Internet of Things 30 (2025) 101500

Contents lists available at ScienceDirect

Internet of Things

journal homepage: www.elsevier.com/locate/iot

Research article

Creating interpretable synthetic time series for enhancing the design and implementation of Internet of Things (IoT) solutions☆

Dimitris Gkoulis

Department of Informatics and Telematics, Harokopio University of Athens, Omirou 9, Athens, 17779, Greece

ARTICLE INFO

Keywords:
Internet of Things (IoT)
Synthetic data
Synthetic time series
IoT simulation

ABSTRACT

This study establishes a foundation for addressing the challenge of developing Internet of Things (IoT) solutions in the absence of real-world data, a common obstacle in the early stages of IoT design, prototyping, and testing. Motivated by the need for reliable and interpretable synthetic data, this work introduces a structured approach and a dedicated library for creating realistic time series data. The methodology emphasizes flexibility and modularity, allowing for the combination of distinct components–such as trends, seasonality, and noise–to create synthetic data that accurately reflects real-world phenomena while maintaining interpretability. The approach's utility is demonstrated by creating synthetic air temperature time series, which are rigorously compared against real-world datasets to assess their fidelity. The results validate the proposed methodology's and library's effectiveness in producing data that closely mirrors real-world patterns, providing a robust tool for IoT development in data-constrained environments.

1. Introduction

The Internet of Things (IoT) [1] is a paradigm that describes an intricate structure where intelligent things are interconnected and interact with each other. This structure also involves humans and systems. The fundamental characteristic of these things is their capability for sensing (via sensors) and acting (via actuators) and, inherently, their ability to communicate (through a network or the Internet).

This concept is indisputably well-established and rapidly evolving. As technology advances, revolutionary solutions with significant technical, social, and economic impact are continuously built upon the concept of the IoT [2–5].

Research interest is particularly high in designing and creating user-centric IoT solutions despite numerous technological and non-technological challenges [6,7]. This study aims to mitigate or even tackle a major technological challenge: designing, implementing, and testing IoT solutions in the absence of data, a critical element of the IoT.

Starting with limited resources, potentially without any things, there is a need for fast prototyping [8] to understand the physical environment and monitor it effectively [9]. This is necessary not only for developing new solutions but also for testing existing ones. This study will focus on the early stages of designing and modeling an IoT solution under the condition that no things are available, and at the same time, utilizing a testbed is impractical due to the complexity and potential costs [10].

The study focuses on the initial conceptual challenge of IoT: the point at which a phenomenon or event occurring in the physical world is transformed into information [11]. This involves an intelligent (or non-intelligent) thing sensing, measuring, and producing a datum, which, either alone or in combination with another, transforms into meaningful information in a different conceptual layer.

☆ **Funding:** This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.
E-mail address: gkoulis@hua.gr.

https://doi.org/10.1016/j.iot.2025.101500
Received 20 August 2024; Received in revised form 26 November 2024; Accepted 7 January 2025
Available online 17 January 2025

This information then informs humans or systems, potentially leading to decision-making.

Typically, the conception and design of an IoT solution begin with defining the context and determining what, how, and when to measure [12,13]. However, designing and developing IoT solutions in the absence of original real-time or historical data presents a unique challenge. There are three general approaches: using open real data similar to the domain, creating data, or initially not using data. Not using data is not viable, as evidence suggests that creating a solution without testing is infeasible.

Using real data is extremely difficult due to issues such as data quality, availability, licensing, and completeness [14]. Despite these challenges, such data remains valuable for visual and statistical analysis, modeling, and extracting information to understand underlying phenomena, repeated patterns, uncertainty, correlations, dependencies, and occurrences.

This study emphasizes the creation of synthetic data [15], which is generated using purpose-built models to solve specific tasks. Synthetic data mimics real-world data's structure, properties, and characteristics but is not collected from actual observations or events [16]. Instead, it is created through mathematical models, algorithms, or other simulation methods. The term "synthetic data" is often associated with GANs and machine learning techniques, while simulated data is commonly used in traditional statistics. These data are sometimes also referred to as artificial, fake, or dummy.

This research focuses on synthetic data's temporary nature, particularly time series data [17], as it is a crucial representation for analyzing and understanding phenomena or events within a broader context. Time series data, representing the context under study, is essential for analyzing temporal patterns and dependencies.

The motivation behind this research is the critical need for data in the effective design, prototyping, and testing of IoT solutions. Given the significant challenges associated with acquiring real-world data–such as data quality issues, availability constraints, licensing complications, and incomplete datasets–this study proposes an approach to generate interpretable synthetic time series data. Such synthetic data can serve as a valuable substitute or complement to real data, supporting the essential phases of IoT solution development. The objective is to develop a flexible and generic library capable of generating time series data that accurately reflects various monitoring phenomena, events, and sensor behaviors. The created data can then be employed to test APIs, test middleware, feed simulators, perform what-if scenarios, emulate sensor behavior, understand the effects of phenomena in decision-making, modernize business processes, and perform other essential tasks involved in creating a modern reactive IoT application.

The document is structured as follows: Section 2 reviews existing methods and approaches for generating synthetic data in the IoT domain, highlighting their advantages and limitations. Section 3 introduces a library for creating synthetic time series, detailing its design, functionalities, and capabilities. Section 4 showcases the practical application of the library, providing concrete examples of its use. Section 5 examines the implications of this work, potential improvements, and directions for future research. Finally, Section 6 summarizes the findings and contributions of this study.

2. Related work

The literature reveals several methods for generating synthetic data in the context of IoT [18], including rule-based methods, simulation-model-based methods, traditional machine-learning-based methods, and deep-learning-based methods. Each approach has its advantages and limitations.

Several frameworks have been proposed to generate synthetic IoT data, replicating the complex characteristics of real-world datasets without compromising privacy. One such framework presented the design and implementation of a synthetic IoT data generation system to support research while preserving privacy [19]. Another study introduced the Generic Methodology for Constructing Synthetic Data (GeMSyD), which generates synthetic datasets emulating human–device interactions for smart devices, demonstrating its utility in machine learning model development [20].

In precision agriculture, the use of generative adversarial networks (GANs) to generate synthetic temperature data for greenhouses significantly improved AI/ML model accuracy [21]. Similarly, an approach leveraging GANs for generating time series data on edge devices showcased the potential of real-time synthetic data generation for machine learning applications [22].

The need for high-quality datasets in smart city applications led to the development of AirGen, a GAN-based synthetic data generator for air quality monitoring, which improved prediction accuracy for deep learning models [23]. For smart home environments, synthetic data generation incorporating time dependency and usage continuity of appliances helped model user behavior patterns more reliably [24]. The Synthetic Time Series Data Generator (SynTiSeD) used multi-agent-based simulations to generate meaningful energy data for smart living applications, demonstrating superior performance in machine learning models trained on synthetic data [25].

In the security domain, synthetic data has been employed to enhance intrusion detection systems (IDS) in IoT networks. A model-hybrid approach was proposed to generate bespoke datasets for training IDS to predict a wide spectrum of real-world attacks effectively [26]. Another study explored the use of GANs to generate synthetic data for network intrusion detection, achieving high performance metrics and reducing dependency on real-world data [27]. The Gotham Testbed provided a reproducible IoT testbed for security experiments, facilitating the generation of up-to-date datasets for machine learning-based security systems [28].

Healthcare applications have also benefited from synthetic data generation. The SynSys framework improved the realism and complexity of synthetic sensor data for healthcare applications, enhancing activity recognition accuracy through semi-supervised learning [29]. Digital twins for stress management employed generative models to predict stress scores from wearable device data, demonstrating high accuracy and data quality [30]. Recurrent GANs were used to generate realistic medical time series data, proving useful for supervised training in early warning systems [31]. Another study combined GANs with differential privacy mechanisms to generate realistic and private healthcare datasets, preserving the statistical properties of original data [32].

In manufacturing, synthetic data has enabled the development of digital twins for production systems analysis. A strategy for generating random production lines and simulating their behavior was proposed to create synthetic data for various applications, including bottleneck analysis [33]. Addressing challenges in data generation for advanced manufacturing, a study explored the issues and requirements for building virtual data generators, emphasizing the importance of data provenance [14]. The GENLOG approach generated reliable event and time series data for process monitoring and analysis, boosting small datasets for online process mining applications [34].

Synthetic data generation methods have been applied to wind speed simulation, where reconstruction methods were developed to generate time series data with accurate statistical properties [35]. Markov chain models were used to generate wind speed data, preserving the statistical characteristics of observed data [36]. A GMM-HMM-based method generated correlated power output time series for multiple wind farms, demonstrating superior statistical characteristics compared to traditional methods [37].

In cloud computing, a novel method for creating synthetic workload traces for data centers was proposed, facilitating realistic evaluation of cloud environments [38]. For maritime applications, synthetic data generation techniques were employed to improve the accuracy of machine learning models for steam turbine analysis [39]. Another study mapped time series data on process patterns to generate synthetic data, contributing to better causal-effect understanding in time series analysis [40]. Synthetic data was also used to enhance weather classification models, increasing training efficiency and addressing dataset bias [41].

The Synthetic Data Vault (SDV) system was developed to create synthetic data for relational databases, showing that synthetic data can effectively replace real data in data science endeavors [42]. A flexible data generation tool for relational databases was introduced, capable of generating realistic data for various applications [43]. Another framework provided a scalable solution for database generation, supporting varied data distributions and facilitating reproducibility [44]. The synthpop package in R enabled the creation of synthetic tabular data, preserving relationships between variables while ensuring data privacy [45].

Generative adversarial networks (GANs) have been adapted for time series data generation, preserving temporal dynamics and improving sample realism [46]. The RTSGAN framework addressed challenges in generating real-world time series with missing values, demonstrating superior performance in downstream tasks [47]. The GRATIS approach utilized mixture autoregressive models to generate diverse and controllable time series for benchmarking purposes [48]. Another study proposed interpretable methods for generating synthetic data using features from continuous functions, achieving indistinguishable patterns from real data [49].

While these approaches provide valuable frameworks and tools for creating synthetic data, they also present notable limitations. Rule-based methods, although straightforward, often lack the sophistication required to emulate complex behaviors. Simulation-based approaches, while accurate, demand significant domain expertise and computational power. Traditional machine learning methods depend on the availability of extensive historical data and the precision of the models used. Deep learning approaches, especially GANs, although capable of generating highly realistic data, face challenges in training complexity, computational resource requirements, and sometimes fail to capture temporal dependencies accurately. highlights how this study and the proposed approach differ, complement, or advance over the existing approaches presented.

The next section introduces a proof-of-concept library designed to facilitate the creation of IoT time series data in the absence of real data. This library aims to establish a structured approach to generating time series, emphasizing interpretability and extensibility. While it does not exclude the current methods, it integrates them as long as they contribute to the interpretability of the time series creation process, thus addressing some of the limitations of existing approaches.

3. iMimic: an open-source library for creating realistic and interpretable synthetic time series

The iMimic library is a modular, flexible, and generic tool designed to address the need for realistic synthetic time series data in the IoT domain. Developed with a focus on interpretability, iMimic allows users to create time series data that closely resembles real-world phenomena, even in the absence of actual data. The library is open-source and structured to support extensive customization, enabling users to combine different components–such as trend, seasonality, and noise–to produce data that meets specific requirements. iMimic is designed to be extensible, allowing users to integrate their own models and methods, whether simple statistical models or complex machine learning algorithms. Additionally, the library includes a suite of tools for analyzing and validating the generated data.

The following subsections provide a detailed examination of the iMimic library's core functionalities, including time point generation, data point generation, and sampling from existing sequences.

3.1. *Time (point) generators*

Time Generators generate time point sequences. There are two categories of time generators: regular and irregular. Each time generator is defined by specific parameters (e.g., start, stop, step, length, distribution, etc.), which can be adjusted to reflect the desired characteristics.

There are four methods for creating time point sequences: two for regular time point sequences and two for irregular ones:

First Method for Creating Regular Time Point Sequences: This method is defined by three parameters: start, stop, and step. It generates a sequence of time points beginning from the start value (inclusive) and incrementing by the step size until the stop value (exclusive) is reached. The consistent step interval ensures that the time points are uniformly spaced. This method is particularly useful when the objective is to maintain a fixed temporal resolution across the sequence.

Second Method for Creating Regular Time Point Sequences: This method also utilizes three parameters: start, stop, and length. It generates a sequence of evenly spaced time points, with the number of points specified by the length parameter. The time points

Table 1
Comparison of related works.

