GenFormer: A Deep-Learning-Based Approach for Generating Multivariate Stochastic Processes

Haoran Zhao*, Wayne Isaac Tan Uy†

February 6, 2024

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 preserves the target marginal distributions and approximately captures 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.

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

1 Introduction

Stochastic generators are tools to produce synthetic data which preserves desired statistical properties. They are crucial in situations wherein the number of available records is inadequate while a large amount of data is required. This is especially the case in reliability analysis and risk management. For example, performance-based engineering requires synthetic data that characterizes excitations from natural hazards to provide accurate reliability estimates of building systems [25, 27]. Robust risk assessments of parametric insurance products necessitate the generation of supplemental synthetic loss events for various perils such as hurricane and excess rainfall [23, 42]. The underlying data in the aforementioned examples, e.g., wind pressure 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 [32] 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 [36] and translation processes [40] 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

*hz289@cornell.edu
†wtu4@cornell.edu

match the marginal distributions, respectively. Recently, a mix of the above two models is proposed in [35]. 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 [7], K-nearest neighbor algorithm [2], and copula models [39]. A prevalent challenge in many of the aforementioned models is the curse of dimensionality in which increasing the number of variates, i.e. spatial locations, and the simulation time horizon leads to substantial computational demands or significant decline in model performance of approximating target statistical properties [37].

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 [18, 9], natural language processing [38, 33], and time series forecasting [29, 21]. 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 [41, 34]. Due to the attention mechanism, Transformers are capable of modeling long-term dependencies and patterns in sequential data, leading to markedly-improved prediction accuracy.

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 [7]. 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 [7] 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 rainfall 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 [7] is of interest in this subsection. It is designed for multi-station precipitation data, i.e., $X_i(t)$ 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 $\hat{\boldsymbol{x}}(t_1), \ldots, \hat{\boldsymbol{x}}(t_n)$ 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^{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 $j > p + 1$ [17]. Under the assumption that Y_j is discrete-valued, the above definition readily implies the well-known Markov property

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

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 4 states, i.e., $Y_j \in \{0, 1, 2, 3\}$. 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 $Y_j, j = 1, \ldots, n$, can be fully characterized by the order p and the transition probability $P(y_j | y_{j-1}, \ldots, y_{j-p})$, $j \geq p + 1$. Likelihood measures such as the Akaike information criterion (AIC) [19] or the Bayesian information criteria (BIC) [30] can be employed to estimate p. The corresponding transition probability, represented in matrix form, can then be estimated given the realizations of $Y_1, \ldots, Y_n$ [8]. Based on (2.1), a sample sequence of Y_j is simulated in a sequential manner for time stamps $t_{p+1}, \ldots, t_n$. Given a realization $y_1, \ldots, y_p$ of $Y_1, \ldots, Y_p$, we initialize $\tilde{y}_1, \ldots, \tilde{y}_p$ by using $y_1, \ldots, y_p$. Subsequently, we simulate a realization $\tilde{y}_{p+1}$ from the transition probability $P(y | y_1, \ldots, y_p)$. This is then used to simulate a realization $\tilde{y}_{p+2}$ from the transition probability $P(y | y_2, \ldots, y_p, \tilde{y}_{p+1})$. This procedure is repeated to produce $\tilde{y}_{p+1}, \ldots, \tilde{y}_n$.

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

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

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

In the next step, preliminary synthetic realizations of the rainfall sequence $\tilde{\boldsymbol{x}}(t_1), \ldots, \tilde{\boldsymbol{x}}(t_n)$ are generated by resampling subsequences of rainy sequences present in the series $\tilde{y}_1, \ldots, \tilde{y}_n$. 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 since 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 [16]. Suppose we have a synthetic realization $\tilde{y}_1, \ldots, \tilde{y}_n$ 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 $t_j, t_{j+1}, t_{j+2}, j \in \{1, \ldots, n-2\}$. Since 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 $\tilde{y}_1, \ldots, \tilde{y}_n$. This results in the synthetic realization $\tilde{\boldsymbol{x}}(t_1), \ldots, \tilde{\boldsymbol{x}}(t_n)$ 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, e.g., 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 [7], the authors suggest to characterize the marginal distributions of $\boldsymbol{X}(t)$ with parametric forms, e.g., 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 in-

dices, $j_1, \ldots, j_n \in \{1, \ldots, n\}$, such that $\tilde{x}_i(t_{j_1}) \geq \cdots \geq \tilde{x}_i(t_{j_n})$. 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 $\hat{\boldsymbol{x}}_i = [\hat{x}_i(t_1), \ldots, \hat{x}_i(t_n)]$ is then defined by

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

To illustrate, consider a 2-station rainfall process from the resampling step with a time horizon of 5 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 $\hat{\boldsymbol{x}}_1 = [2.58, 4.68, 1.26, 4.34, 1.76]$ and $\hat{\boldsymbol{x}}_2 = [1.52, 5.53, 5.27, 2.75, 4.34]$ following (2.2).

(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

Table 2: Reshuffling of a hypothetical 2-station example with 5 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 $\tilde{\boldsymbol{r}}_2$.

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 [7], e.g., 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 [7].

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. Recently, commonly-adopted models in NLP typically follow the Transformer architecture which has an encoder-decoder structure [31, 4, 28]. 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 $\boldsymbol{\mathcal{T}}^{\text{enc}} = [t_1, \ldots, t_{q_{\text{in}}^{\text{enc}}}]$ be a vector of increasing time stamps of length $q_{\text{in}}^{\text{enc}} < n$ and $\boldsymbol{\mathcal{X}}^{\text{enc}} \in \mathbb{R}^{m \times q_{\text{in}}^{\text{enc}}}$ be the time series matrix where each row represents the stochastic process $X_i(t)$ for the i^{th} location, $i = 1, \ldots, m$, at the time stamps indicated in $\boldsymbol{\mathcal{T}}^{\text{enc}}$. Given $\boldsymbol{\mathcal{X}}^{\text{enc}}$ and $\boldsymbol{\mathcal{T}}^{\text{enc}}$, the deep learning model predicts the subsequent sequence $\boldsymbol{\mathcal{X}}^{\text{out}} \in \mathbb{R}^{m \times q_{\text{out}}}$ at q_{out} time stamps specified in $\boldsymbol{\mathcal{T}}^{\text{out}}$. The model proceeds by passing the inputs $\boldsymbol{\mathcal{X}}^{\text{enc}}$, $\boldsymbol{\mathcal{T}}^{\text{enc}}$ through the embedding layer (red block) to obtain the hidden representation $\boldsymbol{\mathcal{Z}}^{\text{enc}}$. The hidden representation refers to the output matrices of the hidden layers, which all have dimension d_{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 $\boldsymbol{\mathcal{Z}}^{\text{enc}}$ and $\boldsymbol{\mathcal{T}}^{\text{out}}$, the decoder (green block) combined with a linear layer (purple block) performs generative inference to produce the output sequence $\boldsymbol{\mathcal{X}}^{\text{out}}$ (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.

