TS-Pothole: Automated Imputation of Missing Values in Univariate Time Series

Brell Sanwouo, Clément Quinton, Romain Rouvoy

TS-Pothole: Automated Imputation of Missing Values in Univariate Time Series

Brell Sanwouo, Clément Quinton, and Romain Rouvoy

Univ. Lille, CNRS, Inria, UMR 9189 CRIStAL, F-59000 Lille, France
{brellpeclard.sanwouochekam;clement.quinton;romain.rouvoy}@univ-lille.fr

Abstract. Time series data are pivotal in diverse fields such as finance, meteorology, and health data analysis. Accurate analysis of these data is crucial for identifying temporal trends and making informed decisions. However, frequent occurrences of missing values, often due to device failures or data collection errors, pose a significant challenge. In this work, we introduce TS-Pothole, a method for imputing missing values in univariate time series. This method leverages cyclic pattern analysis and a recursive strategy to handle univariate datasets in which missing values are distributed both continuously and randomly. We evaluate TS-Pothole on four real-world datasets representing different configurations of missing values, and assess its performance in terms of accuracy and execution speed. In particular, we compare our approach with state-of-the-art methods, such as GANs and autoencoders. Our experiments show that TS-Pothole outperforms such methods by providing more accurate (up to 1.5 times) and faster (up to 2 times) imputations, even as the proportion of missing data increases, representing the best alternative in handling univariate time series with missing values.

Keywords: Time series · Imputation · Machine Learning.

1 Introduction

In various fields, such as finance, meteorology, and health data analysis, analyzing time series and how variables change over time is critical to reveal temporal trends and make informed decisions [1]. However, due to various issues, such as *e.g.*, device breakage or data collection errors, time series often report missing values, preventing a fine-grained and accurate analysis. To fix this problem, several approaches have been proposed for imputing missing data in the case of incomplete time series, including multivariate time series approaches (imputing several variables at once) or univariate approaches (imputing a single variable at a time) [2]. Furthermore, while many methods focus on imputing data with defined trends and seasonalities, few tackle the imputation of data characterized by the lack of fixed trends and seasonalities.

Yet, classical methods exhibit limitations, specifically when dealing with the inherent complexity of many real-world time series [3]. As a result, deep learning techniques, such as *Generative Adversarial Networks* (GANs) and *Auto-Encoders* (AEs), have been explored to address these shortcomings [4–12].

This article addresses the problem of imputing missing values in univariate time series with no fixed trends or seasonal patterns. Our main contributions are:

1. we propose TS-Pothole, an imputation method that covers long- and short-gap missing values;
2. we assess TS-Pothole on 4 real-world datasets of varying sizes, demonstrating its applicability to real-life scenarios, whether the missing values are randomly or continuously distributed in the dataset;
3. we discuss the limitations of TS-Pothole, particularly in contexts with weak or non-linear temporal patterns, and avenues for future research.

The experimental results are promising. Compared with the best state-of-the-art approaches, we observe that:

- TS-Pothole maintains consistent performance and low error rates, even when missing data increases, showing its reliability and relevance for real-life applications,
- TS-Pothole consistently outperforms other methods, such as GANs and AEs [4] to impute missing values in time series.

In the remainder of this article, Section 2 introduces the fundamentals of missing data mechanisms and describes the addressed problem. Section 3 analyzes related work and highlights limitations of existing approaches. Section 4 describes TS-Pothole and details how it leverages the extraction and imputation of valued sequences. Section 5 presents the design of our experiments, and Section 6 provides our experimental results. Section 8 concludes the article and highlights future research opportunities.

2 Fundamentals

This section provides an overview of the fundamental concepts related to missing data mechanisms and explains the data imputation problem we address in this paper.

2.1 Mechanisms Causing Missing Data

Various mechanisms contributing to the occurrence of missing data values must be taken into account when attempting to retrieve such values. Accordingly, definitions of missing data are formulated in the literature based on these underlying mechanisms. Rubin established the theory of missing data imputation [13], classifying it into three main mechanisms, each defined according to data availability or missingness. To define these mechanisms more comprehensibly, we adopt the definitions proposed by Emmanuel *et al.* [14].

Let S be a vector of all the data, decomposed into S_o and S_m, with $|S| = |S_o| + |S_m|$. S_o represents the observed data and S_m the missing data. Let R be a vector of missing values defined by:

$$\forall i \in |S|, R(i) := \begin{cases} 0, & \text{if } S(i) \in S_o \\ 1, & \text{if } S(i) \in S_m \end{cases} \tag{1}$$

To illustrate the concept of observed data S_o and missing data S_m, we represent a vector S and a vector R in Table 1. The vector R indicates for each observation whether the data is missing or not. The vector S represents a data set of 10 observations, some of which are missing. A "0" as value of R indicates observed data (present in S_o), while a "1" indicates missing data (present in S_m).

Index	Values of S	Values of R
1	5	0
2	7	0
3	NA	1
4	10	0
5	13	0
6	NA	1
7	4	0
8	21	0
9	23	0
10	NA	1

Table 1. Observations (S) and missingness (R)

Let q be a vector of values that indicates the association between the absence in R and the dataset S. Missing value mechanisms are, therefore, defined by the probability that a value is observed or missing. One can distinguish between 3 missing data mechanisms [13]:

- *Missing Completely At Random* (MCAR): This happens when missing observations do not depend on observed and unobserved measurements. The probability of MCAR is defined as follows: $p(R|q)$;
- *Missing At Random* (MAR): The probability of a missing value in MAR is mainly related to the observable data. The probability of MAR can be defined as $p(R|S_o, q)$;
- *Missing Not At Random* (MNAR): This applies when the missing data is neither MCAR nor MAR. Missing data depends equally on missing and observed values. The MNAR probability is defined as $p(R|S_o, S_m, q)$. The probability that a position R is missing or observed depends on S_o and S_m.

2.2 Problem Statement

Our study focuses on solving the fundamental problem of imputing missing values in univariate time series with missing or weakly fixed trends and seasonality.

Initially, we consider the more general case of multivariate time series, where we have a set of series, $S_1, S_2, \ldots, S_k$, defined over the same time interval $T = \{1, 2, \ldots, n\}$. Each series S_i consists of observations $s_{i,t}$ at each time $t \in T$. The multivariate imputation problem can be formulated as follows: let $M \subset T$ be the set of time indices for which observations in one or more of the series S_i are missing. The goal is to define an imputation function $I : M \to \mathbb{R}^k$ such that, for each $t \in M$, $I(t)$ is a precise estimate of the set of observations $(s_{1,t}, s_{2,t}, \ldots, s_{k,t})$ that would have been observed in the absence of gaps. Formally, our multivariate optimization problem can be expressed as the minimization of a global cost function C defined over M:

$$\min_I C = \sum_{t \in M} L(I(t), (s_{1,t}, s_{2,t}, \ldots, s_{k,t}))$$

where L is a loss function measuring the discrepancy between the imputed value $I(t)$ and the grounded observation $(s_{1,t}, s_{2,t}, \ldots, s_{k,t})$.

Considering the univariate case, we can reduce the problem by considering a single time series variable, S, with missing or weakly fixed trend and seasonality. This is justified when the missing data follows a pattern of either MCAR or MAR. Thus, S becomes the main series, and the missing time indices are defined by $M \subset T$. The goal remains the same: to define an imputation function $I : M \to \mathbb{R}$ minimizing the global cost C according to the following formulation:

$$\min_I C = \sum_{t \in M} L(I(t), s_t)$$

This transition from the multivariate to the univariate case allows us to specialize our approach while focusing on the specific problem of imputing missing values in a univariate time series with missing or weakly fixed trends and seasonality.

3 Related Work

Recovering missing values in time series poses a persistent challenge in the field of temporal data analysis [15–18]. State-of-the-art approaches [4, 14, 19–21] exhibit limitations in their ability to handle the inherent complexity of many real-world time series. In this regard, recent research has explored innovative avenues, ranging from traditional methods to advanced techniques borrowed from machine learning.

3.1 Statistical Approaches for Missing Data

In the field of missing data, statistical methodologies serve as integral tools, employing descriptive statistics to address the challenge of missing values in

datasets. This literature review examines fundamental techniques, starting with the *deletion case* (DC) [19], a method imposing a complete analysis through the systematic removal of observations or features containing missing values. However, judicious application is required, particularly in cases characterized by MCAR. The *simple imputation method* [4,14] offers an alternative, involving the substitution of missing values with features derived from the available data. Despite its simplicity, this method should be used with caution due to its potential consequences, such as artificially reducing variance and overlooking correlation structures. Furthermore, an exploration of *similarity-based methods* [20, 21], including *k-Nearest Neighbors Imputation* (kNNI), *Hot Deck Imputation* (HDI) and *Cold Deck Imputation* (CDI), unveils strategies that replace missing values with estimates derived from analogous samples. kNNI relies on sample similarity, HDI favors samples with fewer missing values, and CDI requires an additional dataset for imputation. These methodologies emphasize the crucial importance of similarity measures and distance calculations in addressing the complex challenge of missing data but neglect more subtle patterns, such as temporality, seasonality, cycles, or correlation between missing data.

3.2 Machine Learning-based Imputation Methods

Machine learning-based missing data imputation methods exploit machine learning techniques to estimate and replace missing values in a dataset. This section explores 4 commonly-used methods:

1. *Decision tree methods* [22, 23], introduced by Breiman *et al.* [24], can be used for classification (Decision Tree Classifier) and regression (Decision Tree Regressor) problems [25]. In data imputation, they build trees based on non-missing values to predict missing values. Variants include *XGBoost imputation* (XGBI) [26] and *MissForest imputation* (MissFI) [27] using the XGBoost [26] and Random Forest [28] algorithms to predict missing values, respectively;
2. *Regression methods* [14] rely on the creation of regression models to predict missing values based on features containing values. Multivariate methods, such as *Multivariate Imputation by Chained Equations* (MICE) [29], build several linear regression models for each feature containing missing values;
3. *Compression models* [4, 21] involve a compression step followed by reconstruction [30]. These methods, such as *Soft-impute* (SI) [31], *Matrix Factorization Imputation* (MFI) [32] and *Principal Component Analysis Imputation* (PCAI) [33], focus on creating models to compress incomplete data. While effective for high-dimensional data, they can require significant computing resources;
4. *Multilayer neural networks (MLP)* [34] : The *Multilayer Perceptron Imputation* (MLPI) approach [35] involves the individual creation of an MLP model [34] for each incomplete feature. MLPI uses these models to estimate missing values. The choice of error function during training depends on the nature of the features, with options such as mean square error or cross-entropy error [36].

These machine learning-based approaches present robust alternatives to traditional methods of imputing missing data. While decision tree and regression methods exploit predictive models to estimate missing values, they can be sensitive to data complexity and non-linear relationships. Compression models, while effective for high-dimensional data, can require significant computational resources.

3.3 Deep Learning-Based Imputation Methods

Deep learning-based missing data imputation methods take advantage of neural networks and deep learning techniques to estimate and replace missing values in a dataset. In this section, we explore *Generative Adversarial Network* (GANs) and *Auto-Encoders* (AEs), the two main approaches for learning deep generative imputation models [4].

AE-based Imputation Methods The fundamental structure of AE-based imputation involves taking an incomplete data matrix and its corresponding mask matrix as inputs. The AE architecture [37] consists of two main components: an *encoder* and a *decoder*. The *encoder* module compresses the input data into a latent representation, while the *decoder* module reconstructs this latent representation into an output that closely resembles the original input data matrix. In the AE-based imputation algorithm, both the encoder and decoder are trained by minimizing a loss function, which quantifies the reconstruction error between the observed values in the input data matrix (X) and the values generated by the decoder to ensure an accurate reconstruction of missing or incomplete data. Many methods are emerging from AE. *Variational AutoEncoder Imputation* (VAEI) [5] uses the *Variational AutoEncoder* (VAE) [6] for single imputation. VAE introduces a prior distribution over latent variables. VAEI involves two phases: model training, where VAE is trained using initially imputed data, and iteration imputation, which iteratively imputes missing components using the trained VAEs. *Heterogeneous-Incomplete VAE* (HI-VAE) [7] is another single imputation framework based on VAE designed for handling missing data. It uses an input dropout encoder to address missing values and a decoder with different likelihood models for diverse data types. It captures statistical dependencies among features using a neural network. HI-VAE optimizes parameters through an evidence lower bound computed on observed components.

However, it is important to note some limitations of VAEI. While this method may be effective in certain scenarios, it can pose challenges when confronted with complex data structures [9, 10] or non-linear dependency models [8]. Additionally, the quality of imputation may be sensitive to the parameters of the VAE model [11], and the iterative nature may introduce cumulative errors. These aspects underscore the need for a thorough evaluation of the performance of VAEI in specific contexts before widespread adoption.

GAN Imputation methods GANs [38] are a groundbreaking innovation in the field of machine learning. First proposed by Good *et al.* [38], GANs provide a

powerful approach to generating realistic data from scratch. The central concept of GANs revolves around the idea of a competition between two neural networks: the *generator* and the *discriminator*. The *generator* creates synthetic data, while the *discriminator* tries to distinguish this synthetic data from real data. This rivalry drives both networks to continuously improve, with the generator becoming increasingly skilled at producing data that is indistinguishable from real data, and the discriminator becoming more competent at telling them apart. For imputation, some GAN-based techniques have already been proposed, such as GAIN and VGAIN.

Generative Adversarial Imputation Network (GAIN) [12] is an innovative approach that comprises two fundamental components. Firstly, there is a *generator* responsible for imputing missing data, a process conditioned on the observed data, ultimately generating a completed vector. Secondly, a *discriminator* evaluates the output from the generator and endeavors to pinpoint the parts of the data that have been imputed. To guarantee the uniqueness of the distribution produced by the generator, the authors introduced a hint vector that correlates with the missing data pattern. This hint vector plays a crucial role in enhancing the imputation process, ensuring that the generated data aligns with the missing pattern.