#	Methodology	Application domain	Key focus	Comparison summary
Proposed approach	Flexible library; custom models	General IoT time series	Realistic, interpretable synthetic data	Offers a general-purpose, flexible library for generating synthetic time series across IoT domains.
[19]	Rule-based	General IoT data	Privacy-preserving data	Focuses on privacy using predefined rules; the proposed approach offers a flexible library for interpretable synthetic time series without relying on real data.
[20]	Rule-based	Smart devices	Modeling temporal dependencies	Emphasizes domain-specific data; the proposed approach provides a general framework applicable across various IoT phenomena, offering more flexibility.
[21]	GAN-based	Precision agriculture	Improving ML accuracy	Uses GANs for temperature data; the proposed approach focuses on interpretable synthetic time series without relying on complex models.
[22]	GAN-based	Edge computing	Real-time data generation	Relies on GANs on edge devices; the proposed approach offers a flexible library without relying on GANs, also suitable for resource-constrained IoT development.
[23]	GAN-based	Smart city air monitoring	Enhancing ML prediction	Improves ML models with GANs; the proposed approach provides interpretable time series generation without relying on GANs.
[24]	Rule-based	Smart homes	Modeling user behavior	Domain-specific rules; the proposed approach offers a flexible framework applicable across various IoT domains.
[25]	Simulation-based	Smart energy data	Generating energy data	Uses agent-based simulations; the proposed approach provides a flexible library without relying on specific simulations.
[26]	Model-hybrid	IoT security	Intrusion detection systems	Focuses on security datasets; the proposed approach offers a general-purpose library not limited to security.
[27]	GAN-based	IoT cybersecurity	Training ML models	Emphasizes GANs for security data; the proposed approach provides an interpretable library without relying on GANs or focusing on security.
[28]	Simulation-based	IoT security testing	Security experiments	Provides a testbed for security; the proposed approach offers a flexible library across IoT applications.
[29]	ML-based	Healthcare sensors	Improving activity recognition	Uses ML models in healthcare; the proposed approach provides a general framework without relying solely on ML models.
[30]	GAN/VAE-based	Healthcare wearables	Predicting stress scores	Relies on complex models; the proposed approach offers a flexible library without relying complex generative models.
[31]	GAN-based	Medical time series	Assisting supervised training	Uses GANs; the proposed approach focuses on interpretable time series with controlled computational complexity.
[32]	GAN-based	Smart healthcare	Realistic private data	Combines GANs with privacy; the proposed approach provides a flexible method without focusing on privacy or healthcare.
[33]	Simulation-based	Manufacturing	Data for production analysis	Focuses on simulations; the proposed approach offers a general library beyond manufacturing.
[34]	Neural networks	Process monitoring	Boosting small datasets	Uses recurrent networks; the proposed approach provides a flexible library without relying on neural networks.
[35]	Reconstruction	Wind speed simulation	Preserving statistics	Specific methods; the proposed approach offers flexibility without specific techniques.
[36]	Markov models	Wind speed series	Preserving characteristics	Limited to wind speed; the proposed approach provides a general framework without being model-specific.
[37]	GMM-HMM	Wind farm output	Correlated time series	Focuses on energy models; the proposed approach offers an interpretable library across IoT domains.
[38]	Model-based	Cloud workloads	Replicating workloads	Targets cloud; the proposed approach provides a flexible library for IoT applications.
[39]	ML-based	Maritime engineering	Steam turbine analysis	Uses complex models; the proposed approach offers a general framework without relying on such models.

(*continued on next page*)

Table 1 (*continued*).

#	Methodology	Application domain	Key focus	Comparison summary
[40]	Simulation	Time series mapping	Understanding causal effects	Focuses on process mapping; the proposed approach provides a flexible library without specific simulations.
[41]	Simulation-based	Weather images	Improving classification	Deals with images which time series analysis and classification; the proposed approach focuses on time series data for IoT.
[42]	Multivariate modeling	Data science	Synthetic relational data	Focuses on databases; the proposed approach targets time series for IoT.
[43]	Graph-based	Database testing	Realistic test data	Specific to databases; the proposed approach offers a flexible library for IoT time series.
[44]	Flexible framework	Database evaluation	Modeling distributions	Targets databases; the proposed approach provides a library for time series in IoT.
[45]	Statistical modeling	Tabular data	Preserving relationships	Focuses on tabular data; the proposed approach generates time series for IoT.
[46]	GAN-based	General time series	Preserving dynamics	Uses GANs; the proposed approach emphasizes interpretability without relying on complex models.
[47]	GAN-based	Time series with missing values	Using RTSGAN	Relies on complex GANs; the proposed approach offers an interpretable library with controlled complexity.
[48]	MAR models	Benchmarking	Diverse time series	Aims at benchmarking; the proposed approach focuses on IoT development with interpretable time series.
[49]	GAN-based	Industrial data	Control over components	Uses GANs; the proposed approach provides flexibility without relying on GANs.
[14]	Virtual data	Manufacturing	Exploring challenges	Discusses challenges; the proposed approach offers practical tools for IoT domains.

are distributed uniformly between the start (inclusive) and stop (inclusive) values. This approach allows for precise control over the number of time points within a defined range, offering flexibility in determining the temporal resolution.

First Method for Creating Irregular Time Point Sequences: This method utilizes four parameters: start, stop, step, and random distribution. Initially, a regular grid of time points is generated using the start, stop, and step parameters. Random perturbations, derived from the specified random distribution, are then applied to each time point in the grid. The perturbed time points are subsequently sorted to maintain temporal order, ensuring a consistent sequence. This method effectively introduces controlled irregularity into a fundamentally regular time sequence.

Second Method for Creating Irregular Time Point Sequences: This method employs two parameters: length and random distribution. It generates time points by producing a sequence of random increments representing the intervals between consecutive points based on the specified random distribution. These increments are cumulatively summed to generate the initial time points. However, these time points may not be in order due to the randomness in the intervals. Therefore, the time points are subsequently sorted to ensure they are in ascending order, finalizing the irregular time sequence. This method focuses on creating time intervals with variability rather than altering a pre-established regular grid, allowing for generating inherently irregular time sequences.

In application, regular time sampling is utilized in scenarios such as simulating sensor data collection at fixed intervals, ensuring consistency and predictability in the dataset. Conversely, irregular time sampling is employed to reflect real-world conditions where data collection times may be influenced by external factors, such as network latency or varying response times, thus providing a more realistic temporal representation.

3.2. Data (point) generators

Data Generators generate data point sequences. They are parameterizable methods that, given a sequence of time points, produce corresponding data points based on specified mathematical or algorithmic models. These generators provide structured outputs following patterns or distributions, such as linear trends, exponential growth, or random processes. The fundamental characteristic of these data generators is that they operate under the assumption that no specific values are known beforehand. Instead, only the parameters or characteristics that define how these values are generated, always concerning time, are provided.

The **Flat Sequence** generator produces a sequence with a constant value over time, representing an unchanging state.

The formula is:

$$y(t) = c$$

where c is a constant. This generator is useful for modeling systems in equilibrium, baseline levels, or control conditions, serving as a reference level for comparison with more complex data patterns.

The **Linear Sequence** generator produces a sequence where values change linearly over time.

The formula is:

$$y(t) = mt + b$$

where m is the slope and b is the intercept. This generator is applicable for modeling steady changes over time, such as gradual increases or decreases in a variable, and can be used to identify trends or long-term changes in data.

The **Polynomial Sequence** generator creates data points that follow a polynomial function of the independent variable.

The formula is:

$$y(t) = a_n t^n + a_{n-1} t^{n-1} + \cdots + a_1 t + a_0$$

where $a_n, a_{n-1}, \ldots, a_0$ are polynomial coefficients. This generator captures complex, non-linear relationships in data, such as acceleration or deceleration in growth rates, and is useful in modeling scenarios where the rate of change itself changes over time.

The **Logarithmic Sequence** generator produces data points that follow a logarithmic function, often used to model diminishing or saturation effects.

The formula is:

$$y(t) = a \ln(bt)$$

where a and b are constants. This generator is ideal for modeling phenomena where growth decreases over time, such as learning curves or diminishing effects.

The **Exponential Sequence** generator creates a sequence of data points based on an exponential function, capable of representing both growth and decay.

The formula is:

$$y(t) = y_0 e^{rt}$$

where y_0 is the initial value and r is the rate of change. This generator is effective in modeling rapid changes, describing processes that double or halve at a constant rate.

The **Geometric Sequence** generator produces a sequence where each term is a constant ratio of the previous term.

The formula is:

$$y(t) = y_0 r^t$$

where y_0 is the initial value and r is the common ratio. This generator is useful for modeling multiplicative processes, where each step involves scaling by a constant factor.

The **Logistic Sequence** generator produces data points that follow a logistic function, which models bounded growth.

The formula is:

$$y(t) = \frac{L}{1 + e^{-k(t-t_0)}}$$

where L is the maximum value, k is the growth rate, and t_0 is the midpoint. This generator is ideal for modeling growth that saturates at a certain level, such as when a system approaches a maximum capacity or limit.

The **Gaussian Noise Sequence** generator adds normally distributed noise to a sequence.

The formula is:

$$y(t) = \mathcal{N}(\mu, \sigma^2)$$

where $\mathcal{N}$ represents the normal distribution with mean μ and variance σ^2. This generator is used to simulate random fluctuations and measurement errors in data, providing a realistic noise component for testing and analysis.

The **Sinusoidal Wave Sequence** generator produces a smooth, periodic waveform.

The formula is:

$$y(t) = A \sin\left(\frac{2\pi t}{T} + \phi\right)$$

where A is the amplitude, T is the period, and ϕ is the phase shift. This generator is useful for modeling cyclic phenomena such as oscillations, seasonal variations, or any periodic behavior in data.

The **Sawtooth Wave Sequence** generator creates a waveform with a linear rise and a sharp drop.

The formula is:

$$y(t) = 2A\left(\frac{t}{T} - \left\lfloor \frac{1}{2} + \frac{t}{T} \right\rfloor\right)$$

where A is the amplitude and T is the period. This generator is used to model repetitive processes that reset after a certain point.

The **Triangular Wave Sequence** generator produces a waveform with linear rises and falls.

The formula is:

$$y(t) = 2A\left|\frac{t}{T} - \left\lfloor \frac{t}{T} + 0.5 \right\rfloor\right|$$

where A is the amplitude and T is the period. This generator is suitable for representing repetitive processes with symmetrical rises and falls.

The **Step Transition Sequence** generator creates a sequence with abrupt changes.

The formula is:

$$y(t) = k \left\lfloor \frac{t}{\Delta t} \right\rfloor$$

where k is the step size and Δt is the interval between steps. This generator is useful for modeling sudden shifts or step changes in a system, such as transitions between discrete states.

The **Sigmoid Transition Sequence** generator produces a smooth, sigmoid-shaped transition.

The formula is:

$$y(t) = \frac{1}{1 + e^{-k(t-t_0)}}$$

where k controls the steepness and t_0 is the midpoint of the transition. This generator is ideal for modeling gradual transitions between states, useful in scenarios involving smooth shifts or growth approaching a limit.

The **Autoregressive (AR) Sequence** generator creates data based on a linear combination of past values.

The formula is:

$$y(t) = \phi_1 y(t-1) + \phi_2 y(t-2) + \cdots + \phi_p y(t-p) + \epsilon(t)$$

where ϕ_i are the AR coefficients and $\epsilon(t)$ is white noise. This generator is useful for capturing time dependencies and serial correlations in data, allowing for better understanding and controlling of time-dependent phenomena.

The **Moving Average (MA) Sequence** generator models data based on past errors in the series.

The formula is:

$$y(t) = \epsilon(t) + \theta_1 \epsilon(t-1) + \theta_2 \epsilon(t-2) + \cdots + \theta_q \epsilon(t-q)$$

where θ_i are the MA coefficients and $\epsilon(t)$ is white noise. This generator is useful for modeling systems with short-term shocks or noise, where the effect of a disturbance diminishes over time.

The **Autoregressive Moving Average (ARMA) Sequence** generator combines AR and MA processes for data generation.

The formula is:

$$y(t) = \phi_1 y(t-1) + \cdots + \phi_p y(t-p) + \epsilon(t) + \theta_1 \epsilon(t-1) + \cdots + \theta_q \epsilon(t-q)$$

where ϕ_i and θ_i are coefficients for the AR and MA parts, respectively. This generator provides a comprehensive model for capturing both time-dependent structures and random shocks in time series data.

The **Random Walk Sequence** generator simulates a stochastic process where each step is random.

The formula is:

$$y(t) = y(t-1) + \epsilon(t)$$

where $\epsilon(t)$ is a random step. This generator is commonly used to represent random processes or phenomena that evolve over time without a clear direction.

While all generators can accept any kind of time point sequences, stochastic processes specifically require regularly spaced sequences to model time dependencies and random variations accurately. This regular spacing ensures consistent intervals (crucial for the statistical properties of stochastic processes, such as autocorrelation and variance) to be accurately represented and analyzed.

The data generation methods employed in this study are grounded in established mathematical and statistical frameworks, incorporating both classical and advanced models such as algebraic equations, polynomial sequences, and stochastic processes [50–55]. Additionally, the iMimic library features a modular and extensible design, allowing practitioners to adapt, extend, or integrate their own formulas to meet specific domain requirements.

3.3. Samplers

In contrast to data point generators, which operate under the assumption that no actual values are known except for the characteristics used to generate them, samplers require access to the entire or partial set of time series data (i.e., both time points and data points). The fundamental logic behind samplers is that, given a known portion or entirety of a time series, they can conduct sampling to find the data points corresponding to any given time point sequence. This is achieved through the process of interpolation.

