known, as carried out in the previous example. The inference is initialized with data from the first 2 days (48 hours), corresponding to 48 time stamps, while the inference horizon consists of the subsequent 26 days. Synthetic realizations of the wind speeds are generated using the deep learning model in an auto-regressive manner for 13 iterations since $q_{\text{out}}=2$ days. Figure 12 plots the wind speeds simulated by the deep learning model as well as the observed wind speeds for $X_1(t), X_3(t), X_6(t)$. Data to the left of the dotted red line are used to initialize the deep learning model. Although the time series data encompasses more spatial locations and the inference is carried out over a longer simulation period with increased volatility compared to the previous example, the plots demonstrate that the trained deep learning model is still able to closely approximate the target time series.

We now assess how well the proposed GenFormer model captures statistical properties of interest. Figure 13 compares the target spatial correlation matrix (Figure 13(a)) computed from the collected wind speed observations with various approximations. Figure 13(b) shows the estimate computed from samples generated by the trained deep learning model. Figure 13(c) is the estimate from samples to which a transformation based on Cholesky decomposition has been applied while Figure 13(d) is the resulting estimate provided by the GenFormer model. The relative errors (4.3) between the matrices in (a) and (b), (a) and (c), (a) and (d) are given by 0.0862, 0.0031, and 0.0072, respectively. It can be seen that the Figure 13(c) best aligns with the target since the applied transformation corrects for discrepancies in the spatial correlation. In addition, the estimate due to the GenFormer model is still comparable to the



(a) $X_1(t)$ (b) $X_3(t)$ (c) $X_6(t)$

***Figure 12.*** *Target versus synthetic time series produced by the deep learning model for inference of transformed wind speed time series. Even though the time series data in this example is higher-dimensional and exhibits more volatility, the autoregressive inference from the Transformer-based deep learning model can still effectively approximate the target time series using a modest computational time. The inferred time series does not diverge from the target despite the long simulation horizon.*

$$\begin{bmatrix} 1 & 0.71 & 0.61 & 0.68 & 0.68 & 0.68 \\ 0.71 & 1 & 0.61 & 0.68 & 0.78 & 0.67 \\ 0.61 & 0.61 & 1 & 0.68 & 0.62 & 0.60 \\ 0.68 & 0.68 & 0.68 & 1 & 0.67 & 0.60 \\ 0.68 & 0.78 & 0.62 & 0.67 & 1 & 0.72 \\ 0.68 & 0.67 & 0.60 & 0.60 & 0.72 & 1 \end{bmatrix}$$

(a) Estimate from collected data

$$\begin{bmatrix} 0.93 & 0.77 & 0.69 & 0.74 & 0.74 & 0.74 \\ 0.77 & 0.94 & 0.69 & 0.74 & 0.83 & 0.74 \\ 0.69 & 0.69 & 0.98 & 0.74 & 0.69 & 0.67 \\ 0.74 & 0.74 & 0.74 & 0.92 & 0.72 & 0.66 \\ 0.74 & 0.83 & 0.69 & 0.72 & 0.92 & 0.77 \\ 0.74 & 0.74 & 0.67 & 0.66 & 0.77 & 0.91 \end{bmatrix}$$

(b) Estimate from deep learning model

$$\begin{bmatrix} 1 & 0.71 & 0.61 & 0.68 & 0.68 & 0.68 \\ 0.71 & 1 & 0.61 & 0.68 & 0.78 & 0.67 \\ 0.61 & 0.61 & 1 & 0.68 & 0.62 & 0.60 \\ 0.68 & 0.68 & 0.68 & 1 & 0.67 & 0.60 \\ 0.68 & 0.78 & 0.62 & 0.67 & 1 & 0.72 \\ 0.68 & 0.67 & 0.60 & 0.60 & 0.72 & 1 \end{bmatrix}$$

(c) Estimate from Cholesky-based transformation

$$\begin{bmatrix} 1 & 0.71 & 0.61 & 0.68 & 0.68 & 0.68 \\ 0.71 & 1 & 0.61 & 0.68 & 0.78 & 0.67 \\ 0.61 & 0.61 & 1 & 0.68 & 0.62 & 0.60 \\ 0.68 & 0.68 & 0.68 & 1 & 0.67 & 0.60 \\ 0.68 & 0.78 & 0.62 & 0.67 & 1 & 0.72 \\ 0.68 & 0.67 & 0.60 & 0.60 & 0.72 & 1 \end{bmatrix}$$