Variational autoencoder Generative Adversarial Imputation Network (VGAIN) builds on the GAN model by adding a VAE to the generator [39]. The VAE encoder is regularized by imposing a prior on the latent distribution, which means adding Gaussian noise to the latent distribution. This makes the features learned by the generator more robust and effective. VGAIN also uses the reconstruction error of the VAE to encourage the model to better represent the training data.

Overall, imputation methods address the handling of missing data through various approaches, each with its own specific limitations. Statistical techniques, while effective, sometimes struggle to capture subtle nuances such as trends or seasonality. Machine learning-based methods, capable of handling non-linear relationships and data complexity, require considerable computational resources. deep learning strategies, such as AE, may be less effective when faced with complex data structures. Finally, GAN-based [40] and AE approaches are limited by their ability to handle a high level of missing data and are strongly influenced by model architecture and parameters.

While these methods are generally designed to handle data with fixed trends and seasonality, and dominant features in traditional datasets, they may be insufficient for univariate datasets where these patterns are undefined or absent. In the following, we propose a method named TS-Pothole that distinguishes from other approaches by focusing exclusively on univariate datasets, offering a robust solution when traditional methods fail to capture the underlying dynamics.

4 TS-Pothole

This section outlines TS-Pothole, our approach for recursively imputing missing values in univariate time series.

4.1 Overview

TS-Pothole utilizes cyclic pattern analysis and a recursive methodology. The process starts by identifying and extracting sequences from the data. It then determines the longest sequence and uses the *Partial AutoCorrelation Function* (PACF) to measure the direct relationship between an observation and its previous values at different lag levels. Applying this function, TS-Pothole recursively determines the optimal number of *lags* to take into account, starting with the longest available sequence. From the optimal number of lags, it creates new *lagged features* representing previous observations of the time variable lagged by several time steps. The *lagged features* enhance the dataset and provide a solid basis for TS-Pothole to form a linear regression model, which is continuously validated with subsequent sequences to ensure accuracy and robustness. This validation enables dynamic adjustments to be made to the model in response to new data. Once validated, the model is used recursively to impute missing values, relying on the relationships identified to predict these gaps. This recursive application progressively refines the model, guaranteeing imputation accuracy and adaptability.

4.2 Extraction of Sequences without Missing Values

The essential prerequisite of TS-Pothole lies in dividing the univariate time series S into continuous sequences, denoted $s_1, s_2, ..., s_n$. Each sequence s_i is rigorously defined as a subpart of S characterized by the consecutive presence of data. In the example proposed in Table 1, three sequences are presented. Sequence s_1 covers indices 1 to 2 inclusive, sequence s_2 extends from indices 4 to 5 inclusive, and sequence s_3 spans from indices 7 to 9 inclusive.

Formally, each s_i is represented as follows:

$$s_i = \{s_{i,1}, s_{i,2}, ..., s_{i,k}\} \quad \text{with} \quad s_{i,j} \in S, \forall j$$

Thus, s_i encapsulates a data sequence where each observation follows uninterruptedly from the previous one, establishing a fundamental temporal continuity. Mathematically, the complete set of extracted sequences, denoted $S_{\text{extracted}}$, is formalized as follows:

$$S_{\text{extracted}} = \{s_1, s_2, ..., s_n\}$$

This initial step serves as a robust foundation for our analysis of cyclic patterns and guides the implementation of the missing value imputation strategy. Each $s_i \in S_{\text{extracted}}$ represents a continuous temporal unit, devoid of gaps, thus constituting crucial elements for the subsequent steps of TS-Pothole. For example, in Fig. 1, the grey areas represent present valued data, while the white areas reflect missing data.

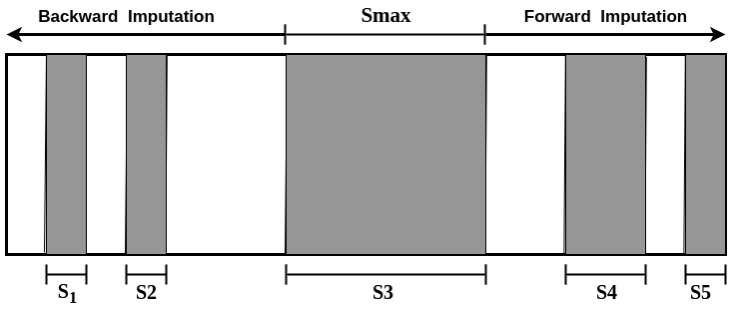


Fig. 1. TS-Pothole Imputation Process

4.3 Selection of the Longest Sequence ($s_{\max}$)

After extracting continuous sequences without missing values from the input univariate time series S, the longest sequence, denoted $s_{\max}$, is identified as the basis for model training and subsequent prediction. This sequence is essential for TS-Pothole, as it provides a substantial representation of the time series, capturing extended cyclic patterns and significant temporal structures.

Mathematically, the longest sequence $s_{\max}$ is determined by evaluating the length of each sequence s_i in the set $S_{\text{extracted}}$ as follows:

$$s_{\max} = \arg \max_{s_i \in S_{\text{extracted}}} |s_i| \tag{2}$$

Here, $|s_i|$ represents the length of sequence s_i, and arg max indicates that we select the sequence s_i that maximizes this length. Thus, we identify $s_{\max}$ as the sequence with the greatest number of consecutive observations. Fig. 1 depicts the maximum sequence S_3 among all sequences we used as an example $\{S_1, ..., S_5\}$.

Once the maximum sequence has been captured, the rest of our approach revolves around the prediction of missing values on both the right (*forward imputation*) and left (*backward imputation*) sides of this bootstrap sequence, as depicted in Fig. 1.

4.4 Forward Imputation

Forward imputation in TS-Pothole begins by determining the optimal number of lagged features by analyzing partial autocorrelations. Next, a linear regression model is built and validated with additional data. The final step consists of iteratively imputing the missing values, and adjusting and re-evaluating the model until the missing data is complete.

Determination of the optimal number of lagged features This phase aims to determine the optimal number of *lagged features*, leveraging the potential of the PACF [41, 42] in modeling time series. The process is divided into three

steps: *Step 1.1 - Calculation of PACF.* First, we compute the PACF for potential lags of the maximum sequence ($s_{\max}$). According to Gasparini *et al.* [43], lags refer to a delay—or time gap—between cause and effect in a series of data observations. It is defined as a temporal lag embedded in the data observation, where each lag represents a time interval separating two consecutive data points. PACF measures the correlation between observations at specific lags as follows, eliminating the influence of intermediate lags:

$$\mathrm{PACF}(s_{\max}, k) = \frac{\mathrm{cov}(s_{\max}[t], s_{\max}[t-k])}{\sqrt{\mathrm{var}(s_{\max}[t]) \cdot \mathrm{var}(s_{\max}[t-k])}} \tag{3}$$

Equation 3 describes the computation of the PACF at lag k. In statistics, partial autocorrelation measures the correlation between 2 variables (here, the values of a time series at different points in time), while controlling for the effects of other lags up to $k-1$. $\mathtt{cov}(s_{\max}[t], s_{\max}[t-k])$ represents the covariance between the time series values at time t and at time $t-k$—*i.e.*, to what extent $s_{\max}$ vary. If the values tend to vary in the same direction, the covariance is positive; if they vary in opposite directions, it is negative. $\mathtt{var}(s_{\max}[t])$ and $\mathtt{var}(s_{\max}[t-k])$ represent the variances of the time series values at times t and $t-k$, respectively—*i.e.*, how much such values are spread out from their mean. Finally, $\sqrt{\mathtt{var}(s_{\max}[t]) \cdot \mathtt{var}(s_{\max}[t-k])}$ normalizes the covariance, the whole formula resulting in a correlation coefficient. Overall, the PACF at lag k is the correlation between the values of the time series separated by k time units, after removing the effect of correlations at intermediate lags.

Step 1.2 - Identification of significant lags. This step is guided by the statistical confidence threshold, typically set at $\pm 1.96/\sqrt{N}$, as recommended by Herrey [44]. 1.96 corresponds to the limits within which 95% of the values of a standard normal distribution lie and N represents the sample size.

Lags whose PACF exceeds this threshold indicate a significant partial correlation, revealing potential relationships useful for prediction or imputation. Precise identification is crucial for model quality: insufficient lags can reduce accuracy, while too many lags can introduce noise and unnecessarily complicate the model, leading to overfitting. Therefore, an informed choice of significant lags ensures a balance between accuracy and complexity, guaranteeing robust and reliable time series modeling. Fig. 2 shows a graph of partial autocorrelation. The vertical blue lines depict the range of potential changes. The light blue band illustrates the confidence interval, while the vertical bars outside this interval indicate significant *lagged features*.

Step 1.3 - Selection of the optimal number of lagged features. The optimal lag selection process is critical in time series modeling. It is generally performed after plotting the PACF. This graph aids in identifying lags with statistically significant partial correlations. The optimal lag is generally the largest lag where the value of PACF (on the y-axis) surpasses the confidence threshold, suggesting a significant impact on the series. This selection aims to include enough historical data for modeling temporal dependencies while minimizing model complexity. For example, in Fig. 2, the optimal number of *lagged features* is 4.

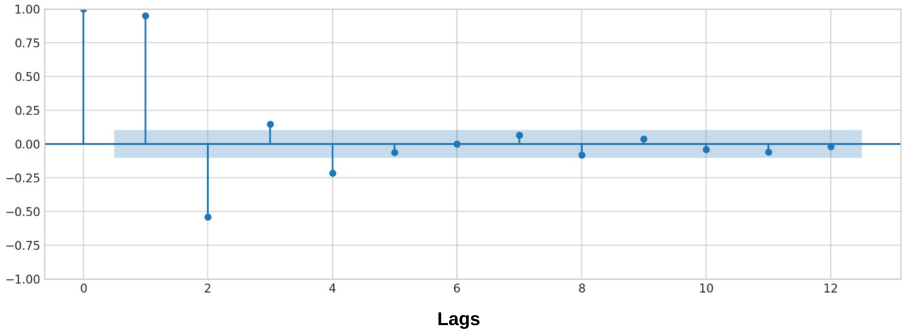


Fig. 2. Confidence interval (partial autocorrelation) used to determine significant lags and optimal lag.

To perform this selection and determine the optimal number of *lagged features* (l_{optimal}), we rely on Algorithm 1. The algorithm initiates with the loading of time series data (line 1), a critical step to ensure the readiness of necessary data for subsequent analysis. After loading data, the function `calculatePACF()` (line 2) is executed to compute the PACF for various lags.

Next, the statistical significance threshold is determined using the `setSignificanceThreshold()` function (line 3), establishing a criterion to distinguish significant temporal correlations from random fluctuations. This threshold is computed based on the specific properties of the data, allowing a tailored adaptation to the unique characteristics of each time series data set.

From line 5 to line 9, Algorithm 1 proceeds to analyze the computed PACF values. For each lag *lag*, we check whether the absolute value of the correlation at this specific lag, `PACF`(lag), exceeds the significance threshold. This step is crucial, as it determines whether the influence of a past lag on the current value is statistically significant or not. If a PACF value surpasses the threshold, it indicates that the corresponding lag has a significant impact on the current values of the time series.

The optimal lag l_{optimal} is initially set to zero and is updated as the algorithm iterates through the lag values. If a lag value exceeds the significance threshold, l_{optimal} is adjusted to match this lag. This iterative approach ensures that the returned lag is the one with the strongest significant correlation, thereby providing a solid foundation for further analyses.

In short, the initial stage of forward imputation in TS-Pothole *begins with an analysis using the partial autocorrelation function to identify significant dependencies between observations. The most relevant lags are then selected based on their influence and statistical relevance. This process allows the model to effectively capture and leverage the inherent dynamics in the data, enabling reliable and accurate prediction of future trends.*

Algorithm 1 Computing Optimal Number of Lagged Features

```
TS ← loadTimeSeriesData()
PACF_Values ← calculatePACF(TS)
λ ← setSignificanceThreshold(TS)
l_optimal ← 0
for lag ∈ PACF_Values do
    if |PACF(lag)| > λ then
        l_optimal ← lag
    end if
end for
return l_optimal
```

Building the Linear Regression model This phase consists of building a linear regression model and validating it with the following sequence.

Step 2.1 - Initial model training. The *lagged features* retrieved from l_{optimal} are then used as training features to train a linear regression model. Relying on these features, one can create a model able to make decisions based on previous observations.

Step 2.2 - Validation with sequence s_{max+1}. Finally, to validate the relevance of the selected features, we use $s_{\max+1}$ as validation data. Using this approach, we can see how well the model predicts missing values between $s_{\max}$ and $s_{\max+1}$. By doing so, we use PACF to guide the feature selection, hence improving time series modeling in TS-Pothole.

Step 2.3 - Model saving. After training the model, we save the model for potential future use. To do this, we create a dictionary *model_dict* that contains the following information for each item:

- l_{optimal}, the optimal lag number for the training data,
- *training_data_start_index*, the starting index of the training data,
- *training_data_end_index*, the ending index of the training data,
- *model*, the linear regression model itself.

Through this phase, we can determine how well the model predicts missing values between $s_{\max}$ and $s_{\max+1}$.

In summary, the second stage of forward imputation in TS-Pothole *consists of building the linear regression model. This stage begins with the initial training of the model, using the previously selected lagged features to learn the underlying relationships in the data. Next, the model is validated using the following sequence, represented by* s_{max+1}*, to test its performance and ability to generalize to new data.*

Iterative Imputation of Missing Values In this phase, we adopt an iterative imputation strategy using a dynamic dataset formed from $s_{\max,1}$ to $s_{\max+1,k}$. The procedure unfolds as follows:

Step 3.1 - Adjustment of lagged features. At each iteration, we reevaluate the optimal number of *lagged features*, using the sequence $s_{\max}$ as a starting point. This ensures constant adaptation to potential changes in the temporal structure. Precisely, the optimal number of *lagged features* l_{optimal} at iteration t is determined by:

$$l_{\text{optimal}}^{(t)} = \arg\max_{l} \text{PACF}(l) \tag{4}$$

Step 3.2 - Iterative model training. At this step, if the new l_{optimal} is already present in the *model_dict* dictionary, then the corresponding *model* is updated with the new data. This involves adjusting the existing model to reflect the recent trends in the time series. Conversely, if the new l_{optimal} is not present in *model_dict*, then a new temporal configuration is handled. In such a case, a new linear regression model is trained with the current data, and the information related to this new model is saved in the dictionary, including the number of lags, the start and end indices of the training data, and the model itself.

