EDITOR: Multi-Resolution Cleaning of Multivariate Time Series via Detect-Localize-Repair

Chenyang Li*, Chaohong Ma†, Xiaohui Yu‡, Cailong Li§, Xiaofeng Meng*
*Renmin University of China, Beijing, China; †Hebei Normal University, Shijiazhuang, China
‡York University, Toronto, Canada; §Northeastern University, Shenyang, China
{chenyangli, xfmeng}@ruc.edu.cn, chaohma@hebtu.edu.cn, xhyu@yorku.ca, cailongli.china@gmail.com

***Abstract*—Multivariate time series (MTS) power downstream decisions, yet real-world MTS often contain errors with varying magnitudes and granularities (e.g., points, subsequences, cross-variable), breaking temporal and inter-variable dependencies. Most existing cleaning methods rely on fixed windows, which (i) miss subtle deviations and (ii) over-clean correct values. We introduce EDITOR, a multi-resolution framework that separates error detection, localization, and repair to improve cleaning precision while preserving MTS structure: (i) High-Sensitivity Detection (HSD) identifies erroneous windows (coarse regions likely to contain errors) using a dual-attention UNet, a difference-enhanced UNet, and dynamic thresholding to capture a broad range of anomalies. (ii) Multi-Granularity Localization (MGL) operates within each detected window, casting fine-grained localization as constrained optimization under learned temporal and cross-variable dependencies. It leverages evolutionary search to pinpoint erroneous points, subsequences or cross-variable concurrence while minimizing collateral changes. (iii) Context-Aware Repair (CAR) applies a two-stage bidirectional correction: a Temporal Convolutional Network (TCN) restores temporal coherence, followed by a Graph Convolutional Network (GCN) for cross-variable consistency, both in forward and backward directions. EDITOR outperforms strong baselines across five datasets and improves downstream tasks. Ablation studies confirm the necessity of each module.**

***Index Terms*—data cleaning, MTS, multi-resolution**

I. Introduction

Background. Multivariate time series (MTS) consist of multiple interrelated variables recorded over time. They are widely generated in critical domains such as healthcare (e.g., patient vital signs including heart rate, blood pressure, and oxygen saturation [1]), industrial monitoring (e.g., machinery sensors [2]), and financial analysis [3]. These datasets exhibit complex relationships characterized by short- and long-term temporal dependencies as well as linear or nonlinear variable interactions. Downstream tasks (e.g., clustering and regression) leverage these relationships for reliable decision-making.

However, real-world MTS are rarely clean. Noise, inconsistent formatting, or data drift introduce errors that disrupt both temporal and variable dependencies [4–7]. As shown in Fig. 1, even a few corrupted points at timestamps 48–51 and 58–59 can break nonlinear dependencies (e.g., Var3-Var5 at timestamps 45-50) or long-range consistency (e.g., Var5 across segments W_A and W_B), severely degrading downstream performance.

Motivation. Classical constraint-based [4, 7, 10, 11] and statistical [12, 13] methods tend to extract short-term temporal

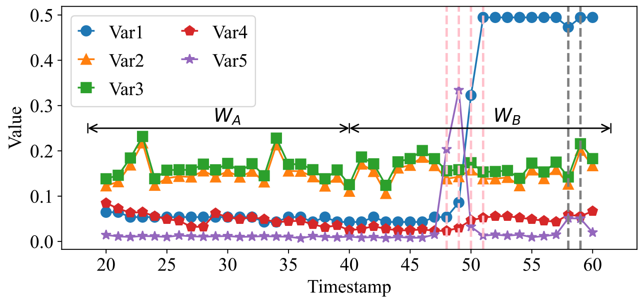


Fig. 1. Motivating example: Server Machine Dataset (SMD) [8], which provides the specific error points marked by pink and gray dashed lines (i.e., Var5 at timestamps 48–51 and 58–59, and Var1 at timestamp 58) [9].

dependencies or linear inter-variable relationships in MTS, but struggle to comprehensively capture complex dependencies and relationships. Deep learning (DL)-based approaches [14, 15] improve performance by modeling nonlinear relationships through reconstruction or forecasting, yet they face two persistent challenges.

On one hand, real-world MTS errors manifest at multiple granularities, including single-point errors (e.g., Var1 at timestamp 58), short subsequences (e.g., Var5 at timestamps 48–51), and concurrent distortions across multiple variables (e.g., Var1 and Var5 at timestamp 58). Existing DL-based methods typically identify fixed-size windows containing mixed-granularity errors (rather than the above specific errors) as erroneous and perform cleaning on these fixed-size windows [16–19], leading to over-cleaning by misidentifying correct data as erroneous [14]. This rigidity introduces unnecessary cleaning (deuterogenic errors), reducing overall cleaning accuracy.

On the other hand, errors in MTS exhibit varying magnitudes, ranging from large, easily detectable deviations (e.g., Var5 at timestamps 48–49 in Fig. 1) to subtle, low-amplitude anomalies (e.g., Var5 at timestamps 58–59 or Var1 at timestamp 58). DL-based methods [17–19] are efficient at identifying large deviations, but often overlook subtle errors, mistaking them for normal fluctuations [18, 20], as empirically studied in §VI-C. This limitation in handling errors across varying magnitudes inevitably leads to missed detections.

Methodology. We propose EDITOR, a purpose-built, multi-resolution MTS cleaning framework that explicitly separates *window-level detection* from *intra-window localization and repair*. This separation allows EDITOR to capture complex dependencies while reducing collateral modification to clean data. Unlike prior work, EDITOR systematically addresses challenges posed by errors of different magnitudes, granularities, and variable–temporal dependencies. Specifically, EDITOR operates in three coordinated stages.

- **High-Sensitivity Detection (HSD)** identifies erroneous windows across varying magnitudes by combining dual-attention and difference-enhancement mechanisms with adaptive thresholding.
- **Multi-Granularity Localization (MGL)** pinpoints specific erroneous points, subsequences or cross-variable concurrence within each window via constraint-guided optimization, using learned temporal and variable constraints to reduce over-correction.
- **Context-Aware Repair (CAR)** applies a two-stage, bi-directional strategy combining temporal modeling with Temporal Convolutional Networks (TCN) and variable-aware refinement with Graph Convolutional Networks (GCN) to generate contextually consistent corrections.

Relevance and Novelty. From a data-engineering perspective, EDITOR tackles a core issue in time-series data pipelines: ensuring *data quality* without sacrificing *data integrity*. It advances beyond anomaly detection or imputation by providing a full **Detect-Localize-Repair** operator that can be embedded into data management workflows. The coordination among detection, localization, and repair is *non-trivial*: detection guides localization to reduce missed errors, localization identifies precise targets for repair, and repair leverages identified errors and MTS dependencies for context-aware corrections. This coordinated design preserves clean data integrity while handling multi-granularity and multi-magnitude errors, distinguishing EDITOR from existing DL or constraint-based methods. Its novelty lies in (i) decoupling detection and localization to reduce collateral damage, (ii) constraint-guided optimization that formalizes temporal/variable consistency as learnable constraints rather than hard rules, and (iii) multi-resolution design that handles errors across both magnitude and granularity. These features make EDITOR not only an algorithmic contribution but also a data quality framework for scalable, reliable, and interpretable MTS management.

