Missing Value Imputation for Multi-attribute Sensor Data Streams via Message Propagation

Xiao Li
Roskilde University, Denmark
xiaol@ruc.dk

Huan Li
Zhejiang University, China
lihuan.cs@zju.edu.cn

Hua Lu†
Roskilde University, Denmark
luhua@ruc.dk

Christian S. Jensen
Aalborg University, Denmark
csj@cs.aau.dk

Varun Pandey
TU Berlin, Germany
varun.pandey@tu-berlin.de

Volker Markl
TU Berlin, BIFOLD, Germany
volker.markl@tu-berlin.de

ABSTRACT

Sensor data streams occur widely in various real-time applications in the context of the Internet of Things (IoT). However, sensor data streams feature missing values due to factors such as sensor failures, communication errors, or depleted batteries. Missing values can compromise the quality of real-time analytics tasks and downstream applications. Existing imputation methods either make strong assumptions about streams or have low efficiency. In this study, we aim to accurately and efficiently impute missing values in data streams that satisfy only general characteristics in order to benefit real-time applications more widely. First, we propose a message propagation imputation network (MPIN) that is able to recover the missing values of data instances in a time window. We give a theoretical analysis of why MPIN is effective. Second, we present a continuous imputation framework that consists of data update and model update mechanisms to enable MPIN to perform continuous imputation both effectively and efficiently. Extensive experiments on multiple real datasets show that MPIN can outperform the existing data imputers by wide margins and that the continuous imputation framework is efficient and accurate.

PVLDB Reference Format:
Xiao Li, Huan Li, Hua Lu, Christian S. Jensen, Varun Pandey, and Volker Markl. Missing Value Imputation for Multi-attribute Sensor Data Streams via Message Propagation. PVLDB, 17(3): 345-358, 2023.
doi:10.14778/3632093.3632100

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/XLI-2020/MPIN.

1 INTRODUCTION

With the increasing deployment of the Internet of Things (IoT), *multi-attribute* sensor data streams (also known as multi-attribute time series) are found in numerous application domains such as medical services [11, 47], meteorology [13, 39] and transportation [4, 33]. For example, in a hospital Intensive Care Unit (ICU), medical professionals may need to continuously monitor patients' health status through a system that tracks vital signs such as heart rate, blood pressure, and body temperature. As another example, it is imperative for an air quality monitoring system to reliably track metrics like PM2.5 and SO_2 at multiple locations in a city. Such systems produce continuous and unbounded streams of data instances. Each data instance in turn is characterized by a vector of attribute values, as illustrated in Figure 1. In the context of the ICU scenario, a data instance is a vector of health-related values such as heart rate, blood pressure, and body temperature. These values typically capture information about the condition of a patient using multiple sensors at a given point in time.

In real-world systems, however, data streams may contain missing values in their instances due to factors such as sensor failures, depleted batteries, and communication errors [5, 36, 46]. All such factors can result in data instances with missing values (see observed and missing attributes in Figure 1). Despite the fact that a

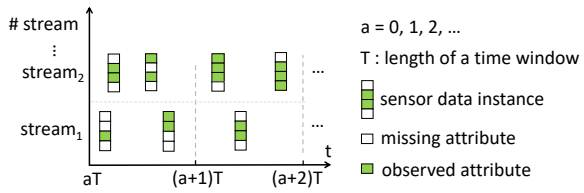

Figure 1: Sensor data streams.

system might continue to function with partially observed data, inaccurate or ill-defined results [8, 9, 36] may be produced. To make it worse, the sparsity can be quite high. For example, in ICU and Wi-Fi datasets that we use in this paper, the data sparsity exceeds 80% (see Table 2). The presence of high sparsity in data streams can cause major issues for online analytical tasks and downstream applications. In the ICU example, a doctor may make an incorrect, life-critical conclusion about the health of a patient based on monitored vital signs with missing values. In the example of air quality monitoring, missing values in the data streams may cause the system to miss the critical conditions of a fire emergency and consequently fail to give timely alerts. Therefore, it is crucial to accurately recover missing values in data streams in real time.

However, it is nontrivial to do so given that sensor data streams may exhibit characteristics as follows. **C1 (Aperiodicity)**: Data

† Corresponding author.

Proceedings of the VLDB Endowment, Vol. 17, No. 3 ISSN 2150-8097.
doi:10.14778/3632093.3632100

instances occur at non-fixed intervals. **C2 (Concurrency)**: Concurrent streams exist, such as multiple air quality monitoring stations or simultaneous monitoring of multiple patients in an ICU scenario. **C3 (Heterogeneity)**: Data instances contain diverse attributes, such as body temperature and blood pressure in an ICU scenario. **C4 (Sparsity)**: Streams often exhibit high ratios of missing values.

Existing studies on data stream imputation either have low efficiency [10, 15, 16] or make strong assumptions on the homogeneity of streams [30, 34, 49]. Such methods generally fall into two categories. First, neural network-based imputers [10, 15, 16] employ sequential neural networks, e.g., Recurrent Neural Networks (RNN), to iteratively fill in missing values within a sequence in a transductive learning mechanism. Such a sequential structure-based model often cannot contend with characteristics C1 and C2 — the aperiodicity of streams can prevent a model from capturing the temporal dependencies for data imputation, while the concurrency of streams can render the imputation inefficient as the model needs to be trained sequentially on all streams.

Second, matrix-based imputers [30, 34, 49] focus solely on homogeneous, single-attribute streams and they mostly fail to contend with characteristics C3 and C4. The heterogeneity of attributes and high ratios of missing values may render matrix-based imputers less effective, as values in a matrix may carry different meanings, and high sparsity in a matrix can compromise matrix completion operations that attempt to recover missing values.

To address these challenges, we propose a set of techniques to overcome the problems in a tumbling window. First of all, we construct a similarity graph to link data instances that originate from different streams or different timestamps within a time window. We do not assume that a stream is periodic; neither do we need to know from which stream an instance originates. We primarily construct a graph based on the similarities of data instances, as we believe this to be most significant for imputation. Also, compared to matrix-based imputers [30, 34, 49], operating on a graph can benefit from positive relational biases [7]. Namely, an instance only needs to be related to the most similar instances instead of to all other instances. Such a positive bias has been demonstrated to be beneficial to missing value imputation [14].

Subsequently, we propose a message propagation process on the similarity graph and design a Message Propagation Imputation Network (MPIN) that can exploit the correlations among instances and attributes to impute missing values accurately. We also provide a theoretical study for why MPIN is more effective in exploiting correlations for data imputation than a recently proposed feature propagation process [37]. Although MPIN conducts imputation in a transductive learning fashion, as do existing neural network-based imputers, MPIN is significantly more time-efficient since it is a graph-based model and can conduct imputation on multiple graph nodes (i.e., instances) in parallel, thus being able to exploit available computing resources sufficiently.

Although MPIN is both effective and efficient at imputing missing values of data instances in a time window, there are other challenges when MPIN is applied to continuous imputation for data streams. Essentially, the aperiodicity of data streams causes the number of data instances to vary from window to window. Consequently, there may not be enough data for training MPIN effectively. To contend with this, we propose a data update mechanism that can keep and update the important data instances in the streams and utilize them to enable more effective continuous imputation. Furthermore, as MPIN relies on transductive learning, retraining is needed at every current time window in order to impute newly arriving data instances. In order to lower the retraining cost, we propose a model update mechanism that allows to resume training from the best model state along the timeline so far. This makes continuous imputation with MPIN more efficient.

In summary, we make the following contributions.

- We construct a similarity graph with data instances in a time window having data instances as graph nodes. In addition, we propose a message propagation process on the graph to enable capturing correlations and exploiting positive relational bias.
- Based on the message propagation process, we propose a message propagation imputation network (MPIN) to exploit correlations among instances to impute their missing values in a time window. We also give a theoretical analysis of why MPIN is effective.
- To use MPIN for continuous imputation, we propose a framework with data update and model update mechanisms that help achieve both effective and efficient continuous imputation.
- We report on extensive experiments showing that the proposed MPIN can outperform competitors at data imputation and that the continuous imputation framework is effective and efficient.

The rest of the paper is organized as follows. Section 2 presents preliminaries and the research problem. Section 3 details the MPIN model for snapshot data imputation in a time window. Section 4 presents an MPIN based continuous imputation framework. Section 5 reports on extensive experiments. Section 6 reviews related work. Section 7 concludes and covers future research.

2 PRELIMINARIES AND PROBLEM

Table 1 presents commonly used notation.

Table 1: Notation.

Symbol	Description
$\mathbf{x}$	sensor data instance
$\mathbf{m}$	mask of a sensor data instance
$\mathcal{X}$	sensor data streams
$\mathcal{M}$	mask of sensor data streams
$\mathcal{X}_a$	sensor data chunk of a time window
$\mathcal{M}_a$	mask of a sensor data chunk

2.1 Data and Notation

Definition 1 (Sensor Data Instance). *A* sensor data instance *is represented as a* $\mathtt{D}$*-dimensional vector* $\mathbf{x} \in \mathbb{R}^{\mathtt{D}}$*, where each dimension (i.e., attribute)* $\mathbf{x}[d]$ *(*$0 \leq d < D$*) captures a sensor measurement.*

When clear from the context, we use "instance" to refer to a sensor data instance. An instance may consist of homogeneous or heterogeneous sensor measurements. In the aforementioned ICU example, an instance refers to a vector of heterogeneous health-related measures such as heart rate and blood pressure.

Definition 2 (Sensor Data Streams). *A* sensor data stream *is an unbounded, time-ordered sequence of sensor data instances.* $\mathtt{J}$ *concurrent sensor data streams are organized as a tensor* $\mathcal{X}$ *of size* $\mathtt{J} \times \mathtt{N} \times \mathtt{D}$ *such that* $\mathtt{N} \rightarrow \infty$ *corresponds to the time dimension. We use* $\mathcal{X}^j = \mathcal{X}[j, :, :]$ *to denote the* j*-th (*$0 \leq j < \mathtt{J}$*) data stream and*

$\mathcal{X}^j(n) = \mathcal{X}[j, n, :]$ *to denote the n-th* ($0 \le n < \mathsf{N}$) *sensor data instance of the j-th data stream.*[1]

The instances in a stream may contain *missing values* in their attributes due to factors such as sensor failures, depleted batteries, communication errors, and unforeseen malfunctions [5, 36, 46]. To indicate the missing values in sensor data streams, we define the notion of mask for sensor data streams as follows.

Definition 3 (Mask for Sensor Data Streams). *Given the sensor data streams $\mathcal{X}$, its corresponding* mask $\mathcal{M}$ *is a binary tensor with the same shape as $\mathcal{X}$:*

$$\mathcal{M}[j, n, d] = \begin{cases} 0, & \mathcal{X}[j, n, d] \text{ is missing;} \\ 1, & \text{otherwise.} \end{cases} \quad (1)$$

Typically, a mask $\mathcal{M}$ is sparse with many zeros. The sparsity characteristic of sensor data streams often renders online analytical tasks or downstream real-time applications (e.g., air quality monitoring) inaccurate or even non-functional [8, 9, 36].

2.2 Research Problem Formulation

Data streams are often processed using **tumbling window** technique [35, 40]. Specifically, a tumbling window is a fixed-length time window that moves through a stream at a constant time interval T in a non-overlapping fashion. With such windows, we discretize the sensor data streams $\mathcal{X}$ into data chunks and represent each chunk as $\mathcal{X}_a = \mathcal{X}[:, a\mathsf{T} : (a+1)\mathsf{T}, :]$, where $a \in \mathbb{Z}^{0+}$. For ease of presentation, we assume that each time window contains T time units and at most T data instances. Formally, $\mathcal{X}_a \in \mathbb{R}^{\mathsf{J}\times\mathsf{T}\times\mathsf{D}}$.