Step 3.3 - Prediction of missing values. Using the newly trained model, we predict the missing values between $s_{\max}$ and $s_{\max+1}$. This is formulated as:

$$Y_{\text{predicted}}^{(t)} = \text{Model}(X_{\text{current}}^{(t)}) \tag{5}$$

where $Y_{\text{predicted}}^{(t)}$ represents the predicted values, and $X_{\text{current}}^{(t)}$ is the current dataset.

Step 3.4 - Update of maximum sequence (s_{max+1}). The new maximal sequence (from $s_{\max,1}$ to $s_{\max+1,k}$) becomes the new basis for the next iteration. This allows for continuous adaptation to changes in the time series.

Step 3.5 - Repetition of the process. This iterative process (steps 3.1 to 3.4) is repeated using each new sequence ($s_{\max+1,1}$ to $s_{\max+2,k}$, $s_{\max+2,1}$ to $s_{\max+3,k}$, etc.) until all missing values up to s_n are imputed.

This iterative approach ensures progressive and adaptive imputation, optimizing the quality of predictions at each iteration and adjusting to potential variations in the time series.

The final stage of forward imputation in TS-Pothole *is the iterative imputation of missing values, crucial for maintaining the integrity and efficiency of the linear regression model in time series analysis. This process starts by adjusting the lagged features to accurately fit the existing data. The model is then trained iteratively, ensuring each version is optimized for the fitted data context. Missing values are predicted using the current model, and the maximum sequence is updated by incorporating new observations, denoted by s_{max+1}. This cycle continues until all missing values are imputed, ensuring the model's robustness and the data's completeness for further analysis.*

4.5 Backward Imputation

This phase consists of predicting time sequences before the maximum sequence. This process involves reversing the time series from the maximum sequence ($s_{\max}$) and repeating the imputation process. The procedure unfolds as follows.

Reversal of the series. The time series is reversed from the maximum sequence ($s_{\max}$). This reversal allows for considering the data preceding the maximum sequence as the new portion of the series to be imputed.

Application of the imputation step in a loop. In this step, we apply all steps of forward imputations on the inverted series, starting from the maximum sequence $seq_{\max}$. These steps, presented in Section 4.4, include adjusting the optimal number of *lagged features*, training the model, predicting missing values, and updating the maximum sequence.

Restoration of temporal order. Once the preceding sequences are imputed, the time series is restored to its original temporal order using Algorithm 2. This iterative process of predicting preceding sequences is executed until all missing values before the maximum sequence are imputed. This approach strengthens the model's ability to reliably anticipate and complete missing data throughout the time series.

Backward imputation in TS-POTHOLE *begins by reversing the temporal order of the series to handle the data as if they were progressing backward in time. Next, the imputation steps are applied in a loop, using similar techniques as for Forward Imputation, but with the series reversed. Finally, once the missing values have been filled, the original temporal order of the series is restored, thus ensuring the integrity and continuity of the temporal data.*

Initially, Algorithm 2 employs the function `extractSequences`(*series*) (line 1) to partition the time series data, *series*, into distinct sequences. It then utilizes `selectLongestSequence`(*sequences*) (line 2) to determine $seq_{\max}$, the longest sequence among these, which sets a benchmark for subsequent imputation steps. The `selectOptimalNbLaggedFeatures`($seq_{\max}$) function then calculates the optimal number of *lagged features*, denoted as l_{optimal} (line 3). This number is essential in defining the span of past values to be considered for imputation and prediction in the sequences. Using functions `getForwardSequences(`*sequences*`)` (line 4) and `getBackwardSequences(`*sequences*`)` (line 5), Algorithm 2 creates two sets $forward_{seq}$ and $backward_{seq}$. $forward_{seq}$ contains all the sequences that are next to the maximal sequence (including the $seq_{\max}$ sequence) and $backward_{seq}$ contains all the sequences that precede the maximal sequence (including the $seq_{\max}$).

During the forward imputation process (line 6 to 14), Algorithm 2 iterates through each sequence *seq* in $forward_{seq}$. If *seq* matches $seq_{\max}$, it calls `createLaggingFeatures`(*seq*, l_{optimal}) to generate new features based on the computed lags. It then progresses to `trainModel`(*seq*) for model training, using $seq + 1$ for model validation. This is followed by `imputeMissingValues`(*seq*) to impute missing values between *seq* and *seq*+1. The imputed sequence is stored in the variable $s_{\text{forward\_imputed}}$. If the optimal number of lagged features is already present in the dictionary *model_dict*, the corresponding model is retrieved and updated with the new data. Otherwise, a new linear regression model is trained

Algorithm 2 Forward & Backward Data Imputation

1: $sequences \leftarrow$ `extractSequences(`$series$`)`
2: $seq_{\max} \leftarrow$ `selectLongestSequence(`$sequences$`)`
3: $l_{\text{optimal}} \leftarrow$ `selectOptimalNbLaggedFeatures(`$seq_{\max}$`)`
4: $forward_{seq} \leftarrow$ `getForwardSequences(`$sequences$`)`
5: $backward_{seq} \leftarrow$ `getBackwardSequences(`$sequences$`)`

6: **for** $seq \in forward_{seq}$ **do**
7: **if** $seq = seq_{\max}$ **then**
8: `createLaggingFeatures(`seq, l_{optimal}`)`
9: $model\_dict \leftarrow$ `trainModel(`seq`)`
10: $s_{\text{forward\_imputed}} \leftarrow$ `imputeMissingValues(`seq`)`
11: **else**
12: $s_{\text{forward\_imputed}} \leftarrow$`imputeAuto(`$seq$, l_{optimal}`)`
13: **end if**
14: **end for**

15: $s_{\text{inverse}} \leftarrow$ `reverseTimeSeries(`$backward_{seq}$`)`
16: **for** $seq_{\text{rev}} \in s_{\text{inverse}}$ **do**
17: $l_{\text{optimal}} \leftarrow$ `selectOptimalNbLaggedFeatures(`seq_{rev}`)`
18: `createLaggingFeatures(`seq_{rev}, l_{optimal}`)`
19: $model\_dict \leftarrow$ `trainModel(`seq_{rev}`)`
20: $s_{\text{backward\_imputed}} \leftarrow$ `imputeMissingValues(`seq_{rev}`)`
21: **end for**
22: $s_{\text{imputed}} \leftarrow$ `concatenate(`$s_{\text{forward\_imputed}}$, $s_{\text{backward\_imputed}}$`)`
23: **return** s_{imputed}

with the current data and the information of this new model is saved in the dictionary.

In cases where seq differs from $seq_{\max}$, function `imputeAuto(`seq, l_{optimal}`)` automatically imputes missing values that are next to the maximal sequence. The value of the updated sequence is stored in the variable $s_{\text{forward\_imputed}}$.

Subsequently, for backward imputation (line 15 to 21), the algorithm reverses $backward_{seq}$ using `reverseTimeSeries(`$backward_{seq}$`)` to put the maximal sequence (initially at the end of $backward_{seq}$) at the beginning of the sequence. It then processes each sequence seq_{rev} in the reversed set with the same succession of functions (`selectOptimalNbLaggedFeatures`, `createLaggingFeatures`, `trainModel`, and `imputeMissingValues`), finally returning $s_{\text{rev\_imputed}}$.

Finally, the imputed sequences from the forward and backward imputation phases are concatenated to form s_{imputed}, the final dataset (line 22).

5 Experimental Methodology

This section describes our hardware settings and our experimental protocol. All experimental materials and performance data monitored during our experiments

as well as a TS-Pothole implementation are publicly accessible at `https://zenodo.org/records/10518265`.

5.1 Hardware Settings

In our experimental setup, two consistent hardware configurations were employed for training our deep learning and traditional machine learning models. For deep learning tasks, we utilized the Kaggle platform, leveraging an NVIDIA Tesla P100 GPU featuring 3584 CUDA cores and 16 GB of dedicated memory. For traditional machine learning, a DELL Latitude 7520 laptop was employed, equipped with an 11^{th} generation Intel® Core™ i7-1185G7 vPro® processor, 32 GB of LPDDR4x memory clocked at 4267 MHz in a dual-channel configuration, and a 512 GB M.2 PCIe x4 NVMe SSD with self-encrypting drive capabilities. This deliberate selection of distinct hardware configurations catered to the specific requirements of each task, facilitating a comprehensive and comparative assessment of TS-Pothole's performance.

5.2 Datasets

We evaluated TS-Pothole on two incomplete data scenarios, with either random or continuous missing values. These scenarios were assessed with 4 different datasets, as reported by Table 2. In particular, we used the Beijing air quality dataset [45] as the baseline to assess TS-Pothole on missing data. This dataset covers the period from March 1, 2013, to February 28, 2017, and is sourced from 12 nationally monitored sites managed by the Beijing Municipal Environmental Monitoring Center. As this dataset does not exhibit missing values, we deliberately removed some data from the dataset by voiding values following a continuous (all values are introduced one after the other) and random distribution, focusing on the fine particulates PM2.5 variable, essential for assessing air quality due to its impact on human health. We then applied TS-Pothole to the resulting dataset, *i.e.*, the one where missing values were introduced. We then relied on three datasets commonly used in the literature to compare univariate time series imputation methods [46]. On the one hand, the Groundwater[1] dataset includes a time series containing more than five million samples of groundwater levels from a monitoring wall in Butte County, California, spanning from 1919 to 2023. This dataset represents the groundwater elevation above the mean sea level, critical for understanding environmental and climatic influences on groundwater resources. On the other hand, the Fluxnet[2] dataset encompasses measurements of CO_2 and water levels, energy exchange, and other meteorological and biological variables from 1999 to 2014 at the Morgan Monroe State Forest site, Indiana, United States. This dataset focuses on the *vapor pressure deficit* variable, key for studying atmosphere-biosphere interactions. In this study, we focused on the CO_2 variable and two derived subsets: the

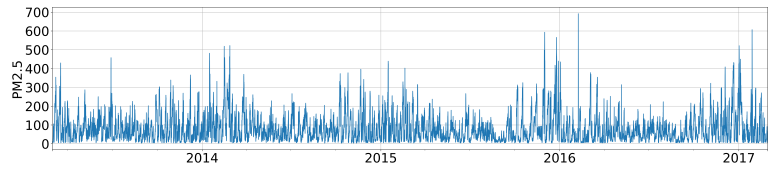
[1] `https://data.cnra.ca.gov/dataset/periodic-groundwater-level-measurements`
[2] `https://fluxnet.org/data/fluxnet2015-dataset/`

Dataset	Metric	Size (# samples)
Beijing [45]	PM2.5	$\sim 35,065$
Fluxnet500 [46]	CO_2	$\sim 500,000$
Fluxnet2M [46]	CO_2	$\sim 2,000,000$
Groundwater [46]	Water elevation	$\sim 5,400,000$

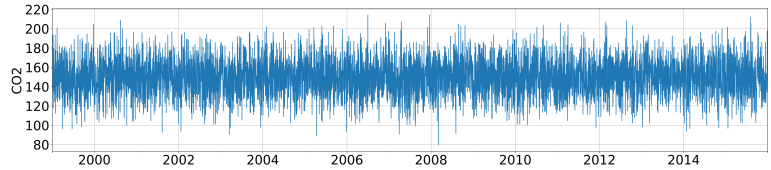
Table 2. Datasets considered in this study.

first, designated Fluxnet500, comprises $\sim 500,000$ samples, while the second, designated Fluxnet2M, contains two million samples.

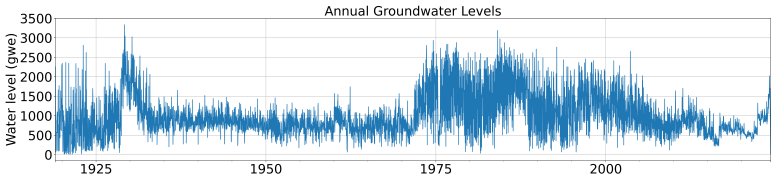
These datasets, presented in Figure 3, were chosen for their relevance in assessing the effectiveness of time series imputation methods, such as TS-Pothole, in handling missing data. We continuously and randomly introduced missing values across different features in each of these two datasets to simulate realistic incomplete data scenarios and evaluate the robustness and versatility of TS-Pothole.



(a) PM2.5 from Beijing Air Quality



(b) Air temperature from Fluxnet



(c) Water level from Groundwater

Fig. 3. Plot of datasets

5.3 Implementation and Configuration

To build TS-Pothole, we integrated several machine learning models, each with distinct architectures and configurations, to address the challenges of imputation. In particular, we implemented a Python-based linear regression model, using Python version 3.10.9. This linear regression model is developed using the Scikit-learn library, version 1.3.2. This particular version of Scikit-learn is selected to leverage its latest advancements and optimizations in machine learning algorithms. We have adopted the following parameters for the linear regression model in Scikit-learn, which provides a standardized baseline for our study.

- fit_intercept : True,
- copy_X : True,
- n_jobs: -1,
- positive: True.

The GAN-based models we evaluated, specifically GAIN and VGAIN architectures, have been developed using generative adversarial networks. These GAN-based models, as well as AE-based models, VAEI, and HI-VAEI models, were constructed using TensorFlow 2.15.0 and Keras 2.14.0. Each of them was built using the following parameters:

- latent_dim $\in \{100, 200, \ldots, 500, 600\}$,
- Epochs $\in \{50, 100, \ldots, 250, 300\}$,
- learning_rate $\in \{0.0001, 0.001, 0.01\}$,
- activation_discriminator $\in \{'sigmoid', 'linear'\}$,
- activation_generator $\in \{'tanh', 'relu'\}$,
- layers $\in \{2, 3, 4\}$,
- Dropout rate $\in \{0.0, 0.1, \ldots, 0.4, 0.5\}$,
- optimizer $\in \{'Adam', 'SGD'\}$,
- loss_function $\in \{'mean\_squared\_error'\}$.