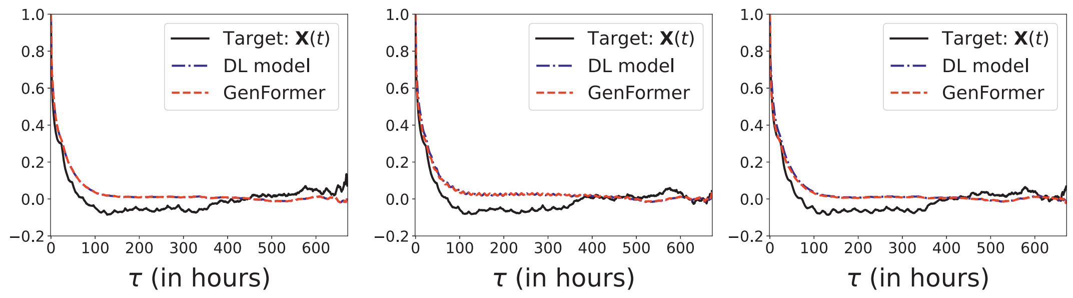
(d) Estimate from the GenFormer model

***Figure 13.*** *Target spatial correlation matrix of collected wind speeds (a) and various approximations (b), (c), (d). The estimate of the spatial correlation matrix due to the GenFormer model is 12 times more accurate than the estimate produced by the deep learning model without post-processing. On the other hand, the transformation based on Cholesky decomposition preserves the spatial correlation matrix irrespective of the number of locations m.*

target which highlights that the reshuffling procedure in this example mostly preserves the effect of correcting the spatial correlation in the post-processing step.

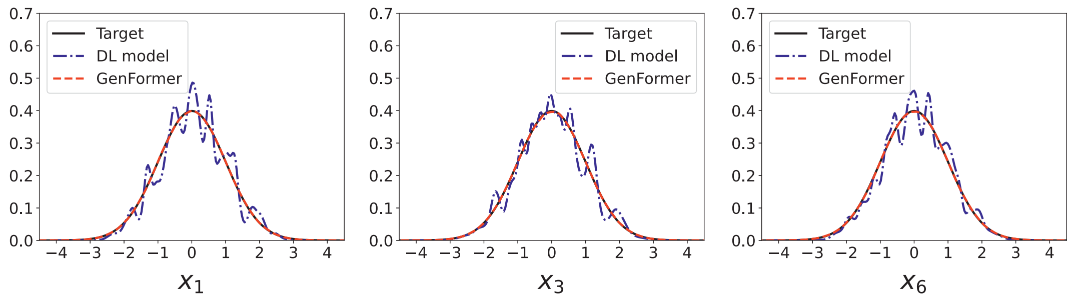
Figure 14 plots the auto-correlation functions of $X_1(t)$, $X_3(t)$, and $X_6(t)$ obtained from the collected wind speed data (black solid line), samples simulated from the trained deep learning model (blue dashdotted line), and samples generated using the GenFormer model (red dashed line). The estimate due to the proposed approach offers a satisfactory approximation to the Monte Carlo estimate from the given observations.

In Figure 15, we show estimates of the marginal densities of $X_1(t)$, $X_3(t)$, and $X_6(t)$ in the Gaussian space. The black solid line marks the target standard Gaussian density. The blue dashdotted line is the estimate computed using inferences from the deep learning model prior to the model post-processing steps with average $L_1$ relative error across the 6 spatial locations being 0.1756. The red dashed line represents the estimate due to the GenFormer model which attains an average $L_1$ relative error of 0.0157. Thus, the model post-processing procedure in the proposed GenFormer model serves to correct the marginal density



(a) $X_1(t)$ (b) $X_3(t)$ (c) $X_6(t)$

***Figure 14.*** *Auto-correlation functions of $X_1(t), X_3(t), X_6(t)$ and various approximations. The trained deep learning model provides satisfactory approximations to the auto-correlation functions with the post-processing procedure introducing only minimal and visually-indiscernible deviations.*



(a) Marginal density of $X_1(t)$ (b) Marginal density of $X_3(t)$ (c) Marginal density of $X_6(t)$

***Figure 15.*** *Marginal density estimates of $X_1(t), X_3(t), X_6(t)$. The model post-processing procedure in the GenFormer model, specifically the reshuffling technique, reduces the $L_1$ relative error by a factor of 11. The deep learning model alone is unable to produce samples with accurate marginal distributions because the training procedure only penalizes the discrepancy in the inferred values.*

of the synthetic realizations which the deep learning model alone does not account for in its training process.

