Editor: A Multi-Resolution Framework for Multivariate Time Series Data Cleaning

Abstract

Multivariate Time Series (MTS) data, consisting of multiple variables recorded over time, are widely used in applications such as healthcare monitoring and industrial analytics. Such data are often prone to errors (e.g., noise and inconsistencies), leading to unreliable outcomes or even system failures. These errors display diverse patterns, occurring over varying time spans and involving different variables. Most data cleaning methods perform error detection and repair with a fixed resolution, typically within a predefined time window and across all variables, which struggle to accommodate the diverse and irregular error patterns. In this paper, we present Editor, a multi-resolution framework for MTS data cleaning, which integrates deep learning models to effectively handle diverse error patterns. At a high level, Editor performs coarse-grained **detection** at the window level to ensure efficiency, followed by fine-grained **location** within each erroneous window to preserve accuracy, and concludes with context-aware **repair** to correct identified errors while avoiding over-cleaning. Specifically, the detection module employs an enhanced autoencoder that sensitively identifies errors of varying magnitudes across time windows, effectively reducing missed detections. Next, the localization module adaptively pinpoints multi-scale errors by solving an optimization problem that minimizes disruptions to the original clean data, thereby reducing false positives. Finally, the repair module introduces a two-stage strategy that captures both temporal dependencies and inter-variable relationships in MTS, enabling accurate corrections across diverse error patterns. Extensive experiments show that Editor improves accuracy by 10%–25% over state-of-the-art methods, demonstrating its effectiveness in reliable MTS cleaning.

CCS Concepts

• **Information systems** → **Data cleaning**.

Keywords

Data cleaning, MTS, Multi-Resolution

ACM Reference Format:

. 2018. Editor: A Multi-Resolution Framework for Multivariate Time Series Data Cleaning. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 13 pages. https://doi.org/XXXXXXX.XXXXXXX

Conference acronym 'XX, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Multivariate Time Series (MTS) data consist of multiple interrelated variables recorded sequentially over time, each representing distinct but interconnected dimensions. MTS data are ubiquitously generated across critical application domains, including healthcare (e.g., patient vital signs such as heart rate, blood pressure, and oxygen levels [25]), industrial monitoring (sensor readings from machinery [27]), and financial analysis [31]. Accurate analysis of MTS data underpins essential tasks such as classification, clustering, and predictive modeling, significantly influencing decision-making quality and effectiveness.

However, real-world MTS data frequently suffer from quality issues, including noise, inconsistent formatting, and data drift [14, 15, 17, 33]. Such quality degradation can seriously impair the accuracy and reliability of subsequent analyses, potentially leading to incorrect or biased insights. Additionally, certain applications, such as astronomical observations, exhibit seemingly random error patterns due to external interferences like extreme weather, significantly complicating error identification and correction [21].

Although MTS cleaning is critical to ensure downstream analytical quality, existing cleaning methods fall short in effectively addressing three fundamental challenges (illustrated in Fig. 1 using Server Machine Data [41]):

(C1) Errors of varying magnitudes. Errors in MTS data range from clearly detectable high-magnitude anomalies (e.g., timestamps 48–49 on Var5 in Fig. 1) to subtle, low-magnitude deviations (e.g., timestamps 58–59 on Var5 or timestamp 58 on Var1). Constraint-based methods [14, 32, 33, 39] excel at capturing large deviations but fail to detect subtle anomalies, mistaking them for normal variations. Conversely, probabilistic approaches [36, 42] effectively capture subtle errors but misinterpret extreme legitimate values as anomalies [20]. No existing approach uniformly handles these varying magnitudes effectively.

(C2) Errors at multiple granularities. Errors in MTS data occur at multiple granularities—single data points (e.g., timestamp 58 on Var1 in Fig. 1), short subsequences (e.g., timestamp 48-51 on Var5), or concurrent errors across multiple variables (e.g., timestamp 58 on both Var1 and Var5). Current approaches, predominantly operating at fixed-size windows [4, 8, 12, 35], often lead to over-generalization, flagging correct data as errors or missing nuanced anomalies. This rigidity generates unnecessary corrections (deuterogenic errors) and undermines cleaning accuracy.

(C3) Errors violating complex temporal and variable dependencies. MTS data inherently contain complex temporal dependencies and variable interactions, both linear and nonlinear, that existing approaches insufficiently leverage. Expression-based methods [7, 13–15] handle linear relationships (e.g., Var2 and Var3 between timestamps 20–23 in Fig. 1) but miss nonlinear interactions (e.g., Var2 and Var3 between timestamps 23–28), while smoothing techniques [9, 16] capture short-term dependencies (e.g., Var5 within W_A) but fail on long-term patterns (e.g., Var4 spanning W_A

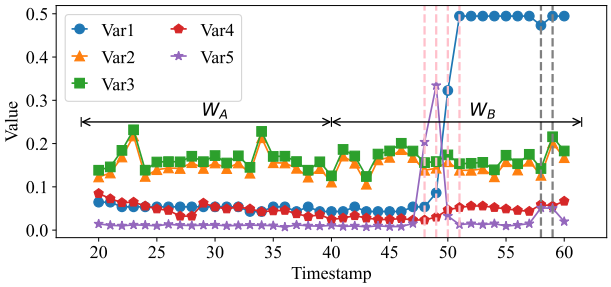


Fig. 1: Example of Server Machine Dataset [41].

and W_B). Deep learning methods [4, 12, 35] model nonlinear and long-term dependencies but inadequately address challenges C1 and C2, limiting the overall cleaning effectiveness.

To address these challenges, we propose **Editor**, a novel multi-resolution MTS cleaning framework that incorporates deep learning models to effectively handle diverse and complex error patterns. Specifically, Editor offers specialized detection, localization, and repair modules:

(1) **Detection: Sensitive Identification of Erroneous Windows.** This module employs an enhanced autoencoder augmented with dual-attention mechanisms and a difference-enhanced technique, enabling accurate window-level detection of errors spanning various magnitudes (addressing C1).
(2) **Localization: Precise Identification within Erroneous Windows.** Editor adopts an adaptive, scale-aware localization strategy formulated as an optimization problem solved via ameliorative evolutionary algorithms. This ensures precise pinpointing of erroneous data points, subsequences, and co-occurrences, which significantly reduces false positives and collateral data damage (addressing C2).
(3) **Repair: Context-aware Data Restoration.** Utilizing a two-stage approach combining Temporal Convolutional Networks (TCN) and Graph Convolutional Networks (GCN), Editor captures both short-term and long-term temporal dependencies and linear/nonlinear variable interactions. This comprehensive modeling ensures robust repair aligned with contextual patterns (addressing C3).

The main contributions of this paper are summarized as follows.

- We propose a unified multi-resolution MTS cleaning framework (Editor) that effectively handles errors of varying magnitudes, scales, and dependencies.
- We develop a sensitive and efficient detection module enhancing accuracy across diverse error magnitudes.
- We devise an adaptive localization method ensuring precise, scale-aware identification of erroneous data points, minimizing unnecessary corrections.
- We present a robust, context-aware repair module that comprehensively integrates temporal and variable relationships.
- We conduct extensive evaluation of Editor on four real-world datasets, and results demonstrate that Editor achieves significant accuracy improvements (10%–25%) over state-of-the-art methods.

The remainder of this paper is organized as follows: Section 2 presents preliminaries and provides an overview of Editor. Sections 3, 4, and 5 detail the detection, localization, and repair modules, respectively. Section 6 provides experimental evaluations. Section 7 discusses related work, and Section 8 concludes the paper.

Tab. 1: Notations

Symbol	Description
D	Multivariate time series data with M variables
x_t	Observation at the t-th timestamp in D
x_t^m	Observed value of the m-th variable at timestamp t
W	A detection window of length l on M variables
C	A cell in W with length v on d variables
S	The final optimized repair values of a window
ϕ_{ts}, ϕ_{tl}	Short- and long-term temporal dependencies
ϕ_{vl}, ϕ_{vn}	Linear and nonlinear variable relationships

2 Preliminaries and Editor Overview

This section introduces the preliminaries, including key concepts and requirements associated with MTS cleaning. We also present the high-level design of Editor. The notations used throughout this paper are summarized in Tab. 1.

2.1 Preliminaries

Definition 1. ***Multivariate Time Series (MTS).*** *A multivariate time series data $D = \{x_1, x_2, ..., x_T\}$ consists of observations recorded over T timestamps. At each timestamp $t \in [0, T]$, the observation $x_t = \{x_t^1, x_t^2, ..., x_t^M\}$ represents the values of M variables, where x_t^m denotes the observation of the m-th ($m \in [1, M]$) variable at timestamp t. MTS inherently exhibit two dimensions: temporal (time-dependent relationships) and variable (inter-variable dependencies).*

Definition 2. ***Detection Window and Cell.*** *We define a detection window as $W[t, t+l]$, where $[t, t+l]$ represents a consecutive time interval from t to $t+l$, and W corresponds to the projection of D on $[t, t+l]$, with $0 \le t < t+l \le T$. Likewise, we term $C^d[t, t+v]$ as a cell to denote the projection of W on d variables (i.e., a subset of the M variables) within a time interval $[t, t+v]$, where $1 \le d \le M$ and $1 \le v \le l$. For brevity, we simplify $W[t, t+l]$ and $C^d[t, t+v]$ as W and C, respectively, when there is no ambiguity.*

Requirements for Effective MTS Cleaning. Driven by the inherent characteristics of MTS data, we derive several key requirements for effective MTS Cleaning.