These parameter ranges were selected by common practice in deep learning to ensure a comprehensive yet manageable experimental setup. Applying a method similar to GridSearch CV[3], we were able to train $62,208$ distinct model configurations for each dataset, totaling $248,832$ deep learning models over four datasets. This ensured exhaustive coverage of the usual parameter space used while limiting experimentation time to around $1,085$ hours.

5.4 Systems under Study

In the development of our GAIN model, we anchored our implementation on the codebase available in the GAIN-Pytorch repository.[4] This repository served as a foundational framework, which we adapted and enhanced to align with the specific requirements and objectives of our research. Similarly, for the construction

[3] `https://scikit-learn.org/stable/modules/grid_search.html`
[4] `https://github.com/dhanajitb/GAIN-Pytorch`

Algorithm 3 Creating Missing Values in Time Series

```
1: validateInputs(series, rate, dis)
2: missing_ind ← getMissingIndices(series, rate, dis)
3: for index ∈ missing_ind do
4:    if dis is CONTINUOUS then
5:       series ← createContMissValues(series, index)
6:    else if dis is RANDOM then
7:       series ← createRandMissValues(series, index)
8:    end if
9: end for
10: return series
```

of our VAE models, we drew inspiration from and built upon the code found in the PyTorch-VAE repository.[5] This source provided a valuable starting point, enabling us to integrate advanced features and customize the models following our study's needs.

For the VGAIN and HI-VAE models, our development strategy was based predominantly on their descriptions in the scientific articles [7, 39], respectively. These articles provided the essential theoretical and architectural guidelines, directing our model construction. We meticulously analyzed the methodologies detailed in these publications, developing our VGAIN and HI-VAE models from the ground up. This approach ensured that our implementations were not only theoretically sound, as per the descriptions in the respective articles, but also finely tuned to fit the particularities and objectives of our study.

5.5 Experimental protocol

Step 1 - Missing data generation. In the initial phase of our time-series experimentation, we systematically introduced missing values into our univariate datasets, ensuring that the modified datasets retained the same distribution as the original data. This process involved the random deletion of 10%, 15%, 20% and 25% of the selected features data points. Using Algorithm 3, we distributed these missing values in two distinct patterns: continuously, to simulate interruptions or gaps in data collection, and randomly, to mimic irregular data loss. Algorithm 3 relies on three variables: the time series (*series*), the missing rate to be introduced (*rate*), and the distribution type (*dis*) which determines how missing values will be introduced into the dataset. The creation of missing values begins with `validateInputs` (line 1), a subroutine that ensures the inputs are within acceptable ranges and formats. This step is critical to maintain data integrity and prevent computational errors. Subsequently, the function `getMissingIndices` (line 2) computes a set of indices in the time *series* where missing values will be introduced, based on the specified *rate* and distribution type (*dis*).

[5] `https://github.com/AntixK/PyTorch-VAE`

The algorithm then iterates through each *index* of the $missing_{ind}$ set and, depending on the distribution specified, it employs either `createContMissValues` (line 5) or `createRandMissValues` (line 7). The former introduces missing values in a continuous block, mimicking scenarios like sensor outages or data transmission losses. The latter randomly distributes missing values throughout the *series*, representing scenarios where data is missing randomly and without a specific pattern. This dual distribution strategy provides a comprehensive assessment of TS-Pothole's capability to handle different missing data scenarios, thus reflecting a variety of real-life situations. For each type of distribution and ratio of missing data, we ran 100 generations to ensure robust statistical significance. Finally, the *series* now containing the specified percentage of missing values as per the chosen distribution type, is returned. This output is pivotal for subsequent analyses, *e.g.*, for testing the efficacy of imputation algorithms under different conditions of missing data.

Step 2 - Identification of continuous / longest sequences. The resulting time series is then analyzed to identify present sequences and determine the longest continuous one. We sought these sequences within the time series after introducing missing values. The emphasis was placed on identifying continuous sequences of non-missing data.

Step 3 - Determination of the optimal number of lagged features. During this phase, the goal is to define the optimal number of lagged features for the considered time series data by carefully examining the inherent cycles in the time series and analyzing the PACF. To achieve this, Algorithm 1 is used together with the protocol described in Section 4.4 to determine such an optimal number of lagged features.

Step 4 - Recursive imputation. The fourth step of TS-Pothole involves recursively training a machine learning model on data from different sequences to impute missing values and progressively expand the dataset. To achieve this, we relied on Algorithm 2 following the protocol described in Sections 4.5 and 4.4 to recursively impute data.

5.6 Evaluation Metrics

Upon completing the iterative process of imputing missing values, one needs to assess the effectiveness of TS-Pothole in such an imputation process. This assessment is performed by comparing TS-Pothole imputed data with actual data from the initial time series. In particular, we considered some of the evaluation metrics most commonly used to assess prediction accuracy in the context of time series, namely the *Root Mean Squared Error* (RMSE), the *Mean Absolute Error* (MAE), and the *NashSutcliffe Model Efficiency* (NSE). In the remainder of this section, $\hat{Y}_t$ denotes predicted values at a given point in time and Y_t denotes observed values at a given point in time.

– **RMSE (Root Mean Squared Error)**: This metric evaluates the square root of the mean of the squares of the deviations between the predicted values and the actual values. It is particularly sensitive to large deviations,

making it ideal for identifying significant errors in imputations. The formula for computing the RMSE is as follows:

$$RMSE = \sqrt{\frac{\sum_{t=1}^{n}(\hat{Y}_t - Y_t)^2}{n}}$$

– **MAE (Mean Absolute Error)**: The MAE measures the mean of the absolute deviations between the predictions and the real values, providing a direct and easily interpretable perspective on mean errors in imputations. The MAE is computed as follows:

$$MAE = \frac{\sum_{t=1}^{n}|\hat{Y}_t - Y_t|}{n}$$

– **NSE (Nash-Sutcliffe Efficiency)**: The NSE is a normalized metric that compares the variance of the errors with the variance of the original observations. It is often used to assess the performance of hydrological models, but adapts well to other temporal contexts. A perfect NSE score is 1, indicating a perfect match between the imputed values and the real values. The formula for computing the NSE is:

$$NSE = 1 - \frac{\sum_{t=1}^{n}(\hat{Y}_t - Y_t)^2}{\sum_{t=1}^{n}(Y_t - \bar{Y})^2}$$

These three metrics provide a quantitative understanding of the fidelity of our imputations to ground truth. Lower RMSE and MAE values indicate a better fit between model predictions and observed reality, while higher NSE values indicate better predictive performance.

In addition to utilizing these metrics to assess the accuracy of our imputations, we will also employ mean imputation, standard deviation imputation, and linear interpolation imputation as baseline methods, as they are among the most commonly used techniques in the field.

The first baseline method, **mean imputation**, involves replacing missing values in the dataset with the overall mean ($\bar{Y}$) of the observed data. This method is particularly straightforward and well-suited for datasets with minimal variations and without strong trends or seasonality. The formula for mean imputation is :

$$\text{Imputed value} = \bar{Y} = \frac{\sum_{i=1}^{n} Y_i}{n}$$

where Y_i represents the observed values and n is the number of observed data points.

The second method, **standard deviation imputation** (σ), involves replacing missing values with the mean of the observed values plus or minus a multiple of the standard deviation of the observed data. This method is useful when one wishes to retain some variability in the imputed data. The formula for standard deviation imputation is:

$$\text{Imputed value} = \bar{Y} \pm k\sigma$$

where k is a factor determined based on the distribution of the data and the desired level of confidence, $\bar{Y}$ is the mean of the observed values, and σ is the standard deviation of the observed values, defined as:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \bar{Y})^2}$$

The third method, **linear interpolation**, estimates missing values by creating a function that mathematically models the existing data points. Linear interpolation new data points within the range of a discrete set of known data points. The formula for linear interpolation between two points Y_a and Y_b at times t_a and t_b respectively, and estimating the value at time t is:

$$\text{Imputed value} = Y_a + \frac{(Y_b - Y_a)}{(t_b - t_a)} \times (t - t_a)$$

Finally, we will use the calculation of mean ranks, as used in the Nemenyi test, to rank the different methods. The Nemenyi test calculates the mean ranks of the methods to determine their relative performance order. The formula for calculating the mean rank R_i for a method i is:

$$R_i = \frac{1}{N}\sum_{j=1}^{N} r_{ij}$$

where:

- r_{ij} is the rank of the i-th method on the j-th data set.
- N is the number of data sets or experiments on which the methods were tested.

The parameter N therefore represents the number of comparisons or experiments carried out, not the number of individual data sets. By using only the calculation of mean ranks, we can establish a ranking of the imputation methods according to their performance, without performing a full statistical comparison between each pair of methods. In our case, the methods are compared using three metrics (RMSE, MAE, and NSE) in contexts where the missing values are either continuously or randomly distributed. Therefore, we have six comparisons ($N = 6$) presented in Figures 4, 5 and 6.

6 Results

6.1 Results Overview

Table 3 to Table 10 present a comparison of the performance of various imputation methods on the four datasets detailed in Table 2 (F-500 stands for

Fluxnet500, F-2M for Fluxnet2M and Ground. for Groundwater), with missing data rates ranging from 10% to 25%. Each table illustrates a specific evaluation metric (see Section 5.6) based on the distribution of missing values, either randomly or continuously. The imputation methods evaluated include TS-Pothole (TS-P), GAIN, VGAIN, VAE, HI-VAE, as well as commonly used basic methods such as mean imputation, standard deviation imputation (Std), and linear interpolation (L.I).

The values in the tables represent the means of 100 executions of each method. For each missing data rate, each dataset, and each metric, each method was executed and evaluated 100 times. The values presented in the tables are the means of these 100 evaluations, ensuring that the performance comparison is robust and not influenced by random variations. For each row, green cells indicate that TS-Pothole has the best performance compared to the other methods, while red cells indicate that a method other than TS-Pothole is better.

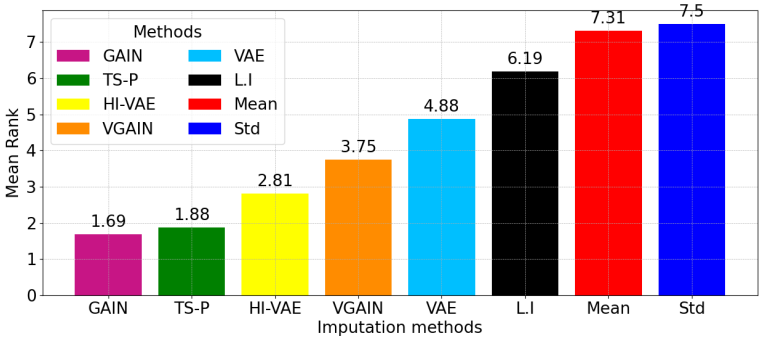
Although the imputation methods used as baselines present a very low execution time (Table 5 and Table 6), they are less efficient than the other imputation methods as shown in Figure 11, Figure 12, Figure 13, Figure 14, Table 3 and Table 10.

To better present these results, we have ranked all imputation methods using the mean rank method. This method involves assigning a rank to each method for each data sample, based on performance: the best-performing method receives rank 1, the next best receives rank 2, and so on. In the event of a tie, the methods receive the mean rank of the positions they occupy. Figure 4, Figure 5 and Figure 6 show the mean RMSE, MAE, and NSE ranking for continuous and random missing data.

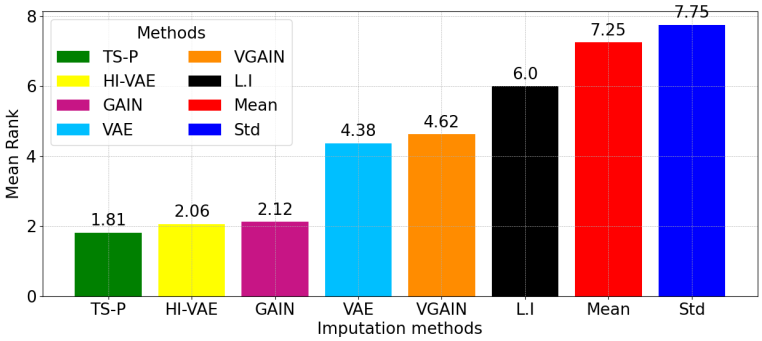
We observe that missing data imputations made by linear interpolation (column L.I), mean (Mean) and standard (Std) deviation perform less well than the other methods. de même, Similarly, in Tables 3 to 10, the error rates of these baseline methods are very high compared to the other methods.

Although imputation by linear interpolation, mean, and standard deviation are generally used, they are not suitable for efficient imputation of missing values in datasets with scenarios similar to those used in our experiments.

In the following, Figure 7 to 10 thus show the application of each imputation method but the baseline ones (*i.e.*, TS-P, GAIN, VGAIN, VAE, and HI-VAE) for each evaluation metric (RMSE, MAE, and NSE) and each dataset described in Section 5.2, with different ratios of missing values and for both continuous and random distributions. Additionally, to facilitate reading, some of the figures and tables have been moved to the Appendices. Each row relates to a missing rate, and each column to an imputation method. The darker and greener, the better (low error rate), and conversely, dark red cells indicate a higher error rate.