Figure 16 shows the records of the synthetic wind speed data in the Gaussian space produced by the GenFormer model at selected stations. A visual inspection of these time series, compared to the collected wind speed data shown in Figure 12, demonstrates that the synthetic samples appear realistic and look similar to the given observations. These synthetic samples can thus be used for various downstream applications.

The downstream application we pursue in this example is defined as the maximum of the monthly averaged hourly wind speeds across stations given by $S = \max_{1 \le i \le 6} \left\{ \int_0^{T_{\text{sim}}} X_i(t)dt / T_{\text{sim}} \right\}$. Such a metric can be used as a measure of the local hurricane intensity which is of interest in parametric insurance (Ng et al., 2021). In Figure 17, we plot the exceedance probabilities of the metric of interest $p(s) = P(S > s)$ computed using the observed data and synthetic realizations generated by the GenFormer model. As in the previous example, the translation process, MLP, and LSTM are adopted as the baseline models for comparison. We see that the approximations due to all baseline models deteriorate as $s$ increases while the proposed GenFormer model is able to provide an accurate estimate. This is further evidenced by the return-period-based relative $L_1$ errors of the translation process, MLP, and LSTM model which are 0.9713, 0.5061, and 0.2972 compared to that of the GenFormer which is 0.1281. The GenFormer offers a 7.6, 4.0, and 2.3 times improvement, respectively, over the aforementioned models. This improvement

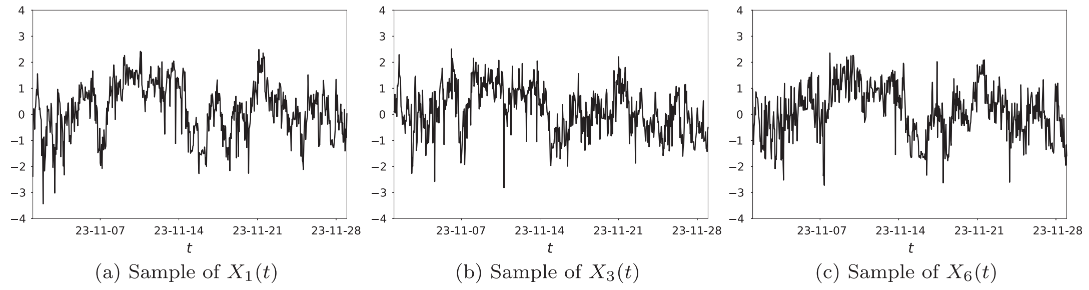


***Figure 16.*** *Synthetic realizations of $X_1(t), X_3(t), X_6(t)$ produced by the GenFormer model. The synthetic transformed wind speed records appear realistic and can therefore be used for downstream applications of interest.*

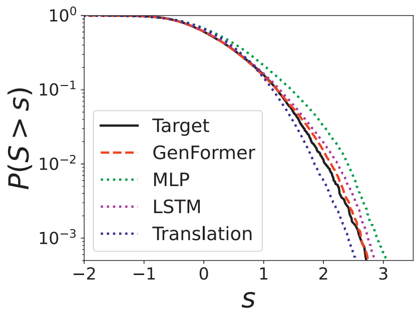


***Figure 17.*** *Exceedance probability of S. The estimate obtained from the GenFormer model is* 7.6, 4.0, *and* 2.3 *times more accurate than the translation, MLP, and LSTM models. The predictive capabilities of the Transformer-based deep learning model coupled with the statistical post-processing techniques enable the GenFormer model to capture high-order statistical properties.*

can become more substantial at specific values of $s$. For example, at $s = 2.7$ with a target return period of 1684 months, the estimates from the GenFormer, the translation model, MLP, and LSTM are 1376, 6088, 376, and 758 months, reflecting an at most 14-fold enhancement in the relative error. This improved accuracy is attributed to the capability of GenFormer in capturing higher-order statistical properties, even when $m$ is large and the simulation horizon is long.