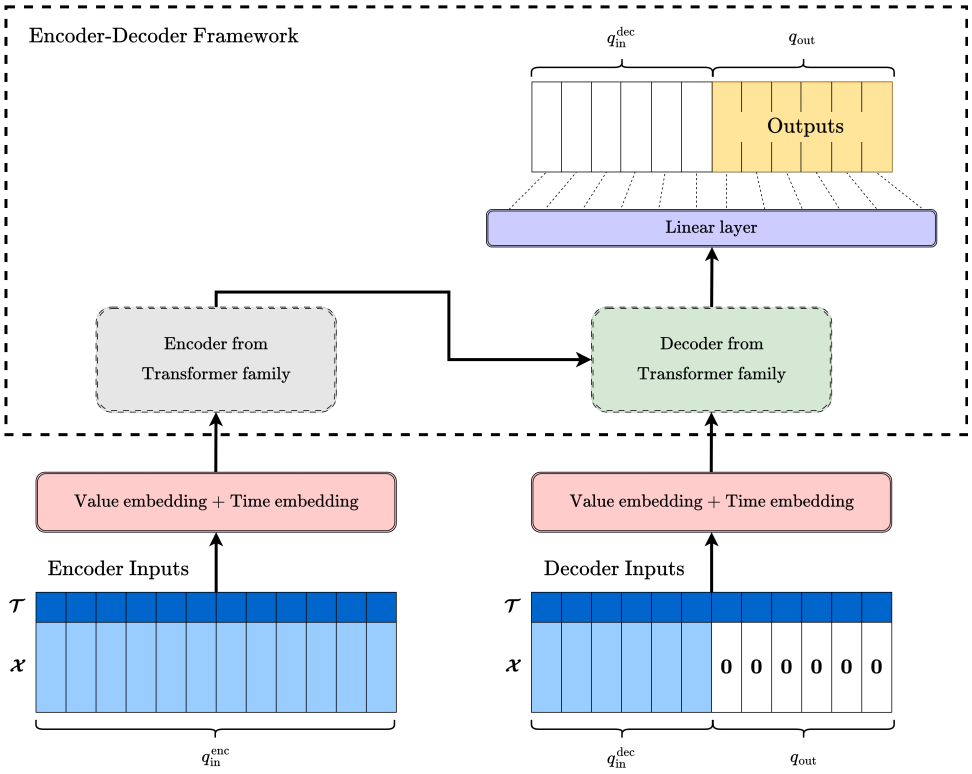


Figure 1: Deep learning model architecture based on the encoder-decoder framework. The model processes inputs through an embedding layer (red block), generating the hidden representation which undergoes further updates in the encoder layers (gray block). The decoder (green block), in conjunction with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

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 $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{m \times q}$ and the time sequence $\boldsymbol{\mathcal{T}}$ of length q, respectively.

The value embedding is a 1D-convolutional layer [1] with specified kernel size and circular padding to map the spatial dimension m to the hidden dimension d_{model}. We denote the operation of the value embedding applied to an input $\boldsymbol{\mathcal{X}}$ by ValueEmbedding($\boldsymbol{\mathcal{X}}$) which produces a hidden representation of dimension $d_{\text{model}} \times q$.

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 [33] 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 $\sin(j/(10000^{2k/d_{\text{model}}}))$ and $\cos(j/(10000^{2k/d_{\text{model}}}))$, respectively. In contrast, if the time argument contains unit information, i.e., the data is in the format of year-month-day-hour-minute-second or a subset of any of these units, [41] 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 $\boldsymbol{\mathcal{T}}$ by TimeEmbedding($\boldsymbol{\mathcal{T}}$) which produces a matrix of dimension $d_{\text{model}} \times q$.

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

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

2.2.2 Encoder-decoder framework

The encoder-decoder framework is commonly used for sequence-to-sequence tasks, e.g., time series forecasting and machine translation in the NLP domain. The work [33] 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 [33], 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 $\boldsymbol{\mathcal{Z}} \in \mathbb{R}^{d_{\text{model}} \times q}$. Mathematically, self-attention is defined as

$$\begin{aligned} \text{Self-Attention}(\boldsymbol{\mathcal{Z}}) &= \boldsymbol{V}\ \text{Softmax}\left(\frac{\boldsymbol{Q}^T\boldsymbol{K}}{\sqrt{d_{\text{model}}}}\right), \\ \boldsymbol{Q} &= \boldsymbol{W}_Q\boldsymbol{\mathcal{Z}}, \\ \boldsymbol{K} &= \boldsymbol{W}_K\boldsymbol{\mathcal{Z}}, \\ \boldsymbol{V} &= \boldsymbol{W}_V\boldsymbol{\mathcal{Z}}, \end{aligned} \tag{2.4}$$

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, Softmax$(\cdot)$ denotes the softmax function defined in [12], 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_{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_{model}-dimensional query to the d_{model}-dimensional key-value pairs once. As suggested in [33], it is beneficial to adopt a multi-head mechanism, i.e., we perform n_{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_{model}. More advanced attention mechanisms have been developed in the deep learning community to improve its performance. For example, the probsparse-attention is developed for Informer [41] with the aim of significantly increasing the efficiency of the attention mechanism. The attention is distilled such that each key can only attend to the dominant queries. The Autoformer [34] 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 $\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{X}}^{\text{enc}}$ across the temporal and spatial dimensions. It is composed of a stack of n_{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 [15] and can be mathematically expressed as

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

where $\boldsymbol{\mathcal{Z}}$ 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_{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 [3]. 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 $\boldsymbol{\mathcal{X}}^{\text{out}}$ based on the time sequences $\boldsymbol{\mathcal{T}}^{\text{out}}$. Similar to NLP applications wherein we apply a start token for dynamic decoding [10], we initiate the inference with $\boldsymbol{\mathcal{X}}^{\text{dec}}_{\text{start}} \in \mathbb{R}^{m \times q^{\text{dec}}_{\text{in}}}$ and $\boldsymbol{\mathcal{T}}^{\text{dec}}_{\text{start}} \in \mathbb{R}^{q^{\text{dec}}_{\text{in}}}$ which are subsets of $\boldsymbol{\mathcal{X}}^{\text{enc}}$ and $\boldsymbol{\mathcal{T}}^{\text{enc}}$, respectively, representing the last $q^{\text{dec}}_{\text{in}}$ columns of the aforementioned matrices. The input of the decoder is the embedding (2.3) applied to the following matrices:

$$\begin{aligned} \boldsymbol{\mathcal{X}} &= \boldsymbol{\mathcal{X}}^{\text{dec}} = \text{Concat}(\boldsymbol{\mathcal{X}}^{\text{dec}}_{\text{start}}, \mathbf{0}), \\ \boldsymbol{\mathcal{T}} &= \boldsymbol{\mathcal{T}}^{\text{dec}} = \text{Concat}(\boldsymbol{\mathcal{T}}^{\text{dec}}_{\text{start}}, \boldsymbol{\mathcal{T}}^{\text{out}}), \end{aligned} \tag{2.6}$$