Interpolation [56] involves fitting the known time series data to an interpolator model, which can then generate values for any time point within the range of the series. If the series is non-periodic, the interpolator can produce values for any time point within the time series range. For periodic patterns, the interpolator can generate values for any time point that falls within the defined repetitive pattern limits.

Technically, a sampler is an abstraction that accepts at least time points (x) and data points (y). Both x and y must be one-dimensional and free of NaN values (i.e., explicit missing values), although some values may be implicitly missing. The x and y must be equal in length, meaning each pair (based on the index) represents a data-time point pair. The data should also be sorted in ascending order.

Samplers provide an abstract method called `sample` that accepts the time points for which corresponding data points are needed, the interpolation method to be used for estimating the values, and optionally, a time scaler, which is a lambda function that adjusts the sampler's time points to match the resolution of the provided time points.

The **Constrained Non-Periodic Pattern Sampler** (CNPPS) is designed to handle time series data representing the known portion of a non-periodic pattern. This sampler requires the presence of at least the initial and final known values of the time and data sequences, respectively, which define the boundaries within which the sampling can occur. The primary assumption in using the

CNPPS is that the points at the sequence's start and end are known, enabling interpolation within this range.

The **Constrained Periodic Pattern Sampler** (CPPS) is designed to handle time series data representing the known portion of a periodic pattern. This sampler operates on data points (y) and their corresponding time points (x), requiring the explicit definition of the period length. The time points (x) are constrained within the interval $[0, \text{period length} - 1]$.

The values of x must satisfy the condition $0 \leq x \leq \text{period length} - 1$ to maintain the integrity of the periodic cycle. Given that the period length is defined as an integer, the resolution of the time series is inherently fixed at 1 unit, regardless of the actual temporal resolution implied by the data. This design necessitates that if the number of data points (y and x) is less than the period length, the sampler infers that there are missing data points. Conversely, if the number of data points matches the period length, the sampler assumes a complete data set.

The CPPS addresses potential data incompleteness and points marginally outside the boundaries of known points of the defined period through a cyclical extension mechanism. This mechanism involves extending the data sequence by prepending the last known data point and appending the first known data point, along with their respective time points. This extension allows the sampling process to cover the entire period, including points that may be marginally outside the initial time bounds due to division and modulo operations. The indices for the prepended and appended points are calculated using the following formulas:

- Prepend index: $((\text{period length} + 1) - x[-1]) \times -1$
- Append index: $(\text{period length} + 1) + x[0]$

These calculations ensure that the time points x are adjusted to maintain the continuity of the periodic pattern, facilitating consistent sampling across the defined period length.

In summary, samplers provide a method for generating data points based on known time series data by interpolating between known values. This process is particularly useful for reconstructing missing data points or predicting/imputing values at new time points within the range of the existing data. The use of interpolation ensures that the generated data points are consistent with the observed trends or patterns in the available data.

3.3.1. Interpolators

Interpolators are mathematical tools used within samplers to estimate unknown data points based on known values from a time series [56]. In the context of time series data, interpolation involves creating a continuous function that passes through the given data points, allowing for estimating values at intermediate time points. The choice of interpolation method affects the smoothness and accuracy of the estimated values.

Several types of interpolators can be used, depending on the desired properties of the interpolated data:

- **Linear Interpolation**: This method connects data points with straight lines. Linear interpolation is the default method and is useful when a simple and quick estimation is needed.
- **Nearest and Nearest-Up Interpolation**: These methods use the nearest data point for the interpolation. Nearest selects the nearest data point, while nearest-up rounds up for half-integers. These methods are useful when the interpolated values need to match the nearest discrete data points, such as in stepwise functions.
- **Spline Interpolation**: This category includes several methods:
  - **Zero**: A zeroth-order spline returns a step function where each interval is constant.
  - **Slinear**: A first-order spline, similar to linear interpolation but implemented using splines.
  - **Quadratic and Cubic**: These second and third-order splines provide smoother transitions between data points and are useful when the data is expected to follow a curved path.
- **Previous and Next**: These methods return the previous or next value at the interpolation point.

Additionally, the **Piecewise Cubic Hermite Interpolating Polynomial (PCHIP)** is used for monotonic cubic interpolation [57]. PCHIP maintains the monotonicity of the data, preventing overshoot and ensuring that the interpolated data does not exceed the range of the known values. This property is particularly important in datasets where preserving the shape and range of the original data is crucial.

Each of these interpolators serves different purposes and is chosen based on the characteristics of the data and the requirements of the analysis. By selecting an appropriate interpolation method, samplers can accurately estimate data points for any given time point sequence, providing a versatile tool for handling incomplete or irregular time series data.

3.3.2. Time scalers

Time Scalers are used in the sampling process for time series data to manage variations in the resolution of time points. They are typically implemented as lambda functions that adjust the time points to ensure proper alignment for interpolation, accommodating differences in resolution. This adjustment is crucial for maintaining consistency in the characterization of the sampled data, allowing interpolators to function effectively across different time scales.

3.4. Randomness and reproducibility

Randomness is a fundamental aspect of generating synthetic time series, as it allows for the introduction of variability and uncertainty, which are inherent characteristics of real-world data. In the context of synthetic time series, randomness can simulate unpredictable fluctuations and trends, capturing the complexities of phenomena such as environmental changes, sensor behaviors, and system performances. Additionally, randomness is crucial for driving stochastic processes such as Autoregressive (AR) models, Moving Average (MA) models, and ARMA models, which are used as data generation processes, as mentioned in previous sections.

Reproducibility in the context of synthetic time series refers to the ability to generate the same data sequence across different runs of a simulation or experiment. This is crucial for verifying and validating models, allowing researchers to replicate experiments and confirm results.

iMimic achieves randomness by utilizing a comprehensive range of statistical distributions provided by the NumPy library [58]. This capability is essential for simulating various data types and processes in artificial time series generation. The (statistical) distributions [59] are categorized based on their typical usage and characteristics:

- **Normal (Gaussian) Distribution**: This distribution is used for modeling data that cluster around a central value, capturing variability around the mean.
- **Uniform Distribution**: This distribution generates random numbers with equal probability across a specified range, ensuring no bias toward any particular outcome.
- **Exponential Distribution**: This distribution is used to model the time between independent events that occur continuously and independently at a constant average rate.
- **Poisson Distribution**: This distribution models the number of events occurring within a fixed interval of time or space, given a known average rate of occurrence.
- **Binomial Distribution**: This distribution describes the number of successes in a fixed number of independent binary trials, where each trial has the same probability of success.
- **Beta Distribution**: This distribution is commonly used to model probabilities and proportions, particularly in scenarios where the data is bounded between 0 and 1.
- **Gamma Distribution**: This distribution is used to model the time until an event occurs or the time between events, particularly where the waiting time can vary.

In summary, the library supports a wide range of distributions provided by the NumPy library, including Chi-Square Distribution, Log-Normal Distribution, Student's t Distribution, Weibull Distribution, Gumbel Distribution, Laplace Distribution, Logistic Distribution, Triangular Distribution, Cauchy Distribution, Rayleigh Distribution, Dirichlet Distribution, and others.

Reproducibility is achieved through the use of a random seed. A random seed initializes the random number generator, ensuring that the sequence of random numbers generated is consistent across different runs given the same seed.

Moreover, a utility called RDC0 (Random Distribution Callable) provides an abstraction for managing the random state and random distributions flexibly. By setting a seed, RDC0 ensures that the same random values are generated each time the data generation process is executed, provided the seed and parameters remain unchanged.

3.5. A typical procedure for creating synthetic time series

The process of creating synthetic time series using the iMimic library can be generally structured into several key steps, which are outlined below. This procedure is intended to provide a general framework rather than a prescriptive method, illustrating how the library can be utilized to create synthetic data tailored to specific research or application needs.

Generate Initial Time Points: The first step involves generating a sequence of time points that will serve as the temporal framework for the synthetic data. This can be accomplished using the library's time generators, which support both regular (e.g., hourly or daily intervals) and irregular time sequences. The choice of time sequence should align with the intended application of the time series, such as modeling hourly sensor data or daily environmental measurements.

Generate Data Points for Components: After establishing the time points, the next step is to generate data points corresponding to each component of the time series. Components may include trends (e.g., a linear or exponential progression over time), seasonal patterns (e.g., diurnal cycles), and noise (e.g., random variability). Each component is created independently using the appropriate data generators provided by the iMimic library, such as linear or polynomial sequences for trends, sinusoidal sequences for seasonality, or Gaussian noise sequences for stochastic variations.

Combine Components: Once the individual components are generated, they are combined to form the synthetic time series. This combination can be achieved through additive, multiplicative, or hybrid interactions, depending on the characteristics of the phenomenon being modeled. For example, an additive combination might involve summing a trend component with a seasonal component and noise, while a multiplicative approach could involve scaling the seasonal component by the trend. The library allows for flexibility in how these components interact, enabling users to model complex relationships within the time series.

Create a Sampler: Following the combination of components, a sampler is created to enable interpolation and further data manipulation. The sampler is designed to fit the generated data points to an interpolation model, facilitating the estimation of values at new time points or filling gaps within the existing data.

Generate New Time Points: Users may define a new sequence of time points at which they wish to sample the combined data. These new time points can be generated regularly or irregularly, depending on the requirements of the analysis. For instance, users might interpolate data at finer intervals or simulate the effects of irregular data availability due to factors such as network latency.

Sample the Data: With the sampler and new time points defined, the data is then sampled to produce the final synthetic time series. During this step, users can apply a time scaler to adjust the resolution of the time points and select an appropriate interpolation method (e.g., linear, spline) to best represent the underlying patterns in the data.

Visualize the Results: The final step involves plotting the synthetic time series to visually inspect the results. This allows users to verify that the generated series accurately reflects the intended components and interactions. The iMimic library provides tools for visualizing and analyzing the generated data, facilitating the assessment of its realism and suitability for the intended application.

This typical procedure is exemplified in Section 4, where the practical application of the iMimic library is showcased. The demonstration illustrates how these steps can be effectively implemented to generate realistic synthetic time series, highlighting the library's flexibility and relevance across various IoT-related scenarios.

3.6. Architectural choices to enable interoperability, scaling, modularity, and flexibility

The iMimic library is designed with architectural principles aimed at ensuring interoperability, scalability, modularity, and flexibility. These design choices facilitate the application of the library across diverse IoT use cases, its integration with existing systems, and its adaptability to accommodate larger datasets and more complex scenarios.

3.6.1. Interoperability

The library's components for generating time points and data points are decoupled from any specific format, ensuring compatibility with various data structures. After generating a time series, iMimic provides interfaces to convert the data into widely used formats such as JSON, XML, Parquet, MessagePack, and binary formats. Furthermore, it supports the transformation of time series into discrete events or network packets, allowing integration with platforms like MQTT [60,61] brokers, Kafka [62], or network simulators such as ns-3 [63]. This design allows the generated data to be utilized across different IoT platforms and frameworks, including commercial cloud solutions. Compression mechanisms are also available to reduce the size of generated datasets, making the library suitable for use in environments where storage or bandwidth may be constrained.

3.6.2. Scaling

The architecture of the library supports both stateless and stateful computations to ensure efficient performance across various computing environments.

- Stateless computations refer to operations that produce deterministic results based solely on the input parameters. These tasks are inherently parallelizable, allowing for horizontal scaling in distributed systems. This is particularly effective for generating large volumes of synthetic data where each process operates independently.
- Stateful computations, by contrast, involve interdependent processes where subsequent steps rely on prior results. These operations are less amenable to parallel execution and typically benefit from vertical scaling, where performance is enhanced by increasing computational power on a single machine. This approach is often necessary for generating multivariate time series, where the interdependencies between variables must be preserved.

Through a hybrid approach, both stateless and stateful computations can be executed in parallel across distributed targets. A coordination mechanism is employed to combine the output from these parallel tasks, ensuring that the final time series is accurately assembled. This method leverages the strengths of parallel processing while maintaining the necessary interdependencies between stateful processes.

The library is capable of operating on commodity hardware for small to medium-sized datasets but can also be deployed in cloud environments for more computationally intensive tasks, offering scalability to meet diverse processing requirements.

It is worth noting that while some synthetic time series generation tasks can be computationally demanding, the clear separation between the generation and usage phases in this library significantly enhances its efficiency. This design allows time series to be generated efficiently and then used as input for testbeds, simulators, or constrained-resource devices and their corresponding emulators. By decoupling these processes, iMimic minimizes the computational burden, ensuring that subsequent stages of testing and implementation are not hindered. This makes the library highly adaptable for a wide range of IoT applications, particularly in resource-constrained edge environments.

3.6.3. Modularity

iMimic follows a modular design, enabling users to extend its capabilities without modifying its core components. This structure allows for seamless integration of:

- Custom models: Users can define new models for generating time points and data points, or extend existing methods. This supports the generation of both regular and irregular time series.
- Custom samplers and interpolation methods: The library supports user-defined methods for sampling or interpolating data, offering flexibility in how missing or irregular time points are handled.
- Advanced models: Beyond basic statistical models, the architecture accommodates more sophisticated approaches, including machine learning models and high-dimensional data generation. This allows for broader application in domains where more complex relationships or patterns need to be modeled.