## 5. Conclusion

We presented GenFormer, a novel stochastic generator for producing synthetic realizations of spatio-temporal multivariate stochastic processes. The model integrates a univariate discrete-time Markov process capturing spatial variation with a Transformer-based deep learning model mapping the Markov states to time series values. GenFormer offers a scalable alternative for simulating multivariate processes in high dimensions and long horizons as well as Markov state sequences of large Markov orders. It exploits the predictive power of deep learning models coupled with modern computing capabilities while leveraging statistical post-processing techniques to guarantee that key statistical behavior is preserved. Numerical experiments applying the GenFormer model to simulate wind speeds in Florida demonstrated its ability to produce samples that approximate statistical properties beyond the second moment, unlike traditional methods. The GenFormer model can thus be reliably employed in various downstream applications in engineering. Future work includes incorporating additional spatial modeling techniques, such as those described in (Wu et al., 2023), to more effectively capture spatial dependencies, implementing novel attention mechanisms in the Transformer architecture to further improve its performance, and extending the proposed approach to non-stationary multivariate stochastic processes.

**Data availability statement.** All the required data and code are available in the Github repository: https://github.com/Zhaohr1990/GenFormer.

**Author contribution.** H. Z.: Conceptualization, data curation, methodology, project administration, software, validation, visualization, writing - original draft, writing - review & editing. W. I. T. U.: Methodology, software, validation, writing - original draft, writing - review & editing.

**Competing interest.** There are no competing interests from the authors in this publication.

**Funding statement.** This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

**Ethical standard.** The research meets all ethical guidelines, including adherence to the legal requirements of the study country.

## References

**Albawi S, Mohammed TA and Al-Zawi S** (2017) Understanding of a convolutional neural network. In *International Conference on Engineering and Technology (ICET). IEEE*, pp. 1–6.

**Apipattanavis S, Podesta G, Rajagopalan B and Katz RW** (2007) A semiparametric multivariate and multisite weather generator. *Water Resources Research 43*(11).

**Ba JL, Kiros JR and Hinton GE** (2016) Layer normalization. *arXiv:1607.06450*.

**Bahdanau D, Cho K and Bengio Y** (2014) Neural machine translation by jointly learning to align and translate. *arXiv:1409.0473*.

**Bayram M, Partal T and Buyukoz GO** (2018) Numerical methods for simulation of stochastic differential equations. *Advances in Difference Equations 17*.

**Bibby BM, Skovgaard IM and Sørensen M** (2005) Diffusion-type models with given marginal distribution and autocorrelation function. *Bernoulli 11*(2), 191–220.

**Breinl K, Turkington T and Stowasser M** (2013) Stochastic generation of multi-site daily precipitation for applications in risk management. *Journal of Hydrology 498*, 23–35.

**Brémaud P** (1999) *Discrete-Time Markov Models*. New York, New York, NY: Springer, pp. 53–93.

**Deng L, Hinton G and Kingsbury B** (2013) New types of deep neural network learning for speech recognition and related applications: An overview. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 8599–8603.

**Devlin J, Chang MW, Lee K and Toutanova K** (2018) Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805*.

**Frazier PI** (2018) A tutorial on bayesian optimization. *arXiv:1807.02811*.

**Goodfellow I, Bengio Y and Courville A** (2016) *Deep Learning*. MIT Press.

**Grigoriu M** (2013) *Stochastic Calculus*. New York, NY: Springer.

**Hartigan JA and Wong MA** (1979) A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics) 28*(1), 100–108.

**Hendrycks D and Gimpel K** (2016) Gaussian error linear units (gelus). *arXiv:1606.08415*.

**Horowitz JL** (2001) *Chapter 52 - the Bootstrap. Volume 5 of Handbook of Econometrics*. Elsevier, pp. 3159–3228.

**Ibragimov R** (2009) Copula-based characterizations for higher order markov processes. *Econometric Theory 25*, 819–846.

**Islam MT, Siddique BMN, Rahman S and Jabid T** (2018) Image recognition with deep learning. In *2018 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, 3, pp. 106–110.

**Katz RW** (1981) On some criteria for estimating the order of a markov-chain. *Technometrics 23*(3), 243–249.

**Kingma DP and Ba J** (2015) Adam: A method for stochastic optimization. *arXiv:1412.6980*.

**Lai G, Chang W, Yang Y and Liu H** (2018) Modeling long- and short-term temporal patterns with deep neural networks. In *ACM SIGIR*, pp. 95–104

**Li F, Guo G, Han F, Zhao J and Shen F** (2024) Multi-scale dilated convolution network for long-term time series forecasting. *arXiv:2405.05499*.

**Li S, Jin X, Xuan Y, Zhou X, Chen W, Wang Y and Yan X** (2019) Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Advances in Neural Information Processing Systems 32*.

**Lin T, Goyal P, Girshick R, He K and Dollar P** (2017) Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

**Liu Y, Gong C, Yang L and Chen Y** (2020) Dstp-rnn: A dual-stage two-phase attention-based recurrent neural networks for long-term and multivariate time series prediction. *Expert Systems with Applications 143*(113082).