To further facilitate actual processing, we convert a data chunk $\mathcal{X}_a$ into a 2-dimensional matrix $\mathcal{X}_a \in \mathbb{R}^{(\mathsf{J}\cdot\mathsf{T})\times\mathsf{D}}$ by merging streams (i.e., J) and time (i.e., T). As a result, a **data chunk** $\mathcal{X}_a$ encompasses at most $\mathsf{J}\cdot\mathsf{T}$ data instances, to which each of the J streams contribute at most T instances in a time window. The corresponding mask can be discretized likewise, resulting in a mask chunk $\mathcal{M}_a$ ($a \in \mathbb{Z}^{0+}$) for the corresponding time window $[a\mathsf{T} : (a+1)\mathsf{T}]$.

We proceed to define the research problem.

Research Problem (Imputation of Sensor Data Streams, IDS). *Given sensor data streams $\mathcal{X}$,* online imputation of sensor data streams *continuously takes the current data chunk $\mathcal{X}_a$ as input and returns a complete data chunk $\hat{\mathcal{X}}_a$ by replacing each missing value in $\mathcal{X}_a$ with a proper data value. The objective here is to recover those missing values in $\mathcal{X}_a$ accurately and efficiently.*

We solve the IDS problem at two different, yet correlated, levels. First, we focus on snapshot imputation for a single time window, i.e., imputing missing values of data instances in one data chunk. In Section 3, we construct a similarity graph to capture the correlations among data instances and propose a Message Propagation Imputation Network (MPIN) that exploits the graph to impute the missing values in the current data chunk. Second, we study efficient and effective imputation for continuous time windows. In Section 4, we design a framework that uses MPIN as a building block for continuous imputation for data chunks from consecutive windows.

[1] In general, n can be a timestamp or an index (sequence number) but it takes only one form in a given data stream. Meanwhile, a stream is either periodic or aperiodic.

3 SNAPSHOT IMPUTATION FOR A WINDOW

This section focuses on snapshot imputation for a single time window. The key innovation is the Message Propagation (MsgProp) Imputation Network (MPIN) that takes a data chunk with missing values as input and outputs a data chunk without missing values. MPIN works on a similarity graph that captures the correlations among data instances in the window. MPIN's MsgProp component extends and generalizes the recent feature propagation [37] (FeaProp) for graph node feature imputation. Section 3.1 presents the similarity graph, Section 3.2 compares MsgProp and FeaProp, and Section 3.3 details MPIN.

3.1 Similarity Graph

Given a data chunk $\mathcal{X}_a \in \mathbb{R}^{(\mathsf{J}\cdot\mathsf{T})\times\mathsf{D}}$, we construct a *similarity graph* $G = (V, E)$ to organize $\mathcal{X}_a$'s data instances. Figure 2 illustrates similarity graphs. Specifically, each data instance $\mathbf{x}_i$ ($0 \le i < \mathsf{J}\cdot\mathsf{T}$)

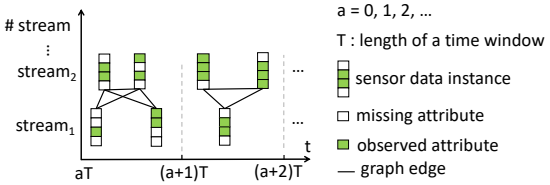

Figure 2: Similarity graphs for different data chunks.

corresponds to a graph node in V, where $|V| = \mathsf{J}\cdot\mathsf{T}$. For each pair of instances $\mathbf{x}_i$ and $\mathbf{x}_k$, we determine whether they should be linked by an edge e_{ik} as follows. First, we fill-in each missing value of each instance using the mean of the observed values in the same dimension of all other instances in V. Second, we compute the similarity between $\mathbf{x}_i$ and $\mathbf{x}_k$ and then create an edge between them only if they are sufficiently similar. The similarity can be implemented based on Euclidean distance or Cosine distance [44]. For example, we may simply check whether the Euclidean or Cosine distance between the vectors of $\mathbf{x}_i$ and $\mathbf{x}_k$ is below a given threshold. Alternatively, we may check whether $\mathbf{x}_i$ is among $\mathbf{x}_k$'s K nearest neighbors (KNN) in terms of Euclidean or Cosine distance, or vice versa. As our preliminary experiments in Appendix A.3 in an extended version [1] show, similarity graphs built using the Euclidean distance-based KNN method achieve the best performance. Therefore, we adopt this method in the experiments.

The similarity graph correlates data instances from different timestamps (but within the same time window) or across different streams. This offers a view of relationships among data instances, that is broader than if we focus on a single stream only. The graph also allows us to design a message propagation mechanism to exploit correlations among nodes (i.e., instances) and impute the missing attribute values for them.

3.2 FeaProp versus MsgProp

Given the similarity graph $G(V, E)$, we obtain a $|V| \times |V|$ adjacency matrix $\mathbf{A}$. Formally, $\mathbf{A}[i, k] = 1$ if graph edge e_{ik} exists and 0, otherwise. Subsequently, we compute the diagonal degree matrix $\mathbf{D} = \text{diag}(\sum_i \mathbf{A}[i, 1], \ldots, \sum_i \mathbf{A}[i, |V|])$ and the normalized adjacency matrix $\tilde{\mathbf{A}} = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$.

Feature Propagation. Given graph G, the FeaProp process aims to minimize the Dirichlet energy of the graph, i.e., $\min \sum_{ik} \big(\tilde{\mathbf{A}}[i, k](\mathbf{x}_i -$

$\mathbf{x}_k)^2$). This is a widely used approach to promote feature homophily in graphs [37]. Essentially, it aims to make a node's feature similar to those of its neighboring nodes. For instance, if $\tilde{\mathbf{A}}[i,k] = 1$, $\mathbf{x}_i$ and $\mathbf{x}_k$ are adjacent nodes. As a result, x_i and x_k should be similar in order to make $(\mathbf{x}_i - \mathbf{x}_k)^2$ small, thus minimizing the objective function. This way, the missing values of $\mathbf{x}_i$ can be interpolated from the corresponding values of $\mathbf{x}_k$ or vice versa. As the Dirichlet energy is convex, an efficient solution is the iterative weighted-sum of neighbors' information as follows.

$$\tilde{\mathbf{x}}_i^{(l+1)} = \sum\nolimits_{\mathbf{x}_k \in \mathcal{N}_i} \tilde{\mathbf{A}}_{ik} \mathbf{x}_k^{(l)}, \tag{2}$$

where $\tilde{\mathbf{A}}_{ik}$ refers to $\tilde{\mathbf{A}}[i,k]$, $\tilde{\mathbf{x}}_i^{(l+1)}$ is the imputation result of $\mathbf{x}_i$ in the $(l+1)$-th iteration, and the set $\mathcal{N}_i$ contains all nodes that are adjacent to $\mathbf{x}_i$. Theoretical analyses can be found elsewhere [37]. Originally, $\tilde{\mathbf{x}}_i^{(l+1)}$ is directly taken into the right-hand side of Equation 2 to start the next iteration, but this will modify the observed, correct values in $\mathbf{x}_i$ and cause a degradation in performance.

To solve the potential problem, a bound condition on the reconstructed features is added, which keeps the values of observed features equal to their original values during iterations. The bound condition is computed as follows.

$$\mathbf{x}_i^{(l+1)} \leftarrow \mathbf{x}_i^{(0)} \odot \mathbf{m}_i + \tilde{\mathbf{x}}_i^{(l+1)} \odot (\mathbf{1} - \mathbf{m}_i), \tag{3}$$

where $\mathbf{m}_i$ is the corresponding part of the mask for $\mathbf{x}_i$ and $\odot$ is the element-wise product operator.

After a certain number of iterations according to Equations 2 and 3, we obtain the converged feature $\hat{\mathbf{x}}_i$ for each data instance $\mathbf{x}_i$. In FeaProp, $\hat{\mathbf{x}}_i$ is used as the reconstructed version of $\mathbf{x}_i$. The proof of convergence of FeaProp is available elsewhere [37].

Message Propagation. The FeaProp process utilizes iterative adjacency matrix multiplication with bound conditions to impute missing values of features of nodes (i.e., instances). However, the adjacency matrix (i.e., $\tilde{A}$) treats a node's neighboring nodes equally, and this may not be accurate in reality as a node may have different correlations with different neighboring nodes. Next, the different attributes of an instance may also be correlated. Thus, we introduce two factors, namely correlations among instances (i.e., nodes) and among attributes of a node, to enhance imputation performance. Specifically, we extend Equation 2 in FeaProp as follows.

$$\tilde{\mathbf{x}}_i^{(l+1)} = \sum\nolimits_{\mathbf{x}_k \in \mathcal{N}_i} c(\mathbf{x}_i^{(l)}, \mathbf{x}_k^{(l)}) \mathbf{x}_k^{(l)} \mathbf{W}. \tag{4}$$

Equation 4 differs from Equation 2 in a number key points. First, Equation 4 replaces the constant component of adjacency matrix $\tilde{\mathbf{A}}_{ik}$ by a learnable correlation function $c(\mathbf{x}_i^{(l)}, \mathbf{x}_k^{(l)})$. Second, a feature transformation matrix $\mathbf{W} \in \mathbb{R}^{\mathsf{D}\times\mathsf{D}}$ is added to Equation 4 in order to capture correlations among different attributes of $\mathbf{x}_i$.[2] Above, the shape of $\mathbf{W}$ is restricted to $\mathsf{D} \times \mathsf{D}$ to make the output dimensions consistent with the input dimensions. To lift the restriction, we factorize $\mathbf{W}$ into $\mathbf{W}_1 \times \mathbf{W}_2$, where $\mathbf{W}_1 \in \mathbb{R}^{\mathsf{D}\times\mathsf{F}}$, $\mathbf{W}_2 \in \mathbb{R}^{\mathsf{F}\times\mathsf{D}}$, and $\mathsf{F}(> \mathsf{D})$ is a user-specified hyperparameter. Similarly, the bound condition in Equation 3 is added during iterations.

As a result, MsgProp possesses the following properties.

Lemma 1. *The feature propagation (FeaProp) process is a special case of the message propagation (MsgProp) process.*

[2]We omit the bias vector for the sake of brevity.

Proof. FeaProp is a special case of MsgProp in which the correlation $c(\mathbf{x}_i^{(l)}, \mathbf{x}_k^{(l)})$ equals $\tilde{\mathbf{A}}_{ik}$ and $\mathbf{W}_1$ and $\mathbf{W}_2$ are both set to identity matrices. □

Lemma 1 indicates that MsgProp has a higher learning capability than FeaProp.