In summary, our **contributions** are summarized as follows:

1) We propose EDITOR, a multi-resolution MTS cleaning framework that performs Detect–Localize–Repair to handle errors of varying magnitudes and granularities while preserving temporal and variable dependencies.
2) We develop high-sensitivity detection to accurately identify erroneous windows of different magnitudes.
3) We design multi-granularity localization to precisely pinpoint errors of varying granularities in erroneous windows.
4) We introduce context-aware repair to robustly generate reliable corrections for each localized error.
5) Extensive experiments demonstrate EDITOR's superiority in MTS cleaning, achieving 10%–25% improvement over state-of-the-art baselines. We further evaluate its impact on downstream tasks, such as regression and clustering, observing performance gains of up to 15%.

Tab. 1
NOTATIONS

Symbol	Description
D	Multivariate time series with M variables
x_t	Observation at the t-th timestamp in D
x_t^m, $\hat{x}_t^m$	Observed value of the m-th variable at timestamp t and its true value
$\mathscr{C}$	A set of constraints defined by complex dependencies of MTS
W	A detection window of length l on M variables
E	Errors with varying granularities E_g and magnitudes E_m in W
δ_t^m	Deviation amplitude between x_t^m and $\hat{x}_t^m$
ϕ_{ts}, ϕ_{tl}	Short- and long-term temporal dependencies
ϕ_{vl}, ϕ_{vn}	Linear and nonlinear inter-variable relationships
δ_w	Differences between the reconstructed W_r and original W_o
$\mathcal{T}$	Adaptive threshold based on deviation distribution for detection
γ_i	The deviation score for each window W
$\mathcal{C}$, c, c_k	Cluster set, each cluster and its center
α, β	Thresholds in variable and temporal constraint condition g_v, g_t
f, χ	Objective function and incentive function
F_o, S_o	The 1^{st}- and 2^{nd}-stage correction values

Paper outline. §II presents preliminaries and an overview of EDITOR. §III, §IV, and §V detail the HSD, MGL, and CAR. §VI gives experiments. §VII discusses related work, and §VIII concludes the paper.

II. PRELIMINARIES AND EDITOR OVERVIEW

We first introduce the key concepts, followed by the core requirements of MTS cleaning that motivate EDITOR's design (§II-A), and finally present EDITOR overview (§II-B). The notations used throughout this paper are summarized in Tab. 1.

A. 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 $x_t = \left\{x_t^1, x_t^2, ..., x_t^M\right\}$ represents the values of M variables at timestamp t, where x_t^m denotes the observation of the m-th variable ($m \in [1, M]$). As described in §I, MTS inherently exhibit complex relationships, including temporal dependencies over time (short- & long-term) and inter-variable correlations across different variables (linear & non-linear).*

Definition 2. ***MTS cleaning.*** *MTS cleaning aims to fully exploit the temporal dependencies and inter-variable dependencies inherent in multivariate time series, and to formalize them as a set of constraints $\mathscr{C}$ for detecting and repairing errors in a dataset D. Under this definition, data points or subsequences that satisfy all constraints are regarded as correct data, whereas those that violate any temporal or variable constraint are considered erroneous data.*

Definition 3. ***Detection window.*** *We define the detection window as $W[t, t+l]$, where $[t, t+l]$ denotes a continuous time interval from t to $t+l$, and W corresponds to the projection of D on $[t, t+l]$, with $0 \leq t < t+l \leq T$. For simplicity, we denote $W[t, t+l]$ as W, when there is no ambiguity.* EDITOR *employs W to efficiently capture temporal and variable dependencies for effective error detection.*

Definition 4. ***Multi-granularity errors** (E_g).* *To further characterize the errors, we first define E_g as corruptions that differ in their affected temporal and variable scopes. Formally, let* *the corrupted segment of variable m over the time interval $[t, t+\Delta t]$ be denoted as $x^m_{[t,t+\Delta t]}$, where $t \in [0,T]$. When $\Delta t = 0$, the corruption affects a single isolated timestamp, forming a single-point error: $e_p = \{(x^m_t)\}$. When $1 \le \Delta t \le l$, the corruption spans consecutive timestamps, forming a subsequence error: $e_s = \{x^m_{[t,t+\Delta t]} \mid 1 \le \Delta t \le l\}$. When multiple variables are concurrently corrupted, it forms a concurrent error: $e_c = \{x^{(m_1,m_2,\ldots,m_i)}_{[t,t+\Delta t]} \mid 0 \le \Delta t \le l, m_i \in [1,M]\}$. The set of multi-granularity errors is thus $E_g = \{e_p, e_s, e_c\}$.*

Definition 5. ***Varying-magnitude errors*** E_m**.** *We then define E_m as the continuous deviations of corrupted values x^m_t from their true values $\hat{x}^m_t$: $E_m = \{\delta^m_t = x^m_t - \hat{x}^m_t \mid t \in [0,T], m \in [1,M]\}$, where $\delta^m_t \in \mathbb{R}$ denotes the deviation amplitude of variable m at timestamp t. Since MTS values are continuous, these deviations vary continuously rather than remaining fixed or discrete. Such varying magnitudes manifest as irregular deviations, making them more challenging to uniformly detect. We simplify E_g and E_m as E, when no ambiguity arises.*

Driven by the above characteristics of MTS data, we derive the following **key requirements** for effective MTS cleaning:

R1. **Handle varying-magnitude errors.** Errors in MTS can vary greatly in magnitude, as shown in Definition 5. Most existing methods cannot simultaneously deal with varying-magnitude errors [14]. For example, DL- or constraint-based approaches [4, 10, 11] typically rely on fixed threshold to identify errors, tending to be conservative and are effective mainly for large-magnitude errors. While statistical methods [12, 13] excel at identifying low-deviation errors. MTS Cleaning methods should be capable of detecting and repairing errors across this full spectrum of magnitudes, without omission or insufficient correction.

R2. **Model temporal and variable dependencies.** As discussed in Definition 1, errors often disrupt complex MTS dependencies across time (short- and long-term) and among variables (linear and non-linear). Classical strategies (e.g., constraint, arithmetic, and smoothing) [4, 6, 7, 15, 21, 22] mainly focus on short-term or linear dependencies. DL-based approaches [8, 23, 24] can capture both temporal and variable dependencies by processing MTS with windows, but often causing collateral damage (as discussed in **R3**). MTS Cleaning methods should capture these complex dependencies and leverage them to achieve both accurate detection and repairs while minimizing distortion.

R3. **Avoid collateral damage in fixed-size cleaning.** To efficiently satisfy **R2**, cleaning methods should process MTS in windows. However, most approaches [16–19] operate at fixed-size windows W for both error detection and repair, inevitably damaging the correct values within the same W and resulting in new errors (termed *deuterogenic errors*). Since W is a mixture of erroneous and correct values, and errors in MTS occurs at multi-granularity (as discussed in Definition 4). MTS Cleaning methods should mitigate the mismatch between detection and repair to prevent such collateral damage.