- R1. **Detecting Errors of Varying Degrees of Deviation.** Errors in MTS can vary greatly in magnitude, from subtle deviations (e.g., small noise in a sensor reading) to large outliers (e.g., a sensor malfunction). Cleaning methods must detect and repair errors across this spectrum without omitting errors, which is especially challenging for subtle deviations.
- R2. **Handling Errors at Multiple Scales.** Errors in MTS can vary in durations and affect different numbers of variables, ranging from point-level errors (e.g., transient signal loss due to poor ECG contact) to subsequence-level errors (e.g., prolonged low battery in an ECG device), and even concurrent or seemingly random errors cross different variables (e.g., simultaneous or sporadic failures of sensor devices due to a power outage). Cleaning methods must accurately locate and repair these multi-scale errors without damaging valid data.
- R3. **Modeling Temporal Dependencies (short- and long-term).** Errors in MTS violate the relationships between observations over time. We use intra-window dependencies ($\phi_{ts}(W)$) to refer to relationships over adjacent points within a window (i.e.,

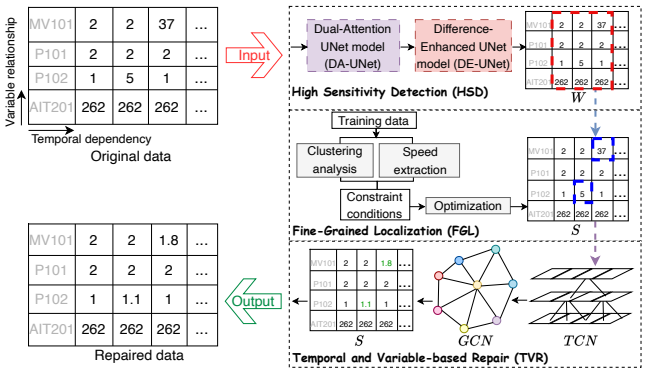


Fig. 2: Editor overview

short-term), and use inter-window dependencies ($\phi_{tl}(W_i, W_j)$) to refer to those that span time intervals larger than a window (i.e., long-term). Cleaning methods must model these dependencies to distinguish valid variations from errors, such as identifying correlations between a spike and subsequent stabilization in a sensor reading.

- R4. **Capturing Variable Relationships (linear and nonlinear).** Errors disrupt the inter-variable correlations in MTS, either linear (e.g., one variable being a scaled version of another) or nonlinear (e.g., temperature-dependent pressure). We use notations ϕ_{vl} and ϕ_{vn} to refer to linear and nonlinear relationships, respectively. Cleaning methods must incorporate these relationships to avoid erroneous repairs that disrupt the inter-variable structure.

2.2 Editor Overview

In response to the above requirements, we propose Editor, a multi-resolution cleaning framework for MTS that integrates deep learning models to handle diverse error patterns and capture complex temporal and inter-variable dependencies. Specifically, as depicted in Fig. 2, Editor consists of three modules: High Sensitivity Detection (HSD), Fine-Grained Localization (FGL), and Temporal and Variable-based Repair (TVR).

High Sensitivity Detection (HSD). This module detects errors at the window level, balancing efficiency and accuracy. Specifically, HSD adopts a Dual-Attention UNet (DA-UNet), enhanced with a Difference-Enhanced UNet (DE-UNet), to capture both high- and low-deviation errors. The dual-attention mechanism effectively models the complex temporal and inter-variable characteristics of MTS, while the difference-enhanced technique amplifies deviations, enabling sensitive detection of errors across varying magnitudes and reducing missed errors (Satisfying R1, R3, R4).

Fine-Grained Localization (FGL). After HSD identifies an erroneous detection window, FGL pinpoints the specific error cells within the window. This process reduces false positives and minimizes interference with correct data through adaptive scale awareness. FGL formalizes error localization as an optimization problem, using inherent variable relationships and temporal dependencies in MTS as constraints. An ameliorative evolutionary algorithm is employed to precisely locate fine-grained error cells within the detection window (Satisfying R2).

Temporal and Variable-based Repair (TVR). Once error cells are identified, TVR repairs them with accurately generated values. Specifically, TVR follows a two-stage repair process: first, TCNs model temporal dependencies and generate preliminary repair values; then, GCNs further refines these by incorporating variable relationships within the temporal context, obtaining the final optimized repair values. This ensures robust correction of diverse error patterns by integrating both temporal and variable features (Satisfying R3, R4).

In Editor, to ensure consistent and comparable processing across variables in MTS data, all observed values x_t^m are normalized to the range $[0, 1]$. This step mitigates the impact of varying dimensions across variables, simplifies computations, and enhances the performance of the algorithms and machine learning models used throughout Editor.

Workflow. Editor processes the dirty MTS data as follows: HSD detects erroneous detection windows by comparing reconstructed and input data, FGL further locates error cells within the windows, and TVR generates repaired values for these cells, resulting in clean and accurate MTS data.

3 High sensitivity detection

This section first introduces the components of the High Sensitivity Detection (HSD) module, and then describes its training and detection processes.

3.1 Components of HSD module

HSD aims to identify whether there exist errors in the detection window W. To achieve this, we propose a reconstruction-based strategy, that utilizes an encoder to extract features of W, and then reconstructs W from features by decoder. Considering the complex characteristics of MTS and varying deviations mentioned in Section 2.1, we design the Dual-Attention UNet model (DA-UNet) and Difference-Enhanced UNet model (DE-UNet), rather than directly applying the traditional AutoEncoder [6] (as described in next section). Below, we first illustrate the DA-UNet, which is capable of efficiently generating reconstructed data, then describe the DE-UNet to detect different deviations.

3.1.1 Dual-Attention UNet model (DA-UNet). To encode MTS, one of the most commonly used models is AutoEncoder (AE), which is widely employed for data reduction and feature representation [6]. Nonetheless, the original encoder in AE may lose the indispensable features necessary for the data reconstruction [26]. Therefore, we apply UNet as the base model, which is a popular deep learning model in image segmentation and video processing tasks [29, 34]. The UNet adopts the same encoder-decoder structure as AE, but address its shortcomings with more complex designs that enhance the recovery of detailed features during decoding, enabling more efficient error detection.

However, the existing UNet models [29, 34] are insufficient to accurately generate reconstructed data, as it assigns equal weights to all features in the process of encoding and decoding, failing to dynamically adjust their importance.

To this end, we propose the DA-UNet, which not only inherits the advantages of UNet but also addresses its limitations. DA-UNet integrates two distinct attention mechanisms in the encoder and decoder, respectively, thereby achieving dual-attention optimization. Unlike existing studies [23] that apply attention only at encoder's

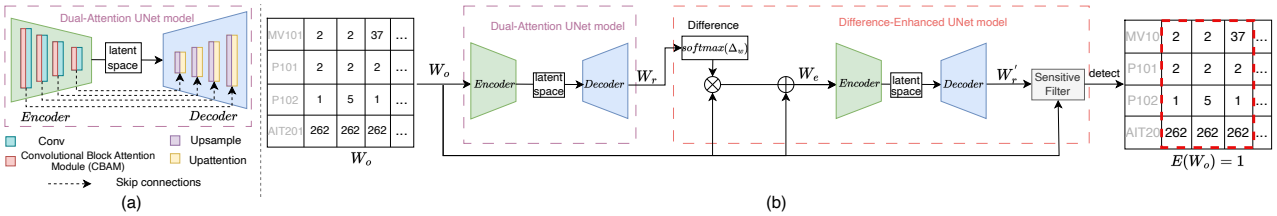


Fig. 3: (a) Dual-Attention UNet model, (b) High Sensitivity Detection based on DA-UNet and DE-UNet

final layer, our DA-UNet incorporates attention into all layers of both the encoder and decoder. This allows dynamic weighting features across temporal and variable dimensions, effectively capturing both temporal and variable dependencies. As a result, it enhances the network's ability to understand both local and global temporal patterns as well as cross-variable dependencies. Specifically, DA-UNet effectively captures local features, including short-term patterns in the time dimension (e.g., short-term volatility within a window (ϕ_{ts})) and localized relationships in the variable dimension (e.g., simple interactions between two variables (ϕ_{vl})). At the same time, it extracts global features, including long-term trends in the time dimension (e.g., stability across windows (ϕ_{tl})) and complex dependencies among multiple variables (ϕ_{vn}). Furthermore, DA-UNet achieves effective feature representation in the encoder and accurate reconstruction in the decoder, while maintaining high computational efficiency and inference speed.

As shown in Fig. 3(a), the DA-UNet model comprises an encoder and a decoder. On the one hand, the encoder process includes the Convolution layer (Conv) and Convolutional Block Attention Module (CBAM). In Conv, a convolutional kernel slides over W_o (a detection window) with a fixed stride to extract local features among the dimensions of timestamps and variables. While CBAM further consists of channel and spatial modules. The channel module of CBAM computes the *average pooling* and *max pooling* for features obtained from Conv, and then generates attention weights based on the results of these two pooling operations (via a shared fully connected layer and a *sigmoid* activation function as per [34]). In this way, the channel module assists the model in dynamically adjusting the weights of extracted features in latent space according to the importance of different temporal and variables. After the channel module, the spatial module of CBAM further weights the features at each variable location, emphasizing key variable features and suppressing unimportant ones. Moreover, the encoder employs a stacked structure that combines Conv and CBAM, gradually expanding the receptive field to capture the global temporal and variable characteristics of the input W_o.

On the other hand, the decoder of DA-UNet is composed of a stacked structure that combines Upsample, Upattention and Skip Connections. The Upsample operation restores the sizes of features extracted from the previous layer. Meanwhile, Upattention is the second attention mechanism we incorporate, which calculates an attention weight map on the features from both the encoder and decoder, emphasizing the important features. The Skip Connections directly pass the features extracted at each layer of the encoder to the corresponding layer of the decoder, and fuse them (through *concatenation* operation) with the features at that layer of the decoder. Its advantage lies in preserving the low-level detailed features from the encoder, which might be otherwise gradually lost during the layer-by-layer Conv and CBAM process. By directly passing these features to the decoder, it helps the decoder to more accurately reconstruct the details, thereby improving the generation quality of the model, making up for the disadvantages of traditional AE. Through these interconnected components, the extracted features are progressively restored to the size of input W_o, enabling the efficient reconstruction from the encoded features.