Lemma 2. *The MsgProp process incorporates the classical message passing mechanism [7, 18] in graph learning.*

Proof. We rewrite Equation 4 as

$$\mathbf{z}_i^{(l+1)} = \sum\nolimits_{\mathbf{x}_k \in \mathcal{N}_i} c(\mathbf{x}_i^{(l)}, \mathbf{x}_k^{(l)}) \mathbf{x}_k^{(l)} \mathbf{W}_1, \tag{5}$$

$$\tilde{\mathbf{x}}_i^{(l+1)} = \mathbf{z}_i^{(l+1)} \mathbf{W}_2. \tag{6}$$

In particular, computing $\mathbf{z}_i^{(l+1)}$ in Equation 5 is a classical message passing process in graph learning. More specifically, Equation 5 implements the graph convolution operator [25] if $c(\mathbf{x}_i^{(l)}, \mathbf{x}_k^{(l)})$ equals $\tilde{\mathbf{A}}_{ik}$ and implements the graph attention operator [43] if $c(\mathbf{x}_i^{(l)}, \mathbf{x}_k^{(l)})$ is an attention function. □

Since MsgProp employs message passing, MsgProp is able to utilize existing message passing modules such as the Graph Attention Unit (GAT) [43], the Graph Convolution Unit (GCN) [25], and GraphSAGE [20], as well as their accompanying optimization techniques. Next, we propose the MsgProp imputation network that reconstructs the missing values of the data instances based on the MsgProp process over the similarity graph.

3.3 MsgProp Imputation Network (MPIN)

We first give the architecture of MPIN and then show how it imputes the missing values of data instances of an input data chunk $\mathcal{X}_a$.

Architecture. MPIN is constructed using a stack of *MsgProp layers* (MPL for short), a basic internal unit. Specifically, each MPL$(\cdot)$, consists of two modules, namely the *message passing module* and the *reconstruction module*, as depicted in Figure 3.

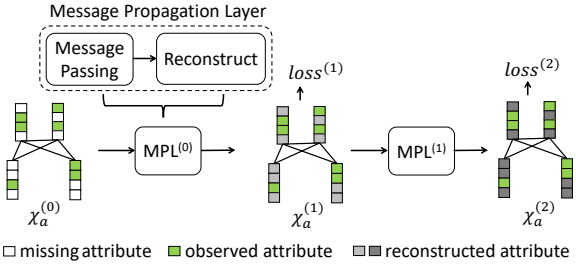

Figure 3: Message propagation imputation network.

The message passing module corresponds to the computational process presented in Equation 5 (see Lemma 2). The reconstruction module is the combination of a linear transformation and a bound condition, corresponding to the processes in Equations 6 and 3, respectively. An MPL transforms the raw data instance $\mathbf{x}_i^{(0)}$ to $\mathbf{x}_i^{(1)}$ by aggregating the features of its similar nodes in the similarity graph. This process thus reconstructs the missing values in $\mathbf{x}_i^{(0)}$ by utilizing correlations among the data instances.

The process is iterated by our MPIN to find the optimal reconstruction results for the missing values in the data instances. Typically, we stack two MPLs to build the imputation network MPIN:

$$\mathcal{X}_a^{(1)} = \text{MPL}(\mathcal{X}_a^{(0)}), \quad \mathcal{X}_a^{(2)} = \text{MPL}(\mathcal{X}_a^{(1)}), \tag{7}$$

where $\mathcal{X}_a^{(0)}$ denotes the raw input data chunk and $\mathcal{X}_a^{(1)}$ denotes the reconstructed data chunk after the first MPL. Next, $\mathcal{X}_a^{(2)}$ performs an enhanced reconstruction based on $\mathcal{X}_a^{(1)}$ and is taken as the final reconstructed result of $\mathcal{X}_a$.

Transductive Learning. Unlike FeaProp that performs matrix multiplications iteratively until convergence, MPIN adopts a transductive learning paradigm [42]. Specifically, given an input data chunk, MPIN imputes the missing values in the input based on backpropagation of the reconstruction loss corresponding to the observed values (see Equation 8). In other words, there are no separate phases of training and inference, and training data (i.e., observed values) and testing data (missing values) are all included in the training process. This paradigm has been proven effective at imputing missing values effectively [10, 15, 16]. The intuition is that if the reconstructed results corresponding to the observed values are close to the observed values themselves, then the reconstructed results corresponding to the missing values should also be close to their ground-truth values though these are unknown in reality. Accordingly, the training loss of MPIN is defined based on the reconstruction error between the observed values in $\mathcal{X}_a$ and their corresponding values as imputed by MPIN.

Since MPIN generates multiple reconstruction results recursively (see Equation 7), we inform MPIN about the reconstruction error of each reconstruction process instead of only at the last one. Thus, we use a linear combination of the losses computed from each of the reconstruction results as the overall loss:

$$\begin{aligned} \mathcal{L}^{(1)} &= \text{MSE}(\mathcal{X}_a^{(0)} \odot \mathcal{M}_a, \tilde{\mathcal{X}}_a^{(1)} \odot \mathcal{M}_a), \\ \mathcal{L}^{(2)} &= \text{MSE}(\mathcal{X}_a^{(0)} \odot \mathcal{M}_a, \tilde{\mathcal{X}}_a^{(2)} \odot \mathcal{M}_a), \\ \mathcal{L} &= \lambda_1 \cdot \mathcal{L}^{(1)} + \lambda_2 \cdot \mathcal{L}^{(2)}, \end{aligned} \tag{8}$$

where $\text{MSE}(\cdot,\cdot)$ calculates the mean squared error and $\mathcal{L}^{(1)}$ and $\mathcal{L}^{(2)}$ refer to the reconstruction loss of the first and second MsgProp layer, respectively. Further, $\mathcal{X}_a^{(0)} \odot \mathcal{M}_a$ is a masking process that picks up only the observed values of the data chunk $\mathcal{X}_a^{(0)}$. Also, $\tilde{\mathcal{X}}_a^{(1)}$, instead of $\mathcal{X}_a^{(1)}$, is used when computing the loss since the imputed results of the observed values exist only in $\tilde{\mathcal{X}}_a^{(1)}$ (see Equation 6), which is an imputation result before the bound condition (i.e., Equation 3) is applied. Finally, λ_1 and λ_2 are hyperparameters.

Algorithm 1 Training for Imputation (input data chunk $\mathcal{X}_a$, training epochs P, validation ratio β, model state θ')

1: validation dataset $\Phi_a \leftarrow$ fraction β of observed values in $\mathcal{X}_a$
2: $\mathcal{X}'_a \leftarrow \mathcal{X}_a \backslash \Phi_a$
3: initialization: $\texttt{MAE}^* \leftarrow \infty$; $\mathcal{X}^*_a \leftarrow \mathcal{X}_a$; $\theta^*_a \leftarrow \theta'$
4: initialize MPIN $\leftarrow$ MPIN of state θ'
5: **for** p = 1 to P **do**
6: $\hat{\mathcal{X}}_a, \hat{\Phi}_a, \hat{\theta}_a \leftarrow$ one-pass backpropagation with MPIN, $\mathcal{X}'_a$
7: $\texttt{MAE} \leftarrow \text{MAE}(\hat{\Phi}_a, \Phi_a)$ ▷ measuring validation error
8: **if** $\texttt{MAE} \leq \texttt{MAE}^*$ **then**
9: $\texttt{MAE}^* \leftarrow \texttt{MAE}$; $\mathcal{X}^*_a \leftarrow \hat{\mathcal{X}}_a$; $\theta^*_a \leftarrow \hat{\theta}_a$ ▷ update optimal
10: $\mathcal{X}^*_a \leftarrow \mathcal{X}^*_a \oplus \Phi_a$ ▷ recover manually removed values
11: return $\mathcal{X}^*_a$, θ^*_a

Algorithm 1 shows how to impute missing values of $\mathcal{X}_a$ by training MPIN. First, we generate a validation dataset by randomly removing a fraction (e.g., 5%) of the observed values in $\mathcal{X}_a$ and use the removed values as the ground truth values to validate the imputation error in each iteration of training (lines 1–2). Specifically, we enter the pre-processed data chunk $\mathcal{X}'_a$ into MPIN. Through backpropagation of reconstruction loss (see Equation 8), we obtain an imputed data chunk $\hat{\mathcal{X}}_a$ that contains the imputed results of the validation part, i.e., $\hat{\Phi}_a$, and a state of the model (i.e., parameters) denoted by $\hat{\theta}_a$ (line 6). By measuring the mean absolute error (MAE) on the validation dataset, we can find the optimal results of imputation and the optimal state of the model during the iterative training (lines 7–9). Finally, we recover the manually removed original values in $\mathcal{X}^*_a$, and return it as the final reconstructed result (lines 10-11). The optimal model parameters are returned as well. Their use will be detailed in Section 4.3.

3.4 Discussion

The MPIN model is inspired by a recent study [37] that uses feature propagation (FeaProp) to improve the learning effectiveness on graphs with missing values in the node features. Nevertheless, MPIN differs from FeaProp in three key aspects. First, our imputation process is based on the novel MsgProp process that exploits a correlation function instead of an adjacency matrix to capture correlations among nodes (i.e., instances). Second, MsgProp utilizes the correlations among the attributes for imputation, whereas FeaProp does not. As we have shown, FeaProp is a special case of MsgProp. Third, we reconstruct the missing values in node features via transductive learning based on observed values in the node features. In contrast, FeaProp does not involve learning but simply imputes missing values in node features based on iterative adjacency matrix multiplications. The more comprehensive design makes our approach more effective, as to be shown in Section 5. We also give an example below to illustrate the difference between FeaProp and MsgProp at imputation. For simplicity, we only show one of their iterative processes.

Example 1. *Suppose that a data chunk $\mathcal{X}_1$ in the time window $[\mathsf{T} : 2\mathsf{T}]$ contains data instances $\mathbf{x}_2 = [12, 5, 6]$, $\mathbf{x}_3 = [8, 5, 3]$, and $\mathbf{x}_4 = [3, 2, 1]$. If we use $K = 2$ to build the similarity graph, both $\mathbf{x}_3$ and $\mathbf{x}_4$ are adjacent nodes to $\mathbf{x}_2$ in the graph. For some reason (e.g., a sensor error), $\mathbf{x}_2$ misses its first component and becomes $\mathbf{x}_2 = [null, 5, 6]$. In order to impute the null value in $\mathbf{x}_2$, FeaProp multiplies the normalized adjacency matrix with neighboring instances (see Equation 2) and obtains an imputed result $\tilde{\mathbf{x}}_2 = 0.5 \cdot \mathbf{x}_3 + 0.5 \cdot \mathbf{x}_4 = [5.5, 3.5, 2]$. With the bound condition, $\tilde{\mathbf{x}}_2$ becomes $[5.5, 5, 6]$. Thus, the null value in $\mathbf{x}_2$ is replaced by 5.5. In contrast, instead of adopting the equal weight (i.e., 0.5) from the normalized adjacency matrix, MsgProp learns the correlation between $\mathbf{x}_2$ and its neighboring nodes through transductive learning based on the observed values. For simplicity, the learning process is simulated by our observation that finds $\mathbf{x}_3$ to be much closer to $\mathbf{x}_2$ than $\mathbf{x}_4$. Hence, we assume the following correlation values: $\mathbf{c}(\mathbf{x}_2, \mathbf{x}_3) = 0.8$ and $\mathbf{c}(\mathbf{x}_2, \mathbf{x}_4) = 0.2$. Applying MsgProp (see Equation 4)[3], we get $\tilde{\mathbf{x}}_2 = 0.8 \cdot \mathbf{x}_3 + 0.2 \cdot \mathbf{x}_4 = [7.0, 4.4, 2.6]$. Due to the same bound condition, $\mathbf{x}_2$ becomes $[7.0, 5, 6]$. As 7.0 is closer to the ground truth (i.e., 12) than 5.5, MsgProp is more effective at imputation than FeaProp.*

[3]For simplicity, we skip the discussion of the correlation among attributes in the example and assume that $\mathbf{W}$ in Equation 4 is an identity matrix.