where $\text{Concat}(\cdot)$ 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 $\boldsymbol{\mathcal{X}}^{\text{out}}$, $\boldsymbol{\mathcal{X}}^{\text{dec}} \in \mathbb{R}^{m \times (q^{\text{dec}}_{\text{in}} + q_{\text{out}})}$, and $\boldsymbol{\mathcal{T}}^{\text{dec}} \in \mathbb{R}^{(q^{\text{dec}}_{\text{in}} + q_{\text{out}})}$. The structure of the decoder is similar to that of the encoder, which is composed of a stack of n_{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 $\boldsymbol{Q} = \boldsymbol{W}_Q \boldsymbol{\mathcal{Z}}^{\text{dec}} \in \mathbb{R}^{d_{\text{model}} \times (q^{\text{dec}}_{\text{in}} + q_{\text{out}})}$, $\boldsymbol{K} = \boldsymbol{W}_K \boldsymbol{\mathcal{Z}}^{\text{enc}} \in \mathbb{R}^{d_{\text{model}} \times q^{\text{enc}}_{\text{in}}}$, and $\boldsymbol{V} = \boldsymbol{W}_V \boldsymbol{\mathcal{Z}}^{\text{enc}} \in \mathbb{R}^{d_{\text{model}} \times q^{\text{enc}}_{\text{in}}}$. The output representation from the cross-attention layer and that of the decoder has dimension $d_{\text{model}} \times (q^{\text{dec}}_{\text{in}} + q_{\text{out}})$. It is then transformed to have size $m \times (q^{\text{dec}}_{\text{in}} + q_{\text{out}})$ via a linear layer with bias. Only the last q_{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 $\hat{\boldsymbol{x}}(t)$ 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 Φ is the standard Gaussian CDF. The marginal distribution F_i can be characterized by parametric distributions as in [7], by its empirical distribution, or by a mixture of both [40].

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 generator 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 is 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 comprised 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_{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 [7] 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 [14] 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_{obs} time stamps into n_{clusters} clusters, where n_{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.

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

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.

3.1.2 Deep learning model with Markov state embedding

The resampling procedure introduced in [7] 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 $\boldsymbol{\mathcal{Y}}^{\text{enc}} \in \mathbb{R}^{q_{\text{in}}^{\text{enc}}}$ be a vector of the Markov state sequence corresponding to the vector of increasing time stamps $\boldsymbol{\mathcal{T}}^{\text{enc}}$ and the matrix of observations $\boldsymbol{\mathcal{X}}^{\text{enc}}$ at the specified time stamps. Given $\boldsymbol{\mathcal{X}}^{\text{enc}}$, $\boldsymbol{\mathcal{T}}^{\text{enc}}$, and $\boldsymbol{\mathcal{Y}}^{\text{enc}}$, the deep learning model aims to infer the subsequent sequence $\boldsymbol{\mathcal{X}}^{\text{out}}$ based on the corresponding Markov state sequence $\boldsymbol{\mathcal{Y}}^{\text{out}}$ specified at time stamps in $\boldsymbol{\mathcal{T}}^{\text{out}}$. The inputs of the model to perform inference starting from time stamp $j+1$ are as follows:

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

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

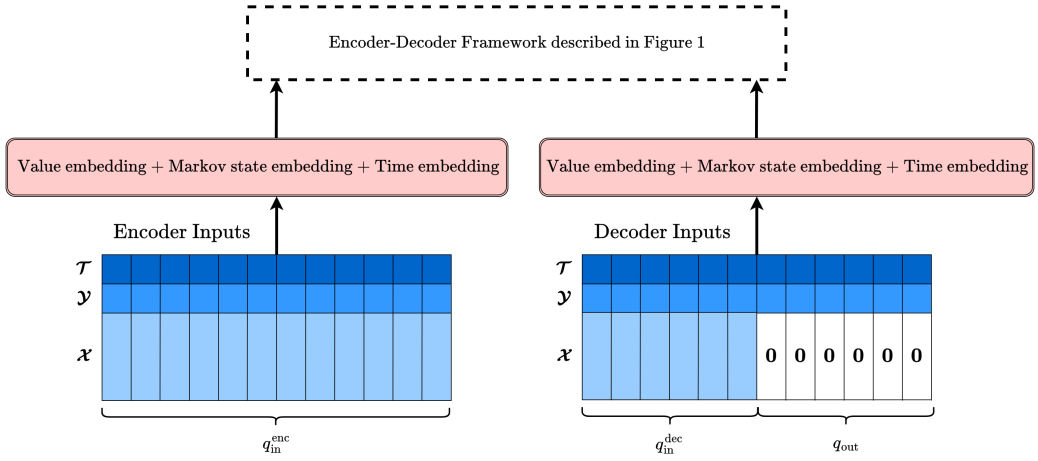


Figure 2: Transformer-based deep learning model with Markov state embedding. The proposed approach includes a Markov state embedding in addition to the value and time embedding present in the embedding layer. The remainder of the model architecture is the same as in Figure 1.

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 $\boldsymbol{\mathcal{Y}}$ 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_{model} learned in the training stage. The operation MarkovStateEmbedding($\boldsymbol{\mathcal{Y}}$) retrieves the respective embedding vectors in the same order as the Markov states in $\boldsymbol{\mathcal{Y}}$ 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 $\boldsymbol{\mathcal{X}}$, the Markov state sequence $\boldsymbol{\mathcal{Y}}$, and the time sequence $\boldsymbol{\mathcal{T}}$, the output representation $\boldsymbol{\mathcal{Z}} \in \mathbb{R}^{d_{\text{model}} \times q}$ from the embedding layer then becomes

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

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

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

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

Note that the hyperparameter $q_{\text{in}}^{\text{enc}}$ is not required to be the same as the Markov order, i.e., $q_{\text{in}}^{\text{enc}} \neq p$ in general. Similarly, setting $q_{\text{out}} = 1$ is not obligatory. In practice, we select $q_{\text{in}}^{\text{enc}} > p$ and $q_{\text{out}} > 1$ because the deep learning model is capable of extracting temporal patterns of long time series of length $q_{\text{in}}^{\text{enc}}$, even though the Markov process is assumed to have a shorter memory of length p. Meanwhile, a larger q_{out} can significantly accelerate the inference by generating samples at q_{out} time stamps at once instead of repeating the inference q_{out} times. On the other hand, $q_{\text{in}}^{\text{enc}}$ and q_{out} cannot be too large since this may potentially hinder the accuracy of the model as will be discussed in Section 3.1.4. 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 $\boldsymbol{\mathcal{Y}}^{\text{out}}$. 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_{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_{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_{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_{clusters} (white block), with each component representing the weight of the respective Markov state in the state space. These weights are then 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 $y_1, \ldots, y_{n_{\text{obs}}}$.

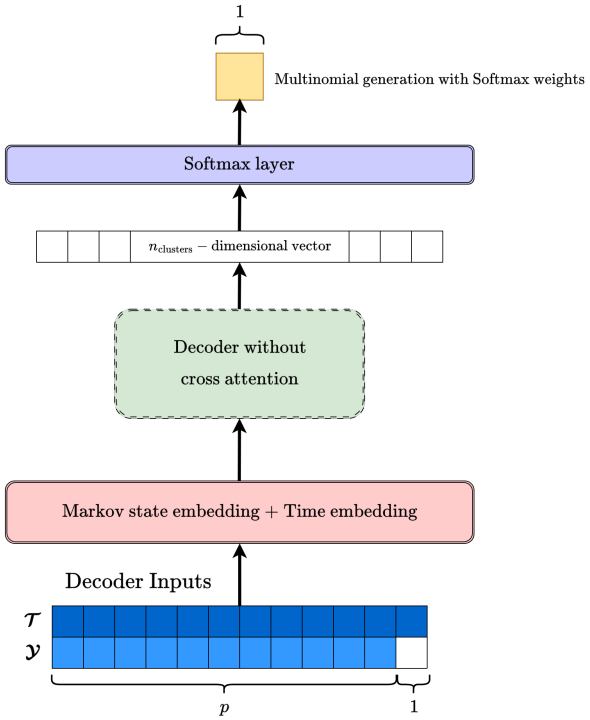


Figure 3: Deep learning model for Markov state sequence generation when Markov order $p \geq 2$. We adopt a decoder-only structure without cross attention mechanism. The input of the model is the Markov states in the previous p time stamps concatenated by a vector of length 1. This is passed to an embedding 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.