3.1.2 Difference-Enhanced UNet model (DE-UNet). Rather than directly applying the output of DA-UNet to identify windows with high-deviation errors, and may lead to omission of low-deviation errors, we propose DE-UNet, which integrates a difference-enhanced technique to efficiently capture the varying deviations.

As illustrated in Fig. 3(b), the DE-UNet consists of three steps: 1) differences-based enhancement, 2) encoding and decoding, and 3) filtering. In the following, we detail each step.

Differences-based enhancement. This step enhances W_o by incorporating differences, producing W_e, which emphasizes the deviation features. As described above, the differences Δ_w between the reconstructed data W_r from DA-UNet and the input W_o struggle to detect low-deviation errors in W_o. Particularly, in some low-deviation regions, the differences are too small to be effectively recognized, which may result in omission. To address this issue, the differences undergo a *softmax* operation, transforming into a probability distribution, where each element falls within the range of [0,1], and the sum of all elements equals 1. Additionally, these transformed differences are multiplied element-wise with the input data W_o to obtain discrepancy coefficients, which are then fused with the input data W_o to generate the enhanced version of W_o, denoted as W_e ($W_e = [softmax(\Delta_w) \otimes W_o] \oplus W_o$). [1] This enhancement ensures that after *softmax* normalization, $\oplus$ and $\otimes$ operations, even low-deviation regions with small reconstruction errors are assigned relatively high weights, ensuring they are not misinterpreted as correct data and ignored during the subsequent encoding-decoding process. The whole process achieves the adaptive adjustment of W_o, ensuring that the deviation features in different regions are effectively emphasized.

Encoding and decoding. To improve sensitivity to low-deviation errors, DE-UNet further leverages the feature information of the enhanced data W_e through an encoding-decoding process. Specifically, the encoder in DE-UNet takes W_e as input to capture features, and since the enhanced information of deviation regions is embedded in W_e, the encoder learns significant features variations, capturing

[1] $\oplus$, $\otimes$ denote element-wise addition and multiplication, respectively. The enhanced data W_e amplifies the deviation regions, allowing the encoder to more effectively capture deviation information in subsequent steps.

the features with discrepancies and de-emphasizing those without. This process facilitates the visibility of deviations, particularly low-deviation errors, in the feature space. Then, the decoder generates reconstructed data $W_r^{'}$ with features of amplified low deviations due to its ability to more accurately reconstruct correct regions [38].

Filtering. This step builds a Sensitive Filter (SF), which designs a dynamic tolerance threshold $\mathcal{T}$, to detect erroneous windows. Specifically, it first calculates the distance between $W_r^{'}$ and W_o, and then computes the L_2 Norm of the distance at each timestamp as the deviation score γ_i, given by $\gamma_i = \|W_{ri}^{'} - W_{oi}\|_2$ (where i denotes the i-th timestamp, $1 \leq i \leq l$, and l is the size of time interval span by W). From the range of these deviation scores γ_i, we iteratively identify the score that effectively distinguishes between erroneous and correct data, defining it as the tolerance deviation threshold $\mathcal{T}$. By comparing each γ_i with $\mathcal{T}$, a γ_i exceeding $\mathcal{T}$ indicates that at least one variable at that timestamp is erroneous, and then we label this window as erroneous. We evaluate the influence of different values of l in Section 6.

Therefore, the DE-UNet reconstructs $W_r^{'}$ with enhanced differences, facilitating the sensitive and accurate detection of errors across varying magnitudes. To verify the high sensitivity of DE-UNet, we conducted an ablation experiment (see "High Sensitivity Detection" of A6 in Section 6), which demonstrates the necessity of DE-UNet.

3.2 HSD training process

Below, we present the training process of HSD, arranged sequentially on DA-UNet and DE-UNet to generate the first and second reconstruction data, respectively. The overall training process (summarized in Algorithm 1) is conducted on training data D_c, divided into W_c using sliding windows.

DA-UNet reconstruction (lines 3-4). The DA-UNet is trained to capture the features from input data in each detection window W_c and generate the first reconstructed data W_r.

Particularly, the training objective of DA-UNet is to reproduce the input data W_c. The input W_c is first processed by the encoder to extract features and map them into the latent space, and then restore the original features to reconstruct the data W_r by the decoder. As described in Eq.1, the loss function for training DA-UNet aims to minimize the gap between W_r and W_c:

$$\mathcal{L}_{DA} = \min_{DA\text{-}UNet} \|W_r - W_c\|_2 \tag{1}$$

where $W_r = DA\text{-}UNet(W_c)$, representing the output of DA-UNet model. And $\|\cdot\|_2$ represents the L_2 Norm.

DE-UNet reconstruction (lines 5-6). The DE-UNet is then trained to learn from the enhanced data W_e derived from W_r and W_c, to generate a more accurate second reconstructed data $W_r^{'}$ with amplified discrepancies.

As shown in Eq. 2, the loss function of DE-UNet training is to minimize the gap between $W_r^{'}$ and W_e:

$$\mathcal{L}_{DE} = \min_{DE\text{-}UNet} \|W_r^{'} - W_e\|_2 \tag{2}$$

where $W_r^{'}$ is the output of DE-UNet model.

Algorithm 1 HSD Training

Require: DA-UNet and DE-UNet; training data D_c
1: Initialize weights → DA-UNet, DE-UNet
2: **for** each W_c in D_c **do** #W_c is obtained by sliding window on D_c
3: DA-UNet$(W_c) \rightarrow W_r$
4: $\|W_r - W_c\|_2 \rightarrow \mathcal{L}_{DA}$
5: DE-UNet$([softmax(\Delta_w) \otimes W_c] \oplus W_c) \rightarrow W_r^{'}$
6: $\|W_r^{'} - W_e\|_2 \rightarrow \mathcal{L}_{DE}$
7: Update weights of DA-UNet, DE-UNet through $\mathcal{L}_{DA}$, $\mathcal{L}_{DE}$

3.3 HSD Detection process

Given the testing data (i.e., the original data D_o with errors), HSD first applies the sliding window to form W_o. As illustrated in Fig. 3(b), the trained DA-UNet model takes W_o as input and generates the reconstructed W_r. Then, the trained DE-UNet model further constructs $W_r^{'}$ based on both W_r and W_o. Finally, we compare the deviation score γ for each timestamp in W_o with the tolerance deviation threshold $\mathcal{T}$ by SF. If any deviation score $any(\gamma)$ exceeds the tolerance deviation threshold $\mathcal{T}$, W_o is detected as erroneous (i.e., $E(W_o) = 1$); otherwise, it detects W_o as correct (i.e., $E(W_o) = 0$).

4 Fine-grained localization

Editor differs from existing two-step cleaning strategies (detection and repair) by introducing the FGL module, to further pinpoint fine-grained error cells within a window, adaptively perceiving error granularity to minimize false positives and collateral data damage. Below, we first formalize FGL as an objective optimization task and then introduce an ameliorative evolutionary algorithm to solve it.

4.1 FGL Rationale

As noted in Section 2.1, FGL refines error detection from window level to cell level to handle multi-granular errors. Moreover, to minimize missed errors, it aims to maximize error identification within data constraints while satisfying fine-grained requirements.

We recognize that this process can be formulated as an Objective Optimization (OO) problem. OO is a common approach to find optimal solutions within a solution space, and several studies [13, 30] have applied it to repair errors. Typically, OO involves defining a set of constraint conditions and an objective function to identify solutions that meet the constraints while optimizing the objective. Constraint conditions, expressed as equalities or inequalities, define the boundaries of the solution space. While the objective function can be single- or multi-objective, we simplify FGL to a single-objective optimization, focusing on a clear prioritization.

Specifically, error cells violate the inherent characteristics of the correct MTS data, denoted as variable relationships ($\mathcal{F}v$) and temporal dependencies ($\mathcal{F}t$) [13], disrupting the constancy and stability of the data. Therefore, we use these characteristics to define the constraint conditions.

As shown in Fig. 4, we obtain variable and temporal constraint conditions g_v and g_t from training data (i.e., correct data) D_c through cluster analysis and speed extraction methods, respectively. Then, we consider the data points within the input window W_o as the solution space and convert the maximization of the distance of violating variable and temporal constraint conditions within the solution space into the objective function.

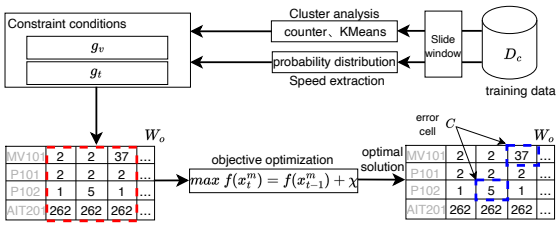


Fig. 4: Fine-Grained Localization based on objective optimization

4.2 FGL Process

Next, we first elaborate on the process of obtaining variable and temporal constraint conditions, and then present the objective function used in the OO problem.