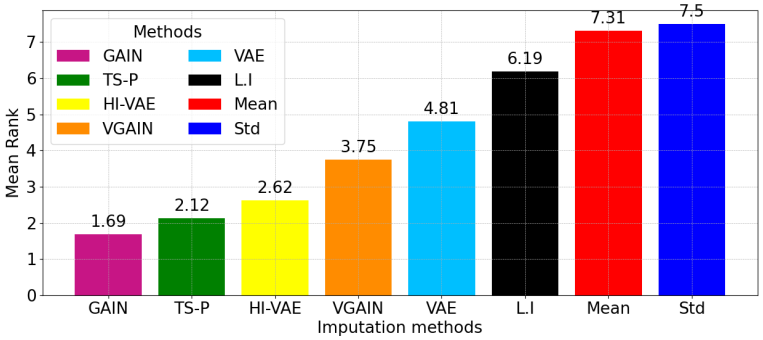


(a) Mean RMSE ranking for continuous missing data

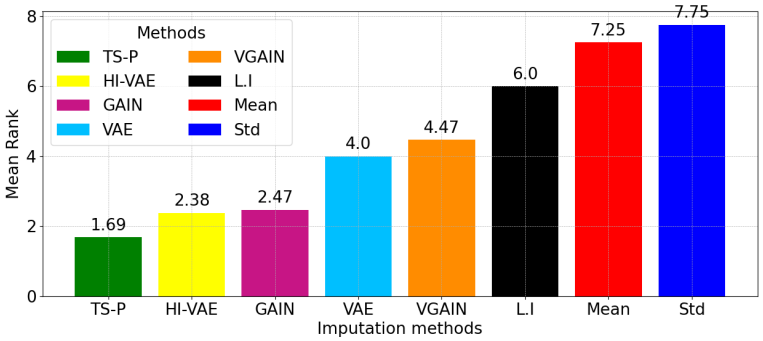


(b) Mean RMSE ranking for random missing data

Fig. 4. Mean RMSE ranking for continuous and random missing data

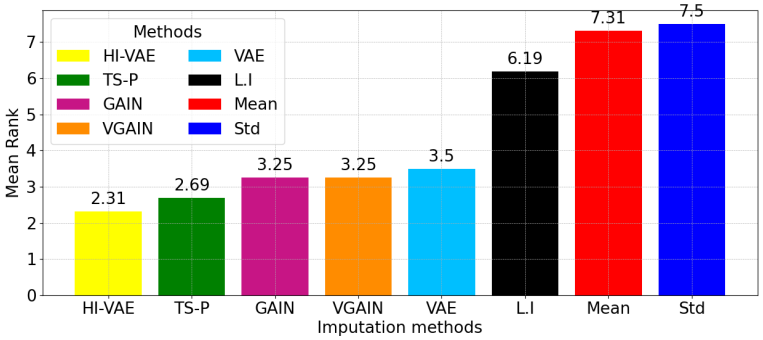


(a) Mean MAE ranking for continuous missing data

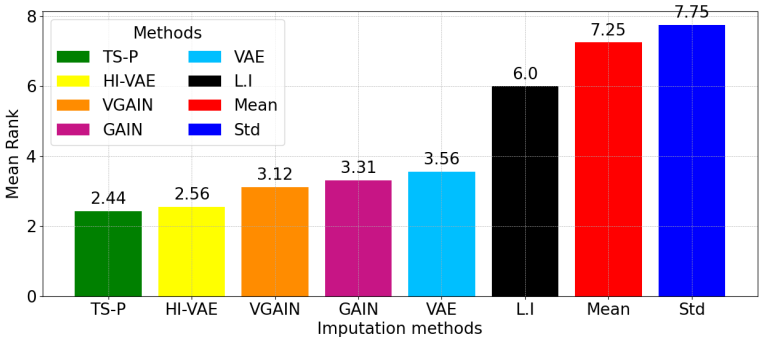


(b) Mean MAE ranking for random missing data

Fig. 5. Mean MAE ranking for continuous and random missing data



(a) Mean NSE ranking for continious missing data



(b) Mean NSE ranking for random missing data

Fig. 6. Mean NSE ranking for continuous and random missing data

Set	Rate	Continuous							
		Mean	**Std**	**L.I**	**TS-P**	**GAIN**	**VGAIN**	**VAE**	**HI-VAE**
Beijing	10%	43.31	44.32	24.027	0.10585	0.109350	0.133650	0.156860	0.128340
	15%	46.72	50.83	24.005	0.150000	0.116550	0.142450	0.243870	0.199530
	20%	48.75	52.03	24.815	0.226500	0.229365	0.280335	0.399685	0.327015
	25%	52.07	54.96	26.74	0.421500	0.441000	0.539000	0.561275	0.459225
F-500	10%	32.870	41.628	28.342	0.20413	0.20561	0.22372	0.2536	0.22073
	15%	38.5301	44.0613	33.4657	0.24585	0.2105	0.23586	0.33828	0.29586
	20%	38.912	43.9008	37.7563	0.32029	0.32189	0.37341	0.49194	0.41965
	25%	41.671	44.71102	37.9472	0.53934	0.51171	0.63665	0.65237	0.55897
F-2M	10%	38.729	33.1271	34.5011	0.6611	0.624	0.8118	0.8359	0.6612
	15%	41.470	38.6003	37.3876	0.6118	0.6837	0.8076	0.8733	0.6496
	20%	41.7999	35.0014	39.3135	0.6667	0.6485	0.8901	0.8908	0.6699
	25%	43.002	35.73399	39.8404	0.6973	0.6666	0.8873	0.924	0.685
Ground.	10%	45.615	53.71	33.6298	0.905602	0.874206	1.068474	1.077731	0.881779
	15%	51.121	55.12	34.7468	0.908360	0.883930	1.080358	1.087955	0.890145
	20%	51.9299	52.90027	34.8774	0.907488	0.933650	1.141128	1.116327	0.913359
	25%	56.001	55.625	37.2109	0.928000	0.950984	1.162314	1.127798	0.922744

Table 3. RMSE for continuous distribution over 100 iterations for all datasets with different rates of missing values.

Set	Rate	Random							
		Mean	**Std**	**L.I**	**TS-P**	**GAIN**	**VGAIN**	**VAE**	**HI-VAE**
Beijing	10%	22.1854	21.4890	15.9271	0.50040	0.69273	0.84667	0.80113	0.65547
	15%	24.6379	21.3142	16.1942	0.51475	0.73458	0.89782	0.90662	0.74178
	20%	23.9821	23.5283	15.7853	0.56325	0.80946	0.98934	0.99726	0.81594
	25%	25.2567	23.6745	17.3618	0.60670	0.89190	1.09010	1.10605	0.90495
F-500	10%	27.3462	317410.	19.4414	0.59303	0.78992	0.94655	0.89696	0.75433
	15%	28.2109	35.6291	18.0305	0.60516	0.82508	0.9933	1.00655	0.83716
	20%	31.8625	33.4728	20.3204	0.91287	0.9065	1.08589	1.09586	0.90974
	25%	31.4756	38.1427	20.2844	0.99515	0.98308	1.1885	1.19794	0.99006
F-2M	10%	28.1254	36.3313	15.6266	0.9083	0.7931	1.0551	0.9953	0.7676
	15%	29.1734	34.7462	15.4583	0.9457	0.8036	1.0907	1.0508	0.8034
	20%	28.1002	39.5127	20.4657	0.9514	0.9446	1.2553	1.222	0.9603
	25%	28.2898	35.9021	20.3862	0.9217	1.0218	1.282	1.2623	0.9834
Ground.	10%	25.8323	38.969	17.2867	1.170750	1.044320	1.276391	1.268825	1.038129
	15%	25.568	36.7121	17.7492	1.177600	1.076355	1.315545	1.290483	1.055849
	20%	26.7812	33.4228	20.0127	1.181451	1.227332	1.500072	1.462126	1.196285
	25%	30.5891	39.1144	21.36	1.184001	1.259971	1.539965	1.529440	1.251360

Table 4. RMSE for random distribution over 100 iterations for all datasets with different percentages of missing values

6.2 Small Univariate Datasets ($\sim 35,000$ samples)

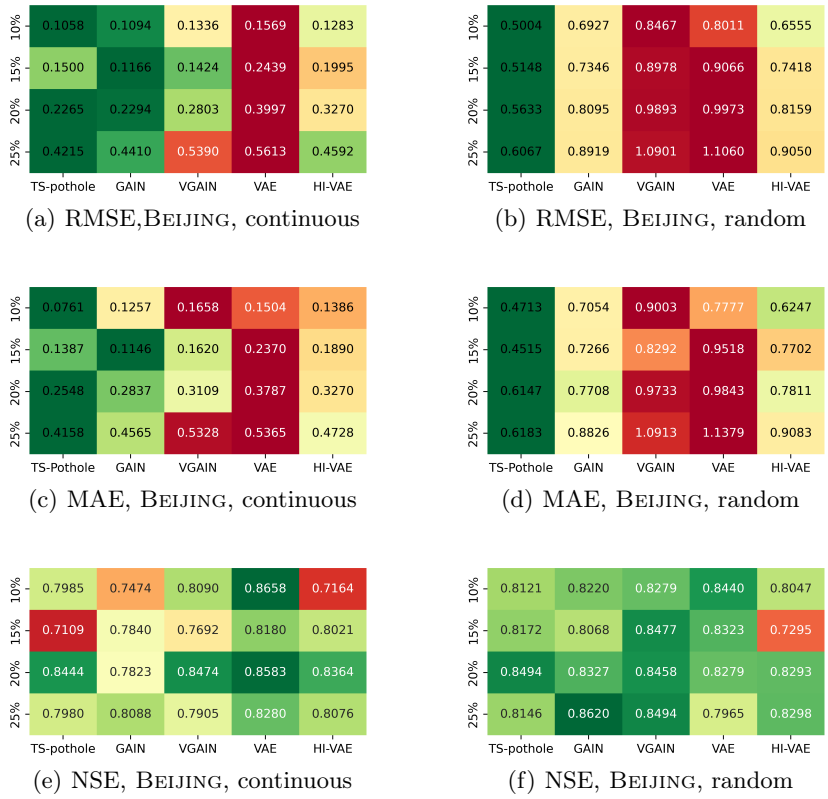
Figure 7 provides an overview of error rate for each imputation method evaluated in this study for the BEIJING dataset, representing a small univariate dataset ($\sim 35,000$ samples). Overall, when missing values are continuously or randomly distributed, TS-POTHOLE outperforms the other methods as presented in Figure 11. For example, TS-POTHOLE achieves a lower MAE and RMSE than competing techniques such as GAIN, V-GAIN, VAE, and HI-VAE in various missing value percentage configurations as shown in Figure 7(a) to 7(d).

TS-POTHOLE thus more accurately estimates missing values while minimizing deviations between predicted and actual values. Regarding *Nash-Sutcliffe Normalized Efficiency* (NSE) in Figure 7(e) and Figure 7(f), although TS-POTHOLE does not always achieve the highest performance, it always maintains scores above 0.70, and in some cases reaches 0.84. These scores demonstrate that TS-POTHOLE is capable of capturing a large proportion of the variance in real data, even in the presence of large missing values. The proximity of these NSE values to the ideal of 1 indicates a very good match between predicted and observed values.

> TS-POTHOLE offers an efficient alternative for imputing continuously or randomly distributed missing data, especially for small datasets ($\sim 35,000$ samples) with up to 25% missing values (Figure 11).

6.3 Medium univariate datasets ($\sim 500,000$ samples)

Figure 8 provides an overview of the error rate for each evaluated imputation method for the FLUXNET500 dataset representing a medium univariate dataset ($\sim 500,000$ samples). Regarding continuously distributed missing values, the heatmaps shown in Fig. 8(a) and 8(c) illustrate that TS-POTHOLE generally underperforms compared to the GAIN method, though it still surpasses VGAIN, VAE, and HI-VAE. However, the difference in performance between GAIN and TS-POTHOLE is not statistically significant given their difference, unlike the one with other methods as shown in Figure 12(a). Regarding the NSE index, Figure 8(e) reveals that values approach the ideal one (1), indicating that TS-POTHOLE achieves an excellent match between predicted and observed values. When missing values are randomly distributed, TS-POTHOLE shows improved performance compared to a continuous distribution (Figure 12(b)). According to Fig. 8(b) and 8(d), TS-POTHOLE outperforms other methods at missing rates of 10% and 15%, with the lowest MAE errors rate and the smallest RMSE errors rate. At missing rates of 20% and 25%, its RMSE remains very close to the best-recorded score. As for the NSE value, Figure 7(f) indicates that values are nearing the ideal (1), also reflecting the accuracy of TS-POTHOLE in predicting observed values.



(a) RMSE,Beijing, continuous (b) RMSE, Beijing, random

(c) MAE, Beijing, continuous (d) MAE, Beijing, random

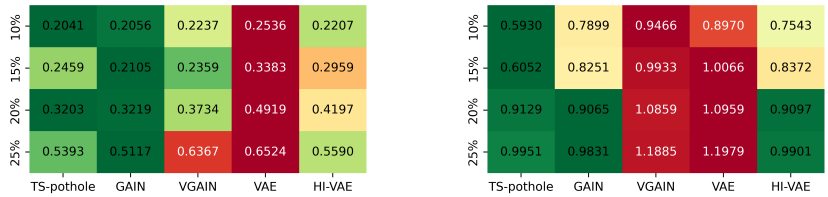
(e) NSE, Beijing, continuous (f) NSE, Beijing, random

Fig. 7. Mean of evaluations metrics for continuous & random distributions over 100 iterations for the Beijing dataset.

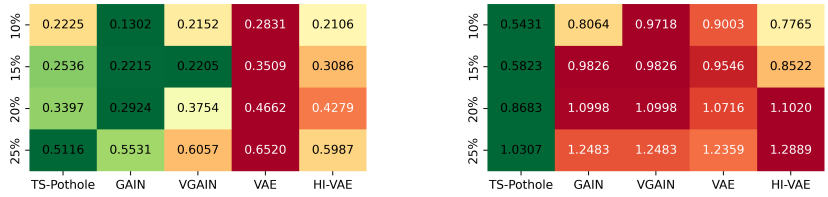
> Although TS-Pothole is only the second-best option for medium-sized datasets ($\sim 500,000$ samples) with continuous distributions, it proves to be the overall best alternative when missing values are randomly distributed.