4 CONTINUOUS IMPUTATION

4.1 Motivation and Overview

To enable continuous imputation on unbounded sensor data streams, a straightforward approach is to apply MPIN to each time window periodically. This approach is called MPIN-P and is depicted in Figure 4 (a). However, this approach suffers from two significant drawbacks. First, from a **data perspective**, the varying number of instances in a data chunk may not provide sufficient data for training MPIN effectively at each time window, leading to potentially poor imputation results. Further, earlier data instances in the stream contain valuable information that can improve current and future imputation outcomes, thus making it unwise to disregard these. Second, from a **model perspective**, since MPIN relies on transductive learning, the entire imputation process (i.e., Algorithm 1) must be started from scratch for each window, which is time-consuming, especially for large-sized data chunks.

To address these drawbacks, we propose an incremental framework for continuously imputing sensor data streams. As shown in Figure 4 (b), the incremental imputation framework is powered by the **data update** and **model update** mechanisms, corresponding to the data and model challenges. First, instead of caching all previous data instances for training the current MPIN, the data update mechanism selects and caches the most valuable data instances to enable a time- and space-efficient transductive learning process during the continuous imputation. Second, to avoid starting from scratch at each window, the model update mechanism regards the model trained at the previous window as a pre-trained model and fine-tunes it for imputing the current data chunk in a transfer learning manner. We use MPIN-DM to refer to our incremental framework.

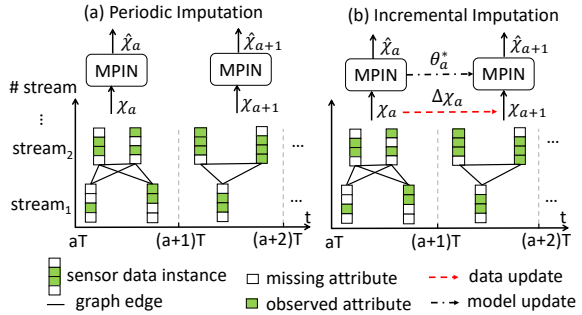

Figure 4: Continuous imputation for sensor data streams.

We proceed to elaborate on the two update mechanisms in Sections 4.2 and 4.3, followed by a complexity analysis of the MPIN-DM framework in Sections 4.4.

4.2 Data Update Mechanism

Data Instance Importance Scores. The data update mechanism aims to identify and retain only the most valuable historical data instances that can help impute the missing values in the current data chunk. We proceed to formulate criteria for quantifying the value of a data instance.

Criterion 1 (Higher Observation Ratio). *Intuitively, a sensor data instance with a higher fraction of observed values can contribute more to the imputation of missing values of other data instances.*

In MPIN, the learning is driven by minimizing the difference between the observed values and their reconstructed counterparts. In this sense, more observed values can improve the effectiveness of transductive learning.

Criterion 2 (Lower Observation Overlap Ratio). *A sensor data instance whose observed dimensions overlap less with those of other instances can contribute more to the imputation of missing values of other instances.*

Generally speaking, the observed values of an instance can provide information for imputing the missing values of other instances only if the former are located in different dimensions than the latter. Referring to the example in Figure 5, suppose that both $\mathbf{x}_2$ and $\mathbf{x}_3$ are used to help impute missing values in $\mathbf{x}_1$. Although $\mathbf{x}_2$ has a higher observation ratio, $\mathbf{x}_3$ may be more helpful in imputing $\mathbf{x}_1$'s missing value since all the observed values of $\mathbf{x}_2$ concern the same dimensions as those of $\mathbf{x}_1$ and thus provide little information to the imputation of $\mathbf{x}_1$'s missing value.

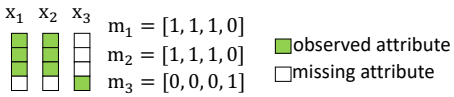

Figure 5: Example of the commonly observed value ratio.

Combining the two criteria, we compute the **importance score** of a data instance $\mathbf{x}_i$, denoted as $\varphi(\mathbf{x}_i)$, as follows.

$$\varphi(\mathbf{x}_i) = OR(\mathbf{x}_i) - \frac{1}{|V|-1}\sum\nolimits_{\mathbf{x}_k \in V \setminus \mathbf{x}_i} OOR(\mathbf{x}_i, \mathbf{x}_k), \tag{9}$$

$$OR(\mathbf{x}_i) = ||\mathbf{m}_i||_0 = \mathbf{m}_i^\top \mathbf{m}_i, \tag{10}$$

$$OOR(\mathbf{x}_i, \mathbf{x}_k) = ||\mathbf{m}_i \cap \mathbf{m}_k||_0 = \mathbf{m}_i^\top \mathbf{m}_k. \tag{11}$$

In particular, the importance score $\varphi(\mathbf{x}_i)$ in Equation 9 equals the observation ratio score $OR(\mathbf{x}_i)$ minus the observation overlap ratio score $OOR(\mathbf{x}_i, \mathbf{x}_k)$ averaged over all other instances. Corresponding to Criterion 1, Equation 10 calculates $OR(\mathbf{x}_i)$ as the number of non-zero elements (i.e., $||\cdot||_0$) in the corresponding mask $\mathbf{m}_i$. Corresponding to Criterion 2, Equation 11 calculates $OOR(\mathbf{x}_i, \mathbf{x}_k)$ for two instances as the number of non-zero elements in the element-wise AND result of their masks $\mathbf{m}_i$ and $\mathbf{m}_k$. In both Equations 10 and 11, we use the number of non-zero elements instead of the ratio, observing that the feature dimension $\mathtt{D}$ is constant.

The importance score has the following important property.

Lemma 3. *$\forall \mathbf{x}_i \in V$, we have $0 \le \varphi(\mathbf{x}_i) \le \mathtt{D} - 1$.*

The proof is given in Appendix B.1 in the extended version [1] for the sake of brevity. Lemma 3 gives the range of the importance score of any data instance. Based on Lemma 3, we are able to normalize the importance scores by dividing them by $(\mathtt{D}-1)$ and use a threshold $\eta \in (0, 1]$ to drop those data instances whose importance score does not exceed η. In practice, η is tuned to a proper value based on preliminary experiments, and a value of around 0.6 has been demonstrated to be able to obtain satisfactory performance, as reported in Appendix A.7 in the extended version [1].

Efficient Computing of Importance Scores. It is time-consuming to compute the importance score per data instance. To accelerate the computations, we use matrix multiplication to obtain importance scores of all instances in one pass. Specifically, we introduce a **gram mask matrix** as follows. First, recall that the similarity graph is built based on the input data chunk, i.e., $\mathcal{X}_a \in \mathbb{R}^{|V| \times \mathtt{D}}$, and

its corresponding mask chunk is denoted as $\mathcal{M}_a \in \mathbb{R}^{|V|\times \mathsf{D}}$. Thus, a gram mask matrix $\mathbf{M}^{\mathrm{GM}}$ is computed as $\mathcal{M}_a\mathcal{M}_a^{\mathsf{T}}$ with size $|V|\times|V|$. Given the gram mask matrix $\mathbf{M}^{\mathrm{GM}}$, the importance scores of all data instances can be computed in one pass via the following lemma.

Lemma 4. *Given the corresponding gram mask matrix $\mathbf{M}^{GM}$, the importance scores of data instances in set V can be computed jointly by*

$$\boldsymbol{\varphi} = \frac{(|V| * \mathrm{diag}^{-1}(\mathbf{M}^{GM}) - \mathbf{M}^{GM}\cdot \mathbf{1}^{|V|\times 1})}{|V|-1}, \tag{12}$$

where $\boldsymbol{\varphi} \in \mathbb{R}^{|V|\times 1}$ and $\boldsymbol{\varphi}[i]$ captures the importance score of the i-th data instance $\mathbf{x}_i$ in data chunk $\mathcal{X}_a$, $\mathrm{diag}^{-1}(\cdot)$ gets the diagonal vector from the input matrix, and $*$ is the element-wise scalar product. The proof is in Appendix B.2 in the extended version [1] for brevity.

Algorithm. Using Lemma 4, we can obtain the importance scores of all instances in parallel with GPU-based matrix multiplication. The mechanism is formalized in Algorithm 2. First, the cached valuable data instances so far, i.e., $\Delta\mathcal{X}_a$, are concatenated with the current input data chunk to form a new data chunk $\mathcal{X}''_{a+1}$ (line 1). The new data chunk instead of $\mathcal{X}_{a+1}$ is used to train MPIN. Likewise, we can obtain a new mask chunk $\mathcal{M}''_{a+1}$ (line 1), based on which we calculate the gram mask matrix and further exploit Lemma 2 to derive the importance score of each data instance (lines 2–3). Finally, we keep those with importance scores above the threshold η and regard them as the updated valuable instances so far (lines 4–5).

Algorithm 2 DataUpdate (last valuable instances $\Delta\mathcal{X}_a$ and mask $\Delta\mathcal{M}_a$, current data chunk $\mathcal{X}_{a+1}$ and mask $\mathcal{M}_{a+1}$, predefined value threshold η)

1: $\mathcal{X}''_{a+1} \leftarrow concat(\Delta\mathcal{X}_a, \mathcal{X}_{a+1})$; $\mathcal{M}''_{a+1} \leftarrow concat(\Delta\mathcal{M}_a, \mathcal{M}_{a+1})$
2: the gram mask matrix $\mathbf{M}^{\mathrm{GM}} \leftarrow \mathcal{M}''_{a+1}{\mathcal{M}''_{a+1}}^{\mathsf{T}}$
3: compute $\boldsymbol{\varphi}$ with $\mathbf{M}^{\mathrm{GM}}$ using Equation 12
4: $\Delta\mathcal{X}_{a+1} \leftarrow \{\mathcal{X}''_{a+1}[i] \mid \boldsymbol{\varphi}[i] \geq \eta\}$
5: **return** $\Delta\mathcal{X}_{a+1}$

To explain the process of data update and its benefits to imputation, we give an example.

Example 2. *To continue Example 1, we assume that another time window $[0:\mathsf{T}]$ is just before the window $[\mathsf{T}:2\mathsf{T}]$. Suppose that the data chunk $\mathcal{X}_0$ in $[0,\mathsf{T}]$ contains 2 data instances $\mathbf{x}_0 = [null, null, 4]$ and $\mathbf{x}_1 = [12, 5, 5]$. Hence, their mask vectors are $\mathbf{m}_0 = [0,0,1]$ and $\mathbf{m}_1 = [1,1,1]$, respectively, and the mask matrix is $\mathcal{M}_0 = [\mathbf{m}_0, \mathbf{m}_1]$. The data update strategy enables us to pass valuable instances to the next window. First, we compute the gram mask matrix $\mathbf{M}^{GM} = \mathcal{M}_0\mathcal{M}_0^{\mathsf{T}} = [[1,1],[1,3]]$ and the importance score vector $[0,1]$ based on Equation 12. Thus, the importance scores of $\mathbf{x}_0$ and $\mathbf{x}_1$ are 0 and 1, meaning that $\mathbf{x}_0$ is much less important since it has very few observed attributes. As a result, $\mathbf{x}_1$ is taken to the next window, i.e., $[\mathsf{T}:2\mathsf{T}]$. Next, we show what difference $\mathbf{x}_1$ can make to the imputation in Example 1. In that example, $\mathbf{x}_1$ will replace $\mathbf{x}_4$ to be the new adjacent node of $\mathbf{x}_2$ in the similarity graph since $\mathbf{x}_1$ is closer to $\mathbf{x}_2$ than $\mathbf{x}_4$. Accordingly, the correlation values are updated to $\mathbf{c}(\mathbf{x}_2,\mathbf{x}_3) = 0.2$ and $\mathbf{c}(\mathbf{x}_2,\mathbf{x}_1) = 0.8$, given that $\mathbf{x}_1$ now is closer to $\mathbf{x}_2$ than is $\mathbf{x}_3$. Following the MsgProp process in Example 1, the imputation result is $[11.2, 5, 6]$, i.e., 11.2 is the imputed value for the null in $\mathbf{x}_2$. This is even closer to the ground truth than the imputed value of 7.0 in Example 1.*

4.3 Model Update Mechanism

To avoid the costly process of training MPIN from scratch for every time window, we propose a model update mechanism that resumes training from the best state so far. The mechanism consists of two components, namely *model state selection* and *model update*.