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_{obs} time stamps $\boldsymbol{x}(t_1), \ldots, \boldsymbol{x}(t_{n_{\text{obs}}})$, the training and validation datasets are partitioned as follows. Let η 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 $\boldsymbol{\mathcal{X}}$ as well as the vectors $\boldsymbol{\mathcal{Y}}$ and $\boldsymbol{\mathcal{T}}$ of the Markov state and time sequences, as shown in Figure 4. To illustrate, applying this procedure to $\boldsymbol{\mathcal{X}}$, we obtain $[\boldsymbol{x}(t_1), \ldots, \boldsymbol{x}(t_{q_{\text{in}}^{\text{enc}}+q_{\text{out}}})]$, $[\boldsymbol{x}(t_2), \ldots, \boldsymbol{x}(t_{q_{\text{in}}^{\text{enc}}+q_{\text{out}}+1})]$, $\ldots$, $[\boldsymbol{x}(t_{\lfloor \eta n_{\text{obs}} \rfloor - q_{\text{in}}^{\text{enc}} - q_{\text{out}}+1}), \ldots, \boldsymbol{x}(t_{\lfloor \eta n_{\text{obs}} \rfloor})]$ for training and $[\boldsymbol{x}(t_{\lfloor \eta n_{\text{obs}} \rfloor+1}), \ldots, \boldsymbol{x}(t_{\lfloor \eta n_{\text{obs}} \rfloor + q_{\text{in}}^{\text{enc}} + q_{\text{out}}})]$, $\ldots$, $[\boldsymbol{x}(t_{n_{\text{obs}} - q_{\text{in}}^{\text{enc}} - q_{\text{out}}+1}), \ldots, \boldsymbol{x}(t_{n_{\text{obs}}})]$ for validation. The first $q_{\text{in}}^{\text{enc}}$ m-dimensional vectors are inputs to the deep learning model, while the subsequent q_{out} vectors constitute the target output sequence for the model. For

$\boldsymbol{\mathcal{Y}}$ and $\boldsymbol{\mathcal{T}}$, 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_{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_{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 hyper-parameters 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 Markov state and time sequences.

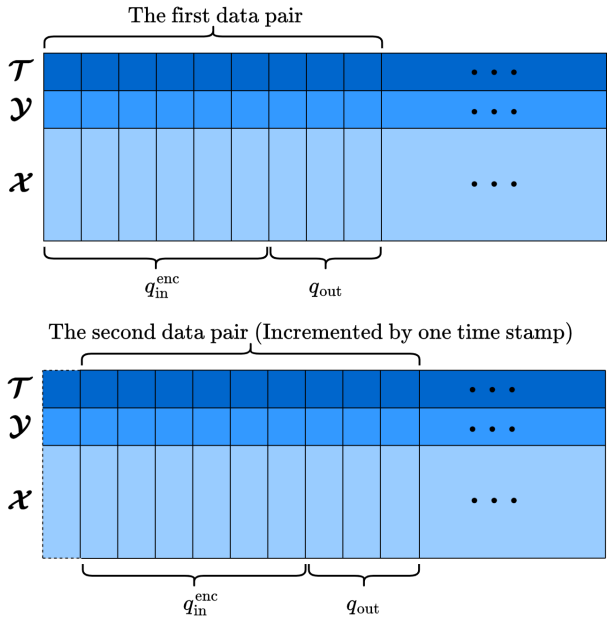


Figure 4: Construction of input-output data pairs. For each sequence of realizations, we apply a sliding window of length $q_{\text{in}}^{\text{enc}} + q_{\text{out}}$ to the time series matrix $\boldsymbol{\mathcal{X}}$ and the vectors $\boldsymbol{\mathcal{Y}}$ and $\boldsymbol{\mathcal{T}}$ of Markov 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_{out} components constitute the target output sequence for the model.

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 [22], 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, [33] 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 [20] 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

Notation	Description
$q_{\text{in}}^{\text{enc}}$	length of the input sequence for the deep learning model
q_{out}	length of the output sequence for the deep learning model
$q_{\text{in}}^{\text{dec}}$	length of the start sequence for the decoder
n_{clusters}	dimension of the Markov state space used for the clustering algorithm
d_{model}	dimension of the hidden embedding and attention layers
d_{ff}	dimension of the hidden feed-forward network
n_{head}	number of heads in the attention mechanism
n_{enc}	number of encoder blocks in the deep learning model for inference of m-variate processes
n_{dec}	number of decoder blocks in the deep learning model for inference of m-variate processes
n_{Markov}	number of decoder blocks in the deep learning model for Markov state sequence generation

Table 4: Standard hyperparameters of the GenFormer model.

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_{enc}, n_{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 [11].

Table 4 lists the standard hyperparameters of the proposed GenFormer model.

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_{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 $\boldsymbol{\mathcal{X}}^{\text{enc}}$, $\boldsymbol{\mathcal{Y}}^{\text{enc}}$, and $\boldsymbol{\mathcal{T}}^{\text{enc}}$ measured at the first $q_{\text{in}}^{\text{enc}}$ time points. Set $q_{\max} = \max(p, q_{\text{in}}^{\text{enc}})$. We therefore assume that data on the m-variate process $\tilde{\boldsymbol{x}}(t_1), \ldots, \tilde{\boldsymbol{x}}(t_{q_{\max}})$ 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 $\boldsymbol{x}(t_1), \ldots, \boldsymbol{x}(t_{q_{\max}})$ of length $q_{\max}$ and its corresponding Markov states $y_1, \ldots, y_{q_{\max}}$ from the given observations.

3.2.1 Simulation of m-variate stochastic processes

Simulation of a univariate Markov sequence. Given the data $\tilde{\boldsymbol{x}}(t_1), \ldots, \tilde{\boldsymbol{x}}(t_{q_{\max}})$ with the corresponding Markov states $\tilde{y}_1, \ldots, \tilde{y}_{q_{\max}}$ at time stamps $t_1, \ldots, t_{q_{\max}}$, sampling the univariate Markov sequence is initialized using the last p Markov states $\tilde{y}_{q_{\max}-p+1}, \ldots, \tilde{y}_{q_{\max}}$. 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 $\tilde{y}_{q_{\max}+2}, \ldots, \tilde{y}_{n_{\text{sim}}}$. 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 $\tilde{y}_{q_{\max}-p+1}, \ldots, \tilde{y}_{q_{\max}}$ and $t_{q_{\max}-p+1}, \ldots, t_{q_{\max}}$ as inputs to produce the sample $\tilde{y}_{q_{\max}+1}$ using the multinomial layer. At iteration l, the sequences $\tilde{y}_{q_{\max}-p+l}, \ldots, \tilde{y}_{q_{\max}+l-1}$ and $t_{q_{\max}-p+l}, \ldots, t_{q_{\max}+l-1}$ are then fed to the model to simulate $\tilde{y}_{q_{\max}+l}$. This process is repeated to generate $\tilde{y}_{q_{\max}+1}, \tilde{y}_{q_{\max}+2}, \ldots, \tilde{y}_{n_{\text{sim}}}$.