6.4 Large univariate datasets ($\sim 2,000,000$ samples)

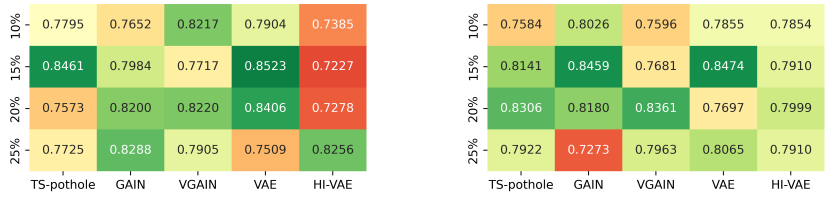
Figure 9 provides an overview of the error rate for each evaluated imputation method for the Fluxnet2M dataset representing a large univariate dataset ($\sim 2,000,000$ samples). When the missing values are continuously distributed, TS-Pothole always comes close to the smallest error, although it does not get the best score. Figure 9(a), 9(c), 13(a) show that TS-Pothole overall gets second or third place, and therefore rarely achieves the lowest error. However, the difference in performance between GAIN and TS-Pothole is not statistically significant, unlike the other methods. As for the NSE index, Figure 9(e) reveals



(a) RMSE,FLUXNET500, continuous (b) RMSE, FLUXNET500, random



(c) MAE, FLUXNET500, continuous (d) MAE, FLUXNET500, random

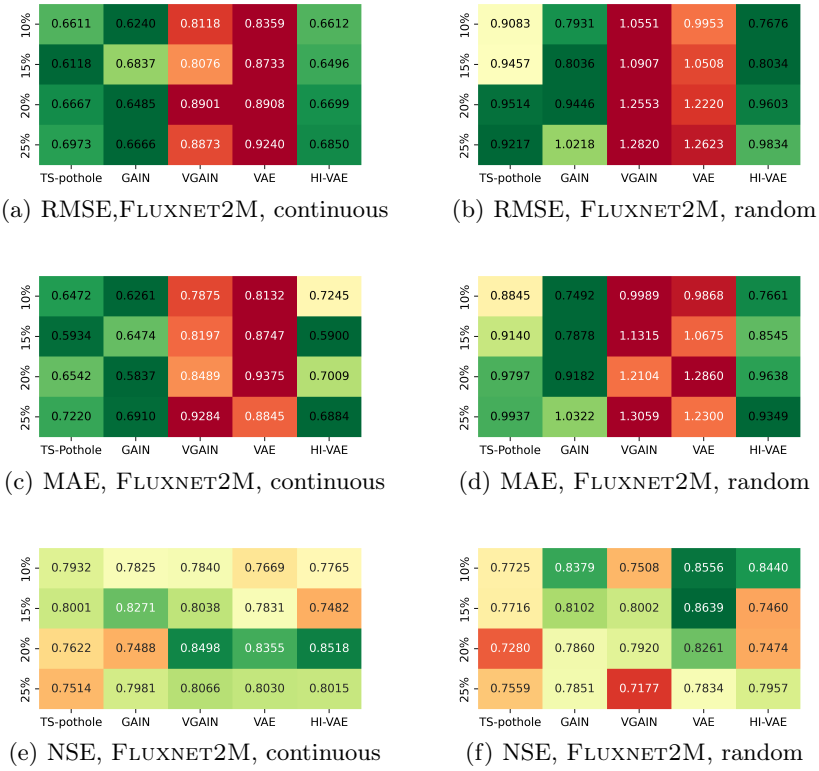


(e) NSE, FLUXNET500, continuous (f) NSE, FLUXNET500, random

Fig. 8. Mean of evaluations metrics for continuous & random distributions over 100 iterations for the FLUXNET500 datasets.

that the values approach the ideal (1), once again indicating that TS-POTHOLE achieves an excellent match between predicted and observed values. When missing values are randomly distributed, the performance of TS-POTHOLE decreases considerably. As illustrated in Figure 9(b), 9(d) and 13(b), TS-POTHOLE does not prove to be the best option in these cases. This observation is corroborated by the NSE index shown in Figure 9(f), which indicates that TS-POTHOLE is often the least efficient of the evaluated alternatives.

> Although TS-POTHOLE is not the most efficient method for imputing missing values in large univariate datasets ($\sim 2,000,000$ samples), it is nevertheless not the worst option and remains superior to the baselines imputation methods considered in previous work as shown in Figure 13.



(a) RMSE,FLUXNET2M, continuous (b) RMSE, FLUXNET2M, random

(c) MAE, FLUXNET2M, continuous (d) MAE, FLUXNET2M, random

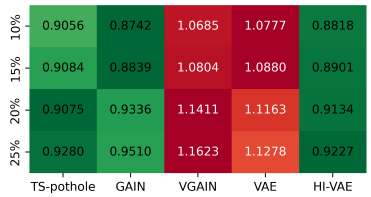
(e) NSE, FLUXNET2M, continuous (f) NSE, FLUXNET2M, random

Fig. 9. Mean of evaluations metrics for continuous & random distributions over 100 iterations for the FLUXNET2M datasets.

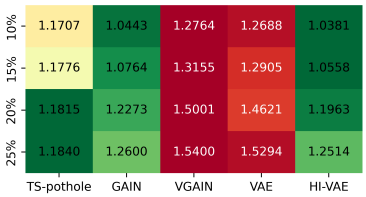
6.5 Very large univariate datasets ($\sim 5,000,000$ samples)

Figure 10 provides an overview of the error rate for each evaluated imputation method, regarding the GROUNDWATER dataset representing a very large univariate dataset ($\sim 5,000,000$ samples). When the distribution of missing values is continuous, the TS-POTHOLE method has the lowest error rate at 20% missing data rate, both in terms of mean absolute error (MAE) and root mean square error (RMSE). For other rates of missing values, TS-POTHOLE approaches the best performance, as shown by Figure 10(a) and 10(c). Regarding the random distribution of missing values, TS-POTHOLE shows the lowest error rate at high missing value rates (20% and 25%), for both MAE and RMSE. At lower missing value rates (10% and 15%), the performance of TS-POTHOLE is less optimal. These results are illustrated in Figure 10(b) and 10(d). Regardless of the mode of distribution of the missing values, the NSE index of TS-POTHOLE remains relatively high, fluctuating between 0.7 and 0.8. This demonstrates that TS-POTHOLE maintains good predictive capability, whatever the conditions of the missing value distribution.

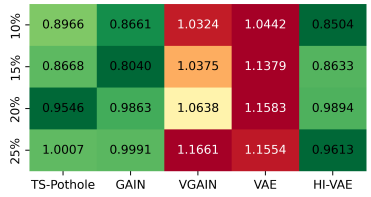
When it comes to very large datasets ($\sim 5,000,000$ samples), TS-Pothole overall exhibits best performances when missing values are continuously distributed (Figure 14(a)). TS-Pothole is particularly well suited to randomly distributed data with a high rate of missing values(Figure 14(b)).



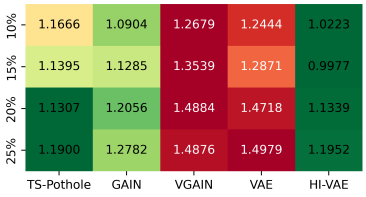
(a) RMSE,Groundwater, continuous



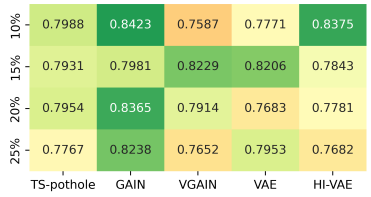
(b) RMSE, Groundwater, random



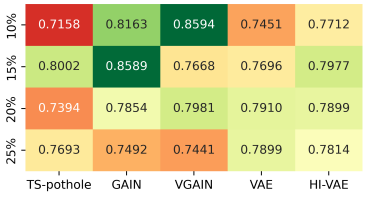
(c) MAE, Groundwater, continuous



(d) MAE, Groundwater, random



(e) NSE, Groundwater, continuous



(f) NSE, Groundwater, random

Fig. 10. Mean of evaluations metrics for continuous & random distributions over 100 iterations for the Groundwater datasets.

6.6 Execution Time

Tables 5 and 6 show the mean imputation time in seconds for the four assessed datasets with varying missing value rates, whether they are datasets with continuous or random missing value distributions.

The methods used as baselines exhibit very low execution times. For example, for continuous missing values, imputation by the mean method is around 27 times faster than TS-Pothole. And when these values are randomly distributed, mean imputation runs around 135 times faster than TS-Pothole. As presented in section 6.1, these baseline methods, although very fast, make significant errors in the imputation process, showing a large difference between the actual and predicted values. Regarding other methods (TS-Pothole, GAIN, VGAIN, VAE, HI-VAE), the execution time is admittedly higher, but we saw that the error rate is lower. Cell highlighted in green in Tables 5 and 6 indicate that TS-Pothole has the best execution time among these methods, while red-colored cells indicate that another method executes faster.

When missing values are continuously distributed, TS-Pothole has slower execution times for small ($\sim 35,000$ samples), medium ($\sim 500,000$ samples), and even large ($\sim 2,000,000$ samples) datasets. However, for very large datasets ($\sim 5,000,000$ samples), TS-Pothole is more than twice as fast as the other methods, regardless of the rate of missing values. When missing values are randomly distributed, TS-Pothole becomes faster for small datasets ($\sim 35,000$ samples). On the Beijing dataset, it demonstrates an execution speed nearly 1.5 times faster than other methods.

Set	Rate	Continuous							
		Mean	**Std**	**L.I**	**TS-P**	**GAIN**	**VGAIN**	**VAE**	**HI-VAE**
Beijing	10%	41	52	110	485	460	470	445	455
	15%	65	75	160	510	490	455	470	440
	20%	95	100	230	530	485	450	495	465
	25%	130	150	280	740	570	545	660	635
F-500	10%	64	71	130	1,630	1,390	1,648	1,312	1,156
	15%	80	86	165	1,810	1,486	1,774	1,697	1,404
	20%	93	92	177	1,930	1,990	1,918	2,024	1,649
	25%	143	156	196	2,410	2,350	2,597	2,089	1,810
F-2M	10%	87	93	121	4,180	3,854	3,745	2,902	2,765
	15%	123	157	235	4,192	3,891	4,138	2,945	4,811
	20%	154	159	289	4,337	4,454	4,848	4,244	4,958
	25%	158	157	333	4,641	5,118	5,312	5,209	5,059
Ground.	10%	158	149	238	10,915	23,612	23,542	16,958	16,932
	15%	247	260	340	12,545	24,188	24,190	18,432	18,499
	20%	348	360	430	12,778	24,768	24,753	20,545	20,492
	25%	570	560	742	13,710	26,230	26,218	22,667	22,680

Table 5. Mean execution time (in seconds) for a continuous distribution over 100 iterations for all datasets with all methods at different percentage of missing values.

Set	Rate	Random							
		Mean	**Std**	**L.I**	**TS-P**	**GAIN**	**VGAIN**	**VAE**	**HI-VAE**
Beijing	10%	44	50	125	1,308	3,690	3,701	3,047	3,049
	15%	67	73	192	2,290	3,965	3,978	3,628	3,604
	20%	102	94	235	3,030	4,289	4,313	4,077	4,092
	25%	135	145	288	4,021	4,404	4,426	4,346	4,327
F-500	10%	68	66	137	4,145	5,221	3,611	6,252	5,168
	15%	85	88	159	6,920	6,013	4,151	7,074	7,841
	20%	97	95	181	7,873	7,676	6,965	7,442	8,725
	25%	147	153	192	8,053	8,587	8,278	8,355	10,735
F-2M	10%	90	85	124	29,118	28,255	27,968	25,717	25,339
	15%	119	150	229	29,159	28,345	29,002	25,833	30,809
	20%	149	165	294	29,548	29,845	30,905	29,297	31,189
	25%	153	163	328	30,357	31,614	32,130	31,872	31,470
Ground.	10%	163	155	234	48,138	40,801	40,820	47,474	47,479
	15%	250	253	345	53,448	42,154	42,136	48,879	48,911
	20%	355	355	428	56,838	47,068	47,086	49,630	49,648
	25%	575	553	748	75,610	52,662	52,675	58,518	58,530

Table 6. Mean execution time (in seconds) for a random distribution over 100 iterations for all datasets with all methods at different percentages of missing values.

Overall, TS-Pothole stands out as an efficient method for imputing data in univariate datasets with randomly or continuously distributed missing values.

- Small datasets ($\sim 35,000$ samples): TS-Pothole performs particularly well for imputing continuously or randomly distributed missing data, up to 25% of missing values. It is also nearly 1.5 times faster than other methods for randomly distributed data.

- Medium datasets ($\sim 500,000$ samples): Although TS-Pothole is not the fastest method, it remains close to the mean in terms of execution time. Regarding imputation performance, TS-Pothole ranks second for sets with continuously distributed missing values but is the best option for those whose missing values follow a random distribution.

- Large datasets ($\sim 2,000,000$ samples): TS-Pothole is not the most efficient method in terms of accuracy and execution time. Not being the worst method in this case, TS-Pothole is close to the mean error and mean execution time, and remains superior to the baseline methods.

- Very large datasets ($\sim 5,000,000$ samples): TS-Pothole achieves optimal performance with very large sets, especially when missing values are continuously distributed. Moreover, TS-Pothole proves to be almost twice as fast as conventional methods.

7 Discussion

The results of our study demonstrate that TS-Pothole is an effective method for data imputation in univariate datasets with randomly or continuously distributed missing values exhibiting little or no trend and seasonality. TS-Pothole outperforms other methods, with a generally low error rate and fast execution time on real-world datasets. As a result, it stands out as a better-performing alternative in certain contexts compared with well-known methods such as GAIN and VGAIN, based on generative adversarial networks (GANs), and VAE and HI-VAE, based on autoencoders. This performance can be attributed to TS-Pothole's ability to fully leverage the temporal structure of univariate data through linear regression optimized by the optimal number of lags. By contrast with methods based on GANs and autoencoders, which can introduce biases and distortions by attempting to capture complex relationships via neural networks, TS-Pothole focuses on linear relationships and precise temporal lags, enabling the statistical and dynamic properties of time series to be better preserved. Our study shows that it is crucial to choose TS-Pothole for data with specific characteristics, such as low seasonality and low trend. In addition, TS-Pothole simplifies the imputation process for practitioners by reducing the need for complex configurations required by GAN-based models and autoencoders, making TS-Pothole more accessible to analysts without deep expertise in artificial intelligence. This simplification also means faster implementation and reduced costs associated with training and fine-tuning sophisticated models. In addition, TS-Pothole is less sensitive to overfitting and the need for large amounts of training data, a major challenge for GAN and autoencoder-based methods.