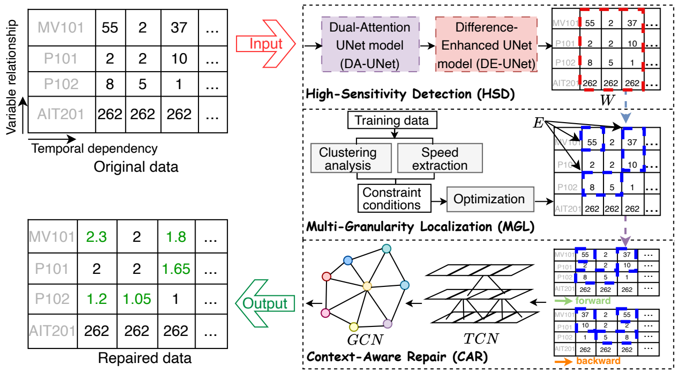


Fig. 2. EDITOR overview, with a numerical example from [25], where the table shows time (horizontal) and variables (vertical). Among them, black and green values show observed data and cleaned data, respectively; while red and blue dashed box labels the erroneous window W and specific errors E.

B. EDITOR *Overview*

To this end, we propose EDITOR, a multi-resolution MTS cleaning framework that addresses errors of varying magnitudes and granularities by effectively exploiting complex MTS relationships via Detect-Localize-Repair.

As depicted in Fig. 2, EDITOR consists of three modules: High-Sensitivity Detection, Multi-Granularity Localization, and Context-Aware Repair. Next, we highlight each module, connected to the requirements in §II-A.

High-Sensitivity Detection (HSD). To leverage temporal and variable dependencies for more efficient cleaning, HSD processes MTS in windows (**R2**). Specifically, it designs a Dual-Attention UNet (DA-UNet) to capture complex dependencies, and develops a Differential-Enhanced UNet (DE-UNet) with dynamic thresholding to facilitate the detection errors of varying magnitudes (**R1**). The dual-attention mechanism models temporal and variable relationships, while the difference-enhanced technique amplifies deviations, enabling high-sensitivity detection of erroneous windows (§III).

Multi-Granularity Localization (MGL). After HSD identifies erroneous windows, MGL precisely localizes errors of different granularities within each window to minimize unnecessary modification of correct data (**R3**). It formulates error localization as a constrained optimization problem, where the constraints are derived from the inherent MTS dependencies. An ameliorative evolutionary algorithm is then proposed to adaptively determine the location of specific errors, achieving multi-granularity localization (§IV).

Context-Aware Repair (CAR). The CAR is designed to accurately repair errors of varying magnitudes and granularities. It adopts a two-stage bidirectional strategy that processes MTS in both forward and backward directions. To capture the temporal and variable context of errors (**R2**), CAR integrates a TCN and a GCN: the TCN establishes temporal dependencies and generates preliminary correction values, while the GCN refines these by incorporating inter-variable correlations within the temporal context. Through the two-stage bidirectional modeling, CAR achieves robust error repair (§V).

In EDITOR, to ensure consistent and comparable processing across variables in MTS data, all observed values x^m_t are normalized to the range $[0, 1]$. This step mitigates the impact of varying dimensions across variables, simplifies computations, and enhances the performance of EDITOR.

Workflow. EDITOR leverages complex temporal and variable dependencies to clean MTS errors with varying magnitudes and granularities. Specifically, HSD detects erroneous windows, MGL pinpoints multi-granularity errors within each window, and CAR generates precise correction values for identified errors, resulting in accurate MTS data.

III. HIGH-SENSITIVITY DETECTION

This section first introduces the components of the High-Sensitivity Detection (HSD) module (§III-A), and then describes its detection process (§III-B).

A. *Components of HSD module*

HSD is designed to efficiently identify whether errors exist in the detection window W by capturing the complex dependencies. To achieve this, we propose a reconstruction-based strategy, which utilizes an encoder to extract features of W, then reconstructs W from features by decoder. Considering the complex dependencies and varying deviations in MTS (§II-A), we design two tailored models—Dual-Attention UNet (DA-UNet) and Difference-Enhanced UNet (DE-UNet)—for MTS data cleaning, rather than general representation learning or directly applying existing transformer-based AutoEncoders [26, 27]. Below, we first present DA-UNet, capturing temporal–variable dependencies for reconstruction, then DE-UNet for detecting errors of different magnitudes.

1) Dual-Attention UNet model (DA-UNet): To encode MTS, transformer-based AutoEncoder (AE) is commonly used for data reduction and feature representation [26], but its encoder may lose crucial features for reconstruction [28]. In contrast, UNet, which has been widely used in image segmentation and video processing [29, 30], preserves fine-grained features through skip connections and hierarchical design, enabling more faithful reconstruction. Accordingly, we adopt UNet as the base model in HSD to enhance feature recovery during decoding and facilitate efficient error detection.

However, generic UNet models [29, 30] struggle to accurately generate reconstructed data, as they assign equal weights to all features during encoding and decoding. Several advanced UNet variants [31–33] address this by incorporating attention. Nonetheless, they typically apply attention only to selected layers (e.g., encoder's endpoint layer, or skip connections) and are primarily designed for vision tasks, where attention operates over homogeneous spatial grids or channels, limiting their ability to capture heterogeneous temporal and variable importance in MTS.

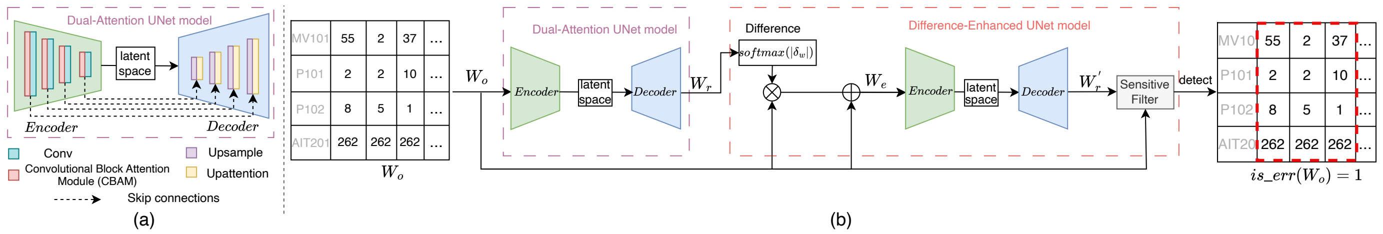
We propose DA-UNet (Fig. 3(a)), which inherits UNet's hierarchical structure while explicitly designing attention mechanisms for MTS cleaning. On one hand, DA-UNet embeds attention in all layers, allowing dynamic weighting of features across temporal and variable dimensions. On the other hand, DA-UNet integrates two distinct attentions in encoder and decoder, optimizing feature representation and facilitating effective reconstruction. This design enables DA-UNet to model both temporal patterns and cross-variable dependencies, including short-term temporal patterns (within-window volatility (ϕ_{ts})) and localized variable relationships (pairwise dependencies (ϕ_{vl})), as well as long-term temporal trends (cross-window stability (ϕ_{tl})) and complex multi-variable dependencies (ϕ_{vn}). Below, we present the detailed design of DA-UNet.