Inference from the deep learning model with Markov state embedding. The second step is to infer the m-variate stochastic process $\tilde{\boldsymbol{x}}(t_{q_{\max}+1}), \ldots, \tilde{\boldsymbol{x}}(t_{n_{\text{sim}}})$ corresponding to the synthetic realization $\tilde{y}_{q_{\max}+1}, \ldots, \tilde{y}_{n_{\text{sim}}}$ 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^{th} iteration, we infer $\tilde{\boldsymbol{x}}(t_{q_{\max}+(l-1)q_{\text{out}}+1}), \ldots, \tilde{\boldsymbol{x}}(t_{q_{\max}+lq_{\text{out}}})$. 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 comprised 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 $\tilde{\boldsymbol{\mathcal{X}}} = [\tilde{\boldsymbol{x}}(t_1), \ldots, \tilde{\boldsymbol{x}}(t_{n_{\text{sim}}})] \in \mathbb{R}^{m \times n_{\text{sim}}}$ 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 $\tilde{\boldsymbol{C}} = \tilde{\boldsymbol{\mathcal{X}}}\tilde{\boldsymbol{\mathcal{X}}}^T/n_{\text{sim}} \in \mathbb{R}^{m \times m}$ due to stationarity and ergodicity. Similarly, let $\boldsymbol{\mathcal{X}} = [\boldsymbol{x}(t_1), \ldots, \boldsymbol{x}(t_{n_{\text{obs}}})] \in \mathbb{R}^{m \times n_{\text{obs}}}$ be the matrix of observations of $\boldsymbol{X}(t)$. The target spatial correlation $\boldsymbol{C}$ of $\boldsymbol{X}(t)$ is approximated via $\boldsymbol{C} = \boldsymbol{\mathcal{X}}\boldsymbol{\mathcal{X}}^T/n_{\text{obs}}$. 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 [24]. 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 $\tilde{\boldsymbol{U}} = \boldsymbol{L}\tilde{\boldsymbol{L}}^T\tilde{\boldsymbol{\mathcal{X}}}$. The updated matrix $\tilde{\boldsymbol{U}}$ has the spatial correlation matrix $\boldsymbol{C}$ since

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

where $\mathbf{I} \in \mathbb{R}^{m \times m}$ 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 $\tilde{\boldsymbol{u}}_i \in \mathbb{R}^{n_{\text{sim}}}, i = 1, \ldots, m$, of $\tilde{\boldsymbol{U}}$ by first simulating n_{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_{sim} is sufficiently large, the reshuffled time series closely approximates the original, thereby minimizing the discrepancy in the spatial correlation matrix. The j^{th} column of the matrix $[\hat{\boldsymbol{x}}_1^T, \ldots, \hat{\boldsymbol{x}}_m^T]^T \in \mathbb{R}^{m \times n_{\text{sim}}}$ is the final realization $\hat{\boldsymbol{x}}(t_j)$.

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 comprised 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_{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_{sim} are large. We demonstrate these properties in Section 4.

Algorithm 1 GenFormer model

Model construction phase

Transform the data $\boldsymbol{x}(t_1), \ldots, \boldsymbol{x}(t_{n_{\text{obs}}})$ to have standard Gaussian marginal distributions
Partition the data into n_{clusters} clusters via K-means clustering in Section 3.1.1 to obtain the Markov state sequence $y_1, \ldots, y_{n_{\text{obs}}}$
if Markov order $p = 1$ **then**
    Estimate the Markov transition matrix from the realizations of $y_1, \ldots, y_{n_{\text{obs}}}$
else if Markov order $p \geq 2$ **then**
    Train the deep learning model for Markov state sequence generation in Section 3.1.3
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

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
if Markov order $p = 1$ **then**
    Simulate $\tilde{y}_{q_{\max}+1}, \ldots, \tilde{y}_{n_{\text{sim}}}$ using the estimated transition matrix
else if Markov order $p \geq 2$ **then**
    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
Apply the trained deep learning model with Markov state embedding on $\tilde{y}_{q_{\max}+1}, \ldots, \tilde{y}_{n_{\text{sim}}}$ to infer $\tilde{\boldsymbol{x}}(t_{q_{\max}+1}), \ldots, \tilde{\boldsymbol{x}}(t_{n_{\text{sim}}})$
Correct spatial correlation by applying transformation based on Cholesky decomposition in Section 3.2.2 to the synthetic realizations
Correct the marginal distributions using the reshuffling technique in Section 2.1.3

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 set the hyperparameters in the model architecture and training process empirically instead of employing hyperparameter tuning. This is because we already observed satisfactory performance of the trained model. However, if adequate computational resources are available, 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 [20] 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 [26] 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.

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 θ [13, p. 435]. More specifically, the mean and variance of the stationary process $Q_i(t)$ are α/β and α/β^2, respectively. The auto-correlation function of $Q_i(t)$ has exponential decay with rate θ and is given by $\exp(-\theta\tau)$, where τ is the time lag. The marginal distribution of $Q_i(t)$ follows the Gamma distribution with shape parameter α and rate parameter β [6], which can be derived through the stationary Fokker-Planck equation defined in [13, p. 482].

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

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

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 [5] 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 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 Gamma$(2, 1)$ and Φ^{-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 [41]. 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 $d_{\text{ff}} = 2048$. Since the time argument is unitless in this case, only the positional embedding is adopted as the time embedding in the embedding layer. Since 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, 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, e.g., n_{head}, d_{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.

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, i.e., $t \in [0, 0.04)$, and their corresponding Markov state sequences, to initialize the simulation of 5 synthetic sequences, resulting in synthetic dataset of 5000 sequences. The trained 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 3×10^6. The model post processing procedures in Section 3.2.2 are then applied to this concatenated matrix to obtain the final synthetic realizations $\hat{\boldsymbol{x}}(t)$.

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.

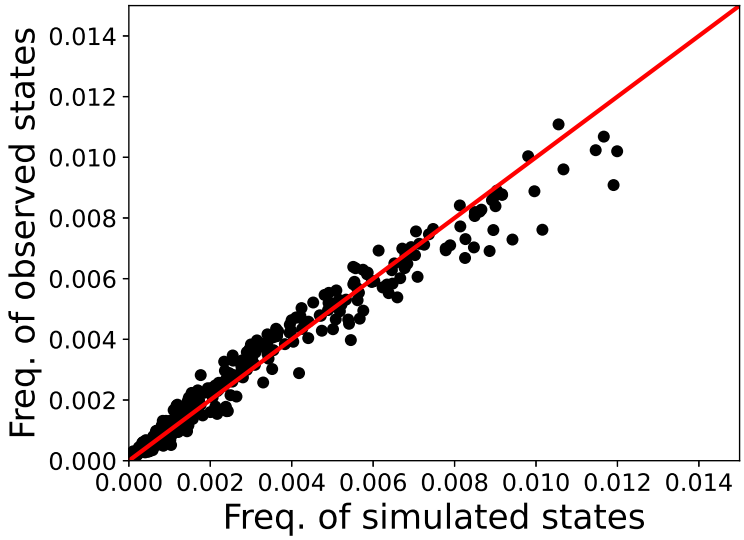
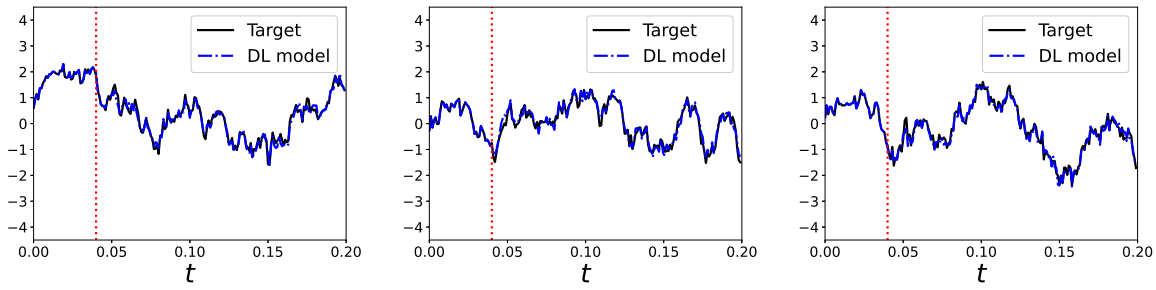