Variable constraint condition g_v. The variable constraints should reflect the inherent characteristics $\mathcal{J}_v$ of MTS on variables, and remain robust against the erroneous data. Consequently, the constraints derived from training data can be applied to testing data, as they follow the principle of independent and identically distributed [15]. To this end, we utilize the statistical information on variables from training data as constraints. First, a *counter* calculates the mean value $\mathcal{M}_m$ for the variable m in each window W_c of the training data D_c. Next, we apply *KMeans* [8] to cluster all $\mathcal{M}$ values, producing k clusters (denoted as C), where each cluster contains the set of variables m belonging to it. Then, the center c_k for each cluster in C is computed and used as the constraint for the variables belonging to this cluster in the corresponding window W_o of the testing data D_o. A value x_t^m is considered erroneous if it violates significantly from the cluster center c_k (assuming m belongs to the k-th cluster). To measure the degree of violation, we apply a variable violation threshold α_k [8]. The threshold α_k, derived from the average distance of all values in the cluster to its respective cluster center, serves to limit the variation between the value x_t^m on m-th variable in the input W_o and the cluster center c_k of its respective k-th cluster. The variable constraint condition is formalized as g_v, as shown in Eq. 3. This process is summarized in Algorithm 2 (lines 2-6).

$$|x_t^m - c_k| \leq \alpha_k \tag{3}$$

Temporal constraint condition g_t. As FGL is designed to locate specific errors in a window, the temporal constraints reflect the inherent temporal dependencies $\mathcal{J}_t$ among adjacent timestamps. As per [33], we impose the speed constraint to limit the degree of variation between adjacent data points. Specifically, we define temporal violation thresholds β_m^l and β_m^u, which represent the lower and upper bounds of speed, respectively. These thresholds are derived from the probability distribution of speeds on the m-th variable in each window W_c of the training data D_c [43]. Specifically, they are determined by calculating the confidence interval (i.e. β_m^l and β_m^u) for the speed using the t-distribution. To elaborate further, we first calculate the speed at the t-th timestamp in the corresponding window W_o of the testing data D_o using $s_t = |\frac{x_t^m - x_{t-h}^m}{h}|$, where h is the time interval, which is set to 1 in our experiments. Then, we compare s_t with the temporal violation thresholds β_m^l and β_m^u. Therefore, the temporal constraint condition is formalized by g_t, as shown in Eq. 4 (see the lines 8-9 in Algorithm 2).

$$\beta_m^l \leq s_t \leq \beta_m^u \tag{4}$$

where the temporal violation thresholds β_m^l is set to be greater than or equal to 0.

Objective function. Specifically, to identify as many error cells as possible while avoiding missed errors (in other words, to maximize the violation distance in the solution space). Based on this goal, and considering that we aim to find data (i.e., error cells) that violates variable and temporal constraint conditions. We apply a positive reward (i.e., adding the violation distance of the data) through an incentive function χ for data x_t^m that violates constraint conditions in input data W_o. Otherwise, we give a reverse penalty (i.e., subtracting the violation distance of the data). On this basis, we propose the objective function $f(x_t^m)$ as follows:

$$max(f(x_t^m)) = f(x_{t-1}^m) + \chi(x_t^m) \tag{5}$$

$$\chi(x_t^m) = \lambda_1 dist_1(|x_t^m - c_k|, \alpha_k) + \lambda_2(dist_2(s_t, \beta_m^u)) \tag{6}$$

In the above incentive function χ, the term $dist_1(|x_t^m - c_k|, \alpha_k) = |x_t^m - c_k| - \alpha_k$ quantifies the violation distance by measuring the deviation between the current variable x_t^m and its cluster center c_k, adjusted by α_k to reflect the variable relationships. While the term $dist_2(s_t, \beta_m^u) = s_t - \beta_m^u$ capture the temporal dependencies. Specifically, it describes the violation distance in terms of speed at the current timestamp t, with adjustments controlled by β_m^u. Additionally, the incentive function χ serves as both positive rewards and negative penalties (depending on whether x_t^m violates the constraint conditions). Both λ_1 and λ_2 are set to 0.5 in our implementation with the assumption that the variable relationships and temporal dependencies are equally important, as they are both fundamental factors that support measurement of deviations in variable and temporal aspects. However, λ_1 and λ_2 can be adjusted according to the specific characteristics of MTS in practice.

4.3 Ameliorative Evolutionary Algorithm

Next, we discuss the approach to solve the optimization problem. Evolutionary algorithms have been widely used for objective optimization. However, classical evolutionary algorithms are not directly applicable to our FGL setting, as they exhibit high computational overhead in exploring the entire solution space through crossover and mutation. To address this limitation, we design an ameliorative evolutionary algorithm (AEA), which whittles the typical crossover and mutation operation in existing methods, while retaining the fitness calculation and selection operations and adjusting them to align with our FGL requirements.

AEA performs an iterative process to locate the erroneous cells within windows, as outlined in Algorithm 2 (lines 11-18). Specifically, in each iteration, AEA initializes the population using the data from each window W_o, where each data point x_t^m represents a solution. The incentive function χ (analogous to the fitness function in classical evolutionary algorithms) is computed for each solution. The χ is designed to evaluate the adaptability of a solution x_t^m to constraint conditions. When x_t^m satisfies the constraint conditions g (i.e., g_v and g_t), the incentive function χ acts as a reverse penalty, always less than or equal to 0; otherwise, χ serves as a positive reward, always greater than 0. After calculating the incentive term

χ for each solution x_t^m, it is continuously accumulated to the objective function $f(x_t^m)$ (see Eq. 5), which is initially 0. Solutions that satisfy the constraint conditions reduce the objective function $f(x_t^m)$, while those that do not increase it. Finally, the optimal solutions that maximize the objective function are returned as error cells. After pinpointing error cells within each window, the window shifts downward, initiating the next iteration until all erroneous windows have been processed.

In summary, as displayed in Fig. 4, FGL takes the erroneous W_o (i.e., the red dashed box) as input and limits the solution space in W_o with constraint conditions g extracted from the training data D_c, and outputs the solutions that maximize the objective function, which are also referred to as finer-grained error cells C (i.e., the blue dashed box) within W_o, achieving multi-scale error localization.

Algorithm 2 Fine-Grained Localization module

Require: Training and testing data D_c, D_o
1: **# Variable constraint condition** g_v
2: **for** m in W_c on D_c **do**
3: $sum(x_t^m)/l \rightarrow \mathcal{M}$
4: $KMeans(\mathcal{M}) \rightarrow C$ # cluster
5: **for** c in C **do**
6: $sum(c)/count(c) \rightarrow c_k$ # calculate and obtain k-th center
7: **# Temporal constraint condition** g_t
8: **for** m in W_c on D_c **do**
9: Calculate the confidence interval of speed s_t using t-distribution
10: **# Ameliorative evolutionary process**
11: **for** each W_o on D_o corresponds to W_c **do** #W_o is obtained by sliding window on D_o
12: Initialize population with W_o and $f(x_t^m) = 0$
13: **for** m in W_o according to C **do**
14: **for** t in l **do**
15: **if** $x_t^m \models g_v, g_t$ **then** $\chi(x_t^m) \leq 0$
16: **else** $\chi(x_t^m) > 0$
17: $max(f(x_t^m)) = f(x_{t-1}^m) + \chi(x_t^m) \rightarrow$ optimal solutions
18: return optimal solutions as the error cells C

5 Temporal and Variable-based Repair

In this section, we introduce the Temporal and Variable-based Repair (TVR) module, which performs context-consistent repairs on error cells identified by the Fine-Grained Localization (FGL) module. We first outline the rationale behind the two-stage repair strategy in TVR, followed by details of its training and repair processes.

5.1 TVR Rationale

To accurately repair error cells C, the TVR module leverages both temporal dependencies and variable relationships inherent in MTS data. As discussed in Section 2.1, temporal dependencies consist of short term within a single window (ϕ_{ts}) and long term across multiple windows (ϕ_{tl}). Meanwhile, variable relationships encompass both linear (ϕ_{vl}) and nonlinear (ϕ_{vn}) interactions. Furthermore, the interwoven relationships between time and variables should also be considered. These characteristics are crucial for predicting reliable values for erroneous cells and ensuring high-quality repairs.

To this end, TVR employs a **two-stage repair strategy**, as shown in Fig. 5. In the first stage, temporal modeling is performed to generate a preliminary repair, while the second stage further refines the repair by fusing variable relationships and producing the final optimized repair. Below, we detail these two stages to demonstrate how TVR is conducted, and capture the complex temporal and variable features, as well as the interwoven relationships between time and variables.

Stage 1: Temporal Modeling with TCN. This stage utilizes TCN to capture both short-term ϕ_{ts} and long-term ϕ_{tl} time dependencies. TCN has been shown to flexibly capture complex and hierarchical time features and outperforms existing methods in training efficiency [10, 11, 22]. Specifically, TCN comprises stacked temporal blocks, each incorporating 1-Dimensional Convolution (1D-Conv), Dilated Convolution, Causal Convolution, and Residual Connections. We deliberately apply 1D-Conv with a kernel size of 3, ensuring that each convolution operation aggregates the current timestamp and its immediate neighbors (i.e., the preceding and succeeding timestamps), to explicitly extract ϕ_{ts}. Moreover, the integration of Dilated Convolution (using a small dilation factor) and Causal Convolution, further enables TCN to efficiently capture ϕ_{ts}. On the other hand, TCN stacks multiple temporal blocks to exponentially expand the receptive field, and gradually passes temporal features extracted within each window upward through Residual Connections, thus integrating these short-term features of different windows to form the long-term temporal features (ϕ_{tl}) across windows. This design makes TCN efficient for modeling temporal patterns [37]. The output of TCN, denoted as T_o, then undergoes a Mask-based Error Correction process (as detailed in Section 5.2) to produce the preliminary repair values F_o for Stage 1.

Stage 2: Variable Modeling with GCN. The second stage takes the preliminary repair F_o as input, to further fuse linear ϕ_{vl} and non-linear ϕ_{vn} variable relationships, using the GCN, which exhibits stronger scalability and is better-suited for handling MTS data with dynamic variable relationships and large volumes, compared to other models [22]. Specifically, GCN consists of multiple graph convolutions, each involving the weight, adjacency, and attention matrix. In our design, variables at each timestamp are treated as nodes, with edges modeling inter-variable relationships. The graph convolution first computes high-level node representations by multiplying the node matrix and the weight matrix. Then, the adjacency matrix, representing the edge connections, is multiplied by the attention matrix to generate a weighted adjacency matrix, highlighting the influence of neighboring nodes. Finally, high-level node features are multiplied by the weighted adjacency matrix to aggregate the information for each node and its neighbors, capturing the linear relationships ϕ_{tl} between variables. Additionally, nonlinear relationships ϕ_{tn} are captured by stacking multiple graph convolution layers. After each graph convolution operation, the addition of ReLU activation functions enables the network to model increasingly complex interactions among variables. The nonlinear transformation applied by ReLU allows GCN to capture further not only linear relationships but also higher-order, intricate dependencies that are crucial for understanding the graph structure. This capability enables GCN to model both linear and nonlinear relationships [11]. The output of GCN is G_o, while the Stage 2 as a whole produces the final optimized repair S_o (as in Section 5.3).