3.6.4. *Flexibility*

The design of iMimic does not impose rigid constraints on the techniques or models used for creating synthetic data. Users can implement any method or algorithm, provided that the output adheres to basic structural requirements, such as format consistency. This includes the ability to integrate machine learning models for synthetic data generation as long as the output format aligns with the expected structure of the library.

This approach ensures that iMimic can be applied to a wide variety of IoT domains and tasks, enabling users to tailor the data generation process to their specific needs. For example, if an application requires data that reflects the output of a particular sensor type or system behavior, the user can define models that produce data in line with those requirements.

4. Demonstration of utility

The utility of the iMimic library is demonstrated through the creation of realistic and interpretable synthetic time series data, showcasing its applicability in various research and practical scenarios. This section details three primary parts: creating air temperature time series data, evaluating the realism of the created time series data, and simulating the incorporation of sensor behavior into the time series data.

4.1. *Constructing air temperature time series reflecting a natural phenomenon*

The first part of the demonstration involves constructing a synthetic time series representing air temperature variations over a week. The process integrates three components: trend, seasonality, and noise.

The trend component models a weekly pattern with distinct phases: a gradual decrease in temperature over the first two days, a gradual increase over the subsequent three days, and a final two-day gradual decrease. These systematic changes reflect broader physical phenomena, with gradual shifts that occur beyond the other components, such as seasonality and noise. This pattern is defined using a constrained non-periodic pattern sampler. The input x and y values are depicted in Table 2.

The seasonality component captures daily temperature cycles by reflecting diurnal variations in temperature. The period length is 24 h, and the input x and y values are depicted in Table 3. These values represent deviations from the corresponding trend values at each hour of the day. Negative values correspond to lower temperatures during the night and early morning, while positive values represent higher temperatures during daylight hours. This cyclic pattern is defined and generated using a constrained periodic pattern sampler with a 24-hour period, ensuring regular temperature variations repeat daily throughout the week.

Hourly fluctuations are introduced through an ARMA (AutoRegressive Moving Average) process, simulating short-term variability and random noise in temperature, which can result from various natural phenomena such as changes in wind speed, cloud cover, solar radiation, atmospheric pressure, and evaporation. The ARMA model parameters include:

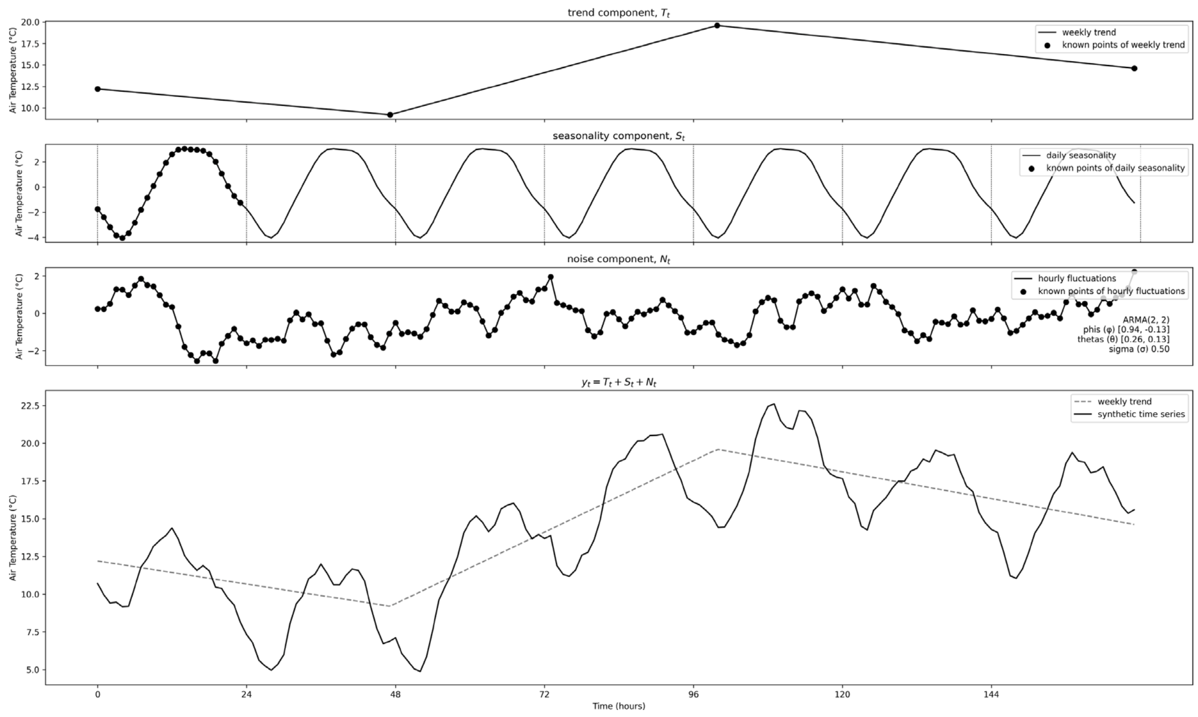

Fig. 1. Synthetic air temperature time series constructed through an additive operation of three components: trend, seasonality, and noise. The trend component reflects long-term changes over the week, the seasonality component captures consistent daily cycles, and the noise component introduces short-term variability.

Table 2
Input x and y to the sampler that defines the weekly trend.

(x)	Time	(y)
0	Monday 00:00:00	12.2
47.13	Tuesday 23:07:48	9.2
99.81	Friday 03:48:36	19.6
167	Sunday 23:00:00	14.62

Table 3
Input x and y to the sampler that defines the daily seasonality.

(x)	Time	(y)	(x)	Time	(y)
0	00:00:00	−1.75	12	12:00:00	2.62
1	01:00:00	−2.39	13	13:00:00	2.98
2	02:00:00	−3.18	14	14:00:00	3.05
3	03:00:00	−3.84	15	15:00:00	2.99
4	04:00:00	−4.04	16	16:00:00	2.95
5	05:00:00	−3.66	17	17:00:00	2.89
6	06:00:00	−2.83	18	18:00:00	2.63
7	07:00:00	−1.82	19	19:00:00	2.01
8	08:00:00	−0.84	20	20:00:00	1.07
9	09:00:00	0.09	21	21:00:00	0.07
10	10:00:00	1.03	22	22:00:00	−0.70
11	11:00:00	1.92	23	23:00:00	−1.25

- AR coefficients: $\phi_1 = 0.94$, $\phi_2 = -0.13$ (capturing the influence of past values)
- MA coefficients: $\theta_1 = 0.26$, $\theta_2 = 0.13$ (capturing the influence of past noise)
- Sigma: $\sigma = 0.50$ (standard deviation of the white noise)

As the methodology for modeling hourly fluctuations (i.e., noise component) is stochastic, explanations and justifications are presented in relation to the assumptions made in order to increase the realism of the example and facilitate discussion. These assumptions ensure that the noise component, representing short-term variability, accurately captures realistic temperature behavior while remaining interpretable.

The ARMA(2,2) model is chosen because it provides an effective balance between capturing temperature dynamics and maintaining simplicity in the model structure. This second-order ARMA model allows for the modeling of both short-term dependencies and random variations, which are typical in temperature data.

The AR coefficients (0.94, −0.13) represent strong continuity with slight damping. The 0.94 coefficient (ϕ_1) indicates a strong positive relationship with the immediate past value, suggesting that the temperature at any given hour is highly correlated with the temperature in the previous hour, thereby capturing the smooth, gradual changes typically observed in temperature data. The −0.13 coefficient (ϕ_2) introduces a smaller negative correlation with the temperature two hours ago, adding a slight damping effect that moderates the persistence introduced by the first coefficient. This accounts for the natural tendency of temperature to adjust slightly over longer intervals, preventing the continuous accumulation of past effects.

The MA coefficients (0.26, 0.13) model short-term shocks with slight persistence. The 0.26 coefficient (θ_1) suggests that the noise component at any given hour is influenced by the noise from the previous hour, reflecting short-term fluctuations that commonly occur in environmental conditions, such as sudden changes in cloud cover or wind speed. The smaller 0.13 coefficient (θ_2) indicates that the noise from two hours ago still has a slight influence on the current value, capturing the lingering effects of past random disturbances.

Finally, the standard deviation of the white noise, represented by a sigma value of 0.50 (σ), ensures moderate variability in the temperature fluctuations. This value reflects natural randomness while keeping the variations within a typical environmental range. It ensures that temperature fluctuations are neither too erratic nor unrealistically stable, maintaining a balance that mimics observed environmental conditions.

Ultimately, with the key components–trend, seasonality, and noise–now defined, the goal is to create an air temperature time series for a week by combining these three components with an hourly resolution. Initially, the time point sequence is generated using the first method for creating regular time point sequences. The start is 0 (Monday 00:00:00, inclusive), the stop is 168 (next Monday 00:00:00, exclusive), and the step is 1, representing one hour.

Next, the weekly trend definition, which has four known points, is sampled to produce a time series with 168 points, covering one week. This is achieved through linear interpolation, where straight lines are drawn between known points to estimate values for the unknown intervals.

The definition of daily seasonality has 24 known points. Therefore, no interpolation is required; the periodic pattern is simply repeated seven times for each day of the week.

For the hourly fluctuations, 168 points have been generated using the respective generator. Thus, a constrained non-periodic pattern sampler is created using the generated hourly fluctuations for the week. Similar to the daily seasonality, no interpolation is

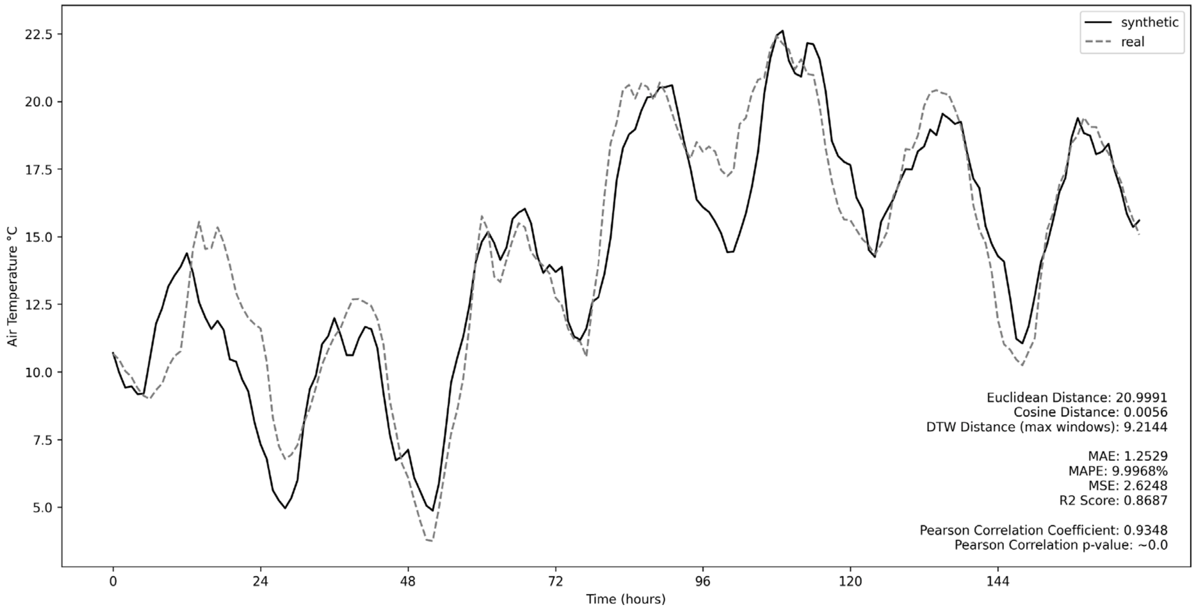

Fig. 2. Comparison of synthetic air temperature time series with real-world data from the Jena Climate Dataset.

needed as the values for each hour of the week are known.

Subsequently, using the `sample` method of each sampler, the three components are generated, as shown in the first three axes of Fig. 1. These components are then combined through an additive operation to create the final time series, depicted in the fourth axis of Fig. 1.

4.2. Evaluating the realism of the synthetic air temperature time series

The second part of the demonstration assesses the realism of the synthetic air temperature time series by comparing it with real-world data from the Jena Climate Dataset, which includes detailed weather observations collected by the Max Planck Institute for Biogeochemistry in Jena, Germany [64]. This dataset serves as a benchmark for validating the synthetic data (see Fig. 2).

According to the classical statistics tutorial of [65] and the empirical evaluation of multiple studies presented in [66,67], Euclidean distance is considered a highly accurate measure for comparing the similarity of time series, especially when the series have similar baselines and scales. Based on this, Euclidean distance is utilized as the primary metric to identify the most similar week in the Jena Climate Dataset. This was done by calculating the Euclidean distance between the synthetic time series for a week and each week in the real dataset. The week with the smallest Euclidean distance is selected for further evaluation, validation, and verification.