Figure 5: Scatter plot of the normalized frequencies of Markov states in the observed and simulated sequences. Generating Markov state sequences by estimating the transition matrix from data is computationally challenging for large Markov order p. This example shows that for large p, the trained deep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

We then evaluate the accuracy of the deep learning model for the mapping from the Markov state y_j to the time series $\boldsymbol{x}(t_j)$. The values of the L_1 losses on the training and validation datasets are 0.1145 and 0.1199, respectively. We visually assess the accuracy of the trained deep learning 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 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.

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 decom-



(a) $X_1(t)$ (b) $X_2(t)$ (c) $X_3(t)$

Figure 6: Target vs. 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.

position 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 [13], $\boldsymbol{X}(t)$ 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.

$$\begin{bmatrix} 1 & 0.47 & 0.48 \\ 0.47 & 1 & 0.45 \\ 0.48 & 0.45 & 1 \end{bmatrix} \qquad \begin{bmatrix} 0.98 & 0.46 & 0.50 \\ 0.46 & 0.94 & 0.44 \\ 0.50 & 0.44 & 0.96 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0.47 & 0.48 \\ 0.47 & 1 & 0.45 \\ 0.48 & 0.45 & 1 \end{bmatrix}$$

(a) Target matrix from observations $\boldsymbol{x}(t)$ (b) deep learning model estimate (c) Cholesky-based transformation estimate

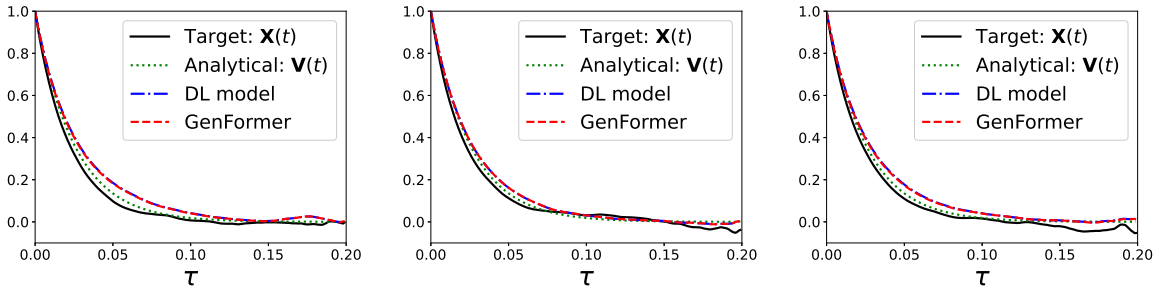
$$\begin{bmatrix} 1 & 0.47 & 0.47 \\ 0.47 & 1 & 0.45 \\ 0.47 & 0.45 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0.5 & 0.5 \\ 0.5 & 1 & 0.5 \\ 0.5 & 0.5 & 1 \end{bmatrix}$$

(d) GenFormer model estimate (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.

In Figure 8, we examine the auto-correlation functions of the components of $\boldsymbol{X}(t)$ and its various ap-

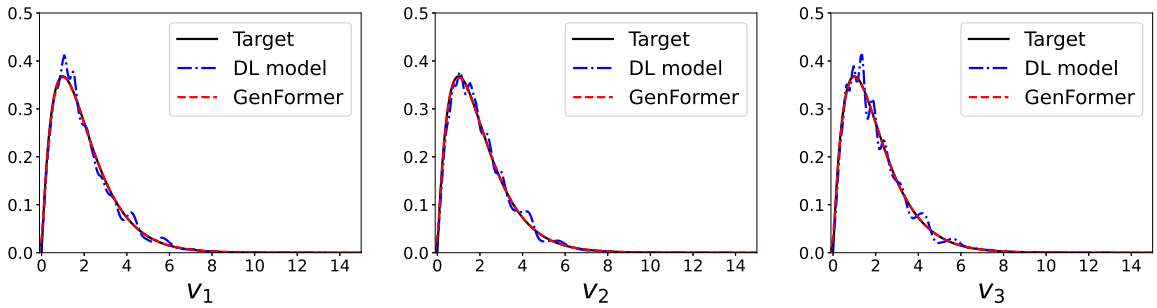
proximations. In each of the panels, the target is represented by the black solid line which is the estimate from the 1000 given realizations $\boldsymbol{x}(t)$. The blue dashdotted and red dashed lines are the estimates based on 5000 synthetic realizations $\tilde{\boldsymbol{x}}(t)$ and $\hat{\boldsymbol{x}}(t)$ 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 [13]. 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.



(a) $X_1(t)$ (b) $X_2(t)$ (c) $X_3(t)$

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.

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 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.



(a) Marginal density of $V_1(t)$ (b) Marginal density of $V_2(t)$ (c) Marginal density of $V_3(t)$

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.

Finally, we consider a downstream application of the GenFormer model in risk management. A met-

ric 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 [40], 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^{th} component is expressed as $V_{T,i}(t) = F_i^{-1}[\Phi(X_i^*(t))]$, where F_i is the marginal distribution of $V_i(t)$ and $\boldsymbol{X}^*(t) = [X_1^*(t), \ldots, X_m^*(t)]^T$ 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 $F_i^{-1}[\Phi(X_i^*(t))]$ is then applied to each component of these samples to obtain samples of $\boldsymbol{V}_T(t)$.

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 $s \in [0, 25]$ estimated using the given observations (black solid line), and the synthetic realizations produced by the translation model (blue 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 and the GenFormer model are 0.3825 and 0.0680, respectively. We notice that the exceedance probability curve due to the proposed GenFormer closely follows the target while the curve produced by the translation model deviates 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 two 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.