Furthermore, to capture the above complex features more accurately, these two stages first jointly model them in the forward direction (i.e., toward future timestamps) as described previously, and then process in the reverse direction (i.e., toward past timestamps). The outputs from forward and reverse modeling are averaged to

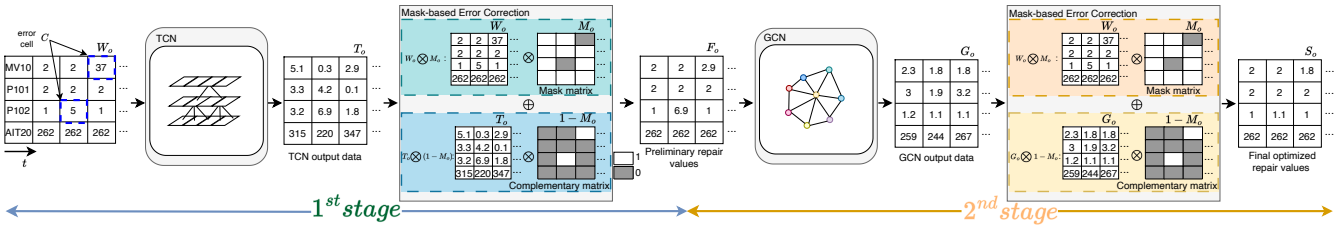


Fig. 5: The two-stage repair strategy in the TVR module.

achieve information fusion, yielding more accurate repaired data and ensuring contextual consistency.

To summarize, in the TVR module, temporal dependencies are extracted through TCN to generate the preliminary repair F_o, which incorporates ϕ_{ts} and ϕ_{tl} features. Then, F_o is refined by fusing variable relationships ϕ_{vl} and ϕ_{vn} over time through GCN. Specifically, given the potentially intricate relationships interwoven between variables and time in MTS data, we treat each timestamp as a node with multiple features. The aggregation operator in GCN is then able to effectively model the variable dependencies over time [11]. Consequently, the TVR module simultaneously accounts for temporal features, variable relationships, and their interwoven relationships. Furthermore, we empirically demonstrate the superiority of TVR in modeling complex relationships in Section 6.5, and highlight the necessity of the two-stage strategy (as shown in Section 6.7, "Temporal and Variable-based Repair").

5.2 TVR Training Process

This section details the training process of TVR, as described in Algorithm 3. The TVR training is performed on the training data D_c, which is iteratively divided into windows W_c using the sliding window (we empirically analyze the impact of window size in Section 6.6). The objective is to minimize the discrepancy between the final optimized repair values and the ground truth. Next, we outline the training steps for each iteration.

(1) *Random Masking (line 3).* Synthetic error cells are introduced into W_c using a random mask matrix M_c, where erroneous cells are set to 0 and correct cells to 1, resulting in E_c, i.e., W_c with the erroneous cells.

(2) *First-Stage Repair (lines 4-5).* TCN processes E_c and outputs T_c. It then applies Mask-based Error Correction by performing element-wise multiplication of the mask matrix M_c and the complementary mask matrix 1-M_c with W_c and T_c, respectively. This is followed by an element-wise addition of the results, to generate the preliminary repair values F_c, which capture temporal dependencies.

(3) *Second-Stage Repair (lines 6-7).* GCN further refines F_c and generates G_c, which models variable relationships over time. A similar Mask-based Error Correction is then conducted to produce the final optimized repair values S_c.

(4) *Loss Minimization (lines 8-9).* TVR minimizes the following loss function to update its parameters:

$$\mathcal{L}_{TV} = \min_{TVR} \|S_c - W_c\|_1, \tag{7}$$

where $\|\cdot\|_1$ represents the L_1 Norm. It is noteworthy that other loss functions, such as the L_2 Norm, can also work well.

Algorithm 3 TVR Training

Require: TVR, training data D_c
1: Initialize weights for TCN and GCN
2: **for** each window W_c in D_c **do**
3: Generate mask M_c for synthetic error cells $\rightarrow E_c$
4: $T_c \leftarrow TCN(E_c)$
5: $F_c \leftarrow (W_c \otimes M_c) \oplus (T_c \otimes (1 - M_c))$
6: $G_c \leftarrow GCN(F_c)$
7: $S_c \leftarrow (W_c \otimes M_c) \oplus (G_c \otimes (1 - M_c))$
8: Compute loss $\mathcal{L}_{TV} \leftarrow \|S_c - W_c\|_1$
9: Update weights of TVR through $\mathcal{L}_{TV}$

5.3 Two-Stage Repair Process

During testing, TVR processes sliding windows W_o from the testing data D_o and performs the following steps, as illustrated in Fig. 5.

1. *Temporal Repair.* TCN generates the output data T_o and then corrects error cells C in W_o using Mask-based Error Correction (as described in Section 5.2), resulting in the first-stage output F_o.

2. *Variable Repair.* GCN further processes F_o to produce its output G_o. Similarly, through Mask-based Error Correction, G_o is used to correct the error cells in W_o, while the correct data points remain unchanged, generating the final optimized repair values S_o in the second stage.

This two-stage repair strategy ensures the comprehensive utilization of temporal and variable characteristics, resulting the high-quality repaired MTS data that maintains contextual pattern consistency. By integrating temporal consistency and variable relationships, it overcomes the limitations of single-stage methods by refining repairs with finer-grained stages.

6 Experiment

In this section, we empirically evaluate the performance of Editor and its components, focusing on the following aspects:

A1. Cleaning evaluation: compare Editor against existing baselines involving detection and repair, and against the combination of the optimal methods for detection, localization and repair, to understand the overall performance.

A2. Detection evaluation: compare HSD with existing detection methods regarding effectiveness and efficiency.

A3. Localization evaluation: evaluate the performance of FGL in pinpointing error cells in a detection window.

A4. Repair evaluation: compare TVR with state-of-the-art methods in terms of repair effectiveness and efficiency.

A5. Parameter analysis: explore the sensitivity of Editor to the changes of window size.

A6. Ablation study: first investigate the necessity of HSD, FGL, and TVR in Editor, then further evaluate the potential revenue sources in HSD, FGL and TVR, respectively.

Tab. 2: Characteristics of datasets

Datasets	variables of #	training data points of #	testing data points of #	error rate (%) in testing data of #
SMD	38	28479	28479	4.5
PUMP	46	76901	143401	11.9
SWaT	51	495000	449919	3.6
WADI	123	1209601	172801	6.1

6.1 Experimental Setup

Editor is implemented in Python. All experiments are conducted on a Ubuntu machine with a 3.6GHz Intel (R) Xeon (R) Gold 5122 CPU, 128GB memory, and 12GB GPU. Below, we introduce the datasets, baselines, and evaluation metrics used in the experiments.

6.1.1 Datasets. We use four real-world datasets for evaluation:

- SWaT: Secure Water Treatment (SWaT) [19] dataset contains the readings of 51 sensors and actuators from an industrial control system, recorded during 7 days of correct operations and 4 days of erroneous operations.
- WADI: Water Distribution (WADI) [3] is an extension of SWaT, increasing the number of sensors and actuators from 51 to 123, which includes 14 days of correct operations and 2 days of erroneous operations.
- SMD: Server Machine Dataset (SMD) [41] contains 5 weeks of resource utilization for servers from a computing cluster in an Internet company.
- PUMP: This dataset [41] contains 5 months of sensor readings from a water pump system in a town.

The characteristics of datasets, including the number of variables, and the number of timestamps (in training data and testing data), as well as the erroneous data rate, are provided in Tab. 2.

Aligned with the latest study on MTS cleaning evaluation [14, 43], we intercept the clean part of the datasets and inject additive Gaussian white noise, simulating real-world errors [32]. Errors are injected by randomly selecting variables, with varying magnitudes and granularities, to evaluate the performance of cleaning on diverse error patterns.

6.1.2 Baselines. Below, we first introduce the cleaning baselines for the overall comparison with Editor. Next, we describe the applicable methods to each sub-task: detection, localization, and repair. Baselines are implemented using official or public code from papers or Github, with their specified parameters.

Cleaning baselines. We compare Editor against Vari, EWMA, and CO, to evaluate the overall cleaning performance.

- **Vari** [39]: A constraint-based approach using variance constraints to find and repair errors within a segment of data.
- **EWMA** [16]: An expression-based approach utilizing the exponential weighted moving average to locate and smooth errors.
- **CO**: Combine the best-performing methods in sub-tasks for each dataset, i.e., USAD [5]+SpeedACC [32]+rGain [24], TranAD [35]+SpeedACC [32]+Brits [10], and DAGMM [44]+EWMA [16]+Brits [10], to detect, locate, and repair errors. We detail the sub-task baselines in the following.

Detection baselines. We compare HSD in Editor against four deep learning-based baselines to sensitively identify whether a detection window contains errors of varying magnitudes, including DAGMM [44] (applying deep autoencoder and Gaussian mixture model), GDN [12] (using graph neural networks and attention-based mechanisms), USAD [5] (combining autoencoder with Generative Adversarial Networks), and TranAD [35] (integrating the self-conditioning, meta-learning and adversarial training framework with Transformer).

Localization baselines. We compare FGL with Vari [39] (employing variance constraints), Speed [33] (utilizing maximum and minimum speed constraints), Speed+Acc [32] (incorporating maximum, minimum speed, and acceleration constraints) and EWMA [16] (leveraging exponentially weighted moving averages), to further adaptively and accurately pinpoint erroneous cells within detection windows.