Specifically, as shown in Fig. 3(a), DA-UNet model comprises an encoder and a decoder. On one hand, the encoder process includes the Convolution layer (Conv) and Convolutional Block Attention Module (CBAM). First, Conv applies a convolutional kernel sliding over W_o with a fixed stride to extract local features among the temporal and variable dimensions. Then, CBAM consists of channel and spatial modules. The channel module computes the *average pooling* and *max pooling* for features obtained from Conv, and 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 [30]). 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. Subsequently, the spatial module further weights the features at each variable location, emphasizing key variable features and suppressing unimportant ones. By stacking Conv and CBAM, the encoder gradually expands the receptive field, efficiently capturing the global temporal and variable characteristics of the input W_o.

On the other hand, the decoder of DA-UNet adopts a stacked structure that combines Upsample, Upattention and Skip Connections. First, the Upsample operation restores the sizes of features extracted from the previous layer. Meanwhile, Upattention, the second attention mechanism we incorporate, calculates an attention weight map on the features from both the encoder and decoder, emphasizing the important features. Then, 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 of MTS.

Overall, DA-UNet achieves rich feature representation in encoder and accurate reconstruction in decoder, while maintaining high computational efficiency and inference speed.

2) Difference-Enhanced UNet model (DE-UNet): Rather than directly using DA-UNet outputs, which identify windows with high-deviation errors but may miss low-deviation ones, existing U-Net–based models [29–31, 33] lack an effective



(a) (b)

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

solution for this issue. To address this, we propose DE-UNet, which integrates a difference-enhancement technique to amplify deviations during encoding and decoding, facilitating the detection of erroneous windows. It further employs dynamic thresholding to sensitively filter errors of varying magnitudes.

As shown in Fig. 3(b), the DE-UNet consists of three steps: 1) differences-based enhancement, 2) encoding and decoding, and 3) sensitive 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. First, as described above, the differences $delta_w$ 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. Then, to address this limitation, the differences δ_w undergo a *softmax* operation, transforming into a probability distribution, where each element is primarily determined by the relative contrast among regions instead of their absolute deviation magnitudes, falling within the range of [0,1] and the sum of all elements equaling 1. Next, these transformed differences are multiplied element-wise with the input data W_o to obtain discrepancy coefficients, which are thereafter 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]. Finally, this enhancement ensures that after *softmax* normalization, $\oplus$ and $\otimes$ operations, correct regions, whose reconstruction differences are typically smooth and temporally consistent, are assigned relatively low weights. Conversely, even erroneous regions with small reconstruction differences, which often disrupt temporal dependencies and reveals high contrast, 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 emphasized.

Example 1: For W_o, regions 55, 37 and 10 as well as 8 and 5 show non-zero reconstruction differences δ_w. Although the absolute deviations at region 8 and 5 are much smaller than at regions 55, 37 and 10, *softmax* normalization assigns them high weights due to their contrast with surrounding near-zero regions. When fused with W_o, these regions are selectively amplified, preventing low-deviation errors from being obscured by correct patterns.

[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 following steps.

Encoding and decoding. To improve sensitivity to low-deviation errors, DE-UNet leverages the feature information of the enhanced data W_e through the same encoding-decoding process as DA-UNet. Specifically, the encoder in DE-UNet takes W_e as input to extract feature representations. Since the enhanced deviation information is embedded in W_e, it learns features variations reflecting the differences between correct regions and erroneous regions, thus highlighting the feature regions with discrepancies while de-emphasizing those without. This facilitates the visibility of deviations, particularly low-deviation errors, in the feature space. Then, the decoder generates reconstructed data W_r', in which the low-deviation features are amplified, as the accurate reconstruction of correct regions makes subtle discrepancies in erroneous regions more distinguishable [34].

Sensitive Filtering (SF). Fixed thresholds struggle to balance the detection of errors with varying magnitudes. A low threshold increases the detection of subtle errors but may lead to over-cleaning, whereas a high threshold is effective for large-deviation errors but tends to miss subtle errors [35]. To address this, we design SF with a dynamic threshold $\mathcal{T}$ that adaptively adjusts with the deviation distribution for robust detection. SF calculates the L_1 Norm between $W_r^{'}$ and W_o as the deviation score $\gamma_i = \|W_r^{'} - W_o\|_1$ for each window, $i \in [1, \frac{T}{l}]$, where $\frac{T}{l}$ denotes the total number of windows. Based on the distribution of γ_i across all windows, SF iteratively determines a threshold $\mathcal{T}$ that separates erroneous from correct windows. Windows with $\gamma_i > \mathcal{T}$ are labeled as erroneous. This dynamic threshold $\mathcal{T}$ effectively detects subtle errors while avoiding misidentification of large but normal variations. We analyze the effect of different l in §VI.

Therefore, DE-UNet reconstructs $W_r^{'}$ with enhanced differences, facilitating sensitive detection of windows containing errors of varying magnitudes. To verify the high sensitivity of DE-UNet, we conduct an ablation study (see "HSD" of A6 in §VI), which demonstrates the necessity of DE-UNet.

B. HSD Detection process

Below, we present the window-based detection process of HSD, sequentially applied to DA-UNet and DE-UNet, which generate the first and second reconstructions by leveraging the complex relationships in MTS and the difference-based enhanced technique, respectively. The entire detection process (summarized in Algorithm 1) is conducted on the testing data D_o (i.e., the original data with errors).

Given the testing data, HSD first applies the sliding window to form W_o (line 1). As shown in Fig. 3(b), the trained DA-

UNet encodes W_o to extract features and map them to the latent space. Since the model has been trained on correct data, the correct regions of W_o align with the learned distribution, while erroneous regions deviate from ones, causing the model to project them toward more plausible patterns and generate a more reasonable feature representation. Then, DA-UNet decodes the feature representation, producing the first reconstructed data W_r (line 2). As DA-UNet's reconstruction difference may be too small for subtle errors, the trained DE-UNet leverages the difference between W_r and W_o to highlight and enhance the erroneous regions in W_o. Through another round of encoding and decoding, DE-UNet produces the second reconstructed data $W_r^{'}$, which adheres to the correct pattern and amplifies the reconstruction difference between $W_r^{'}$ and W_o, improving sensitivity to errors of various magnitudes, particularly low-deviation errors (lines 3–4). Finally, the deviation score γ_i for W_o is compared with the deviation threshold $\mathcal{T}$ by SF; if any γ_i exceeds $\mathcal{T}$, W_o is detected as erroneous ($is\_err(W_o) = 1$); otherwise, it is considered correct ($is\_err(W_o) = 0$) (lines 5–9).