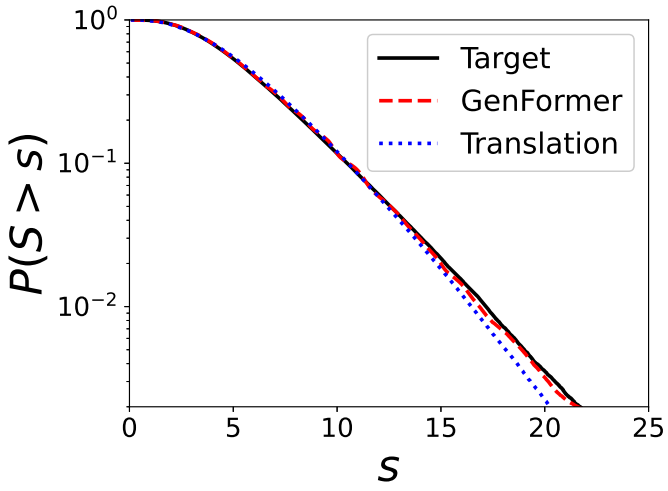


Figure 10: Exceedance probability of $S(t)$. The relative error in the return period attained by the proposed GenFormer model is approximately an order of magnitude lower than that of the translation model. The GenFormer model can capture higher-order statistical properties of $\boldsymbol{X}(t)$ beyond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

Station ID	Station Name	State	Latitude	Longitude
747830	FT LAUD/HOLLYWOOD INTL APT	FL, US	26.079	-80.162
722037	NORTH PERRY AIRPORT	FL, US	26.000	-80.241
722024	OPA LOCKA AIRPORT	FL, US	25.910	-80.283
722020	MIAMI INTERNATIONAL AIRPORT	FL, US	25.788	-80.317
722029	KENDALL-TAMIAMI EXEC ARPT	FL, US	25.642	-80.435
722026	HOMESTEAD AFB AIRPORT	FL, US	25.483	-80.383

Table 5: Weather stations in Florida selected in this work.

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 6 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. 5. 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 is 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 is transformed to the Gaussian space.

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 architecture of the deep learning model for inference of the m-variate process, 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 two days based on the observed wind speeds in the previous two days. Both the encoder and decoder consists of four blocks each, i.e., $n_{\text{enc}} = n_{\text{dec}} = 4$, while d_{model}, n_{head}, and d_{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 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

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

stamps is $n_{\text{sim}} = 672 = 28 \times 24$. The simulation of synthetic realizations begins by extracting the observed data at $q_{\max} = 48$ 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.

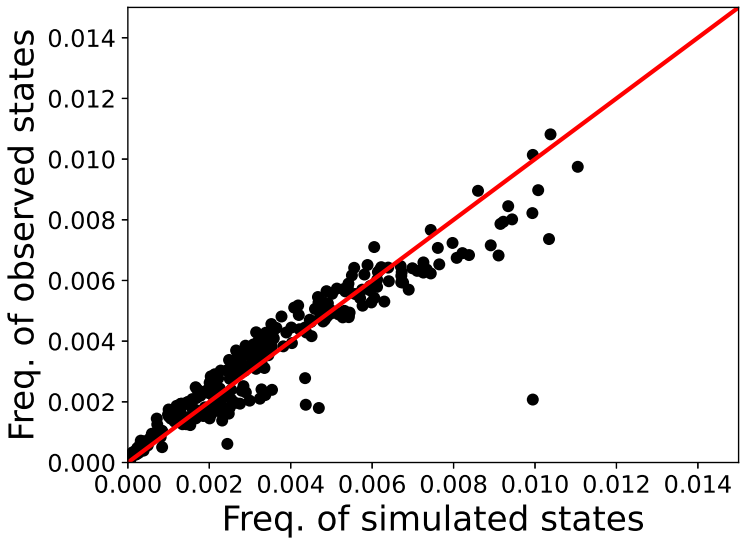
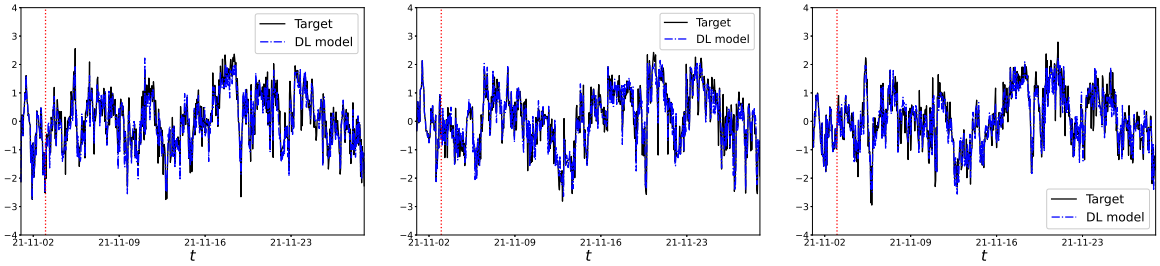


Figure 11: Scatter plot of the normalized frequencies of Markov states in the observed and simulated sequences. Estimating the transition matrix for large p is prohibitive since the transition matrix would have dimension $300^{36} \times 300$. In this example, the trained deep learning model for Markov 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.

The trained deep learning model for inference of the 6-variate wind speed time series achieves L_1 losses of 0.2296 and 0.2504 on the training and validation sets, respectively. We 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 is 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.



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

Figure 12: Target vs. synthetic time series produced by the deep learning model for inference of 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

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

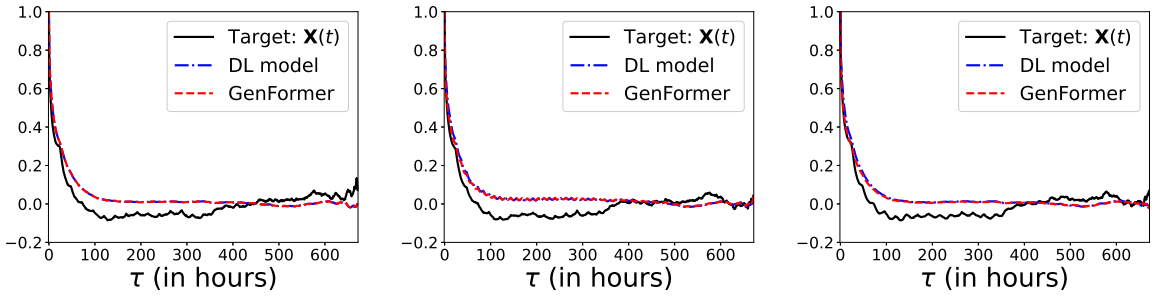
(d) Estimate from the GenFormer model

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.

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 target which highlights that

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

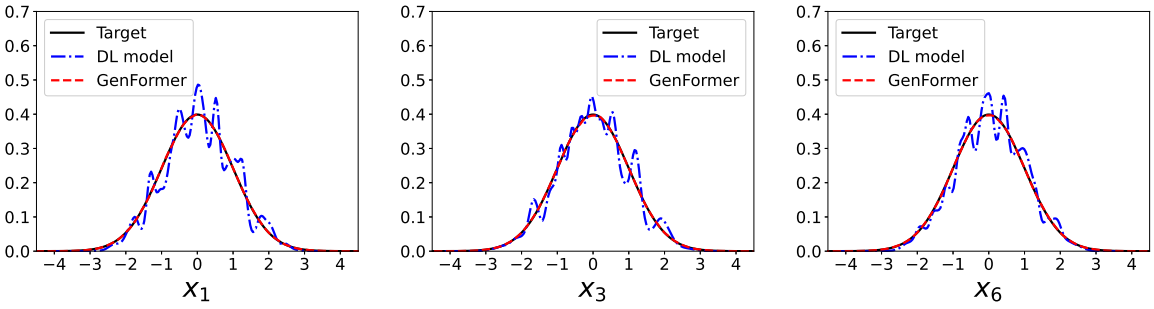
Figure 14 plots the auto-correlation functions of $X_1(t)$, $X_3(t)$, and $X_6(t)$ obtained from the collected wind speed data (black solid line), samples simulated from the trained deep learning model (blue dashdotted line), and samples generated using the GenFormer model (red dashed line). The estimate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



(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.