The best match yielded an Euclidean distance of 20.9991, which indicates moderate similarity between the synthetic and real datasets. This value suggests that while there are differences between the datasets, they are not large enough to disqualify the synthetic data from being useful in certain applications. Fig. 2 shows the visual comparison between the synthetic and real datasets, highlighting their similarities and differences.

To further assess the similarity between the synthetic and real datasets, additional measures were applied [66–69].

Cosine distance measures the angular difference between two time series, indicating how aligned the overall trends are, regardless of scale. A cosine distance of 0.0056 indicates that the synthetic and real datasets are almost perfectly aligned in terms of their direction or general trend. This suggests that despite some pointwise differences, the synthetic data mirrors the overall pattern of the real-world data very closely.

Dynamic Time Warping (DTW) [69] distance is used to compare time series with potential shifts in the timing of patterns, allowing for flexible alignment. A DTW distance of 9.2144 indicates a reasonable alignment between the synthetic and real datasets, with some minor differences in the exact timing of the patterns. This result suggests that the synthetic data captures the key temporal features of the real data but with slight shifts. Further insight can be gained from Fig. 3, which illustrates the warping path between the two time series. The best alignment path, computed with a maximum window size matching the datasets, demonstrates the synthetic data's ability to follow the real dataset closely, even allowing for minor time discrepancies.

In terms of central tendency and variability, the synthetic data closely aligns with the real data. The mean of the synthetic time series is approximately 14.23 °C, while the mean of the comparable week from the Jena Climate Dataset is 14.50 °C. This close proximity indicates that the synthetic data mirrors the central tendency of the real dataset. Similarly, the standard deviation of the synthetic time series is approximately 4.36 °C, compared to 4.47 °C for the real dataset. The near-equal standard deviations suggest

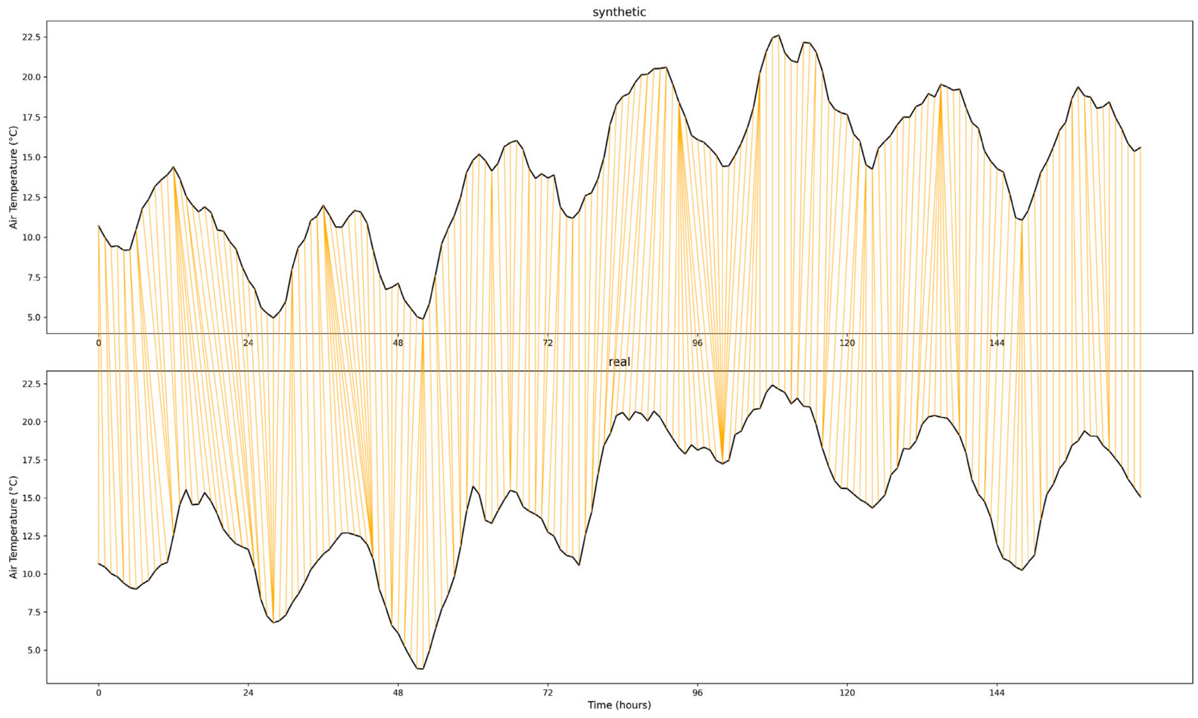

Fig. 3. Dynamic Time Warping path between real and synthetic time series, illustrating flexible alignment and capturing key temporal patterns despite slight shifts.

that the variability in the synthetic data is consistent with real-world fluctuations, further reinforcing the realism of the generated data.

Mean Absolute Error (MAE) measures the average absolute difference between the synthetic and real datasets. A value of 1.2529 indicates that the synthetic data differs from the real data by about 1.25 °C on average, which is an acceptable level of deviation for many practical applications.

Mean Absolute Percentage Error (MAPE) represents the average percentage error between the synthetic and real datasets. A value of approximately 10% shows that, on average, the synthetic data differs by 10% from the real data, suggesting a reasonable level of accuracy, particularly for applications where exact precision is not critical.

Mean Squared Error (MSE) MSE calculates the average squared difference between the synthetic and real datasets, emphasizing larger deviations. A value of 2.6248 indicates that the synthetic data has moderate errors, but these errors are not extreme. This measure confirms that the synthetic data captures the overall patterns with some small deviations in specific instances.

The R^2 score (coefficient of determination) indicates how much of the variance in the real dataset is captured by the synthetic dataset. An R^2 of 0.8687 suggests that the synthetic data explains about 87% of the variance observed in the real data, demonstrating a strong fit, although there is still room for further improvement.

Autocorrelation Function (ACF) measures how current values in a time series relate to past values at various lags, helping to identify patterns or trends over time. Partial Autocorrelation Function (PACF) focuses on the direct relationship between a time series and its lagged values, filtering out indirect correlations. In Fig. 4, both the ACF and PACF for the real and synthetic datasets are visualized, allowing a direct comparison of their temporal dependencies. The plots show that the synthetic data closely mimics the autocorrelation structure of the real data, capturing important features such as seasonality and lag-based dependencies. The ACF and PACF patterns for both the real and synthetic datasets are closely aligned, successfully replicating key temporal dependencies and the overall autocorrelation structure observed in the real data. The synthetic data captures strong autocorrelation at lower lags and also reflects some seasonal or periodic features at higher lags (e.g., around lag 18 in the ACF). The PACF plot further reinforces that the synthetic data follows similar lag dependencies, maintaining alignment even at higher lags.

Pearson correlation measures the linear relationship between the synthetic and real datasets. A coefficient of 0.9348 indicates a strong positive correlation, suggesting that the synthetic data closely follows the linear trends present in the real data. This high correlation value confirms that the overall structure of the synthetic data is well aligned with the real dataset. The *p*-value for the Pearson correlation test indicates the statistical significance of the observed correlation. A *p*-value of approximately 0.0 suggests that the correlation is statistically significant and not due to random chance, reinforcing the reliability of the similarity between the synthetic and real datasets.

This set of metrics provides a highly encouraging evaluation of the similarity between the synthetic and real time series. The results demonstrate that the synthetic data not only captures the overall trends but also reflects key features of the real dataset with

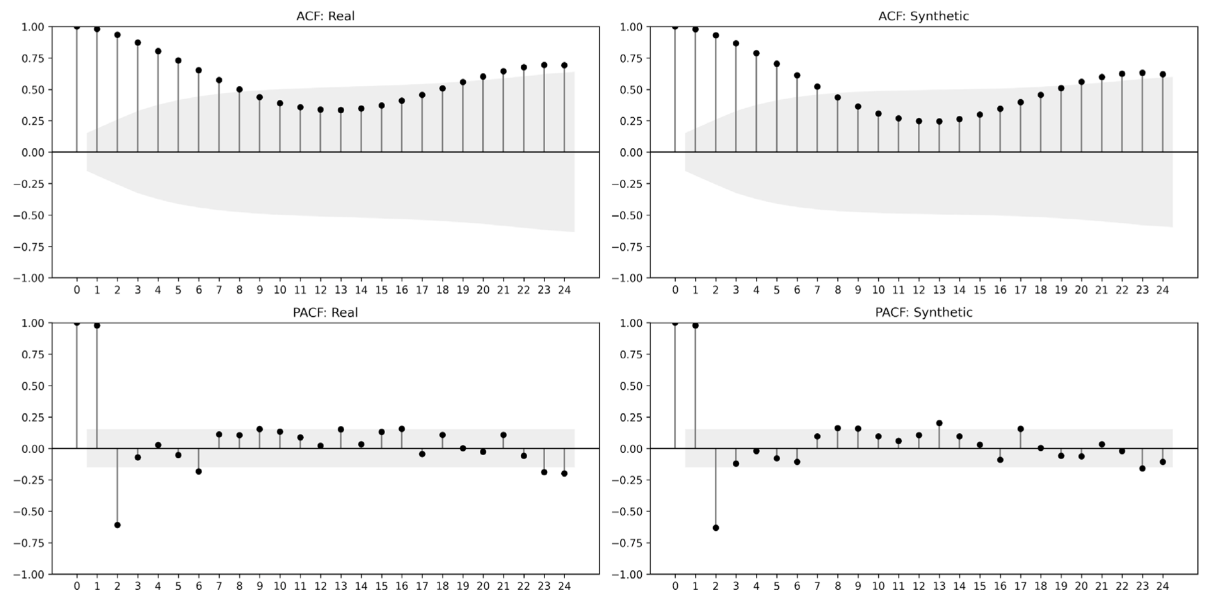

Fig. 4. Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) comparison between real and synthetic datasets, showing closely aligned temporal dependencies and autocorrelation structures.

impressive accuracy, making it a robust and reliable alternative for testing and prototyping in scenarios where real-world data is limited or inaccessible. The close alignment in most metrics underscores the synthetic data's capability to mimic real-world patterns effectively. While there are minor pointwise deviations, the overall performance remains strong, positioning the synthetic data as a potentially highly effective tool for IoT development and analysis.

4.3. Incorporating IoT device behavior into air temperature time series

The final part of the demonstration illustrates how air temperature data could have been made available from an IoT device with sensing and communication capabilities. The ground truth is the hourly synthetic air temperature time series (the created synthetic time series from Section 4.2). Subsequently, four factors are introduced that reflect non-ideal realities in an IoT-enabled environment: drift, deviation and calibration, and noise.

The first component concerns drift, which is a linear decrease in measurement accuracy over time. This drift amounts to approximately 2 degrees Celsius per year. This annual drift is broken down to reflect daily changes, generated using a linear sequence generator, and is depicted on the first axis of Fig. 5.

The second component concerns a constant deviation in the measurement of air temperature. This deviation is a steady 5 degrees Celsius, generated using a flat sequence generator, and is depicted on the second axis of Fig. 5.

The third component involves the detection and correction of the constant deviation. At the 36th hour, the constant deviation is corrected by calibrating the sensor through a firmware update. The correction process takes approximately one hour, during which data is unavailable. This component is generated using a step transition sequence generator, where at the 36th hour, all subsequent values are −5 degrees Celsius, which is the inverse of the constant deviation. This component is depicted on the third axis of Fig. 5.

The fourth component concerns the noise that arises from the sensor and the firmware that converts the analog signal to a temperature measurement. Here, noise is assumed to be random with a mean of 0 and a standard deviation of 0.5 degrees Celsius. This noise is generated using a Gaussian noise sequence generator with parameters mean 0 and standard deviation 0.5, and is depicted on the fourth axis of Fig. 5.

The fifth component is the hourly air temperature produced in the first part in Section 4.1. For convenience and visual comparison with the other components, it is depicted on the fifth axis of Fig. 5.

All five components have regular hourly time points. Through an additive operation, an initial regular hourly sample of the air temperature that could have been produced from a sensor is created and fitted to a constrained non-periodic pattern sample to perform upsampling through interpolation.

To do this, an irregular time points sequence is generated. Irregular time points are created as follows: it is assumed that the IoT device is programmed to sense, measure, and produce data every 10 min. However, there is a time overhead involved due to the aforementioned process as well as network latency. Therefore, this overhead is assumed to follow an exponential distribution with a scale of approximately 16–17 s. Using the second method for creating irregular time sequences for Section 3.1, regular time points are created every 10 min and then, using the exponential distribution, a delay for each data point is generated. The result is the data points at which the data became available to the application.