Model State Selection. This component helps choose the best model state, i.e., the best-trained parameters so far. The best state is then used as the initial state of MPIN in the next window. To make the resumed training of MPIN in the next window effective, we need to ensure a "good" initial state (i.e., parameters) for MPIN from history. We realize this by Algorithm 3. First, we find the optimal state of the model within the current time window using Algorithm 1. This will in turn be conveyed to the next window as the initial state of MPIN for retraining. As we can see, θ^*_{a+1} is derived from θ^*_a, i.e., the best state so far. This way, we can ensure that θ^*_{a+1} is always the best state of the model seen so far.

Algorithm 3 ModelStateSelection (last best state θ^*_a, current data chunk $\mathcal{X}_{a+1}$, training epochs P, validation fraction η)

1: $\mathcal{X}^*_{a+1}, \theta^*_{a+1} \leftarrow$ call Algorithm 1 ($\mathcal{X}_{a+1}$, P, ϵ, θ^*_a) ▷ best state
2: return θ^*_{a+1}

Model Update. After getting the best model state so far, we may simply use it as the initial state to train the whole MPIN for the current window, i.e., from the best state instead of from scratch [17] (see Figure 6 (a)). However, our preliminary experiments show that this yields poor imputation results, probably due to overfitting.

To achieve better results, we adopt another strategy to make use of the best state and to initialize the current MPIN. Specifically, referring to Figure 6 (b), we only utilize the best state parameters relevant to MPIN's message passing module. In addition, we always initialize MPIN's reconstruction module with random parameter settings (i.e., θ_0). This contributes to preventing overfitting. This approach is inspired by transfer learning, where the best state of a model acts as a pre-trained model on a graph (i.e., a data chunk), and we only need to fine-tune it for another graph corresponding to the current data chunk. The fine-tuning process in our context refers to retraining only the reconstruction module from scratch while reusing the previous parameters of the message-passing modules. This way, the knowledge learned by MPIN from the previous window can be transferred to help impute the data chunk in the current window, while significantly reducing the risk of overfitting.

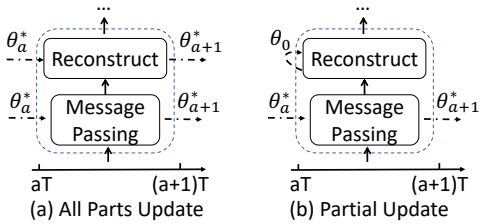

Figure 6: All parts update vs. partial update.

4.4 Complexity Analysis

We present the space and time complexity of the proposed MPIN-DM and the periodic approach, MPIN-P.

Space Complexity. The space cost mainly corresponds to the storage of relevant data in the streams for online imputation, including

the cached data and the new data in the current window. We assume that the average size of a data chunk is V, that the maximum cached data via data update strategy is V^{mc}, and that a is the index of the current time window (i.e., the number of time windows that have passed so far). MPIN-DM takes $O(\mathsf{V}+\mathsf{V}^{mc})$ space. In contrast, MPIN-P consumes $O(\mathsf{V})$. Furthermore, if an imputation method caches all historical data for retraining the current MPIN, it consumes $O(a \cdot \mathsf{V})$. We can regard V and V^{mc} as constants and then get:

$$O(\mathsf{V} + \mathsf{V}^{mc}) \approx O(\mathsf{V}) \approx O(1) \ll O(a \cdot \mathsf{V}) \approx O(a) \tag{13}$$

As we can see, MPIN-DM's space complexity is at the same level as that of MPIN-P, but MPIN-DM is much more effective (see Section 5.3). Also, the caching-all approach is infeasible as space consumption increases constantly as streams evolve.

Time Complexity. The main time cost is associated with training MPIN to impute missing values in a data chunk (see Algorithm 1). While it is hard to perform a detailed time complexity analysis, we provide some insight into the complexity as follows. For any chunk $\mathcal{X}_{a+1}$ in the $(a+1)$th time window, if we train MPIN from scratch until getting the best-imputed results, the time cost is denoted as $\mathcal{T}(\theta^*_{a+1})$. In contrast, utilizing the model update mechanism, the training resumes from the last best state θ^*_a, resulting in a reduced time cost of $\Delta\mathcal{T} = \mathcal{T}(\theta^*_{a+1}) - \mathcal{T}(\theta^*_a)$. When the size of $\mathcal{X}_{a+1}$ is large, we have $\Delta\mathcal{T} \ll \mathcal{T}(\theta^*_{a+1})$, as more data typically entail higher time costs if the current MPIN is trained from scratch.

5 EXPERIMENTAL STUDIES

Section 5.1 covers the overall experimental settings. Subsequently, Sections 5.2 and 5.3 validate the efficacy of the snapshot imputer MPIN (see Section 3) and the continuous imputation framework MPIN-DM (see Section 4), respectively.

5.1 Overall Settings

Datasets. We use three datasets from different scenarios:

- **ICU**[4] consists of 11,988 48-hour time series that record the health condition of patients in Intensive Care Units (ICU) during the initial 48 hours after their admission to ICU. The records are taken hourly and each one contains 37 variables such as temperature, heart rate, and blood pressure.
- **Airquality**[5] contains hourly air quality monitoring data from 12 monitoring sites in Beijing for a continuous period of 48 months (1,461 days). Each data instance is comprised of 11 variables such as PM2.5, PM10, and SO2. Since the application focuses on daily air quality, we aggregate the data from all sites to form concurrent streams spanning a single day, resulting in a total of 17,532 concurrent streams (#days × #sites).
- **Wi-Fi**[6] is a sequence of Wi-Fi signal records that span one hour at a mall. Each instance in the sequence represents a vector of 671 Wi-Fi Received Signal Strength Indicator (RSSI) values, each from one of 671 Wi-Fi access points. The collected RSSI vectors can be used for indoor positioning [26].

Table 2 presents the key characteristics of the datasets. Specifically, ICU and Airquality feature concurrent streams with longer lengths and heterogeneous attributes in periodic instances, whereas Wi-Fi consists of high-dimensional homogeneous aperiodic sensor data. Additionally, ICU and Wi-Fi have originally high data sparsity, whereas Airquality is less sparse.

Table 2: Key characteristics of the datasets.

Datasets	ICU	Airquality	Wi-Fi
Dimensionality of Data Instances	37	11	671
Time Length of Streams (hours)	48	24	1
Number of Concurrent Streams	11988	17532	1
Regular Sampling	Yes	Yes	No
Heterogeneous Attributes	Yes	Yes	No
Original Data Sparsity	80.0%	1.6%	85.6%

Implementation. The project is primarily in Python 3.8 and tested on a Linux server with 3.20 GHz Intel Core i9 CPU and NVIDIA Geforce P8 GPU with 24.5 GB memory. PyTorch 1.8 is used to create the neural network models. The datasets, code, and configuration details are accessible online [2]. To construct the similarity graph, we use Euclidean distance-based KNN with $K = 10$. For MPIN, we use two MsgProp layers and apply GraphSAGE [20] unit to implement the internal message passing module[7]. In model training, we set the learning rate to 0.01, the weight decay rate to 0.1, λ_1 and λ_2 to 1, the epoch count to 200, and use Adam as the gradient descent optimizer. The extended version [1] covers additional experiments on the selection of hyperparameters, such as the similarity function, the value K of the KNN-based method for building similarity graphs (see Section 3.1), and the number of MsgProp layers.

Evaluation Metrics. We concern both effectiveness and efficiency.

(1) Effectiveness. As no ground-truth values are available for truly missing values in the real datasets, following previous studies [10, 15, 16] on data imputation, we randomly remove a fraction of the observed attributes in $\mathcal{X}_a$ and use the removed values as the ground truth values to evaluate the imputation error. Specifically, we employ *Mean Absolute Error* (MAE) and *Mean Relative Error* (MRE) to measure the difference between the imputed results $\hat{\mathcal{X}}_a$ and the input data chunk $\mathcal{X}_a$ with respect to the randomly removed attribute values that are marked by an indicator matrix $\mathcal{M}_e$. They are calculated as follows.

$$MAE(\mathcal{X}_a, \hat{\mathcal{X}}_a, \mathcal{M}_e) = \frac{\sum_{i=1}^{\mathsf{J}\cdot\mathsf{T}} \sum_{d=1}^{\mathsf{D}} \left(|\mathcal{X}_a[i,d] - \hat{\mathcal{X}}_a[i,d]| \cdot \mathcal{M}_e[i,d]\right)}{\sum_{i=1}^{\mathsf{J}\cdot\mathsf{T}} \sum_{d=1}^{\mathsf{D}} \mathcal{M}_e[i,d]},$$

$$MRE(\mathcal{X}_a, \hat{\mathcal{X}}_a, \mathcal{M}_e) = \frac{\sum_{i=1}^{\mathsf{J}\cdot\mathsf{T}} \sum_{d=1}^{\mathsf{D}} \left(|\mathcal{X}_a[i,d] - \hat{\mathcal{X}}_a[i,d]| \cdot \mathcal{M}_e[i,d]\right)}{\sum_{i=1}^{\mathsf{J}\cdot\mathsf{T}} \sum_{d=1}^{\mathsf{D}} \left(|\mathcal{X}_a[i,d]| \cdot \mathcal{M}_e[i,d]\right)}.$$

The lower values of these metrics indicate higher accuracy in imputed results. These metrics have been proven effective in assessing the quality of data imputation [10, 15, 16].

(2) Efficiency. As a crucial factor for online imputation, efficiency is reflected in two metrics: *imputation time cost* required to impute a data chunk, and *memory cost* required by the model online.

5.2 Evaluation on Snapshot Imputation

In this part, we compare the proposed MPIN approach with alternative representative imputers, focusing on data imputation for a single time window.

Baselines. We compare our MPIN with seven existing data imputers from three categories, as listed in Table 3.

[4] https://physionet.org/challenge/2012/
[5] https://archive.ics.uci.edu/ml/datasets/Beijing+Multi-Site+Air-Quality+Data
[6] https://www.kaggle.com/competitions/indoor-location-navigation/data

[7] Other alternative message passing modules such as GAT [43] and GCN [25] achieve lower performance and are covered in Appendix A.6 in the extended version [1].

Table 3: Baseline methods.