**Ng KS, Leckebusch GC, Ye Q, Ying W and Zhao H** (2021) On the use of ensemble predictions for parametric typhoon insurance. *Climate 9*(12), 174.

**Nicholas JH** (1990) Analysis of the cholesky decomposition of a semi-definite matrix. *Reliable Numerical Computation*, 161–185.

**Ouyang Z and Spence SMJ** (2021) Performance-based wind-induced structural and envelope damage assessment of engineered buildings through nonlinear dynamic analysis. *Journal of Wind Engineering and Industrial Aerodynamics 208*, 104452.

**Paszke A**, **Gross S**, **Massa F**, **Lerer A**, **Bradbury J**, **Chanan G**, **Killeen T**, **Lin Z**, **Gimelshein N**, **Antiga L**, **Desmaison A**, **Kopf A**, **Yang E**, **DeVito Z**, **Raison M**, **Tejani A**, **Chilamkurthy S**, **Steiner B**, **Fang L**, **Bai J and Chintala S** (2019) Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems 32*.

**Radu A and Grigoriu M** (2018) An earthquake-source-based metric for seismic fragility analysis. *Bulletin of Earthquake Engineering 16*, 3771–3789.

**Raffel C**, **Shazeer N**, **Roberts A**, **Lee K**, **Narang S**, **Matena M**, **Zhou Y**, **Li W and Liu PJ** (2019) Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv:1910.10683*.

**Salinas D**, **Flunkert V**, **Gasthaus J and Januschowski T** (2020) Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting 36*(3), 1181–1191.

**Schwarz G** (1978) Estimating dimension of a model. *The Annals of Statistics 6*(2), 461–464.

**Sutskever I**, **Vinyals O and Le QV** (2014) Sequence to sequence learning with neural networks. *Advances in Neural Information Processing Systems*, 3104–3112.

**Vandanapu L and Shields MD** (2021) 3rd-order spectral representation method: Simulation of multi-dimensional random fields and ergodic multi-variate random processes with fast fourier transform implementation. *Probabilistic Engineering Mechanics 64*, 103128.

**Vaswani A**, **Shazeer N**, **Parmar N**, **Uszkoreit J**, **Jones L**, **Gomez AN**, **Kaiser L and Polosukhin I** (2017) Attention is all you need. *In Advances in Neural Information Processing Systems*, 5998–6008.

**Wu H**, **Xu J**, **Wang J and Long M** (2021) Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in Neural Information Processing Systems 34*, 22419–22430.

**Wu H**, **Zhou H**, **Long M and Wang J** (2023) Interpretable weather forecasting for worldwide stations with a unified deep model. *Nature Machine Intelligence 5*(602–611).

**Xu H**, **Grigoriu M and Gurley KR** (2023) A novel surrogate for extremes of random functions. *Reliability Engineering and System Safety 239*, 109493.

**Yang L**, **Gurley KR and Prevatt DO** (2013) Probabilistic modeling of wind pressure on low-rise buildings. *Journal of Wind Engineering and Industrial Aerodynamics 114*, 18–26.

**Yao W**, **Zheng X**, **Zhang J**, **Wang N and Tang G** (2023) Deep adaptive arbitrary polynomial chaos expansion: A mini-data-driven semi-supervised method for uncertainty quantification. *Reliability Engineering and System Safety 229*, 108813.

**Yin W**, **Kann MY and Schutze H** (2017) Comparative study of cnn and rnn for natural language processing. *arXiv:1702.01923*.

**Zhao H** (2017) *Probabilistic Models for Wind Loads and Reliability Analysis*PhD thesis. Cornell University.

**Zhao H**, **Grigoriu M and Gurley KR** (2019) Translation processes for wind pressures on low-rise buildings. *Journal of Wind Engineering and Industrial Aerodynamics 184*, 405–416.

**Zhou H**, **Zhang S**, **Peng J**, **Zhang S**, **Li J**, **Xiong H and Zhang W** (2021) Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, *35*, pp. 11106–11115.

**Zhou R**, **Li JS-H and Pai J** (2018) Evaluating effectiveness of rainfall index insurance. *Agricultural Finance Review 78*(5), 611–625.

**Cite this article:** Zhao H and Uy W (2025). GenFormer: a deep-learning-based approach for generating multivariate stochastic processes. *Data-Centric Engineering*, 6, e17. doi:10.1017/dce.2025.6