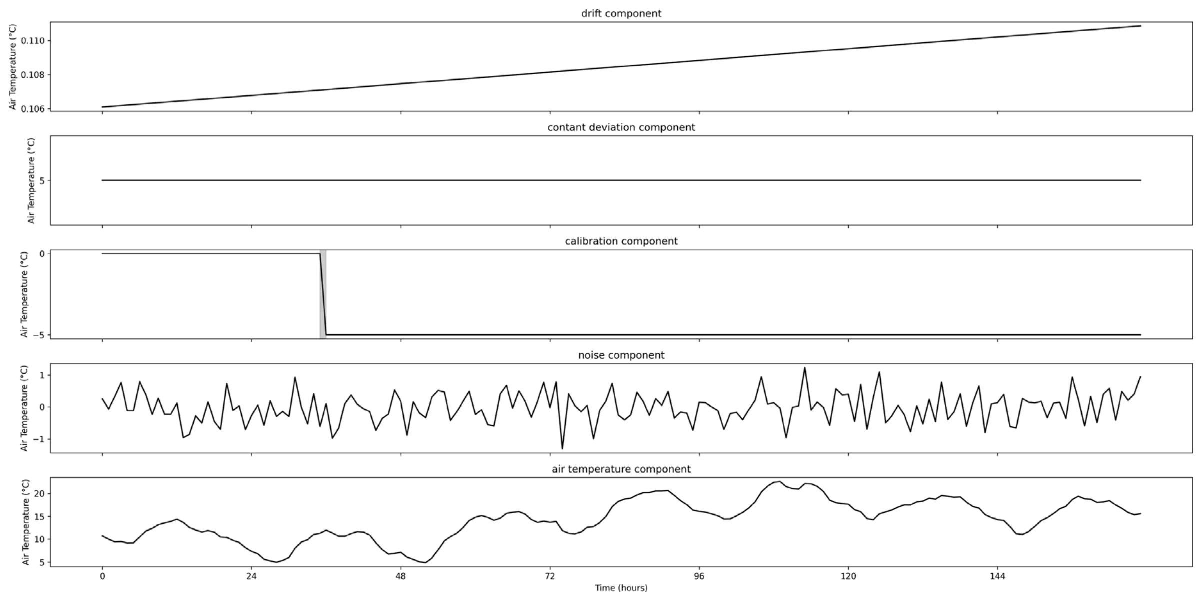

Fig. 5. Components simulating sensor behavior: drift, constant deviation, calibration, and noise, along with the hourly synthetic air temperature time series.

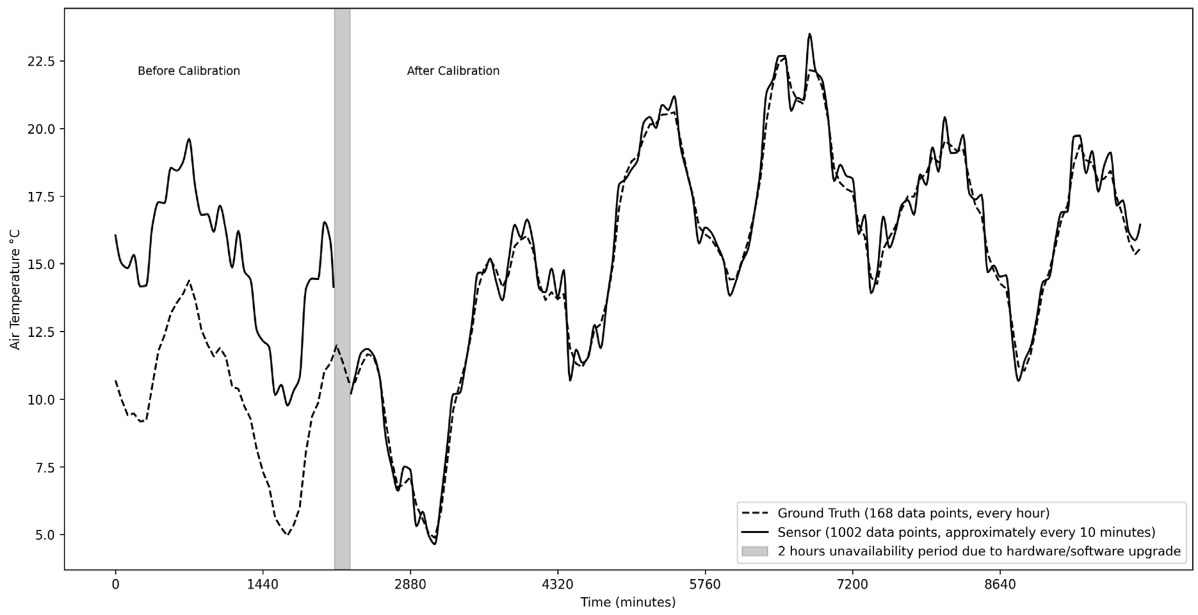

Fig. 6. Sensor air temperature time series with 1002 data points (solid line), including simulated non-ideal factors (drift, deviation, calibration, noise), and the hourly artificial air temperature series (dashed line) with these factors added.

The generated irregular time points are interpolated to create a continuous time series. Using the previously fitted sampler, the Piecewise Cubic Hermite Interpolating Polynomial (PCHIP) method is used for this purpose, ensuring that the data points are accurately estimated between the known values while at the same time preserving their characteristics.

In Fig. 6, the sensor's air temperature with 1002 data points, including the simulated non-ideal factors (drift, deviation, calibration, noise), is depicted along with the hourly air temperature to which these factors were added through an additive operation.

5. Discussion

The motivation for this research is rooted in the critical need for reliable data to facilitate the design, prototyping, and testing of Internet of Things (IoT) solutions. This necessity is compounded by the inherent challenges associated with the lack of high-quality real-world data or, in some cases, the complete absence of data. The research gap presented in Section 2, along with the findings in Section 4, underscore the importance and utility of a flexible approach to creating synthetic time series data, which also offers a significant degree of interpretability.

The library introduced in Section 3, coupled with its application through realistic examples in Section 4, represents a foundational step towards establishing a structured process for creating synthetic data, including time series, within the IoT domain. By complementing the methodologies presented in Section 2 and addressing two key gaps–(i) the need for interpretability and (ii) the absence of real data–the proposed library provides several distinct advantages.

The first major benefit of this approach is that practitioners can create time series data without the need for real-world datasets, effectively addressing a critical challenge in the early stages of IoT solution design. This capability is particularly valuable when real data is unavailable, inaccessible, or insufficient for prototyping and testing purposes.

Secondly, the library's modular approach, which allows for the combination of different components such as trend, seasonality, and noise, not only enhances the realism of the generated time series but also improves its interpretability. Furthermore, this approach enables the customization of various components, facilitating the execution of what-if scenarios. For example, given a specific seasonal pattern and a modeled noise or fluctuation, practitioners can explore how the time series evolves under different conditions, such as upward or downward trends.

Moreover, the library does not impose a rigid methodology for creating synthetic time series. Instead, it provides a suite of data generators, time generators, samplers, and patterns while also allowing the integration of custom methods and approaches. For instance, IoT practitioners can incorporate their own generators, which may range from simple statistical models to complex machine learning algorithms, thereby tailoring the synthetic data creation process to their specific needs.

5.1. Limitations

Limited exploration of cross-domain methodologies. In Section 2, a thorough review is presented, covering a wide range of studies focused on generating synthetic time series for IoT. Given the clear gap in the literature for a generalized, flexible, and interpretable approach to creating univariate or multivariate time series in the IoT context, it may be beneficial to conduct interdisciplinary research across other domains to explore new methods and techniques that could enhance the proposed approach. These explorations could draw from fields like finance, healthcare, or industrial control systems, where time series data generation has been extensively studied and where different types of complex dependencies and interactions between variables are prominent.

Challenges in achieving interpretability without oversimplification. The iMimic library introduces a series of methods for generating time series components that can be combined to create univariate time series. Its architectural choices allow for a flexible, modular, and scalable approach, enabling the production of realistic univariate time series that reflect physical phenomena through an interpretable framework. However, this focus on interpretability can sometimes reduce the complexity that is often required in certain domains. For instance, while high interpretability makes the time series easier to understand and analyze, it may come at the cost of oversimplifying the phenomena being modeled, potentially leading to time series that are not fully representative of real-world conditions. Alternatively, aiming to maintain interpretability may limit the generalization ability of the generated data, especially when trying to apply it to more complex systems or diverse environments.

Dependency on user perception and domain knowledge. A major limitation is that synthetic data generation depends heavily on the user's perception and domain knowledge. In the absence of real-world data, users must rely on their understanding of the phenomena they are trying to model, which can introduce biases. When low-quality datasets are used for example-based synthetic data generation or for validation, the quality and completeness of the resulting synthetic data are directly tied to the quality of the initial dataset. On the other hand, even with high-quality datasets, the differences between the synthetic and real data may not always reflect real-world conditions, despite passing verification checks. Therefore, the selection of models, understanding of the phenomena, and strong statistical foundations are critical to minimizing the risk of generating synthetic data that leads to unrealistic or suboptimal IoT prototypes.

Biases in synthetic time series generation. Biases in synthetic time series creation can manifest in various forms, including sampling bias, representational bias, and procedural bias [70,71]. Sampling bias arises when the underlying dataset used to inform the synthetic generation process is not representative of the target application domain, leading to skewed or incomplete synthetic datasets [72]. Representational bias may occur if the chosen mathematical models or distributions fail to capture the full spectrum of variability in real-world phenomena. Procedural bias is introduced when simplifications or assumptions in the generation process fail to account for essential interactions or dynamics between variables [73]. For IoT scenarios, these biases can result in prototypes that underperform when exposed to real-world conditions, potentially undermining the design, testing, and implementation phases.

Impact of biases on IoT prototype reliability. During the design phase of IoT solutions, biases in synthetic data–such as oversimplified models, unrealistic assumptions, or inadequate representation of environmental variability–can lead to the development of systems that fail to capture the true complexity of the physical phenomena they are intended to monitor or control. These biases may result in designs that are optimized for synthetic scenarios but poorly suited to real-world conditions. In the implementation phase, reliance on synthetic data that lacks edge cases or extreme scenarios can lead to the integration of components that are overly rigid, unable to adapt to unexpected behaviors, or prone to failure in dynamic environments. Furthermore, during testing, synthetic data biases often mask critical vulnerabilities. For example, idealized sensor behavior in synthetic datasets might overlook the impact of noise, calibration drift, or data loss, leading to prototypes that appear robust in simulations but perform inadequately in operational contexts. These biases not only reduce the effectiveness of IoT prototypes but also create false confidence in their readiness, resulting in costly redesigns, delayed deployments, and the potential for failure in real-world applications. To address these challenges, it is crucial to account for synthetic data limitations by incorporating real-world testing and hybrid validation approaches throughout the prototyping process.

Insufficient support for multivariate time series. From a technical standpoint, creating multivariate time series is possible using the iMimic library and the presented approach. This can be achieved through a sequential process, starting with the generation of univariate time series and then combining them to form a multivariate time series. While this approach works for simpler cases, it does not account for the generation of multivariate time series where multiple series are created jointly, with potential interdependencies between them.

Addressing the challenges of generating realistic multivariate time series requires recognizing the inherent complexities of this task. Multivariate time series often exhibit intricate relationships, including temporal dependencies, lagged interactions, and non-linear dynamics across variables. Capturing these aspects involves sophisticated techniques, such as multivariate autoregressive models or advanced statistical frameworks like copulas, which are computationally demanding and require significant domain expertise [55,74]. Moreover, ensuring that the statistical properties of individual series and their interdependencies are preserved in synthetic data further compounds the difficulty.

To address this limitation, a practical example can be found in electric power distribution. Monitoring and predicting electricity demand is a multivariate problem influenced by factors such as time, weather, seasons, and usage patterns. Realistic multivariate time series generation is crucial, particularly when operational variables like transformer oil temperature are involved. A multivariate time series could model electricity load alongside oil temperature, capturing how fluctuations in demand impact transformer efficiency. Future work would involve creating a parameterizable stateful approach within iMimic, combining the statistical methods from Section 4–incorporating long-term seasonal effects and noise–along with a modified explainable version of the Informer estimator [75] to model shorter-term interdependencies, such as unexpected demand spikes or equipment anomalies. This would enable practitioners to prototype solutions to optimize power distribution both safely and efficiently.

Lack of integration with machine learning and deep learning techniques. In connection with this, and as presented in Section 2, various machine learning and deep learning techniques have been proposed for domain-specific tasks in time series generation. Although the presented work does not exclude the use of machine learning approaches, it has not yet incorporated them. As highlighted in studies like [76], machine learning approaches, while powerful, can sometimes compromise interpretability due to their complexity. This does not mean they are unsuitable—on the contrary, the flexible, modular, and scalable nature of the presented approach allows for the integration of machine learning models. However, ensuring that interpretability is maintained and integrating such models seamlessly into the time series generation process will require further investigation.

Demonstration focused solely on air temperature. Section 4 presents a detailed example of generating synthetic air temperature time series. Air temperature is a universal and common physical quantity that impacts many aspects of life—ranging from everyday activities to business, economy, industry, smart cities, and agriculture. This makes it an excellent choice for highlighting the potential of the iMimic library, as it provides a well-understood and relatable example for demonstrating the realism of the generated time series. However, the research could significantly benefit from additional use cases that involve multivariate time series representing multiple physical quantities. Expanding the scope to include diverse IoT applications–such as healthcare, argiculture, smart cities, or industrial IoT–would emphasize the library's versatility and robustness. Demonstrating its capability to handle complex, real-world scenarios with interconnected variables across these domains would further validate its applicability and highlight its potential for addressing a wide range of challenges beyond air temperature data.