Despite the promising results, our study has some limitations that need to be taken into consideration. First of all, our evaluation focused only on univariate datasets. It would be interesting to examine the performance of TS-Pothole in multivariate contexts to assess its generalizability. Furthermore, although TS-Pothole showed superior performance in our experiments, this method relies strongly on the assumption of linearity and optimal lag, which may not apply to any time series, especially those exhibiting complex non-linear behaviors.

For example, we could take advantage of the multi-level feature de-interleaving method mentioned by Chen et al. [47], which could be adapted to improve the understanding of complex patterns in univariate time series. Similarly, the two-stream convolutional LSTM technique described in [48] could offer valuable insights for developing imputation models that account for dynamic variations within time series.

Another limitation is that our study did not exhaustively explore the impact of the configuration parameters of the other methods compared (GAIN, VGAIN, VAE, HI-VAE), which could potentially influence the results. We applied the TS-Pothole method to four real datasets of varying sizes, which may not represent the diversity and complexity of all possible time series. Thus, results could differ for datasets with different characteristics. In addition, although we used standard metrics such as MAE, RMSE, and NSE to assess the performance of imputation methods, these metrics may not capture all aspects of imputation

quality. Finally, although TS-Pothole is relatively simple to implement, further research is needed to develop practical guides and software tools to facilitate its adoption. For large-scale applications running in resource-constrained environments, it is essential to ensure that the model can operate efficiently without requiring excessive computational resources. Optimizing memory usage and improving scalability will be key aspects to address in future research to make TS-Pothole even more applicable.

8 Conclusion

In this article, we introduced TS-Pothole, an automated approach for imputing missing data in univariate time series. TS-Pothole is based on the analysis of cyclical patterns and relies on a recursive strategy to impute missing values efficiently and accurately. TS-Pothole first seeks the longest sequence of continuous data, which then serves as the basis for the imputation of missing data before (backward imputation) and after (forward imputation) that sequence. By assessing TS-Pothole on four real-world data sets of varying sizes, we have demonstrated its applicability to real-life scenarios, whether the missing values are randomly or continuously distributed in the dataset. Our study also revealed that, although TS-Pothole can occasionally be slightly less rapid and precise than other methods in rare cases, it generally proves to be the most effective alternative in terms of precision and execution time, regardless of the dataset size. In certain situations, TS-Pothole has demonstrated the ability to achieve twice the precision in a reduced time or to execute twice as fast. This performance attests to the competitiveness of TS-Pothole in terms of precision, even in the case of large datasets, thereby highlighting its versatility and adaptability, making it an invaluable tool for a wide range of data processing applications.

For future work, we plan to improve TS-Pothole, especially regarding how the longest data sequence is handled and the lag information is identified when the sequence size is reduced. This may involve optimizing current algorithms, using data augmentation methods, generating synthetic data, or creating new techniques to efficiently manage large volumes of temporal data. Finally, we will extend TS-Pothole for multivariate data imputation. This task will involve adapting the algorithm to handle several variables simultaneously and testing its effectiveness in more complex scenarios.

Data Availability

The data that support the findings of this study are publicly available from `https://zenodo.org/records/10518265`.

Conflict of Interests/Competing Interests

The authors have no competing interests to declare that are relevant to the content of this article.

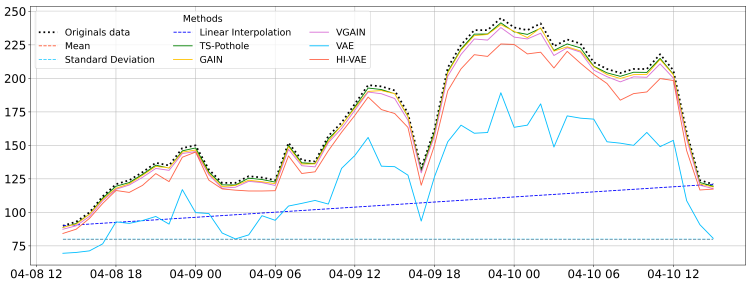
References

1. R. Wu, S. D. Hamshaw, L. Yang, D. W. Kincaid, R. Etheridge, and A. Ghasemkhani, "Data imputation for multivariate time series sensor data with large gaps of missing data," *IEEE Sensors Journal*, vol. 22, no. 11, pp. 10 671–10 683, 2022.
2. A. Zainuddin, M. A. Hairuddin, A. I. M. Yassin, Z. I. Abd Latiff, and A. Azhar, "Time series data and recent imputation techniques for missing data: A review," in *2022 International Conference on Green Energy, Computing and Sustainable Technology (GECOST)*. IEEE, 2022, pp. 346–350.
3. R. Garg, S. Barpanda *et al.*, "Machine learning algorithms for time series analysis and forecasting," *arXiv preprint arXiv:2211.14387*, 2022.
4. X. Miao, Y. Wu, L. Chen, Y. Gao, and J. Yin, "An experimental survey of missing data imputation algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 7, pp. 6630–6650, 2023.
5. J. T. McCoy, S. Kroon, and L. Auret, "Variational autoencoders for missing data imputation with application to a simulated milling circuit," *IFAC-PapersOnLine*, vol. 51, no. 21, pp. 141–146, 2018.
6. M. J. Kusner, B. Paige, and J. M. Hernández-Lobato, "Grammar variational autoencoder," in *International conference on machine learning*. PMLR, 2017, pp. 1945–1954.
7. A. Nazabal, P. M. Olmos, Z. Ghahramani, and I. Valera, "Handling incomplete heterogeneous data using vaes," *Pattern Recognition*, vol. 107, p. 107501, 2020.
8. V. Simkus and M. U. Gutmann, "Conditional sampling of variational autoencoders via iterated approximate ancestral sampling," *arXiv preprint arXiv:2308.09078*, 2023.
9. I. Peis, C. Ma, and J. M. Hernández-Lobato, "Missing data imputation and acquisition with deep hierarchical models and hamiltonian monte carlo," *Advances in Neural Information Processing Systems*, vol. 35, pp. 35 839–35 851, 2022.
10. S. Kumar, P. Payne, and A. Sotiras, "Improving normative modeling for multimodal neuroimaging data using mixture-of-product-of-experts variational autoencoders," *arXiv preprint arXiv:2312.00992*, 2023.
11. R. C. Pereira, P. H. Abreu, and P. P. Rodrigues, "Partial multiple imputation with variational autoencoders: Tackling not at randomness in healthcare data," *IEEE Journal of Biomedical and Health Informatics*, vol. 26, no. 8, pp. 4218–4227, 2022.
12. J. Yoon, J. Jordon, and M. Schaar, "Gain: Missing data imputation using generative adversarial nets," in *International conference on machine learning*. PMLR, 2018, pp. 5689–5698.
13. D. B. Rubin, "Inference and missing data," *ETS Research Bulletin Series*, vol. 1975, no. 1, pp. i–19, 1975.
14. T. Emmanuel, T. Maupong, D. Mpoeleng, T. Semong, B. Mphago, and O. Tabona, "A survey on missing data in machine learning," *Journal of Big Data*, vol. 8, no. 1, pp. 1–37, 2021.
15. S. Wu, L. Wang, T. Wu, X. Tao, and J. Lu, "Hankel matrix factorization for tagged time series to recover missing values during blackouts," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, 2019, pp. 1654–1657.
16. X. Xu, J. Wang, X. Xu, Y. Sun, Q. Chen, X. Li, and G. Xie, "Estimating missing values in multivariate-time-series clinical data using gradient boosting tree on temporal and cross-variable features," in *2019 IEEE International Conference on Healthcare Informatics (ICHI)*, 2019, pp. 1–3.

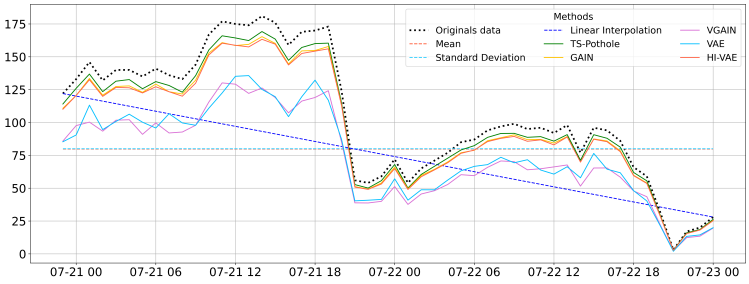
17. P. B. Weerakody, K. W. Wong, and G. Wang, "Cyclic gate recurrent neural networks for time series data with missing values," *Neural Processing Letters*, vol. 55, pp. 1527–1554, 2023. [Online]. Available: https://doi.org/10.1007/s11063-022-10950-2
18. M. Kazijevs and M. D. Samad, "Deep imputation of missing values in time series health data: A review with benchmarking," *arXiv preprint arXiv:2302.10902*, 2023.
19. E. Acuna and C. Rodriguez, "The treatment of missing values and its effect on classifier accuracy," in *Classification, Clustering, and Data Mining Applications: Proceedings of the Meeting of the International Federation of Classification Societies (IFCS), Illinois Institute of Technology, Chicago, 15–18 July 2004*. Springer, 2004, pp. 639–647.
20. G. Chhabra, V. Vashisht, and J. Ranjan, "A comparison of multiple imputation methods for data with missing values," *Indian Journal of Science and Technology*, vol. 10, no. 19, pp. 1–7, 2017.
21. W. M. Hameed and N. A. Ali, "Missing value imputation techniques: A survey," *UHD Journal of Science and Technology*, vol. 7, no. 1, p. 72–81, Mar. 2023. [Online]. Available: https://journals.uhd.edu.iq/index.php/uhdjst/article/view/1086
22. P. Vateekul and K. Sarinnapakorn, "Tree-based approach to missing data imputation," in *2009 IEEE International Conference on Data Mining Workshops*. IEEE, 2009, pp. 70–75.
23. M. G. Rahman and M. Z. Islam, "A decision tree-based missing value imputation technique for data pre-processing," in *The 9th Australasian Data Mining Conference: AusDM 2011*. Australian Computer Society Inc, 2011, pp. 41–50.
24. L. Breiman, J. Friedman, R. Olshen, and C. Stone, "Classification and regression t rees (monterey, california: Wadsworth)," 1984.
25. G. James, D. Witten, T. Hastie, R. Tibshirani, and J. Taylor, "Tree-based methods," in *An Introduction to Statistical Learning: with Applications in Python*. Springer, 2023, pp. 331–366.
26. T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.
27. D. J. Stekhoven and P. Bühlmann, "Missforest—non-parametric missing value imputation for mixed-type data," *Bioinformatics*, vol. 28, no. 1, pp. 112–118, 2012.
28. M. Pal, "Random forest classifier for remote sensing classification," *International journal of remote sensing*, vol. 26, no. 1, pp. 217–222, 2005.
29. P. Royston and I. R. White, "Multiple imputation by chained equations (mice): implementation in stata," *Journal of statistical software*, vol. 45, pp. 1–20, 2011.
30. D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, 1999.
31. R. Mazumder, T. Hastie, and R. Tibshirani, "Spectral regularization algorithms for learning large incomplete matrices," *The Journal of Machine Learning Research*, vol. 11, pp. 2287–2322, 2010.
32. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," *Advances in neural information processing systems*, vol. 13, 2000.
33. J. Josse, J. Pagès, and F. Husson, "Multiple imputation in principal component analysis," *Advances in data analysis and classification*, vol. 5, pp. 231–246, 2011.
34. S. Mitra and S. K. Pal, "Fuzzy multi-layer perceptron, inferencing and rule generation," *IEEE Transactions on Neural Networks*, vol. 6, no. 1, pp. 51–63, 1995.
35. P. J. García-Laencina, J.-L. Sancho-Gómez, and A. R. Figueiras-Vidal, "Pattern classification with missing data: a review," *Neural Computing and Applications*, vol. 19, pp. 263–282, 2010.

36. P. Golik, P. Doetsch, and H. Ney, "Cross-entropy vs. squared error training: a theoretical and experimental comparison." in *Interspeech*, vol. 13, 2013, pp. 1756–1760.
37. M. A. Kramer, "Nonlinear principal component analysis using autoassociative neural networks," *AIChE journal*, vol. 37, no. 2, pp. 233–243, 1991.
38. I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.
39. W. Li, L. Fan, Z. Wang, C. Ma, and X. Cui, "Tackling mode collapse in multi-generator gans with orthogonal vectors," *Pattern Recognition*, vol. 110, p. 107646, 2021.
40. Y. Li, A. Dogan, and C. Liu, "Ensemble generative adversarial imputation network with selective multi-generator (esm-gain) for missing data imputation," in *2022 IEEE 18th International Conference on Automation Science and Engineering (CASE)*, 2022, pp. 807–812.
41. F. L. Ramsey, "Characterization of the partial autocorrelation function," *The Annals of Statistics*, vol. 2, no. 6, pp. 1296–1301, 1974. [Online]. Available: http://www.jstor.org/stable/2958346
42. Y.-H. Xue, R. Chen, J.-G. Wang, W. Liu, Y. Yao, J.-L. Liu, and H.-L. Chen, "Granger-based root cause diagnosis with improved backward-in-time selection," in *2023 IEEE 12th Data Driven Control and Learning Systems Conference (DDCLS)*, 2023, pp. 1853–1858.
43. A. Gasparrini, "Modeling exposure–lag–response associations with distributed lag non-linear models," *Statistics in medicine*, vol. 33, no. 5, pp. 881–899, 2014.
44. E. M. Herrey, "Confidence intervals based on the mean absolute deviation of a normal sample," *Journal of the American Statistical Association*, vol. 60, no. 309, pp. 257–269, 1965.
45. S. Chen, "Beijing Multi-Site Air-Quality Data," UCI Machine Learning Repository, 2019, DOI: https://doi.org/10.24432/C5RK5G.
46. J. Park, J. Müller, B. Arora, B. Faybishenko, G. Pastorello, C. Varadharajan, R. Sahu, and D. Agarwal, "Long-term missing value imputation for time series data using deep neural networks," *Neural Computing and Applications*, vol. 35, no. 12, pp. 9071–9091, 2023.
47. S. Chen, Y. Bo, and X. Wu, "A spatiotemporal motion prediction network based on multi-level feature disentanglement," *Image and Vision Computing*, vol. 146, p. 105005, 2024.
48. S. Chen, X. Xu, Y. Zhang, D. Shao, S. Zhang, and M. Zeng, "Two-stream convolutional lstm for precipitation nowcasting," *Neural Computing and Applications*, vol. 34, no. 16, pp. 13 281–13 290, 2022.