Category	Method	Description
Traditional	MEAN [3]	MEAN imputes the missing values in a data chunk using the mean of each corresponding dimension.
	KNN [41]	KNN imputes the missing values of each instance using the mean of the instance's k nearest neighbors.
	MICE [6]	Multiple Imputation by Chained Equation imputes missing values based on interdependent features in a round-robin fashion.
	MF [21]	Matrix Factorization treats a data chunk as a matrix and employs iterative SVD decomposition to impute missing values.
Time-series based	BRITS [10]	Bi-directional Recurrent Imputation for Time Series data adapts an RNN for iterative imputation.
	SAITS [16]	Self-Attention-based Imputation for Time Series replaces the RNN with the self-attention to capture dependencies for imputation.
Graph-based	FP [37]	Feature Propagation captures the correlation of data instances by iterative adjacency matrix multiplications.

Parameter Settings. Our experiments concern three key parameters. First, we vary the parameter of *missing rate*. Similar to previous work [10, 16], we do not have the ground truth for those truly missing values. Therefore, we randomly remove a ratio of observed values from the raw data. The removal ratio is called the missing rate and the removed values are used as the ground truth of the missing values thus induced. Second, we vary the parameter of *window length*, i.e., changing the length of a time window. A longer window will involve more data instances of the stream, but the number of windows will be less due to the fixed stream length. Third, we vary the parameter of *concurrent streams*, i.e., the number of concurrent streams involved in a time window. We achieve this by varying the ratio of the original number of streams included in the data for our experiments. In general, the lower the ratio, the fewer streams are involved and the smaller a data chunk will be.

Table 4 gives the parameters settings with defaults shown in bold. It is noteworthy that we apply relatively higher missing rates (up to 80%) to Airquality because this dataset originally is much less sparse than the other two (see Table 2). Besides, Wi-Fi has only one single stream and thus there is no variation on the concurrent streams. The parameter of window length is set properly according to the total length of the streams. In particular, the length of time windows for ICU and Airquality are at an hourly level, since the data instances in the streams are sampled each hour. However, the actual data amount within a time window is large due to the existence of concurrent streams. Nevertheless, MPIN's imputation needs a few seconds only (see Table 6).

Table 4: Parameter settings (defaults are in bold).

Datasets	ICU	Airquality	Wi-Fi
Missing Rate (%)	[10, 30, **50**]	[40, 60, **80**]	[10, 30, **50**]
Window Length	[1,2,3,**4**,5,6] (h.)	[1,**2**,3,4,5] (h.)	[2,4,**6**,8,10] (min.)
Concurrent Streams (%)	[1, 10, 20, . . ., **100**]		–

5.2.1 Overall Effectiveness Comparison. We vary the missing rate per dataset to test the robustness of each method under varied data sparsity. The effectiveness measures are reported in Table 5, where the **best** and second-best results per setting are highlighted.

Overall, MPIN always outperforms the competitors with wide margins in terms of both MAE and MRE in almost all settings. Only in very few cases, MPIN is the second-best, with performance very close to the best. Time series models SAITS and BRITS can only exploit temporal dependencies in a series (i.e., a stream), whereas MPIN is able to exploit correlations among data instances that may include those across different streams. Moreover, MPIN operates on a graph and thus can exploit positive relational inductive biases from the graph [7] to further promote the effectiveness of imputation. This also explains why MPIN outperforms those traditional imputers such as MICE and MF.

In addition, compared to FP, MPIN is much more effective in most cases. Unlike FeaProp, the MsgProp mechanism can capture inter-instance and inter-attribute correlations dynamically. Also, we notice that FP performs relatively better on Wi-Fi than on ICU and Airquality. This is because Wi-Fi is a homogeneous dataset where it is easier for the FeaProp in FP to capture correlations.

Moreover, MPIN is much more robust to increasing data sparsity than the other methods. On the one hand, a similarity graph-based data structure involves data instances across streams and timestamps within a window and thus is more likely to find alternative neighbors when the original instances become sparse. On the other hand, MPIN can exploit the correlations in the graph-structured data sufficiently using MsgProp and transductive learning.

5.2.2 Overall Efficiency Comparison. Table 6 compares the efficiency of all imputation methods in terms of time cost. Since the time cost remains consistent across varying missing ratios, we provide a single result for each imputer on the respective dataset. Notably, methods such as MEAN, FP, and MPIN exhibit considerably lower time costs compared to others. On the other hand, the time-series models BRITS and SAITS demonstrate the highest time consumption due to their sequential neural network architecture, which includes computationally intensive components like recurrent units or self-attention units during training. While MPIN is also a neural network model, it capitalizes on parallel processing with its MsgProp layer for graph-structured data, enabling efficient utilization of computational resources and minimizing training time (i.e., the imputation process).

Table 7 provides a comparison of memory cost among the neural network models, namely MPIN, BRITS, and SAITS. Non-neural network models are excluded in this analysis as they generally exhibit lower effectiveness (as shown in Table 5). The results in Table 7 demonstrate that MPIN requires clearly less memory compared to BRITS and SAITS. This distinction arises from the fact that MPIN's MsgProp layer uses a simpler structure than the recurrent unit and self-attention unit used by the others.

Subsequently, we exclude the inferior traditional imputers and narrow our focus to comparing MPIN with BRITS, SAITS, and FP as they exhibit superior effectiveness or efficiency. Furthermore, as the MAE results demonstrate similar trends to MRE, they are reported in Appendix A.1 in the extended version [1].

5.2.3 Effect of Time Window Length. We vary the length of time window to investigate its impact on imputation. We present the

Table 5: Overall effectiveness comparison (the unit of MRE is %).

Dataset	ICU						Airquality						Wi-Fi					
Rate	50%		30%		10%		80%		60%		40%		50%		30%		10%	
Metrics	MAE	MRE	MAE	MRE	MAE	MRE	MAE	MRE	MAE	MRE	MAE	MRE	MAE	MRE	MAE	MRE	MAE	MRE
MEAN	0.88	101.7	0.81	94.98	0.73	84.12	0.48	84.17	0.46	79.56	0.40	68.98	2.21	92.06	1.95	81.39	1.52	63.21
KNN	0.74	85.26	0.62	72.76	0.46	53.33	0.48	84.43	0.47	80.22	0.41	70.44	1.84	76.89	1.43	59.51	0.63	26.37
MICE	0.73	84.85	0.61	71.55	0.44	51.07	0.48	83.52	0.45	78.02	0.39	66.74	2.00	83.68	1.65	68.77	0.99	41.25
MF	0.79	91.81	0.71	83.29	0.57	66.37	0.54	91.57	0.51	86.76	0.46	79.58	1.85	77.19	1.50	62.34	0.81	33.66
FP	0.83	96.03	0.78	91.49	0.70	81.35	0.57	99.78	0.57	98.67	0.56	96.36	1.26	52.77	0.55	22.73	**0.12**	**5.17**
BRITS	0.57	70.76	0.50	63.04	0.44	55.19	0.49	68.64	0.39	55.57	0.35	49.00	0.37	48.28	0.33	43.24	0.30	39.69
SAITS	0.53	65.89	0.47	58.29	0.41	51.63	0.43	60.65	0.31	43.42	0.23	**32.69**	0.47	60.92	0.40	52.04	0.41	54.09
MPIN	**0.39**	**44.92**	**0.38**	**44.17**	**0.39**	**44.54**	**0.20**	**35.89**	**0.21**	**36.06**	**0.20**	34.82	**0.24**	**10.06**	**0.20**	**8.31**	0.16	6.67

Table 6: Overall time cost comparison (unit: seconds).

Method	MEAN	KNN	MICE	MF	FP	BRITS	SAITS	MPIN
ICU	0.05	175.25	84.84	25.28	0.32	2143.61	1985.28	3.19
Airquality	0.01	81.31	0.29	1.33	0.09	1576.1	4201.69	1.28
Wi-Fi	0.05	0.11	42.39	4.21	0.2	7.34	9.46	2.94

Table 7: Memory cost (unit: MB).

Method	BRITS	SAITS	MPIN
ICU	0.374	5.256	**0.183**
Airquality	0.189	5.091	**0.056**
Wi-Fi	23.574	12.305	**3.246**

results for ICU and Airquality. The results for Wi-Fi follow similar patterns and are shown in Appendix A.2 in the extended version [1].

Referring to Figures 7 (a) and (b), as the time window becomes longer, the imputation errors of MPIN, SAITS and BRITS decrease. In particular, both SAITS and BRITS improve more rapidly than MPIN. As both SAITS and BRITS are time series data imputers, they can capture deeper time dependencies and benefit more from longer time series. In contrast, MPIN does not capture temporal dependencies, and the only benefit to MPIN of a longer time window is to provide more training data. However, this benefit may bring about only slight performance gains, as the training data may already be sufficient. Still, MPIN is the most effective in all cases.

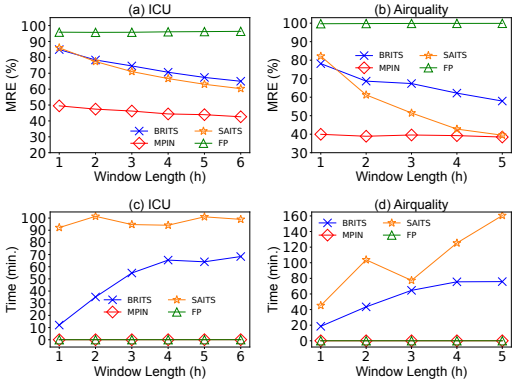

Figure 7: Effect of window length.

Moreover, referring to Figures 7 (c) and (d), despite fluctuations, the time costs of SAITS and BRITS increases as the window length increases. The longer the time window, the longer the time series that SAITS and BRITS need to process with their sequential modules. The fluctuations are due to the random initialization of parameters at the beginning of the training process.

In contrast, MPIN maintains consistently low and stable time costs. This is because MPIN does not process data sequentially but can process graph-structured data in parallel with its MsgProp layer. This enables efficient utilization of computational resources and reduces training time. Overall, the time cost of SAITS and BRITS is by orders of magnitude higher, making them less desirable for online imputation scenarios. Also, FP exhibits similar efficiency to MPIN across different window lengths. However, FP lags behind in terms of effectiveness due to its reliance on the simple matrix multiplications in FeaProp that struggle to effectively capture correlations in heterogeneous datasets.

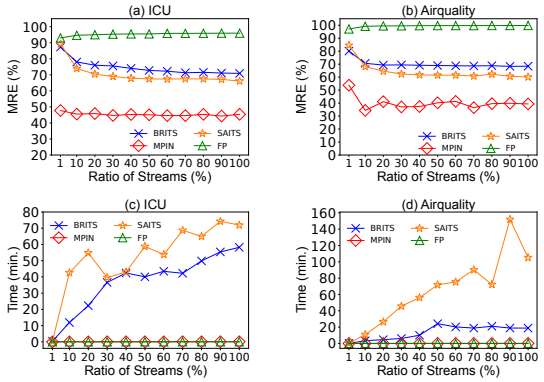

Figure 8: Effect of concurrent streams.

5.2.4 Effect of Number of Concurrent Streams. We vary the ratio (number) of concurrent streams to study its impact on imputation quality. Referring to Figures 8 (a) and (b), as the ratio of streams increases, the MRE of MPIN, BRITS, and SAITS initially decreases rapidly and then stabilizes. When the ratio of streams is low, the three neural network models struggle to impute the streams effectively due to the limited amount of data instances for training (imputation). When the ratio of the streams is higher, their imputation performance is improved due to the additional training data. Furthermore, when the ratio of the streams is excessively high, the advantages diminish as the current data may already be sufficient for training. During the whole course, SAITS and BRITS improve more significantly than MPIN. The former are sequential neural network models, and the higher ratio of the concurrent streams means

that more sequences can be used by their sequential modules to capture temporal dependencies. However, MPIN always outperforms BRITS and SAITS even when the ratio of the streams reaches 100%. On the other hand, FP shows minimal sensitivity to the ratio of the streams as it is not a neural network model. However, FP is always the least effective as it simply relies on adjacency matrix multiplications. These findings indicate that the scarcity of data within a time window adversely affects the imputation effectiveness.