Reliance on a single dataset for evaluation. While the time series similarity measures and evaluation methods presented in Section 4.2 produce optimistic results with high potential, further validation is needed. More comprehensive comparisons with diverse real-world datasets are necessary to establish the robustness and fidelity of the synthetic data. This can be achieved by conducting experiments based on realistic use cases derived from the literature and high-quality datasets where possible, following the same approaches and presenting results similar to those already demonstrated. By doing so, a foundation of concrete, trustworthy examples can be created, showing how IoT practitioners can use the iMimic library to generate domain-specific synthetic time series.

Need for enhanced theoretical rigor. The data (point) generation methods presented in Section 3.2 are grounded in well-established mathematical and statistical frameworks, supported by authoritative literature. Classical algebraic and polynomial models, such as the flat sequence, linear sequence, and polynomial sequence, provide a foundation for modeling equilibrium states, trends, and non-linear growth patterns [50,51]. More advanced techniques, including logarithmic, exponential, and logistic sequences, enable the modeling of phenomena such as decreasing growth, saturation effects, and bounded dynamics [52,53]. Waveform-based generators, such as sinusoidal, sawtooth, and triangular sequences, leverage principles from signal processing and periodic function theory to represent oscillatory and seasonal behaviors [53,54]. Finally, stochastic generators–such as autoregressive (AR), moving average

(MA), and autoregressive moving average (ARMA) sequences–are rooted in foundational time series analysis and effectively capture temporal dependencies, serial correlations, and noise [52,55]. The inclusion of a random walk generator further adds flexibility, enabling the simulation of processes with inherently unpredictable evolutions.

While these methods provide a strong starting point, their application to IoT contexts requires further validation and theoretical refinement. As discussed earlier, the adaptability of the iMimic library to various scenarios relies on these mathematical foundations. However, enhancing theoretical rigor could help address interconnected limitations, such as biases in data generation, oversimplification of complex relationships, and challenges in modeling multivariate dependencies. By ensuring that these methods are robustly validated and theoretically grounded across a broader range of IoT applications, the library could better support scenarios that demand high fidelity and reliability.

Future work should focus on extending the library's theoretical underpinnings to address these gaps. This includes incorporating more sophisticated statistical models, deriving stronger justifications for the applicability of the existing methods, and formalizing the mathematical properties of the generated sequences in the context of IoT use cases. Such improvements could not only strengthen the library's core framework but also mitigate other limitations, such as the dependency on user expertise and the potential biases introduced during the data generation process. Moreover, a rigorous approach to theoretical validation could establish the broader applicability of these methods, fostering their integration with advanced techniques like machine learning while preserving interpretability.

5.2. Additional future directions

Mitigating the limitations presented in the previous section is of high priority. However, there are several future research directions that focus on how the iMimic library and its approach can be integrated into various stages of IoT solution development, including design, implementation, and testing phases. An IoT solution involves understanding the physical environment, discrete event simulation, modeling, digital twins, complex event processing, automation, analytics, networking simulation, and more. So, further extensions are necessary to enable the library to support these aspects, either holistically or as a complementary tool.

One important future direction is to extend the library's capability to emulate sensor behaviors, which is central to IoT applications. In an IoT environment, sensors and actuators are fundamental components that interact with the physical world through sensing and acting processes. The data generated by these components often has non-ideal characteristics due to sensor limitations such as measurement accuracy, precision, resolution, and sampling frequency. Therefore, a crucial extension of the library would be the ability to transform the generated time series into sensor readings that reflect these limitations. For example, a synthetic time series that accurately reflects a physical phenomenon like temperature could be transformed to simulate sensor readings that are constrained by the specific capabilities of a sensor—such as high accuracy but low precision, or high sampling frequency but limited resolution.

By modeling these non-ideal factors, the iMimic library could enable IoT practitioners to test and validate prototypes that are sensitive to the quality of sensor data. This transformation would be highly valuable for validating IoT applications where sensor accuracy plays a critical role in decision-making processes, such as in smart farming, industrial automation, or healthcare monitoring systems. The incorporation of such features would extend the library's utility beyond simply generating synthetic time series, enabling it to more accurately represent real-world sensor behaviors and their impact on IoT systems.

While the flexibility of the library is a significant advantage, it can sometimes introduce a learning curve or added complexity. A more targeted approach that allows for domain-specific customization would be valuable. For example, generating air temperature data for a smart building might require a different approach than producing multivariate data for smart farming, where both temperature and humidity must be modeled together. Additionally, in healthcare, generating data for wearable sensors used in smart hospitals might differ significantly from personal health wearables, where the environment and user context change dramatically. Therefore, the development of ready-to-use, coarse-grained processes that encapsulate domain-specific knowledge could enable fast prototyping and knowledge transfer. These processes could serve as black-box models with configurable parameters that provide domain expertise from IoT practitioners in a simplified format, allowing other users to generate complex synthetic datasets with minimal effort.

Finally, while establishing a strong theoretical foundation and proven practical applications for synthetic time series generation in the IoT context is challenging, the mitigation strategies for the limitations, along with the future research directions, make the iMimic library and its approach highly feasible. Coupled with the promising results observed in Section 4, these factors strengthen the argument for the library's utility and its potential to significantly contribute to IoT solution development.

5.3. Performance considerations

The performance of the iMimic library is influenced by both the desired size of the datasets to be created and the complexity of the time series tasks. Its design prioritizes decoupled computation and usage, which enables versatile application across different environments, from resource-constrained edge devices to high-performance computing systems.

In the demonstrations provided in Section 4, all tasks were executed on commodity hardware (Intel Core i3, 1.2 GHz, 10th generation, 4 GB DDR4 RAM, SSD, running Python 3.11). Despite the modest hardware, the library was able to generate results, excluding figure visualization, in under two seconds. This efficiency underscores its suitability for a wide range of environments, including low-resource setups where computational power is limited. For example, simple synthetic time series can be generated directly on edge devices or microcontrollers, although the generation of complex datasets or those requiring high computational power–such as stochastic processes or machine learning models–may require more capable hardware. In these cases, synthetic datasets can be precomputed in a high-resource environment and subsequently deployed on low-resource devices for testing or simulation purposes. This decoupled approach ensures that iMimic can serve diverse use cases effectively.

6. Conclusions

This paper presents a comprehensive approach to addressing the critical challenge of data availability in the design, prototyping, and testing of Internet of Things (IoT) solutions. By introducing the iMimic library, an open-source tool for creating realistic and interpretable synthetic time series data, this study provides a robust framework that enables practitioners to overcome the limitations posed by the absence of real-world data. The library's flexible, modular, interoperable, and scalable architecture allows for the creation of time series by combining various components such as trend, seasonality, and noise, thereby enhancing both the realism and interpretability of the generated data. The demonstration of utility through realistic examples further underscores the practical applicability and effectiveness of this approach, showing how synthetic data can closely mimic real-world phenomena and support the early stages of IoT solution development.

The iMimic library not only complements existing methods for synthetic data generation but also fills important gaps, particularly in terms of interpretability and flexibility. As discussed, future work should focus on mitigating current limitations and expanding the library's capabilities to support multivariate time series, integrate advanced machine learning models, and tailor the tool to specific application domains. These enhancements will further solidify the library's potential to contribute significantly to the IoT ecosystem, providing a valuable resource for researchers and practitioners across fields such as healthcare, smart cities, and industry. Ultimately, the iMimic library represents a critical step forward in the development of sophisticated, data-driven IoT solutions.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] S. Madakam, R. Ramaswamy, S. Tripathi, Internet of things (IoT): A literature review, J. Comput. Commun. 3 (5) (2015) 164–173.

[2] H. Espinoza, G. Kling, F. McGroarty, M. O'Mahony, X. Ziouvelou, Estimating the impact of the internet of things on productivity in Europe, Heliyon 6 (5) (2020).

[3] A. Khanna, S. Kaur, Evolution of internet of things (IoT) and its significant impact in the field of precision agriculture, Comput. Electron. Agric. 157 (2019) 218–231, http://dx.doi.org/10.1016/j.compag.2018.12.039, URL: https://www.sciencedirect.com/science/article/pii/S0168169918316417.

[4] M. Nasiri, N. Tura, V. Ojanen, Developing disruptive innovations for sustainability: A review on impact of internet of things (IOT), in: 2017 Portland International Conference on Management of Engineering and Technology, PICMET, 2017, pp. 1–10, http://dx.doi.org/10.23919/PICMET.2017.8125369.

[5] J.T. Kelly, K.L. Campbell, E. Gong, P. Scuffham, The internet of things: Impact and implications for health care delivery, J. Med. Internet Res. 22 (11) (2020) e20135.

[6] S. Li, L.D. Xu, S. Zhao, The internet of things: a survey, Inf. Syst. Front. 17 (2) (2015) 243–259, http://dx.doi.org/10.1007/s10796-014-9492-7.

[7] M. Gigli, S. Koo, Internet of things: Services and applications categorization, Adv. Internet Things 01 (02) (2011) 27–31.

[8] D. Mazzei, G. Baldi, G. Montelisciani, G. Fantoni, A full stack for quick prototyping of IoT solutions, Ann. Telecommun. 73 (7) (2018) 439–449, http://dx.doi.org/10.1007/s12243-018-0644-5.

[9] M. Chernyshev, Z. Baig, O. Bello, S. Zeadally, Internet of things (IoT): Research, simulators, and testbeds, IEEE Internet Things J. 5 (3) (2018) 1637–1647, http://dx.doi.org/10.1109/JIOT.2017.2786639.

[10] J. Muñoz, F. Rincon, T. Chang, X. Vilajosana, B. Vermeulen, T. Walcarius, W. van de Meerssche, T. Watteyne, OpenTestBed: Poor man's IoT testbed, in: IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS, 2019, pp. 467–471, http://dx.doi.org/10.1109/INFCOMW.2019.8845269.

[11] M. Wu, T.-J. Lu, F.-Y. Ling, J. Sun, H.-Y. Du, Research on the architecture of internet of things, in: 2010 3rd International Conference on Advanced Computer Theory and Engineering, Vol. 5, ICACTE, 2010, pp. V5–484–V5–487, http://dx.doi.org/10.1109/ICACTE.2010.5579493.

[12] S. De, P. Barnaghi, M. Bauer, S. Meissner, Service modelling for the internet of things, in: 2011 Federated Conference on Computer Science and Information Systems, FedCSIS, 2011, pp. 949–955.

[13] J.P. Dias, A. Restivo, H.S. Ferreira, Designing and constructing internet-of-things systems: An overview of the ecosystem, Internet Things 19 (2022) 100529, http://dx.doi.org/10.1016/j.iot.2022.100529, URL: https://www.sciencedirect.com/science/article/pii/S2542660522000312.

[14] D. Libes, D. Lechevalier, S. Jain, Issues in synthetic data generation for advanced manufacturing, in: 2017 IEEE International Conference on Big Data, Big Data, 2017, pp. 1746–1754, http://dx.doi.org/10.1109/BigData.2017.8258117.

[15] J. Jordon, L. Szpruch, F. Houssiau, M. Bottarelli, G. Cherubin, C. Maple, S.N. Cohen, A. Weller, Synthetic data–what, why and how?, 2022, arXiv preprint arXiv:2205.03257.

[16] P. Patel, Synthetic data, Bus. Inf. Rev. 41 (2) (2024) 48–52.

[17] M.W. Rodrigues, L.E. Zárate, Time series analysis using synthetic data for monitoring the temporal behavior of sensor signals, in: 2019 IEEE International Conference on Systems, Man and Cybernetics, SMC, 2019, pp. 453–458, http://dx.doi.org/10.1109/SMC.2019.8913907.

[18] C. Hu, Z. Sun, C. Li, Y. Zhang, C. Xing, Survey of time series data generation in IoT, Sensors 23 (15) (2023) http://dx.doi.org/10.3390/s23156976, URL: https://www.mdpi.com/1424-8220/23/15/6976.

[19] J.W. Anderson, K.E. Kennedy, L.B. Ngo, A. Luckow, A.W. Apon, Synthetic data generation for the internet of things, in: 2014 IEEE International Conference on Big Data, Big Data, 2014, pp. 171–176, http://dx.doi.org/10.1109/BigData.2014.7004228.

[20] R. Tolas, R. Portase, R. Potolea, GeMSyD: Generic framework for synthetic data generation, Data 9 (1) (2024) http://dx.doi.org/10.3390/data9010014, URL: https://www.mdpi.com/2306-5729/9/1/14.

[21] J. Morales-García, A. Bueno-Crespo, F. Terroso-Sáenz, F. Arcas-Túnez, R. Martínez-España, J.M. Cecilia, Evaluation of synthetic data generation for intelligent climate control in greenhouses, Appl. Intell. 53 (21) (2023) 24765–24781, http://dx.doi.org/10.1007/s10489-023-04783-2.