Repair baselines. We compare TVR against learning-based methods such as KNN [28] (utilizing the linear average of k-nearest neighbors), rGain [24, 40] (combining bidirectional Recurrent Neural Network (RNN) and Generative Adversarial Network), MPGRU [22] (employing diffusion convolutional RNN), Brits [10] (using bidirectional GRU-D-like RNN), and GRIN [11] (using message passing and RNN), to robustly repair the error cells.

6.1.3 Metrics. We measure the performance of Editor from two aspects: effectiveness and efficiency.

Effectiveness. For *cleaning* and *repair*, we report Mean Absolute Error (MAE), Mean Squared Error (MSE), and Mean Relative Error (MRE) to evaluate the performance of repaired data [11]: $MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$, evaluating the average accuracy[2];$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$, assessing the volatility and stability; $MRE = \frac{1}{m}\sum_{i=1}^{n}|\frac{y_i - \hat{y}_i}{y_i}|$, assessing the relative accuracy. For *detection* and *localization*, we apply Precision (P), Recall (R), and F1-score (F1) to evaluate error identification at the window level and error localization at the cell level [5, 13], where $P = \frac{TP}{TP+FP}$, indicating the correctness[3]; $R = \frac{TP}{TP+FN}$, assessing the completeness; $F_1 = 2 \cdot \frac{P \cdot R}{P+R}$, the harmonic mean of P and R.

Efficiency. We apply running time to evaluate the efficiency of different methods[4], including training time and testing time (i.e. cleaning time) [5, 35].

6.2 Cleaning evaluation (A1)

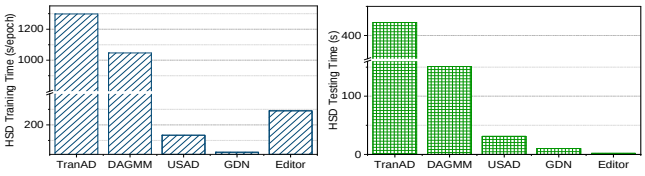
This experiment evaluates the overall performance of Editor in comparison to cleaning baselines. The results for MAE, MSE, MRE and Cleaning Time are provided in Tab. 3.

As shown in Tab. 3, Editor achieves superior performance in handling diverse error patterns compared baselines. Specifically, compared with applicable baselines, Editor achieves an average reduction of 45%, 43.93%, and 23.55% in MAE, MSE, and MRE, respectively, across all datasets, showcasing its effectiveness. This is because the overall design of Editor, which contains HSD, FGL, and TVR modules. Further, each module in Editor is well-designed and offers unique strengths, as evaluated in Section 6.3, 6.4, 6.5. Although several methods show comparable performance to Editor in terms of specific metrics such as MAE, MSE, or MRE on

[2] n is the number of cells, m is the iteration count, y_i is the truth value, and $\hat{y}_i$ is the repaired value.
[3] TP, FP, FN represents true positive, false positive, false negative, respectively.
[4] Note the FGL module in Editor does not involve training and testing phases, so we omit its evaluation and replace it with the overall running time.

Tab. 3: Effectiveness and efficiency of the overall cleaning

Dataset	Methods	*MAE*	*MSE*	*MRE*	*TIME*(*s*)
SWaT	Vari	0.547	0.268	0.303	699.173
	EWMA	0.494	0.231	0.270	1.394
	CO	0.460	0.172	0.253	718
	Editor (ours)	0.099	0.059	0.215	712
WADI	Vari	0.291	0.171	0.463	455.924
	EWMA	0.308	0.224	0.507	0.5767
	CO	0.214	0.157	0.362	901
	Editor (ours)	0.179	0.154	0.303	582.336
SMD	Vari	0.201	0.117	0.656	64.108
	EWMA	0.185	0.098	0.632	0.094
	CO	0.123	0.035	0.699	101.5
	Editor (ours)	0.111	0.029	0.628	166.012
PUMP	Vari	0.497	0.335	0.895	412.989
	EWMA	0.467	0.314	0.874	0.411
	CO	0.321	0.160	0.580	460.945
	Editor (ours)	0.265	0.158	0.478	532.800



(a) HSD training time (b) HSD testing time

Fig. 6: Efficiency of HSD compared with baselines

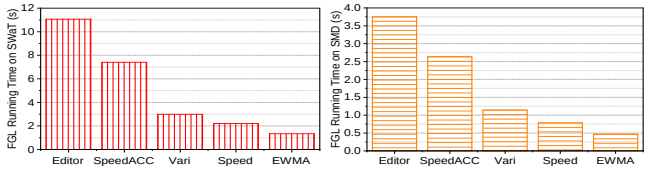
certain datasets, Editor consistently demonstrates stable and effective performance across various datasets and metrics. Beyond effectiveness, Editor is comparable to Vari in terms of efficiency and more efficient than CO, which is also a learning-based method.

6.3 Detection evaluation (A2)

This experiment evaluates the effectiveness and efficiency of HSD, and the results are presented in Tab. 4 and Fig. 6, separately.

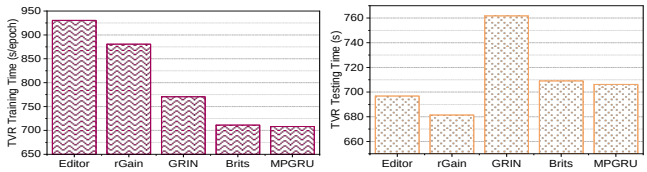
Effectiveness. As observed from Tab. 4, the HSD in our proposed Editor outperforms all detection baselines in terms of F1 across various datasets. Though other methods (e.g., USAD, DAGMM) achieve higher P or R on certain datasets, they often perform poorly on the other metrics. In contrast, HSD exhibits strong and consistent performance on both P and R, leading to optimal overall performance. Specifically, compared with these advanced baselines, HSD improves the F1 by an average of 23.25% on all datasets. One of the main reasons for HSD's superiority is the incorporation of the dual attention mechanisms and the difference-enhanced technique, enables HSD to perceive differences among data, and can effectively detect errors with varying deviations.

Efficiency. The training and testing time of HSD on SWaT are given in Fig. 6, and other datasets exhibit similar trends. For training time, HSD is slightly higher than GDN and USAD, but is 77% and 72% lower than TranAD and DAGMM, respectively. Specifically, HSD designs the advanced DA-UNet and DE-UNet models, which despite possessing slightly more parameters, can process window data in parallel, achieving comparable or even reduced training time compared to deep learning-based detection baselines. However, during testing, HSD requires 80%-98% less time compared to baselines. As testing is a lightweight inference process that operates on the trained model, and does not involve the time-intensive and iterative training procedure.



(a) FGL running time on SWaT (b) FGL running time on SMD

Fig. 7: Efficiency of FGL compared with baselines



(a) TVR training time (b) TVR testing time

Fig. 8: Efficiency of TVR compared with baselines

6.4 Localization evaluation (A3)

This experiment investigates the FGL sub-task, with the results presented in Tab. 5 and Fig. 7.

Effectiveness. As shown in Tab. 5, FGL in Editor demonstrates superior overall F1 compared to localization baselines across all datasets. Most existing constraint-based methods (e.g., Vari, Speed, and SpeedACC) perform poorly in both P and R, whereas expression-based method EWMA achieves decent R but suffers from a relatively low P. Additionally, Vari is effective only on WADI dataset. In contrast, FGL improves P while maintaining high R, leading to an average F1 increase of 38.77%. This improvement stems from FGL's effective transformation of MTS characteristics into temporal and variable constraints, and adaptively identifying cell-level data violations through AEA, allowing scale-aware error localization.

Efficiency. The running time of FGL and localization baselines on SWaT and SMD datasets are illustrated in Fig. 7, with similar trends observed on other datasets. FGL shows a slightly higher running time due to its fine-grained calculations for assessing the violation degree of each data point against constraints. However, the running time remains tolerable, typically completing within a few seconds, and can be further optimized through parallel processing.

6.5 Repair evaluation (A4)

We report the performance of TVR and repair baselines in Tab. 6 and Fig. 8.

Effectiveness. Tab. 6 displays the MAE, MSE, and MRE on various datasets. It is clear that TVR in Editor demonstrates significant and consistent performance across all metrics. Specifically, TVR achieves 10%, 7.5%, 12.75%, 6.42%, and 18.75% lower MRE than KNN, rGain, MPGRU, Brits, and GRIN across all datasets. TVR's superior performance stems from its two-stage repair strategy—TCN for temporal dependencies and GCN for variable relationships—which ensures context-aware, accurate repairs by fully leveraging temporal and inter-variable information.