Algorithm 1 Detection process

```
Input: DA-UNet and DE-UNet; testing data $D_o$
1: for each $W_o$ in $D_o$ do #$W_o$ is obtained by sliding window on $D_o$
2:     DA-UNet($W_o$) $\rightarrow W_r$
3:     $\delta_w = W_r - W_o$
4:     DE-UNet($[softmax(\delta_w) \otimes W_o] \oplus W_o$) $\rightarrow W_r^{'}$
5:     $\gamma_i = \|W_r^{'} - W_o\|_1$
6:     if $\gamma_i > \mathcal{T}$ then
7:         $is\_err(W_o) \leftarrow 1$
8:     else
9:         $is\_err(W_o) \leftarrow 0$
```

IV. Multi-Granularity Localization

This section introduces the rationale of Multi-Granularity Localization (MGL) (§IV-A), formulates its localization task as an objective optimization problem (§IV-B), and presents an ameliorative evolutionary algorithm for its solution (§IV-C).

A. MGL Rationale

MGL precisely and adaptively pinpoints multi-granularity errors within each erroneous window to reduce missed errors while minimizing interference with clean data.

We formulate this process as an *Objective Optimization* (*OO*) problem. *OO* is a common approach to find optimal solutions within a solution space. 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 define the boundaries of the solution space.

Specifically, errors violate the inherent relationships of the correct MTS data, denoted as inter-variable correlations and temporal dependencies [21], disrupting the constancy and stability of data. Therefore, we use these relationships to define the constraint conditions. However, existing constraint-based methods [4, 21] usually apply static constraints that cannot adapt to the evolving nature of MTS, degrading performance.

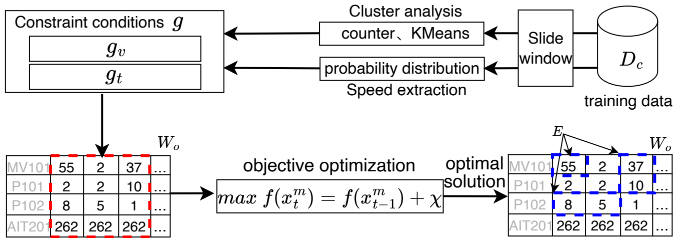


Fig. 4. Multi-Granularity Localization based on objective optimization

To address this issue, as shown in Fig. 4, we obtain learnable variable and temporal constraint conditions g_v and g_t from the window of training data (i.e., correct data) D_c via cluster analysis and speed extraction method. The learnable constraints can then be applied to to the corresponding window of testing data under the independent identically distribution principle as per [6]. We further consider data points within the input window W_o as the solution space and convert the maximization of the distance of violating variable and temporal constraints within the solution space into the objective function. This design enables MGL to effectively localize multi-granularity errors in MTS cleaning.

B. MGL 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 relationships of MTS on variables, and remain robust against the errors. 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* [16] to cluster all $\mathcal{M}$ values, producing k clusters (denoted as $\mathcal{C}$), where each cluster contains the set of variables m belonging to it. Then, the center c_k for each cluster in $\mathcal{C}$ 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 [16]. 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. 1. This process is summarized in Algorithm 2 (lines 2-6).

$$|x_t^m - c_k| \leq \alpha_k \tag{1}$$

Temporal constraint condition g_t. The temporal constraints reflect the inherent temporal dependencies among adjacent timestamps. As per [4], we impose the speed extraction to limit the degree of variation between adjacent data points. And 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 [15]. 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. 2 (see the lines 8-9 in Algorithm 2).

$$\beta_m^l \leq s_t \leq \beta_m^u \tag{2}$$

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 errors 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., specific errors) 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 $\max(f(x_t^m))$ as follows:

$$\max(f(x_t^m)) = f(x_{t-1}^m) + \chi(x_t^m) \tag{3}$$

where $\chi(x_t^m) = \lambda_1 dist_1(|x_t^m - c_k|, \alpha_k) + \lambda_2(dist_2^u(s_t, \beta_m^u) + dist_2^l(s_t, \beta_m^l))$, with violation distances defined w.r.t. g_v (Eq. 1) and g_t (Eq. 2).

In the above incentive function χ, the term $dist_1(|x_t^m - c_k|, \alpha_k) = |x_t^m - c_k| - \alpha_k$ measures the violation distance based on the deviation between the current variable x_t^m and its cluster center c_k, adjusted by α_k to reflect the inter-variable relationships. While the term $dist_2^u(s_t, \beta_m^u) + dist_2^l(s_t, \beta_m^l) = \max(0, s_t - \beta_m^u) + \max(0, \beta_m^l - s_t)$ assesses the violation distance by evaluating the difference of speed at the current timestamp t, with β_m^u and β_m^l controlling the upper and lower bounds, respectively, to capture temporal dependencies. Notably, when $\beta_m^l = 0$, the lower-bound constraint is always satisfied and its corresponding penalty term vanishes. In this case, $\chi(x_t^m)$ simplifies to $\chi(x_t^m) = \lambda_1 dist_1(|x_t^m - c_k|, \alpha_k) + \lambda_2(dist_2(s_t, \beta_m^u))$, where the term $dist_2(s_t, \beta_m^u) = s_t - \beta_m^u$ controlled solely 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 inter-variable relationships and temporal dependencies are equally important, as they are fundamental factors that measure the deviations in variable and temporal aspects. In practice, λ_1 and λ_2 can be adjusted according to the characteristics of MTS.

C. Ameliorative Evolutionary Algorithm

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

AEA performs an iterative process to locate the multi-granularity errors within windows, as outlined in Algorithm 2 (lines 11-18). First, 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. Next, 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. Specifically, 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. Then, after calculating the incentive function χ for each solution x_t^m, it is continuously accumulated to the objective function $f(x_t^m)$ (see Eq. 3), which is initialized to 0. At this stage, solutions that satisfy the constraint conditions reduce the objective function $f(x_t^m)$, while those that do not increase it. After that, all solutions that continuously increase the objective function until it reaches its maximum are returned, represented as errors, i.e., the optimal solutions. Finally, after pinpointing the specific errors within each window, the window is shifted downward to initiate the next iteration, which is repeated until all erroneous windows are processed.

Example 2: As shown in Fig. 4, most data points in W_o (e.g., 1, 2, 262) satisfy the constraints and receive non-positive incentives, causing their objective values to decrease or remain unchanged. In contrast, points that deviate from the normal pattern (e.g., 55, 37, 10, 8, 5) violate the constraints and receive positive incentives, which accumulate and increase the objective function. Then, only data points with the largest accumulated objective values are identified as errors, while the rest are retained as correct.

In summary, as shown in Fig. 4, MGL takes the erroneous W_o (i.e., the red dashed box) as input and limits the solution space in W_o with inherent relationships-based constraint conditions g extracted from the training data D_c, and outputs the solutions that maximize the objective function, which are referred to as the specific errors E (i.e., the blue dashed box) within W_o, achieving multi-granularity error localization.

V. Context-Aware Repair

In this section, we first outline the rationale behind the Context-Aware Repair (CAR) module (§V-A), followed by details of its repair process (§V-B).