[22] M.F. Yousuf, M.S. Mahmud, Generating synthetic time-series data on edge devices using generative adversarial networks, in: 2024 International Conference on Computing, Networking and Communications, ICNC, 2024, pp. 441–445, http://dx.doi.org/10.1109/ICNC59896.2024.10556140.

[23] K.-H. Le Minh, K.-H. Le, AirGen: GAN-based synthetic data generator for air monitoring in smart city, in: 2021 IEEE 6th International Forum on Research and Technology for Society and Industry, RTSI, 2021, pp. 317–322, http://dx.doi.org/10.1109/RTSI50628.2021.9597364.

[24] R. Myung, S. Choi, W. Choi, H. Yu, D. Lee, E. Lee, Elaborate synthetic data generation for internet of things services at smart home environment, in: 2016 International Conference on Computational Science and Computational Intelligence, CSCI, 2016, pp. 226–229, http://dx.doi.org/10.1109/CSCI.2016.0050.

[25] M. Meiser, B. Duppe, I. Zinnikus, SynTiSeD – Synthetic time series data generator, in: 2023 11th Workshop on Modelling and Simulation of Cyber-Physical Energy Systems, MSCPES, 2023, pp. 1–6, http://dx.doi.org/10.1109/MSCPES58582.2023.10123429.

[26] L. Arnaboldi, C. Morisset, Generating synthetic data for real world detection of DoS attacks in the IoT, in: M. Mazzara, I. Ober, G. Salaün (Eds.), Software Technologies: Applications and Foundations, Springer International Publishing, Cham, 2018, pp. 130–145.

[27] S. Rahman, S. Pal, S. Mittal, T. Chawla, C. Karmakar, SYN-GAN: A robust intrusion detection system using GAN-based synthetic data for IoT security, Internet Things 26 (2024) 101212, http://dx.doi.org/10.1016/j.iot.2024.101212, URL: https://www.sciencedirect.com/science/article/pii/S2542660524001537.

[28] X. Sáez-de Cámara, J.L. Flores, C. Arellano, A. Urbieta, U. Zurutuza, Gotham testbed: A reproducible IoT testbed for security experiments and dataset generation, IEEE Trans. Dependable Secur. Comput. 21 (1) (2024) 186–203, http://dx.doi.org/10.1109/tdsc.2023.3247166.

[29] J. Dahmen, D. Cook, SynSys: A synthetic data generation system for healthcare applications, Sensors 19 (5) (2019) http://dx.doi.org/10.3390/s19051181, URL: https://www.mdpi.com/1424-8220/19/5/1181.

[30] S. Kumi, M. Ray, S. Walia, R.K. Lomotey, R. Deters, Digital twins for stress management utilizing synthetic data, in: 2024 IEEE World AI IoT Congress, AIIoT, 2024, pp. 329–335, http://dx.doi.org/10.1109/AIIoT61789.2024.10579038.

[31] C. Esteban, S.L. Hyland, G. Rätsch, Real-valued (medical) time series generation with recurrent conditional GANs, 2017, arXiv:1706.02633.

[32] S. Imtiaz, M. Arsalan, V. Vlassov, R. Sadre, Synthetic and private smart health care data generation using GANs, in: 2021 International Conference on Computer Communications and Networks, ICCCN, 2021, pp. 1–7, http://dx.doi.org/10.1109/ICCCN52240.2021.9522203.

[33] P.V. Lopes, L. Silveira, R.D. Guimaraes Aquino, C.H. Ribeiro, A. Skoogh, F.A.N. Verri, Synthetic data generation for digital twins: enabling production systems analysis in the absence of data, Int. J. Comput. Integr. Manuf. (2024) 1–18, http://dx.doi.org/10.1080/0951192X.2024.2322981.

[34] T. Herbert, J. Mangler, S. Rinderle-Ma, Generating reliable process event streams and time series data based on neural networks, in: A. Augusto, A. Gill, S. Nurcan, I. Reinhartz-Berger, R. Schmidt, J. Zdravkovic (Eds.), Enterprise, Business-Process and Information Systems Modeling, Springer International Publishing, Cham, 2021, pp. 81–95.

[35] N.D. Bokde, A. Feijóo, N. Al-Ansari, Z.M. Yaseen, A comparison between reconstruction methods for generation of synthetic time series applied to wind speed simulation, IEEE Access 7 (2019) 135386–135398, http://dx.doi.org/10.1109/ACCESS.2019.2941826.

[36] A. Shamshad, M. Bawadi, W. Wan Hussin, T. Majid, S. Sanusi, First and second order Markov chain models for synthetic generation of wind speed time series, Energy 30 (5) (2005) 693–708, http://dx.doi.org/10.1016/j.energy.2004.05.026, URL: https://www.sciencedirect.com/science/article/pii/S0360544204002609.

[37] Y. Li, B. Hu, T. Niu, S. Gao, J. Yan, K. Xie, Z. Ren, GMM-HMM-based medium- and long-term multi-wind farm correlated power output time series generation method, IEEE Access 9 (2021) 90255–90267, http://dx.doi.org/10.1109/ACCESS.2021.3091460.

[38] F. Koltuk, E.G. Schmidt, A novel method for the synthetic generation of non-I.I.D workloads for cloud data centers, in: 2020 IEEE Symposium on Computers and Communications, ISCC, 2020, pp. 1–6, http://dx.doi.org/10.1109/ISCC50000.2020.9219577.

[39] S. Baressi Šegota, V. Mrzljak, N. Anđelić, I. Poljak, Z. Car, Use of synthetic data in maritime applications for the problem of steam turbine exergy analysis, J. Mar. Sci. Eng. 11 (8) (2023) http://dx.doi.org/10.3390/jmse11081595, URL: https://www.mdpi.com/2077-1312/11/8/1595.

[40] F. Fonger, M. Aleknonytė-Resch, A. Koschmider, Mapping time-series data on process patterns to generate synthetic data, in: M. Ruiz, P. Soffer (Eds.), Advanced Information Systems Engineering Workshops, Springer International Publishing, Cham, 2023, pp. 50–61.

[41] S. Minhas, Z. Khanam, S. Ehsan, K. McDonald-Maier, A. Hernández-Sabaté, Weather classification by utilizing synthetic data, Sensors 22 (9) (2022) http://dx.doi.org/10.3390/s22093193, URL: https://www.mdpi.com/1424-8220/22/9/3193.

[42] N. Patki, R. Wedge, K. Veeramachaneni, The synthetic data vault, in: 2016 IEEE International Conference on Data Science and Advanced Analytics, DSAA, 2016, pp. 399–410, http://dx.doi.org/10.1109/DSAA.2016.49.

[43] K. Houkjær, K. Torp, R. Wind, Simple and realistic data generation, in: Proceedings of the 32nd International Conference on Very Large Data Bases, VLDB '06, VLDB Endowment, 2006, pp. 1243–1246.

[44] N. Bruno, S. Chaudhuri, Flexible database generators, in: Proceedings of the 31st International Conference on Very Large Data Bases, VLDB '05, VLDB Endowment, 2005, pp. 1097–1107.

[45] B. Nowok, G.M. Raab, C. Dibben, synthpop: Bespoke creation of synthetic data in R, J. Stat. Softw. 74 (11) (2016) 1–26, http://dx.doi.org/10.18637/jss.v074.i11, URL: https://www.jstatsoft.org/index.php/jss/article/view/v074i11.

[46] J. Yoon, D. Jarrett, M. Van der Schaar, Time-series generative adversarial networks, Adv. Neural Inf. Process. Syst. 32 (2019).

[47] H. Pei, K. Ren, Y. Yang, C. Liu, T. Qin, D. Li, Towards generating real-world time series data, in: 2021 IEEE International Conference on Data Mining, ICDM, 2021, pp. 469–478, http://dx.doi.org/10.1109/ICDM51629.2021.00058.

[48] Y. Kang, R.J. Hyndman, F. Li, GRATIS: GeneRAting TIme series with diverse and controllable characteristics, Stat. Anal. Data Mining: ASA Data Sci. J. 13 (4) (2020) 354–376, http://dx.doi.org/10.1002/sam.11461, URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/sam.11461.

[49] H. Klopries, A. Schwung, ITF-GAN: Synthetic time series dataset generation and manipulation by interpretable features, Knowl.-Based Syst. 283 (2024) 111131, http://dx.doi.org/10.1016/j.knosys.2023.111131, URL: https://www.sciencedirect.com/science/article/pii/S095070512300881X.

[50] J. Stewart, Calculus: Early Transcendentals, eight ed., Cengage Learning, Boston, MA, 2015, p. 1368.

[51] E. Kreyszig, Advanced Engineering Mathematics, tenth ed., Wiley, 2011.

[52] E. Parzen, Stochastic Processes, SIAM, 1999.

[53] A.V. Oppenheim, A.S. Willsky, S.H. Nawab, Signals & Systems, Pearson, 1997.

[54] C.M. Bishop, N.M. Nasrabadi, Pattern Recognition and Machine Learning, vol. 4, Springer, 2006.

[55] G.E. Box, G.M. Jenkins, G.C. Reinsel, G.M. Ljung, Time Series Analysis: Forecasting and Control, John Wiley & Sons, 2015.

[56] W.H. Press, Numerical Recipes 3rd Edition: The Art of Scientific Computing, Cambridge University Press, 2007.

[57] F. Fritsch, J. Butland, A method for constructing local monotone piecewise cubic interpolants, SIAM J. Sci. Comput. 5 (2) (1984) 300–304, http://dx.doi.org/10.1137/0905021.

[58] C.R. Harris, K.J. Millman, S.J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N.J. Smith, et al., Array programming with numpy, Nature 585 (7825) (2020) 357–362.

[59] C. Forbes, M. Evans, N. Hastings, B. Peacock, Statistical Distributions, John Wiley & Sons, 2011.

[60] A. Banks, R. Gupta, MQTT Version 3.1.1, Technical Report, OASIS Standard, 2014, URL: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html. Latest version: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html.

[61] A. Banks, E. Briggs, K. Borgendale, R. Gupta, MQTT Version 5.0, Technical Report, OASIS Standard, 2019, URL: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html. Latest version: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[62] J. Kreps, N. Narkhede, J. Rao, et al., Kafka: A distributed messaging system for log processing, in: Proceedings of the NetDB, vol. 11, Athens, Greece, 2011, pp. 1–7.

[63] G.F. Riley, T.R. Henderson, The ns-3 network simulator, in: Modeling and Tools for Network Simulation, Springer, 2010, pp. 15–34.
[64] Max Planck Institute for Biogeochemistry, Current weather and climate data, 2024, https://www.bgc-jena.mpg.de/wetter/. (Accessed: 18 August 2024).
[65] D. Gunopulos, G. Das, Time series similarity measures (tutorial PM-2), in: Tutorial Notes of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '00, Association for Computing Machinery, New York, NY, USA, 2000, pp. 243–307, http://dx.doi.org/10.1145/349093.349108.
[66] J. Serrà, J.L. Arcos, An empirical evaluation of similarity measures for time series classification, Knowl.-Based Syst. 67 (2014) 305–314, http://dx.doi.org/10.1016/j.knosys.2014.04.035, URL: https://www.sciencedirect.com/science/article/pii/S0950705114001658.
[67] S. Lhermitte, J. Verbesselt, W. Verstraeten, P. Coppin, A comparison of time series similarity measures for classification and change detection of ecosystem dynamics, Remote Sens. Environ. 115 (12) (2011) 3129–3152, http://dx.doi.org/10.1016/j.rse.2011.06.020, URL: https://www.sciencedirect.com/science/article/pii/S0034425711002446.
[68] J. Serra, J.L. Arcos, An empirical evaluation of similarity measures for time series classification, Knowl.-Based Syst. 67 (2014) 305–314.
[69] M. Müller, Dynamic time warping, Inf. Retr. Music. Motion (2007) 69–84.
[70] P. Turney, Bias and the quantification of stability, Mach. Learn. 20 (1995) 23–33.
[71] F.J. Van de Vijver, Towards a theory of bias and equivalence, 3, 1998, pp. 41–65, 50967.
[72] R. Fu, Y. Huang, P.V. Singh, AI and algorithmic bias: Source, detection, mitigation and implications, SSRN Electron. J. (2020) http://dx.doi.org/10.2139/ssrn.3681517, Available at SSRN: https://ssrn.com/abstract=3681517.
[73] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, A. Galstyan, A survey on bias and fairness in machine learning, ACM Comput. Surv. 54 (6) (2021) http://dx.doi.org/10.1145/3457607.
[74] J.D. Hamilton, Time Series Analysis, Princeton University Press, Princeton, NJ, 1994, URL: https://press.princeton.edu/books/hardcover/9780691042893/time-series-analysis.
[75] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, W. Zhang, Informer: Beyond efficient transformer for long sequence time-series forecasting, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 35, 2021, pp. 11106–11115.
[76] L.H. Gilpin, D. Bau, B.Z. Yuan, A. Bajwa, M. Specter, L. Kagal, Explaining explanations: An overview of interpretability of machine learning, in: 2018 IEEE 5th International Conference on Data Science and Advanced Analytics, DSAA, 2018, pp. 80–89, http://dx.doi.org/10.1109/DSAA.2018.00018.