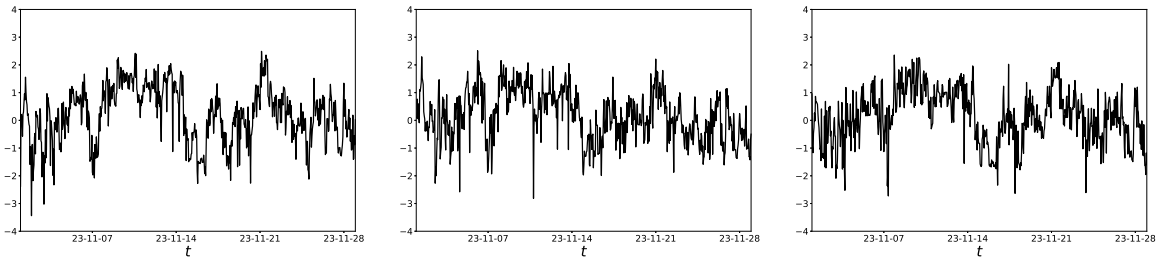
In Figure 15, we show estimates of the marginal densities of $X_1(t)$, $X_3(t)$, and $X_6(t)$ in the Gaussian space. The black solid line marks the target standard Gaussian density. The blue dashdotted line is the estimate computed using inferences from the deep learning model prior to the model post-processing 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 of the synthetic realizations which the deep learning model alone does not account for in its training process.



(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.

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.



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

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

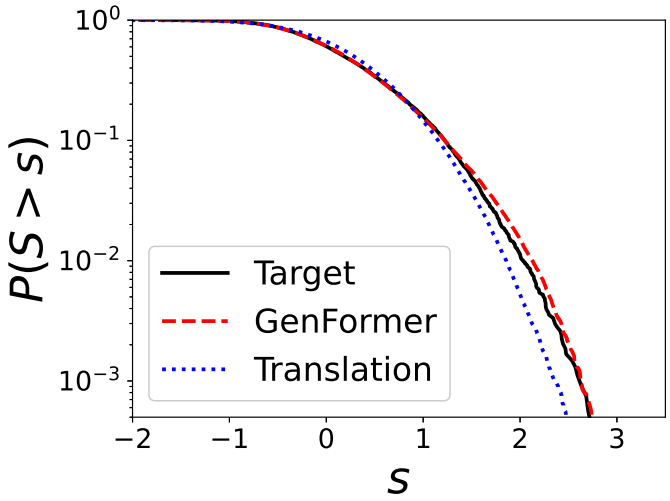


Figure 17: Exceedance probability of S. The estimate obtained from the Genformer model is 7.6 times more accurate than the translation model. The predictive capabilities of the deep learning model coupled with the statistical post-processing techniques enable the GenFormer model to capture high-order statistical properties while the translation model can only match up to the second-moment properties.

The downstream application we pursue in this example is defined as the maximum of the monthly-averaged hourly wind speeds across stations given by $S = \max_{1 \le i \le 6} \{\int_0^{T_{\text{sim}}} X_i(t)dt/T_{\text{sim}}\}$. Such a metric can be used as a measure of the local hurricane intensity which is of interest in parametric insurance [23]. 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. The translation process is adopted as the baseline model for comparison. Similar to the previous example, we see that the approximation due to the translation model deteriorates 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 error of the translation process which is 0.9713 compared to that of the GenFormer which is 0.1281, a 7.6 times improvement over the former model. This improvement 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 and the translation model are 1376 and 6088 months, reflecting a 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.

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. Utilizing state-of-the-art attention mechanisms in the Transformer architecture can further improve the performance of the proposed approach.

References

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

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

[3] J.L. Ba, J.R. Kiros, and G.E. Hinton. Layer normalization. *arXiv:1607.06450*, 2016.

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

[5] M. Bayram, T. Partal, and G. Orucova Buyukoz. Numerical methods for simulation of stochastic differential equations. *Adv Differ Equ*, 17, 2018.

[6] B.M. Bibby, I.M. Skovgaard, and M. Sørensen. Diffusion-type models with given marginal distribution and autocorrelation function. *Bernoulli*, 11(2):191–220, 2005.

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

[8] P. Brémaud. *Discrete-Time Markov Models*, pages 53–93. Springer New York, New York, NY, 1999.

[9] L. Deng, G. Hinton, and B. Kingsbury. 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*, pages 8599–8603, 2013.

[10] J. Devlin, M.W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805*, 2018.

[11] P.I. Frazier. A tutorial on bayesian optimization. *arXiv:1807.02811*, 2018.

[12] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[13] M. Grigoriu. *Stochastic Calculus*. Springer, New York, NY, December 2013.

[14] J.A. Hartigan and M.A. Wong. A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

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

[16] J.L. Horowitz. Chapter 52 - the bootstrap. volume 5 of *Handbook of Econometrics*, pages 3159–3228. Elsevier, 2001.

[17] R. Ibragimov. Copula-based characterizations for higher order markov processes. *Econometric Theory*, 25:819–846, 2009.

[18] M.T. Islam, B.M.N. Karim Siddique, S. Rahman, and T. Jabid. Image recognition with deep learning. In *2018 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS)*, volume 3, pages 106–110, 2018.

[19] R.W. Katz. On some criteria for estimating the order of a markov-chain. *Technometrics*, 23(3):243–249, 1981.

[20] D.P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2015.

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

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

[23] K.S. Ng, G.C. Leckebusch, Q. Ye, W. Ying, and H. Zhao. On the use of ensemble predictions for parametric typhoon insurance. *Climate*, 9(12):174, 2021.

[24] J.H. Nicholas. Analysis of the cholesky decomposition of a semi-definite matrix. *Reliable Numerical Computation*, pages 161–185, 1990.

[25] Z. Ouyang and S.M.J. Spence. 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, 2021.

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

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

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

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

[30] G. Schwarz. Estimating dimension of a model. *Ann. Stat.*, 6(2):461–464, 1978.

[31] I. Sutskever, O. Vinyals, and Q.V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, page 3104–3112, 2014.

[32] L. Vandanapu and M.D. Shields. 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, 2021.

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

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

[35] H. Xu, M. Grigoriu, and K.R. Gurley. A novel surrogate for extremes of random functions. *"Reliability Engineering & System Safety"*, 239:109493, 2023.

[36] L. Yang, K.R. Gurley, and D.O. Prevatt. Probabilistic modeling of wind pressure on low-rise buildings. *Journal of Wind Engineering and Industrial Aerodynamics*, 114:18–26, 2013.

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

[38] W. Yin, k. Kann, M. Yu, and H. Schutze. Comparative study of cnn and rnn for natural language processing. *arXiv:1702.01923*, 2017.

[39] H. Zhao. *Probabilistic Models for Wind Loads and Reliability Analysis*. PhD thesis, Cornell University, 2017.

[40] H. Zhao, M. Grigoriu, and K.R. Gurley. Translation processes for wind pressures on low-rise buildings. *Journal of Wind Engineering and Industrial Aerodynamics*, 184:405–416, 2019.

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

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