Efficiency. As illustrated in Fig. 8, we evaluate the time consumed during training and testing for TVR and other repair baselines on SWaT dataset. Similar trends are observed on other datasets. Note that KNN is excluded from the analysis because it does not

Tab. 4: Effectiveness of HSD on different datasets

Methods	SWaT			WADI			SMD			PUMP		
	P	*R*	*F1*	*P*	*R*	*F1*	*P*	*R*	*F1*	*P*	*R*	*F1*
DAGMM	0.677	0.989	0.804	0.606	0.928	0.733	0.739	0.921	0.820	0.407	0.999	0.579
GDN	0.384	0.992	0.554	0.336	0.952	0.497	0.685	0.877	0.769	0.389	0.466	0.424
USAD	1	0.694	0.819	0.705	0.591	0.643	0.676	0.628	0.651	0.331	0.784	0.465
TranAD	0.570	0.929	0.706	0.800	0.933	0.862	0.594	0.946	0.730	0.125	0.999	0.223
HSD in EDITOR	0.989	0.963	0.976	0.951	0.945	0.948	0.982	0.954	0.968	0.857	0.989	0.918

Tab. 5: Effectiveness of FGL on different datasets

Methods	SWaT			WADI			SMD			PUMP		
	P	*R*	*F1*	*P*	*R*	*F1*	*P*	*R*	*F1*	*P*	*R*	*F1*
Vari	0.444	0.477	0.460	0.259	0.973	0.409	0.281	0.426	0.338	0.246	0.616	0.352
Speed	0.379	0.252	0.303	0.321	0.673	0.435	0.152	0.111	0.128	0.246	0.291	0.267
Speed+ACC	0.418	0.245	0.309	0.325	0.662	0.436	0.154	0.110	0.128	0.245	0.290	0.266
EWMA	0.443	0.998	0.614	0.255	0.998	0.406	0.239	0.978	0.384	0.261	0.997	0.413
FGL in EDITOR	0.738	0.780	0.758	0.622	0.804	0.701	0.876	0.855	0.865	0.801	0.642	0.713

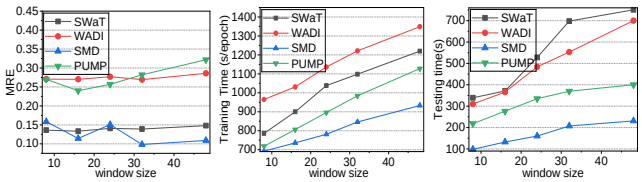
Tab. 6: Effectiveness of TVR on different datasets

Methods	SWaT			WADI			SMD			PUMP		
	MAE	*MSE*	*MRE*	*MAE*	*MSE*	*MRE*	*MAE*	*MSE*	*MRE*	*MAE*	*MSE*	*MRE*
KNN	0.118	0.046	0.239	0.065	0.025	0.276	0.015	0.001	0.117	0.061	0.020	0.311
rGain	0.417	0.138	0.1729	0.323	0.103	0.283	0.046	0.005	0.103	0.280	0.064	0.291
MPGRU	0.483	0.236	0.199	0.474	0.190	0.417	0.069	0.010	0.145	0.323	0.079	0.330
Brits	0.107	0.040	0.173	0.070	0.025	0.274	0.012	0.003	0.087	0.084	0.018	0.283
GRIN	0.483	0.236	0.199	0.514	0.256	0.448	0.061	0.007	0.138	0.499	0.179	0.533
TVR in EDITOR	0.087	0.028	0.141	0.051	0.019	0.202	0.010	0.001	0.080	0.038	0.004	0.126

involve the training process. Fig. 8(a) shows the training time of TVR is slightly longer than other baselines. The main reason is that TVR employs a two-stage repair strategy, incorporating TCN and GCN models to capture both the temporal and variable characteristics by convolution and graph structure, respectively. Note that the training is an offline process. During testing, the time trend differs notably from training. TVR, in particular, demonstrates significant and competitive performance in the testing phase.

6.6 Parameter analysis (A5)

We next investigate the effectiveness and efficiency of EDITOR with various window sizes. The results on different datasets are reported in Fig. 9. As shown in Fig. 9(a), the MREs of EDITOR remain relatively stable as the window size increases, demonstrating the robustness of EDITOR. Notably, the best MREs are achieved at window sizes between 16 and 32, depending on the dataset. However, as the window size increases, the training time and testing time (described in Fig. 9(b)-(c)) grow linearly. This is because a larger window size means a larger amount of data to be processed, leading to higher computational complexity. Therefore, to balance the performance and training/testing time, we set the window size to 32 as default.



(a) Performance (b) Training time (c) Testing time

Fig. 9: Effect of window size in EDITOR

6.7 Ablation Study (A6)

This experiment first conducts ablation studies on the three modules (HSD, FGL and TVR) of EDITOR. The results are obtained by excluding each module from EDITOR, which are presented in Tab. 7. We then carry out the ablation studies on different components in each module, and the results are shown in Tabs. 8-10.

EDITOR. Tab. 7 depicts the MAE, MSE, and MRE on different datasets after replacing HSD, FGL, and TVR with the corresponding best baseline method, respectively. It is clear that the highest performance across all datasets is obtained on EDITOR with its designed modules (HSD, FGL, and TVR). Also, there is an obvious upward trend in MAE, MSE and MRE for nearly all datasets when excluding a specific module. Among them, FGL is the most significant one, as directly repairing the entire window results in most of the original correct data being changed.

High Sensitivity Detection. Tab. 8 displays P, R and F1 on different datasets when either the dual attention or difference-enhanced technique in HSD is omitted. The exclusion of specific components leads to declines in P, R and F1 across all datasets, hindering HSD's ability to focus on critical features in MTS and amplify discrepancies, thereby reducing its effectiveness in identifying high- and low-deviation errors.

Fine-Grained Localization. As observed from Tab. 9, removing the constraint conditions g_v, g_t, or ameliorative evolutionary algorithm (AEA) from FGL results in decreased P, R, and F1 on all datasets. This is because relying on a single constraint condition is insufficient for effectively finding errors, and neglecting the AEA tends to miss errors, as the AEA focuses on maximizing the violation distance of errors to prevent omissions.

Temporal and Variable-based Repair. Tab. 10 indicates that the MAE, MSE and MRE metrics deteriorate significantly when either the TCN or GCN component is excluded from TVR, regardless of datasets. The results highlight the importance of fully learning

Tab. 7: Ablation study of each module in Editor

Module	SWaT			WADI			SMD			PUMP		
	MAE	*MSE*	*MRE*	*MAE*	*MSE*	*MRE*	*MAE*	*MSE*	*MRE*	*MAE*	*MSE*	*MRE*
w/o HSD	0.159	0.094	0.301	0.195	0.124	0.657	0.117	0.030	0.660	0.215	0.090	0.584
w/o FGL	0.172	0.100	0.324	0.163	0.085	0.551	0.124	0.029	0.700	0.236	0.101	0.641
w/o TVR	0.129	0.075	0.282	0.206	0.156	0.348	0.111	0.031	0.631	0.283	0.192	0.512
Editor	0.099	0.059	0.215	0.179	0.154	0.303	0.111	0.029	0.628	0.265	0.158	0.478

Tab. 8: Ablation study in HSD.

Component	SWaT			WADI			SMD			PUMP		
	P	*R*	*F*1	*P*	*R*	*F*1	*P*	*R*	*F*1	*P*	*R*	*F*1
w/o Dual attention	0.884	0.892	0.887	0.890	0.935	0.912	0.747	0.855	0.798	0.789	0.978	0.874
w/o Difference-enhanced	0.925	0.882	0.903	0.951	0.922	0.936	0.950	0.799	0.868	0.836	0.920	0.876
HSD in Editor	0.989	0.963	0.976	0.951	0.945	0.948	0.982	0.954	0.968	0.857	0.990	0.918

Tab. 9: Ablation study in FGL

Component	SWaT			WADI			SMD			PUMP		
	P	*R*	*F*1	*P*	*R*	*F*1	*P*	*R*	*F*1	*P*	*R*	*F*1
w/o Constraint condition ϕ_v	0.511	0.720	0.598	0.522	0.663	0.584	0.563	0.307	0.397	0.447	0.480	0.463
w/o Constraint condition ϕ_t	0.674	0.431	0.526	0.477	0.650	0.550	0.997	0.319	0.484	0.108	0.199	0.140
w/o AEA	0.561	0.843	0.674	0.532	0.438	0.480	0.779	0.385	0.515	0.253	0.608	0.357
FGL in Editor	0.738	0.780	0.758	0.622	0.804	0.701	0.876	0.855	0.865	0.801	0.642	0.713

Tab. 10: Ablation study in TVR

Component	SWaT			WADI			SMD			PUMP		
	MAE	*MSE*	*MRE*	*MAE*	*MSE*	*MRE*	*MAE*	*MSE*	*MRE*	*MAE*	*MSE*	*MRE*
w/o GCN	0.109	0.039	0.177	0.095	0.033	0.334	0.015	0.001	0.116	0.112	0.033	0.379
w/o TCN	0.120	0.046	0.192	0.070	0.023	0.273	0.013	0.002	0.102	0.109	0.034	0.378
TVR in Editor	0.088	0.028	0.141	0.051	0.019	0.202	0.010	0.001	0.080	0.093	0.020	0.326

the temporal and variable characteristics from MTS to achieve accurate repaired values.

7 Related work

Expression-based cleaning. Numerous methods have studied smoothing-based and arithmetic-based cleaning. Smoothing-based methods include MA [9] and EWMA [16], which compute (exponentially) weighted averages of neighboring points to detect anomalies. Arithmetic-based methods include AutoRegressive Integrated Moving Average (ARIMA) [7], Conformance Relationship (CR) [15], and Row-column Relationship (RR) [13]. ARIMA [7] fits linear regression to capture relationships between current and past data points for cleaning. CR [15] models the arithmetic relationships between variables to detect abnormal values. RR [13] uses arithmetic relationships between variables to identify errors, and then repair them based on the temporal context to satisfy variable relationships. MTSClean and MTSClean-soft [14] clean subsequence errors by leveraging linear relationships among temporal and variable dimensions. However, most methods rely on linear expressions and struggle with the complex temporal-variable interactions in MTS.

Constraint-based and Statistical cleaning. Some works have focused on constraint-based and statistical methods. Constraint-based methods build on Speed Constraint [33], subsequently deriving Speed and Acceleration Constraint [32], Multi-speed Constraint [18], as well as Variance Constraint [39]. They utilize the maximum and minimum constraint values extracted from the data to detect and repair erroneous data. These methods are effective for handling high-deviation errors but are unable to detect low-deviation errors. The statistical algorithm [36, 42] aims to maximize the probability of speed changes during the cleaning process. Such representative methods exhibit high sensitivity in detecting low-deviation errors but are relatively ineffective for high-deviation errors [8].

Deep learning-based cleaning. Recently, several deep learning-based anomaly detection methods, such as NumentaHTM [2], TripleES [1], USAD [5] and TranAD [35] have been extended to data cleaning. They directly use prediction or reconstruction to simultaneously detect and repair anomalies. However, they identify windows rather than fine-grained cells as anomalies and repair the entire window, easily leading to originally correct data being misclassified as erroneous and modified, which introduces deuterogenic errors [8, 20].