A. CAR Rationale

CAR performs accurate repair for identified errors E, and is specifically designed for MTS cleaning, rather than directly

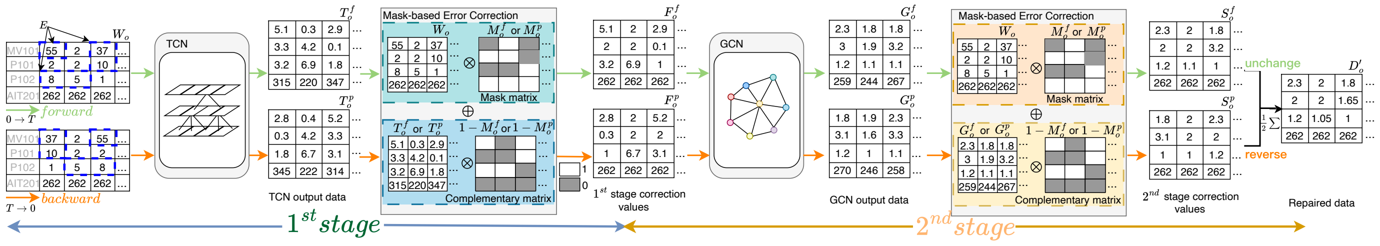


Fig. 5. Context-Aware Repair based on two-stage bidirectional strategy

Algorithm 2 Localization process

```
Input: 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:     Compute mean value: $sum(x_t^m)/l \rightarrow \mathcal{M}$
 4: Cluster $\mathcal{M}$ using KMeans: $\mathcal{C} \leftarrow KMeans(\mathcal{M})$
 5: for each cluster $c$ in $\mathcal{C}$ do
 6:     Compute $k$-th cluster center $c_k \leftarrow sum(c)/count(c)$
 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 each variable $m$ in $W_o$ according to $\mathcal{C}$ do
14:         for each timestamp $t$ in $l$ do
15:             if no violation (as penalty): $x_t^m \models g_v, g_t$ then $\chi(x_t^m) \leq 0$
16:             else violation identified (as reward): $\chi(x_t^m) > 0$
17:             Accumulate and maximize: $max(f(x_t^m)) = f(x_{t-1}^m) +$
    $\chi(x_t^m) \rightarrow$ optimal solutions
18: return optimal solutions as errors $E$
```

using existing backbone models as generic interpolators. CAR is integrated within a collaborative MTS cleaning pipeline to address the core challenge of contextual consistency across time and variables. The context is characterized along two dimensions: temporal dependencies including short-term trends (ϕ_{ts}) within a single window and long-term ones (ϕ_{tl}) span multiple windows; inter-variable relationships involving linear (ϕ_{vl}) and nonlinear (ϕ_{vn}) interactions. While temporal and variable dependencies exhibit distinct characteristics, they are inherently interconnected, creating a heterogeneous and coupled context.

To better capture such context, CAR adopts a two-stage bidirectional strategy following a *divide-and-conquer* principle. Instead of modeling all dependencies in a single pass, which is insufficient for complex MTS data, CAR decomposes the repair process into two dedicated stages with clear functional roles. The first stage (1^{st}-stage) enforces temporal consistency to generate the 1^{st}-Stage correction, while the second stage (2^{nd}-stage) refines it by integrating time-varying inter-variable relationships to yield the 2^{nd}-Stage correction. Both complementary stages run in forward and backward directions to fully exploit context for aligning the repair process with the intrinsic dependency structure of MTS, as shown in Fig. 5. Next, we detail each stage's key design.

1^{st}**-Stage.** This stage utilizes TCN (Temporal Convolutional Networks) to primarily capture temporal dependencies, including both short- and long-term dependencies (ϕ_{ts} & ϕ_{tl}). TCN flexibly captures hierarchical time features and outperforms existing methods in training efficiency [36–38]. 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 to aggregate each timestamp with its immediate neighbors (i.e., the preceding and succeeding timestamps), explicitly extracting ϕ_{ts}. Moreover, the integration of dilated convolution with a small dilation factor and causal convolution further efficiently capture ϕ_{ts}. By stacking multiple temporal blocks, TCN exponentially expands the receptive field and propagates temporal features through residual connections, enabling the extraction of long-term temporal features (ϕ_{tl}) across windows. This design makes TCN efficient for modeling temporal patterns [39]. The output of TCN, denoted as T_o, then undergoes a Mask-based Error Correction process (as detailed in §V-B) to produce the 1^{st}-stage correction values F_o.

2^{nd}**-Stage.** This stage then takes the F_o as input and employs a GCN (Graph Convolutional Networks) to further fuse inter-variable relationships evolving over time, including linear and nonlinear dependencies (ϕ_{vl} & ϕ_{vn}). Compared with other models, GCN offers stronger scalability and is better-suited for MTS with dynamic variable interactions [36], owing to its multiple graph convolutions that involve the weight, adjacency, and attention matrices. In our design, variables at each timestamp are treated as nodes, and their inter-dependencies form edges, enabling flexible modeling of time-varying inter-variable relationships. The graph convolution first computes 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 influential neighboring nodes. Finally, node representations are multiplied by the weighted adjacency matrix to aggregate the information from 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 with ReLU activation functions, enabling the network to model increasingly complex dependencies beyond linear interactions among variables, which are crucial for understanding the graph structure. This capability enables GCN to model both linear and nonlinear relationships [38]. The output of GCN is G_o, while the 2^{nd}-Stage as a whole produces the 2^{nd}-stage correction values S_o (see §V-B).

Furthermore, to capture complex relationships more accurately, these two stages first jointly model them in the forward direction (i.e., towards future timestamps $0 \to T$), then process them in the backward direction (i.e., toward past timestamps $T \to 0$). The outputs from forward and reverse modeling are averaged to achieve bidirectional information fusion, yielding accurate repaired data and ensuring contextual consistency.

Therefore, CAR simultaneously accounts for temporal features, inter-variable relationships, and their bidirectional interactions. We empirically demonstrate the superiority of CAR in modeling complex relationships in §VI-E, and highlight the necessity of the two-stage bidirectional strategy in §VI-G.

B. CAR Repair Process

Below, we describe how CAR repairs localized multi-granularity errors, as shown in Fig. 5 and detailed in Algorithm 3. The repair is performed via a two-stage bidirectional strategy over the sliding windows W_o in the test data D_o (the impact of window size is empirically analyzed in §VI-F).

1) *Preprocessing (lines 1-2).* Errors are flagged in W_o using a mask matrix M_o, where the specific erroneous data are set to 0 and correct data to 1, guiding selective repair operations.

2) 1^{st}*-Stage Correction (lines 3-4).* TCN processes W_o and outputs T_o. It then applies Mask-based Error Correction by performing element-wise multiplication of the mask matrix M_o and the complementary mask matrix 1-M_o with W_o and T_o, respectively. This is followed by an element-wise addition of the results, to generate the 1^{st}-stage correction values F_o, which capture temporal dependencies.

3) 2^{nd}*-Stage Correction (lines 5-6).* GCN further refines F_o and generates G_o, which models time-varying inter-variable relationships. A similar Mask-based Error Correction is then conducted, where G_o corrects the erroneous data in W_o while keeping the correct data unchanged, to produce the 2^{nd}-stage correction values S_o.

4) *Bidirectional Interleaving (lines 7-8).* The process runs in both forward and backward directions, producing S_o^f and S_o^p. By keeping S_o^f unchanged and reversing S_o^p along the temporal axis, their average produces the repaired data D'_o.