A Appendix

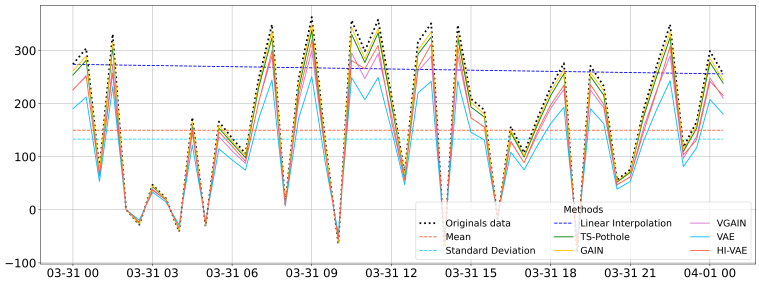


(a) Imputations of continuous missing values, Beijing Dataset

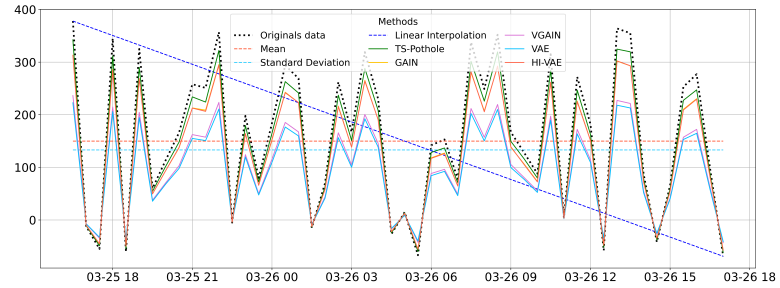


(b) Imputations of random missing values, Beijing Dataset

Fig. 11. Imputation of Beijing Air Quality Dataset

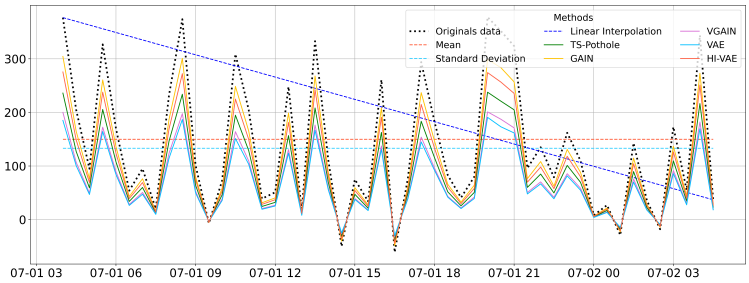


(a) Imputations of continuous missing values, Fluxnet500 Dataset

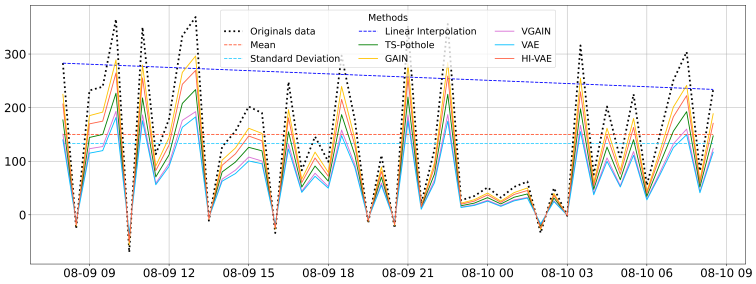


(b) Imputations of random missing values, Fluxnet500 Dataset

Fig. 12. Imputations of Fluxnet500 Dataset

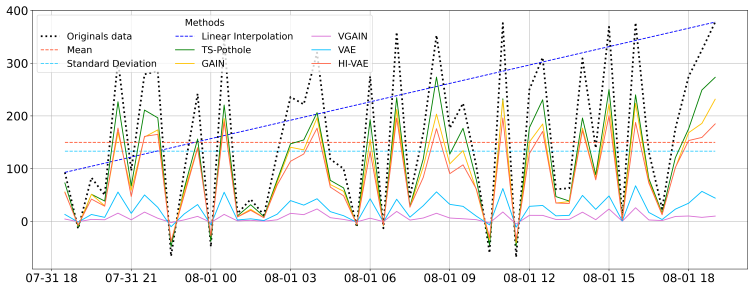


(a) Imputations of continuous missing values, Fluxnet2M Dataset

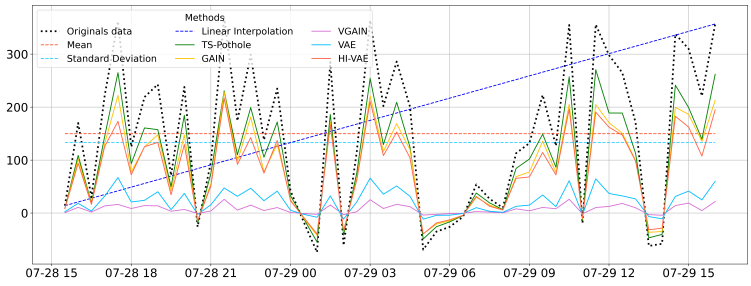


(b) Imputations of random missing values, Fluxnet2M Dataset

Fig. 13. Imputations of Fluxnet2M Dataset



(a) Imputations of continuous missing values, Groundwater Dataset



(b) Imputations of continuous missing values, Groundwater Dataset

Fig. 14. Plot of imputation methods apply to different datasets

Set	Rate	Continuous							
		Mean	**Std**	**L.I**	**TS-P**	**GAIN**	**VGAIN**	**VAE**	**HI-VAE**
Beijing	10%	43.31	44.32	24.027	0.0761	0.1257	0.1658	0.1504	0.1386
	15%	46.72	50.83	24.005	0.1387	0.1146	0.162	0.237	0.189
	20%	48.75	52.03	24.815	0.2548	0.2837	0.3109	0.3787	0.327
	25%	52.07	54.96	26.74	0.4158	0.4565	0.5328	0.5365	0.4728
F-500	10%	32.870	41.628	28.342	0.2225	0.1302	0.2152	0.2831	0.2106
	15%	38.5301	44.0613	33.4657	0.2536	0.2215	0.2205	0.3509	0.3086
	20%	38.912	43.9008	37.7563	0.3397	0.2924	0.3754	0.4662	0.4279
	25%	41.671	44.71102	37.9472	0.5116	0.5531	0.6057	0.6520	0.5987
F-2M	10%	38.729	33.1271	34.5011	0.6472	0.6261	0.7875	0.8132	0.7245
	15%	41.470	38.6003	33.378	0.5934	0.6474	0.8197	0.8747	0.5900
	20%	41.7999	35.0014	39.3135	0.6542	0.5837	0.8489	0.9375	0.7009
	25%	43.002	35.73399	39.8404	0.722	0.691	0.9284	0.8845	0.6884
Ground.	10%	45.615	53.71	33.6298	0.8966	0.8661	1.0324	1.0442	0.8504
	15%	51.121	55.12	34.7468	0.8668	0.804	1.0375	1.1379	0.8633
	20%	51.9299	52.90027	34.8774	0.9546	0.9863	1.0638	1.1583	0.9894
	25%	56.001	55.625	37.2109	1.0007	0.9991	1.1661	1.1554	0.9613

Table 7. MAE for continuous distribution over 100 iterations for all datasets with different percentages of missing values

Set	Rate	Random							
		Mean	**Std**	**L.I**	**TS-P**	**GAIN**	**VGAIN**	**VAE**	**HI-VAE**
Beijing	10%	22.1854	21.4890	15.9271	0.4713	0.7054	0.9003	0.7777	0.6247
	15%	24.6379	21.3142	16.1942	0.4515	0.7266	0.8292	0.9518	0.7702
	20%	23.9821	23.5283	15.7853	0.6147	0.7708	0.9733	0.9843	0.7811
	25%	25.2567	23.6745	17.3618	0.6183	0.8826	1.0913	1.1379	0.9083
F-500	10%	27.3462	317410.	19.4414	0.5431	0.8064	0.9718	0.9003	0.7765
	15%	28.2109	35.6291	18.0305	0.5823	0.9826	0.9826	0.9546	0.8522
	20%	31.8625	33.4728	20.3204	0.8683	1.0998	1.0998	1.0716	1.102
	25%	31.4756	38.1427	20.2844	1.0307	1.2483	1.2483	1.2359	1.2889
F-2M	10%	28.1254	36.3313	15.6266	0.8845	0.7492	0.9989	0.9868	0.7661
	15%	29.1734	34.7462	15.4583	0.914	0.7878	1.1315	1.0675	0.8545
	20%	28.1002	39.5127	20.4657	0.9797	0.9182	1.2104	1.286	0.9638
	25%	28.2898	35.9021	20.3862	0.9937	1.0322	1.3059	1.23	0.9349
Ground.	10%	25.8323	38.969	17.2867	1.1666	1.0904	1.2679	1.2444	1.0223
	15%	25.568	36.7121	17.7492	1.1395	1.1285	1.3539	1.2871	0.9977
	20%	26.7812	33.4228	20.0127	1.1307	1.2056	1.4884	1.4718	1.1339
	25%	30.5891	39.1144	21.36	1.1900	1.2782	1.4876	1.4979	1.1952

Table 8. MAE for random distribution over 100 iterations for all datasets with different percentages of missing values

Set	Rate	Continuous							
		Mean	**Std**	**L.I**	**TS-P**	**GAIN**	**VGAIN**	**VAE**	**HI-VAE**
Beijing	10%	43.31	44.32	24.027	0.7985	0.7474	0.8090	0.8658	0.7164
	15%	46.72	50.83	24.005	0.7109	0.7840	0.7692	0.8180	0.8021
	20%	48.75	52.03	24.815	0.8444	0.7823	0.8474	0.8583	0.8364
	25%	52.07	54.96	26.74	0.7980	0.8088	0.7905	0.8280	0.8076
F-500	10%	32.870	41.628	28.342	0.7795	0.7652	0.8217	0.7904	0.7385
	15%	38.5301	44.0613	33.4657	0.8461	0.7984	0.7717	0.8523	0.7227
	20%	38.912	43.9008	37.7563	0.7573	0.8200	0.8220	0.8406	0.7278
	25%	41.671	44.71102	37.9472	0.7725	0.8288	0.7905	0.7509	0.8256
F-2M	10%	38.729	33.1271	34.5011	0.7932	0.7825	0.784	0.7669	0.7765
	15%	41.470	38.6003	36.3102	0.8001	0.8271	0.8038	0.7831	0.7482
	20%	41.7999	35.0014	39.3135	0.7622	0.7488	0.8498	0.8355	0.8518
	25%	43.002	35.73399	39.8404	0.7514	0.7981	0.8066	0.803	0.8015
Ground.	10%	45.615	53.71	33.6298	0.7988	0.8423	0.7587	0.7771	0.8375
	15%	51.121	55.12	34.7468	0.7931	0.7981	0.8229	0.8206	0.7843
	20%	51.9299	52.90027	34.8774	0.7954	0.8365	0.7914	0.7683	0.7781
	25%	56.001	55.625	37.2109	0.7767	0.8238	0.7652	0.7953	0.7682

Table 9. NSE for continuous distribution over 100 iterations for all datasets with different percentages of missing values

Set	Rate	Random							
		Mean	**Std**	**L.I**	**TS-P**	**GAIN**	**VGAIN**	**VAE**	**HI-VAE**
Beijing	10%	22.1854	21.4890	15.9271	0.8121	0.822	0.8279	0.844	0.8047
	15%	24.6379	21.3142	16.1942	0.8172	0.8068	0.8477	0.8323	0.7295
	20%	23.9821	23.5283	15.7853	0.8494	0.8327	0.8458	0.8279	0.8293
	25%	25.2567	23.6745	17.3618	0.8146	0.862	0.8494	0.7965	0.8298
F-500	10%	27.3462	31.7410	19.4414	0.7584	0.8026	0.7596	0.7855	0.7854
	15%	28.2109	35.6291	18.0305	0.8141	0.8459	0.7681	0.8474	0.791
	20%	31.8625	33.4728	20.3204	0.8306	0.818	0.8361	0.7697	0.7999
	25%	31.4756	38.1427	20.2844	0.7922	0.7273	0.7963	0.8065	0.791
F-2M	10%	28.1254	36.3313	15.6266	0.7725	0.8379	0.7508	0.8556	0.844
	15%	29.1734	34.7462	15.4583	0.7716	0.8102	0.8002	0.8639	0.746
	20%	28.1002	39.5127	20.4657	0.728	0.786	0.792	0.8261	0.7474
	25%	28.2898	35.9021	20.3862	0.7559	0.7851	0.7177	0.7834	0.7957
Ground.	10%	25.8323	38.969	17.2867	0.7158	0.8163	0.8594	0.7451	0.7712
	15%	25.568	36.7121	17.7492	0.8002	0.8589	0.7668	0.7696	0.7977
	20%	26.7812	33.4228	20.0127	0.7394	0.7854	0.7981	0.7910	0.7899
	25%	30.5891	39.1144	21.36	0.7693	0.7492	0.7441	0.7899	0.7814

Table 10. NSE for random distribution over 100 iterations for all datasets with different percentages of missing values