To this end, the proposed Editor aims to be highly sensitive to both high- and low-deviation errors, eliminate the rigidity in perceiving errors at multiple granularities, and fully leverage the characteristics of MTS to improve repair accuracy.

8 Conclusion

This paper presents Editor, a multi-resolution framework for MTS data cleaning that integrates deep learning models to effectively handle diverse error patterns. First, Editor performs coarse-grained detection at the window level, designing an improved autoencoder and a difference-enhanced technique to improve sensitivity to errors of varying magnitudes. Secondly, Editor adaptively locates the erroneous cells within a window, optimized by an ameliorative evolutionary algorithm to mitigate the collateral damage to correct data. Finally, it performs a two-stage and context-aware repair process to generate accurate repairs. For future work, we aim to further improve the time efficiency of Editor, and extend its capability to handle large-scale MTS data, especially when dealing with much higher-dimensional MTS.

References

[1] Adam Aboode. 2018. Anomaly Detection in Time Series Data Based on Holt-Winters Method.

[2] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. 2017. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing* 262 (2017), 134–147.

[3] Chuadhry Mujeeb Ahmed, Venkata Reddy Palleti, and Aditya P Mathur. 2017. WADI: a water distribution testbed for research in the design of secure cyber physical systems. In *Proceedings of the 3rd international workshop on cyber-physical systems for smart water networks*. 25–28.

[4] G Angloher, S Banik, D Bartolot, G Benato, A Bento, A Bertolini, R Breier, C Bucci, J Burkhart, L Canonica, et al. 2023. Towards an automated data cleaning with deep learning in CRESST. *The European Physical Journal Plus* 138, 1 (2023), 1–11.

[5] Julien Audibert, Pietro Michiardi, Frédéric Guyard, Sébastien Marti, and Maria A Zuluaga. 2020. Usad: Unsupervised anomaly detection on multivariate time series. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*. 3395–3404.

[6] Kamal Berahmand, Fatemeh Daneshfar, Elaheh Sadat Salehi, Yuefeng Li, and Yue Xu. 2024. Autoencoders and their applications in machine learning: a survey. *Artificial Intelligence Review* 57, 2 (2024), 28.

[7] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. 2015. *Time series analysis: forecasting and control*. John Wiley & Sons.

[8] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. 2000. LOF: identifying density-based local outliers. In *SIGMOD*. 93–104.

[9] David R Brillinger. 2001. *Time series: data analysis and theory*. SIAM.

[10] Wei Cao, Dong Wang, Jian Li, Hao Zhou, Lei Li, and Yitan Li. 2018. Brits: Bidirectional recurrent imputation for time series. *Advances in neural information processing systems* 31 (2018).

[11] Andrea Cini, Ivan Marisca, and Cesare Alippi. 2022. Filling the g_ap_s: Multivariate time series imputation by graph neural networks. *arXiv preprint arXiv:2108.00298* (2022).

[12] Ailin Deng and Bryan Hooi. 2021. Graph neural network-based anomaly detection in multivariate time series. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 4027–4035.

[13] X Ding, G Li, H Wang, and et al. 2024. Time series data cleaning under expressive constraints on both rows and columns. In *International Conference on Data Engineering*. IEEE, 5689–5698.

[14] Xiaoou Ding, Yichen Song, Hongzhi Wang, Chen Wang, and Donghua Yang. 2024. MTSClean: Efficient Constraint-based Cleaning for Multi-Dimensional Time Series Data. *Proc. VLDB Endow.* 17, 13 (2024), 4840–4852.

[15] Anna Fariha, Ashish Tiwari, Arjun Radhakrishna, Sumit Gulwani, and Alexandra Meliou. 2021. Conformance constraint discovery: Measuring trust in data-driven systems. In *Proceedings of the 2021 International Conference on Management of Data*. 499–512.

[16] Roland Fried and Ann Cathrice George. 2011. Exponential and Holt-Winters Smoothing.

[17] Sainyam Galhotra, Anna Fariha, Raoni Lourenço, Juliana Freire, Alexandra Meliou, and Divesh Srivastava. 2022. Dataprism: Exposing disconnect between data and systems. In *SIGMOD*. 217–231.

[18] Fei Gao, Shaoxu Song, and Jianmin Wang. 2021. Time Series Data Cleaning under Multi-Speed Constraints. *Int. J. Softw. Informatics* 11, 1 (2021), 29–54.

[19] Jonathan Goh, Sridhar Adepu, Khurum Nazir Junejo, and Aditya Mathur. 2017. A dataset to support research in the design of secure water treatment systems. In *Critical Information Infrastructures Security: 11th International Conference, CRITIS 2016, Paris, France, 2016*. Springer, 88–99.

[20] Xiaoyu Han, Haoran Xiong, Zhenying He, Peng Wang, Chen Wang, and X Sean Wang. 2024. Akane: Perplexity-Guided Time Series Data Cleaning. *SIGMOD* 2, 3 (2024), 1–26.

[21] Xinli Hao, Yile Chen, Chen Yang, Zhihui Du, Chaohong Ma, Chao Wu, and Xiaofeng Meng. 2024. From Chaos to Clarity: Time Series Anomaly Detection in Astronomical Observations. In *40th IEEE International Conference on Data Engineering*. IEEE, 570–583.

[22] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2017. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926* (2017).

[23] Wen Liu, Yankui Sun, and Qingge Ji. 2020. MDAN-UNet: multi-scale and dual attention enhanced nested U-Net architecture for segmentation of optical coherence tomography images. *Algorithms* 13, 3 (2020), 60.

[24] 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.

[25] Riccardo Miotto, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T Dudley. 2018. Deep learning for healthcare: review, opportunities and challenges. *Briefings in bioinformatics* 19, 6 (2018), 1236–1246.

[26] Karishma Pawar and Vahida Z Attar. 2020. Assessment of autoencoder architectures for data representation. *Deep learning: concepts and architectures* (2020), 101–132.

[27] Siby Jose Plathottam, Arin Rzonca, Rishi Lakhnori, and Chukwunwike O Iloeje. 2023. A review of artificial intelligence applications in manufacturing operations. *Journal of Advanced Manufacturing and Processing* 5, 3 (2023), e10159.

[28] Chowdhury K R. 2020. KNNImputer: A robust way to impute missing values (using Scikit-Learn).

[29] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*. Springer, 234–241.

[30] Maximilian Schleich, Zixuan Geng, Yihong Zhang, and Dan Suciu. 2021. GeCo: Quality Counterfactual Explanations in Real Time. *Proc. VLDB Endow* 14, 9 (2021), 1681–1693.

[31] Keng Leng Siau, Fiona Fui Hoon Nah, Yuzhou Qian, Brenda L Eschenbrenner, and Langtao Chen. 2022. Artificial intelligence in financial technology. In *15th China Summer Workshop on Information Management (CSWIM 2022)*.

[32] Shaoxu Song, Fei Gao, Aoqian Zhang, Jianmin Wang, and Philip S Yu. 2021. Stream data cleaning under speed and acceleration constraints. *ACM Transactions on Database Systems (TODS)* 46, 3 (2021), 1–44.

[33] Shaoxu Song, Aoqian Zhang, Jianmin Wang, and Philip S Yu. 2015. SCREEN: Stream data cleaning under speed constraints. In *SIGMOD*. 827–841.

[34] Kevin Trebing, Tomasz Stańczyk, and Siamak Mehrkanoon. 2021. SmaAt-UNet: Precipitation nowcasting using a small attention-UNet architecture. *Pattern Recognition Letters* 145 (2021), 178–186.

[35] Shreshth Tuli, Giuliano Casale, and Nicholas R Jennings. 2022. Tranad: Deep transformer networks for anomaly detection in multivariate time series data. *arXiv preprint arXiv:2201.07284* (2022).

[36] Haoyu Wang, Aoqian Zhang, Shaoxu Song, and Jianmin Wang. 2024. Streaming data cleaning based on speed change. *The VLDB Journal* 33, 1 (2024), 1–24.

[37] Yunxiao Wang, Zheng Liu, Di Hu, and Mian Zhang. 2019. Multivariate time series prediction based on optimized temporal convolutional networks with stacked auto-encoders. In *Asian Conference on Machine Learning*. PMLR, 157–172.

[38] Timothy Wong and Zhiyuan Luo. 2018. Recurrent Auto-Encoder Model for Large-Scale Industrial Sensor Signal Analysis. In *Engineering Applications of Neural Networks*. Springer, 203–216.

[39] Wei Yin, Tianbai Yue, Hongzhi Wang, Yanhao Huang, and Yaping Li. 2018. Time series cleaning under variance constraints. In *Database Systems for Advanced Applications: DASFAA 2018 International Workshops: BDMS, BDQM, GDMA, and SeCoP, Gold Coast, QLD, Australia, May 21-24, 2018, Proceedings 23*. Springer, 108–113.

[40] Jinsung Yoon, James Jordon, and Mihaela Schaar. 2018. Gain: Missing data imputation using generative adversarial nets. In *International conference on machine learning*. PMLR, 5689–5698.

[41] Aoqian Zhang, Shuqing Deng, Dongping Cui, Ye Yuan, and Guoren Wang. 2023. An Experimental Evaluation of Anomaly Detection in Time Series. *Proceedings of the VLDB Endowment* 17, 3 (2023), 483–496.

[42] Aoqian Zhang, Shaoxu Song, and Jianmin Wang. 2016. Sequential data cleaning: A statistical approach. In *Proceedings of the 2016 International Conference on Management of Data*. 909–924.

[43] Aoqian Zhang, Zexue Wu, Yifeng Gong, Ye Yuan, and Guoren Wang. 2024. Multivariate Time Series Cleaning under Speed Constraints. *Proceedings of the ACM on Management of Data* 2, 6 (2024), 1–26.

[44] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. 2018. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International conference on learning representations*.