Algorithm 3 Repair process

```
Input: CAR, testing data $D_o$
1: for each window $W_o$ in $D_o$ do
2:     Generate mask $M_o$ for multi-granularity errors in $W_o$
3:     $T_o \leftarrow TCN(W_o)$                          # $1^{st}$ stage
4:     $F_o \leftarrow (W_o \otimes M_o) \oplus (T_o \otimes (1 - M_o))$
5:     $G_o \leftarrow GCN(F_o)$                          # $2^{nd}$ stage
6:     $S_o \leftarrow (W_o \otimes M_o) \oplus (G_o \otimes (1 - M_o))$    # Bidirectional modeling
7:     $S_o^f = GCN^f(TCN^f(W_o))$, $S_o^p = GCN^p(TCN^p(W_o))$
8:     $D'_o = \frac{S_o^f + reverse(S_o^p)}{2}$
```

VI. Experiment

This section evaluates Editor on the following aspects: **A1.** Cleaning evaluation: compare Editor against cleaning baselines. **A2.** Detection evaluation: compare HSD with detection methods. **A3.** Localization evaluation: evaluate MGL with localization methods. **A4.** Repair evaluation: compare CAR with repair methods. **A5.** Parameter analysis: explore Editor's sensitivity to window sizes. **A6.** Ablation study: investigate the advantages of HSD, MGL, and CAR in Editor, then evaluate the potential revenue sources in HSD, MGL and CAR, respectively. **A7.** Downstream evaluation: apply repaired data from Editor and baselines to downstream tasks. **A8.** Analyze AEA in MGL and its simple alternative. **A9.** Analyze potential error propagation in TCN. **A10.** Discuss Editor's memory footprint and scalability for long-horizon MTS.

Tab. 2
Characteristics of datasets

Datasets	variables of #	timestamps in training data of #	timestamps in testing data of #	error rate (%) in testing data of #
SMD	38	28479	25785	4.5
PUMP	46	76901	128993	11.9
SWaT	51	495000	395335	3.6
WADI	123	1209601	162451	6.1
MSL	55	58317	65963	10.5

A. Experimental Setup

Editor is implemented in Python on Ubuntu with 3.6GHz Intel Xeon Gold 5122 CPU, 128GB RAM, 12GB GPU.

1) Datasets: We use five real-world datasets for evaluation. (1) SWaT [25] contains readings from 51 sensors and actuators in an industrial control system over 7 days of correct and 4 days of erroneous operations. (2) WADI [40] extends SWaT with 123 sensors and actuators, including 14 days of correct and 2 days of erroneous operations. (3) SMD [8] contains 5 weeks of server resource utilization from a computing cluster. (4) PUMP [8] contains 5 months of sensor readings from a town water pump system. (5) MSL [41] contains sensor and actuator data from the Mars rover.

2) Implementation: To enable controllable evaluation, we follow the error-injection guidelines in [7, 15] by extracting clean segments from the original datasets and injecting synthetic errors, ensuring the ground truth prior to corruption is known for quantitative evaluation. For datasets like SWaT and WADI, which only provide binary labels indicating event presence and lack ground truth at anomalous timestamps, we inject synthetic errors into clean segments. The synthetic errors are generated using additive Gaussian white noise [7] or random values within the dataset's range [15], simulating both subtle and large real-world errors. For SMD, which provides specific error point locations, we sample errors from the labeled anomalous segments and inject them into clean segments. We randomly selecting variables to inject single-point, subsequence and co-occurrence errors. Thus, for all datasets, we obtain constructed data with reduced volume and varied error rates compared to original data [8, 25, 40, 41]. The characteristics of generated datasets used are listed in Tab. 2.

3) Baselines: We first introduce the cleaning baselines for overall comparison with Editor, then describe the applicable methods for each sub-task: detection, localization, and repair. To ensure fairness in evaluating method performance, all subtask approaches use the same input: HSD outputs for localization and MGL outputs for repair.

Cleaning baselines. We compare Editor with Vari [11] (variance constraints), EWMA [22] (exponential weighted moving average), Clean4MTS [21] (linear inter-variable or short temporal relations), and CO. Among these, Vari, EWMA, and Clean4MTS are methods capable of detecting, localizing, and repairing errors. While CO is the best-per-subtask composition combining top methods per dataset, e.g., USAD[23]+ EWMA[10]+ rGain[42] for SWaT, TranAD[18]+ SpeedACC[10]+ Brits[37] for WADI, DAGMM[24]+ EWMA[22]+ Brits[37] for SMD, serving as the upper-bound comparator, to clean errors. These best-per-subtask baselines are described below.

Detection baselines. We compare HSD with DAGMM [24] (autoencoder and Gaussian mixture model), GDN [19] (graph neural network and attention), USAD [23] (autoencoder with Generative Adversarial Network (GAN)), TranAD [18] (Transformer with self-conditioning, meta-learning, adversarial training), to identify windows with errors of varying magnitudes.

Localization baselines. We compare MGL with Vari [11] (variance constraints), Speed [4] (max and min speed constraints), Speed+Acc [10] (max, min speed, and acceleration constraints) and EWMA [22] (exponentially weighted moving averages), to adaptively pinpoint errors within windows.

Repair baselines. We compare CAR with KNN [43] (linear average of k-nearest neighbors), rGain [42, 44] (bidirectional Recurrent Neural Network (RNN) and GAN), MPGRU [36] (diffusion convolutional RNN), Brits [37] (bidirectional RNN), and GRIN [38] (message passing and RNN), to repair errors.

4) Metrics: We evaluate EDITOR from two aspects: **Effectiveness**. For *cleaning* and *repair*, we report Mean Absolute Error (MAE), Mean Squared Error (MSE), and Mean Relative Error (MRE) to evaluate repaired data [38]: $MAE = \frac{1}{n}\sum_{i=1}^{n}|\tilde{y}_i - \hat{y}_i|$, evaluating the average accuracy; $MSE = \frac{1}{n}\sum_{i=1}^{n}(\tilde{y}_i - \hat{y}_i)^2$, assessing the volatility; $MRE = \frac{1}{m}\sum_{i=1}^{n}|\frac{\tilde{y}_i - \hat{y}_i}{\hat{y}_i}|$, assessing the relative accuracy [2]. For *detection* and *localization*, we use Precision (P), Recall (R), and F1-score ($F1$) to evaluate erroneous window identification and error localization [21, 23], where $P = \frac{TP}{TP+FP}$, indicating the correctness[3]; $R = \frac{TP}{TP+FN}$, assessing the completeness; $F1 = 2 \cdot \frac{P \cdot R}{P+R}$, harmonic mean of P and R. For *downstream tasks*, we use MSE and Silhouette Score (SS) to evaluate regression and clustering. Among them, $SS = \frac{\sum_{i=1}^{n}\frac{b(\tilde{y}_i)-a(\tilde{y}_i)}{max(a(\tilde{y}_i),b(\tilde{y}_i))}}{n}$, where a and b denote the average intra-cluster and nearest-cluster distances. **Efficiency**. We provide execution time, including training and testing time (i.e. running time) [18, 23], as well as the memory footprint in MB.