Regarding time costs, Figures 8 (c) and (d) show that BRITS and SAITS experience degradation as the ratio of concurrent streams increases. Conversely, MPIN and FP maintain fast and stable performance. The increase in concurrent streams implies that more time series need to be processed by the sequential models BRITS and SAITS, resulting in higher time costs. In contrast, MPIN is non-sequential and can process graph-structured data in parallel. In cases where a large data chunk exceeds the available memory, we can split it into medium-sized data chunks and impute each of them individually. This way will minimize the effect of large data chunks and keep the whole process still fast.

5.3 Evaluation on Continuous Imputation

Methods. We assess the integration of MPIN with the techniques proposed in Section 4 for continuous data stream computation, involving the following methods:

- **MPIN-P** periodically calls MPIN to impute the missing values of a data chunk at each time window.
- **MPIN-D** equips MPIN with a data update mechanism to preserve and update valuable data instances for improved imputation at the next window.
- **MPIN-M** equips MPIN with a model update mechanism that avoids training the model from scratch, and instead resumes training from the best-ever state.
- **MPIN-DM** combines both data update and model update mechanisms with MPIN.

Parameter Settings. Since the proposed continuous techniques primarily address concerns related to data quantity, such as data scarcity within a window, we focus on varying the ratios of concurrent streams to create data chunks of different sizes within the window. The ratios of streams are adjusted at intervals of one order of magnitude, ranging from 0.1%, 1%, 10%, to 100%. The Wi-Fi dataset, which consists of a single stream, is excluded from the variation. Other parameters, such as window length, are kept as defaults. The reported results are averaged across time windows.

5.3.1 Effect of Number of Concurrent Streams. Referring to Figures 9 (a) and (b), both MPIN-DM and MPIN-D exhibit lower MRE values compared to MPIN-M and MPIN-P in most tested cases. This difference becomes more prominent when the ratio of streams is small, such as 0.1% or 1%. Both MPIN-DM and MPIN-D incorporate a data update mechanism, allowing them to utilize valuable past instances and enhance the effectiveness of subsequent imputations. When encountering data scarcity within a window, the data update becomes particularly significant as it mitigates the impact of scarcity by augmenting the available data. Moreover, the inclusion of model update in MPIN-DM further enhances its effectiveness compared to MPIN-D, as it enables MPIN to converge to a better local optimum during training. This also explains why MPIN-M is more effective than MPIN-P in most cases. However, MPIN-M may still occasionally be less effective than MPIN-P, e.g., when ratio=1.0% on ICU. We attribute this to data distribution drift between windows, which renders the strategy of reusing states from previous windows less effective. Moreover, we notice that the MRE of methods does not strictly follow a monotonic trend as the ratio of streams increases or decreases. Nonetheless, in general, the MRE tends to be higher when data is scarce (e.g., 0.1%) and lower when data is sufficient (e.g., 100%). Overall, MPIN-DM, MPIN-D, and MPIN-M outperform MPIN-P, highlighting the effectiveness of the proposed incremental techniques.

On the other hand, referring to Figures 9 (c) and (d), in most cases, MPIN-DM is more efficient than MPIN-D, and MPIN-M is more efficient than MPIN-P. This efficiency advantage arises from the model update employed in both MPIN-DM and MPIN-M, which allows them to avoid training from scratch and thus reduce time costs. However, in a few cases (e.g., ratio=1.0%), the time cost of MPIN-M exceeds that of MPIN-P. This is likely because the distribution of data between windows may change occasionally, thus making direct reuse of model states from previous windows less effective at training with the data in the current window. Consequently, training MPIN-M (also imputing) converges more slowly than when training from scratch (i.e., MPIN-P). We also notice that MPIN-DM is more efficient than MPIN-M. The data update strategy, i.e., reusing some data from previous windows, can alleviate data distribution drift between windows and can accelerate the model training based on previous model states. In general, MPIN-D is the least efficient due to its training from scratch with additional data (i.e., those valuable data instances). However, when the ratio of the streams is high (i.e., 100%), the time cost of MPIN-D is reduced considerably. With large amounts of data, training MPIN may need fewer iterations to converge to a local optimum, which accelerates the training. Moreover, the additional valuable data instances may contribute to this as they provide high-value data instances to speed up the convergence process.

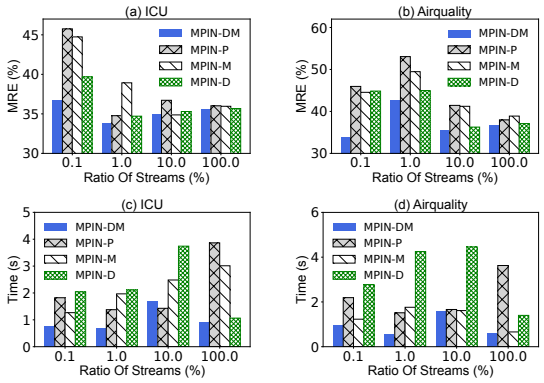

Figure 9: Continuous imputation on concurrent streams.

5.3.2 Effect of Irregular Sensor Streams. In previous experiments, the chosen ratio of concurrent streams remained constant across all windows, resulting in data chunks of the same size. However, in reality, streams may appear in an irregular or aperiodic manner, leading to sequential data chunks of varying sizes. To simulate this scenario, we introduce variations in the ratio of concurrent streams

in *each* time window, including values of 0.1%, 1%, 10%, and 100%. The imputation results are then averaged across all windows.

The results in Figure 10 reveal that methods incorporating data update exhibit lower MRE values compared to those without such a mechanism. Specifically, MPIN-DM outperforms MPIN-M and MPIN-D surpasses MPIN-P in terms of effectiveness.

Regarding time costs, methods using the model update mechanism are more efficient than those without — MPIN-DM outperforms MPIN-D and MPIN-M outperforms MPIN-P. Overall, in the presence of irregular streams, the proposed incremental techniques are also helpful. Notably, MPIN equipped with both data update and model update emerges as the most effective and efficient method.

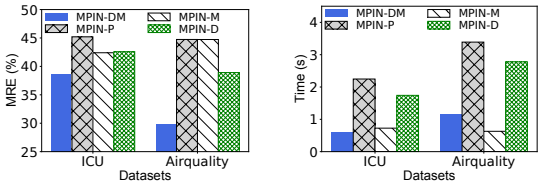

Figure 10: Performance on irregular streams.

Due to the space limit, we present additional experimental results and analyses in the extended version [1].

6 RELATED WORK

Traditional Imputers for Data Streams. Association rules-based data imputation methods [19, 22] target wireless sensor networks only and rely heavily on data characteristics. Next, KNN-based imputation uses the K neighbors' mean to replace the missing values in the instance [50, 51]. but simple averaging operations are ineffective on sparse data. Besides, multiple imputations by chained equation (MICE) [21] exploits chained equations and is more effective but it cannot handle high data sparsity or high dimensionality.

The top-K case matching [45] finds anchor points that share similar patterns across streams of time series. A matrix-based imputer [23] imputes time series streams by means of an incremental centroid decomposition of the matrix capturing the data. Unlike our work, these two studies consider only homogeneous time series and single-attribute cases. An experimental evaluation [24] compares matrix factorization-based methods [30, 34, 49] for data imputation.

Unlike the matrix-based methods, our proposals operate on a similarity graph that not only enables capturing correlation but also contains a positive relational bias [7, 14]. An instance in a graph only relates to the most similar neighbors. This positive bias provides more accurate and relevant information for data imputation.

Neural Network-based Imputers for Data Streams. Che et al. [11] use an adapted gated-recurrent unit (GRU) with masking and time-lag mechanisms to impute the missing values in sequences in an iterative fashion. Cao et al. [10] propose a classical bi-directional recurrent neural network for time series data imputation (BRITS). It adopts a transductive learning style and reconstructs missing values by reasoning from observed values in the sequences. Studies also exist that combine this RNN-based design with a generative adversarial network (GAN) [27–29, 32] to improve effectiveness. However, this combination may increase the training cost and cause unstable imputation results as GANs are often hard to train [31]. While Cini et al. [14] first integrate a message-passing unit with an RNN-based imputer to capture correlations among attributes, they target only homogeneous data. Recently, Du et al. [16] propose a self-attention-based imputation model to improve the efficiency of imputation. However, the full time cost remains high, rendering the model unsuitable for online imputation.

Overall, the imputation models above suffer from the following drawbacks. First, they are all based on sequential structures (either RNN or attention) and usually need much time for training. Second, they only capture intra-sequence temporal dependencies, but not inter-sequence dependencies. Third, some of them only work on periodic time series, while streams in practice are often aperiodic. In contrast, MPIN operates on a similarity graph and thus can exploit correlations among instances that may come from different streams or different times within a time window. Further, MPIN is amenable to fast training because it enables parallel operations on multiple nodes, thus exploiting available computing resources sufficiently.

Imputers for Graph-Structured Data. You et al. [48] propose to represent tabular data as a bipartite graph, where a node represents an attribute or a label, and then exploit an existing GNN and known labels to impute missing values of feature attributes. Spinelli et al. [38] propose an encoder-decoder model to impute missing values of graph nodes using label loss. Chen et al. [12] propose an end-to-end imputation process that takes the known labels and observed feature values into account to impute missing values and predict remaining unknown labels. All these imputation methods are designed to exploit (partially) known labels which, however, do not exist in our problem setting. More importantly, these methods apply existing GNN models directly, whereas we enable a message propagation process, with an accompanying theoretical analysis. Finally, the existing methods target only static datasets (e.g., a table) and fall short on data streams. The closest study to ours is feature propagation [37] (discussed in Section 3.4). That study aims to impute missing values of node features in a pre-existing graph that may lack homophily. In contrast, our approach builds graphs based on the similarity among data instances (see Section 3.1), and the resulting graphs certainly contain homophily to be exploited.

7 CONCLUSION

In this work, we have proposed a message propagation imputation network (MPIN) to accurately and efficiently impute missing values in the data instances of a time window in data streams. Moreover, we have proposed a continuous imputation framework with data and model update mechanisms to enable MPIN to support continuous imputation. Extensive experimental studies have demonstrated that the proposed data imputer is generally more accurate and more efficient than competitors and that the continuous framework is effective at continuous imputation on data streams.

In the future, it is of interest to deploy the proposed techniques in sensor data streaming systems. Also, it is interesting to explore data imputation on heterophilic graphs built on dissimilar data.

ACKNOWLEDGMENTS

This research was funded by Independent Research Fund Denmark (No. 8022-00366B). It was also supported in part by the Innovation Fund Denmark center DIREC. Moreover, Volker Markl gratefully acknowledges funding from the German Federal Ministry of Education and Research under the grant BIFOLD23B.

REFERENCES

[1] 2023. Extended Version. https://arxiv.org/pdf/2311.07344.pdf.

[2] 2023. MPIN project: code and datasets. https://github.com/XLI-2020/MPIN.

[3] 2023. SimpleImputer. https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html.

[4] Daniel J. Abadi, Donald Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stanley B. Zdonik. 2003. Aurora: a new model and architecture for data stream management. *VLDB J.* 12, 2 (2003), 120–139.

[5] Ines Arous, Mourad Khayati, Philippe Cudré-Mauroux, Ying Zhang, Martin L. Kersten, and Svetlin Stalinlov. 2019. RecovDB: Accurate and Efficient Missing Blocks Recovery for Large Time Series. In *35th IEEE International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019*. IEEE, 1976–1979.

[6] Melissa J Azur, Elizabeth A Stuart, Constantine Frangakis, and Philip J Leaf. 2011. Multiple imputation by chained equations: What is it and how does it work? *Int J Methods Psychiatr Res* 20, 1 (2011), 40–49.

[7] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Çaglar Gülçehre, H. Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey R. Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew M. Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. 2018. Relational inductive biases, deep learning, and graph networks. *CoRR* abs/1806.01261 (2018). arXiv:1806.01261 http://arxiv.org/abs/1806.01261

[8] Laure Berti-Équille, Hazar Harmouch, Felix Naumann, Noël Novelli, and Saravanan Thirumuruganathan. 2018. Discovery of Genuine Functional Dependencies from Relational Data with Missing Values. *Proc. VLDB Endow.* 11, 8 (2018), 880–892.

[9] José Cambronero, John K. Feser, Micah J. Smith, and Samuel Madden. 2017. Query Optimization for Dynamic Imputation. *Proc. VLDB Endow.* 10, 11 (2017), 1310–1321.

[10] Wei Cao, Dong Wang, Jian Li, Hao Zhou, Lei Li, and Yitan Li. 2018. BRITS: Bidirectional Recurrent Imputation for Time Series. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. 6776–6786.

[11] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David A. Sontag, and Yan Liu. 2018. Recurrent Neural Networks for Multivariate Time Series with Missing Values. *Scientific Reports* 8, 1 (2018), 6085. https://doi.org/10.1038/s41598-018-24271-9

[12] Katrina Chen, Xiuqin Liang, Zhibin Zhang, and Zheng Ma. 2022. GEDI: A Graph-based End-to-end Data Imputation Framework. *CoRR* abs/2208.06573 (2022). https://doi.org/10.48550/arXiv.2208.06573 arXiv:2208.06573

[13] Yixin Chen, Guozhu Dong, Jiawei Han, Benjamin W. Wah, and Jianyong Wang. 2002. Multi-Dimensional Regression Analysis of Time-Series Data Streams. In *Proceedings of 28th International Conference on Very Large Data Bases, VLDB 2002, Hong Kong, August 20-23, 2002*. Morgan Kaufmann, 323–334.

[14] Andrea Cini, Ivan Marisca, and Cesare Alippi. 2021. Multivariate Time Series Imputation by Graph Neural Networks. *CoRR* abs/2108.00298 (2021). arXiv:2108.00298

[15] Andrea Cini, Ivan Marisca, and Cesare Alippi. 2022. Filling the G_ap_s: Multivariate Time Series Imputation by Graph Neural Networks. In *International Conference on Learning Representations*.

[16] Wenjie Du, David Côté, and Yan Liu. 2023. SAITS: Self-attention-based imputation for time series. *Expert Syst. Appl.* 219 (2023), 119619.

[17] Lukas Galke, Iacopo Vagliano, and Ansgar Scherp. 2020. Incremental Training of Graph Neural Networks on Temporal Graphs under Distribution Shift. *CoRR* abs/2006.14422 (2020). arXiv:2006.14422 https://arxiv.org/abs/2006.14422

[18] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017 (Proceedings of Machine Learning Research)*, Vol. 70. PMLR, 1263–1272.

[19] Mihail Halatchev and Le Gruenwald. 2005. Estimating Missing Values in Related Sensor Data Streams. In *Advances in Data Management 2005, Proceedings of the Eleventh International Conference on Management of Data, January 6, 7, and 8, 2005, Goa, India*, Jayant R. Haritsa and T. M. Vijayaraman (Eds.). Computer Society of India, 83–94.

[20] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* 30 (2017).

[21] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition*. Springer. https://doi.org/10.1007/978-0-387-84858-7

[22] Nan Jiang and Le Gruenwald. 2007. Estimating Missing Data in Data Streams. In *Advances in Databases: Concepts, Systems and Applications, 12th International Conference on Database Systems for Advanced Applications, DASFAA 2007, Bangkok, Thailand, April 9-12, 2007, Proceedings (Lecture Notes in Computer Science)*, Vol. 4443. Springer, 981–987.

[23] Mourad Khayati, Ines Arous, Zakhar Tymchenko, and Philippe Cudré-Mauroux. 2020. ORBITS: Online Recovery of Missing Values in Multiple Time Series Streams. *Proc. VLDB Endow.* 14, 3 (2020), 294–306.

[24] Mourad Khayati, Alberto Lerner, Zakhar Tymchenko, and Philippe Cudré-Mauroux. 2020. Mind the Gap: An Experimental Evaluation of Imputation of Missing Values Techniques in Time Series. *Proc. VLDB Endow.* 13, 5 (2020), 768–782.

[25] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=SJU4ayYgl

[26] Xiao Li, Huan Li, Harry Kai-Ho Chan, Hua Lu, and Christian S. Jensen. 2023. Data Imputation for Sparse Radio Maps in Indoor Positioning. In *39th IEEE International Conference on Data Engineering, ICDE 2023, Anaheim, CA, USA, April 3-7, 2023*. IEEE, 2235–2248.

[27] Yukai Liu, Rose Yu, Stephan Zheng, Eric Zhan, and Yisong Yue. 2019. NAOMI: Non-Autoregressive Multiresolution Sequence Imputation. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. 11236–11246.

[28] Yonghong Luo, Xiangrui Cai, Ying Zhang, Jun Xu, and Xiaojie Yuan. 2018. Multivariate Time Series Imputation with Generative Adversarial Networks. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. 1603–1614.

[29] Yonghong Luo, Ying Zhang, Xiangrui Cai, and Xiaojie Yuan. 2019. E^2GAN: End-to-End Generative Adversarial Network for Multivariate Time Series Imputation. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. ijcai.org, 3094–3100.

[30] Jiali Mei, Yohann de Castro, Yannig Goude, and Georges Hébrail. 2017. Nonnegative Matrix Factorization for Time Series Recovery From a Few Temporal Aggregates. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017 (Proceedings of Machine Learning Research)*, Vol. 70. PMLR, 2382–2390.

[31] Lars M. Mescheder, Andreas Geiger, and Sebastian Nowozin. 2018. Which Training Methods for GANs do actually Converge?. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018 (Proceedings of Machine Learning Research)*, Vol. 80. PMLR, 3478–3487.

[32] Xiaoye Miao, Yangyang Wu, Jun Wang, Yunjun Gao, Xudong Mao, and Jianwei Yin. 2021. Generative semi-supervised learning for multivariate time series imputation. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 8983–8991.

[33] Katsiaryna Mirylenka, Graham Cormode, Themis Palpanas, and Divesh Srivastava. 2015. Conditional heavy hitters: detecting interesting correlations in data streams. *VLDB J.* 24, 3 (2015), 395–414.

[34] Spiros Papadimitriou, Jimeng Sun, and Christos Faloutsos. 2005. Streaming Pattern Discovery in Multiple Time-Series. In *Proceedings of the 31st International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September 2, 2005*. ACM, 697–708.

[35] Kostas Patroumpas and Timos K. Sellis. 2006. Window Specification over Data Streams. In *Current Trends in Database Technology - EDBT 2006, EDBT 2006 Workshops PhD, DataX, IIDB, IIHA, ICSNW, QLQP, PIM, PaRMA, and Reactivity on the Web, Munich, Germany, March 26-31, 2006, Revised Selected Papers (Lecture Notes in Computer Science)*, Vol. 4254. Springer, 445–464.

[36] Weilong Ren, Xiang Lian, and Kambiz Ghazinour. 2019. Skyline queries over incomplete data streams. *VLDB J.* 28, 6 (2019), 961–985.

[37] Emanuele Rossi, Henry Kenlay, Maria I. Gorinova, Benjamin Paul Chamberlain, Xiaowen Dong, and Michael M. Bronstein. 2022. On the Unreasonable Effectiveness of Feature Propagation in Learning on Graphs With Missing Node Features. In *Learning on Graphs Conference, LoG 2022, 9-12 December 2022, Virtual Event (Proceedings of Machine Learning Research)*, Vol. 198. PMLR, 11.

[38] Indro Spinelli, Simone Scardapane, and Aurelio Uncini. 2020. Missing data imputation with adversarially-trained graph convolutional networks. *Neural Networks* 129 (2020), 249–260.

[39] Thanh T. L. Tran, Liping Peng, Yanlei Diao, Andrew McGregor, and Anna Liu. 2012. CLARO: modeling and processing uncertain data streams. *VLDB J.* 21, 5 (2012), 651–676.

[40] Jonas Traub, Philipp M. Grulich, Alejandro Rodriguez Cuellar, Sebastian Breß, Asterios Katsifodimos, Tilmann Rabl, and Volker Markl. 2019. Efficient Window Aggregation with General Stream Slicing. In *Advances in Database Technology - 22nd International Conference on Extending Database Technology, EDBT 2019, Lisbon, Portugal, March 26-29, 2019*. OpenProceedings.org, 97–108.

[41] Olga G. Troyanskaya, Michael N. Cantor, Gavin Sherlock, Patrick O. Brown, Trevor Hastie, Robert Tibshirani, David Botstein, and Russ B. Altman. 2001. Missing value estimation methods for DNA microarrays. *Bioinform.* 17, 6 (2001), 520–525.

[42] Vladimir Vapnik. 2006. *Estimation of Dependences Based on Empirical Data, Second Editiontion.* Springer. https://doi.org/10.1007/0-387-34239-7

[43] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=rJXMpikCZ

[44] Haixun Wang, Wei Wang, Jiong Yang, and Philip S. Yu. 2002. Clustering by pattern similarity in large data sets. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, USA, June 3-6, 2002*. ACM, 394–405.

[45] Kevin Wellenzohn, Michael H. Böhlen, Anton Dignös, Johann Gamper, and Hannes Mitterer. 2017. Continuous Imputation of Missing Values in Streams of Pattern-Determining Time Series. In *Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017*. OpenProceedings.org, 330–341.

[46] Steven Whang and Jae-Gil Lee. 2020. Data Collection and Quality Challenges for Deep Learning. *Proc. VLDB Endow.* 13, 12 (2020), 3429–3432.

[47] Susik Yoon, Jae-Gil Lee, and Byung Suk Lee. 2019. NETS: Extremely Fast Outlier Detection from a Data Stream via Set-Based Processing. *Proc. VLDB Endow.* 12, 11 (2019), 1303–1315.

[48] Jiaxuan You, Xiaobai Ma, Daisy Yi Ding, Mykel J. Kochenderfer, and Jure Leskovec. 2020. Handling Missing Data with Graph Representation Learning. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

[49] Hsiang-Fu Yu, Nikhil Rao, and Inderjit S. Dhillon. 2016. Temporal Regularized Matrix Factorization for High-dimensional Time Series Prediction. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. 847–855.

[50] Peng Zhang, Xingquan Zhu, Jianlong Tan, and Li Guo. 2010. SKIF: a data imputation framework for concept drifting data streams. In *Proceedings of the 19th ACM Conference on Information and Knowledge Management, CIKM 2010, Toronto, Ontario, Canada, October 26-30, 2010*. ACM, 1869–1872.

[51] Shichao Zhang. 2012. Nearest neighbor selection for iteratively kNN imputation. *Journal of Systems and Software* 85, 11 (2012), 2541–2552.