B. Cleaning evaluation (A1)

This experiment evaluates the overall performance of EDITOR, with results shown in Tab. 3.

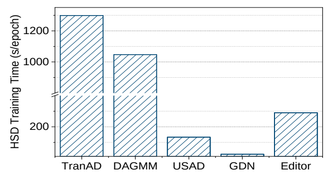
EDITOR outperforms baselines in handling diverse error patterns under complex relationships, achieving average reductions of 76%, 74.46%, and 59.82% in MAE, MSE, and MRE across datasets. This stems from its design (HSD, MGL, CAR), analyzed in §VI-C, VI-D, VI-E. While some methods perform competitively on specific metrics, EDITOR consistently delivers superior performance. Notably, it matches or beats CO across metrics as a unified framework. For efficiency, EDITOR is comparable to Vari [11] and outperforms CO in most cases, which incorporates deep-learning models. Given its cleaning effectiveness, the efficiency trade-off is reasonable.

[2] n is the amount of data, m is the number of batch iterations over the dataset, $\tilde{y}_i$ is the repaired value, and $\hat{y}_i$ is the truth value.

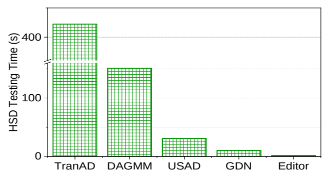
[3] TP, FP, FN denotes true positive, false positive, false negative, respectively.

Tab. 3
EFFECTIVENESS AND EFFICIENCY OF THE OVERALL CLEANING

Dataset	Methods	$MAE \downarrow$	$MSE \downarrow$	$MRE \downarrow$	$Running\ Time(s) \downarrow$
SWaT	Vari	0.547	0.268	0.303	699.173
	EWMA	0.494	0.231	0.270	1.394
	Clean4MTS	0.466	0.187	0.255	936
	CO	0.460	0.172	0.253	718
	EDITOR (ours)	0.075	0.025	0.120	712
WADI	Vari	0.291	0.171	0.463	455.924
	EWMA	0.308	0.224	0.507	0.5767
	Clean4MTS	0.256	0.183	0.419	601.9
	CO	0.214	0.157	0.362	901
	EDITOR (ours)	0.049	0.016	0.187	582.336
SMD	Vari	0.201	0.117	0.656	64.108
	EWMA	0.185	0.098	0.632	0.094
	Clean4MTS	0.179	0.085	0.630	170
	CO	0.123	0.035	0.699	101.5
	EDITOR (ours)	0.010	0.0005	0.077	166.012
PUMP	Vari	0.497	0.335	0.895	412.989
	EWMA	0.467	0.314	0.874	0.411
	Clean4MTS	0.398	0.254	0.697	565.32
	CO	0.321	0.160	0.580	460.945
	EDITOR (ours)	0.025	0.0025	0.083	532.800
MSL	Vari	0.012	0.002	1.240	604.899
	EWMA	0.015	0.004	1.322	0.211
	Clean4MTS	0.011	0.002	1.200	103.10
	CO	0.005	0.004	1.148	180.6
	EDITOR (ours)	0.003	0.002	0.864	292.78



(a) HSD training time



(b) HSD testing time

Fig. 6. Efficiency of HSD compared with baselines

C. Detection evaluation (A2)

This experiment assesses the effects of HSD, and the results are presented in Tab. 4 and Fig. 6.

Effectiveness. As shown in Tab. 4, HSD consistently outperforms all detection baselines in $F1$ across datasets. Though other methods (e.g., USAD, DAGMM) achieve higher P or R on certain datasets, they perform poorly on the other metrics. In contrast, HSD maintains balanced P and R, yielding an average $F1$ improvement of 23.25% over baselines. This superiority stems from the incorporation of dual attention mechanisms and difference-enhanced technique, enables HSD to exploit complex relationships and perceive differences among data to detect errors with varying deviations.

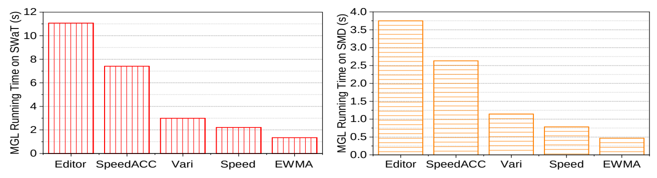
Efficiency. The training and testing time of HSD on SWaT are given in Fig. 6, and other datasets exhibit similar trends. In training, HSD is slightly higher than GDN and USAD, but 77% and 72% lower than the Transformer-based TranAD, which is parameter-intensive, and DAGMM, respectively. This is due to its advanced DA-UNet and DE-UNet designs, which despite possessing more parameters, allow efficient window-level processing. During testing, HSD is highly efficient, requiring 80%–98% less time than baselines, as it performs lightweight inference on the trained model.

Tab. 4
EFFECTIVENESS OF HSD ON DIFFERENT DATASETS

Methods	SWaT			WADI			SMD			PUMP			MSL		
	$P\uparrow$	$R\uparrow$	$F1\uparrow$	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	0.362	0.968	0.527
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	0.146	0.804	0.247
USAD	1	0.694	0.819	0.705	0.591	0.643	0.676	0.628	0.651	0.331	0.784	0.465	1	0.099	0.182
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	0.559	0.125	0.205
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	0.979	0.988	0.984

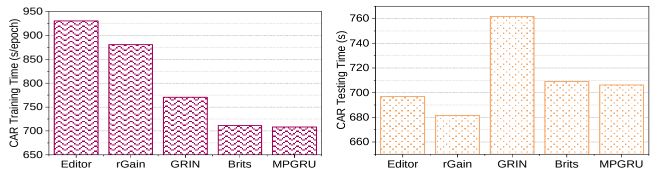
Tab. 5
EFFECTIVENESS OF MGL ON DIFFERENT DATASETS

Methods	SWaT			WADI			SMD			PUMP			MSL		
	$P\uparrow$	$R\uparrow$	$F1\uparrow$	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	0.698	0.719	0.709
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	0.671	0.641	0.656
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	0.640	0.557	0.595
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	0.754	0.978	0.851
MGL 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	0.759	0.969	0.851



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

Fig. 7. Efficiency of MGL compared with baselines



(a) CAR training time (b) CAR testing time

Fig. 8. Efficiency of CAR compared with baselines

D. Localization evaluation (A3)

This experiment investigates the MGL, with the results presented in Tab. 5 and Fig. 7.

Effectiveness. As shown in Tab. 5, MGL achieves superior $F1$ compared to localization baselines across all datasets. Vari, Speed, and SpeedACC perform poorly on P and R, whereas EWMA attains high R but low P. In contrast, MGL improves P while maintaining high R, leading to an average $F1$ increase of 38.77%. This stems from MGL's effective transformation of MTS relationships into temporal and variable constraints, adaptively identifying data point violations through AEA, allowing multi-granularity error localization.

Efficiency. The running time of MGL and localization baselines on SWaT and SMD datasets are illustrated in Fig. 7, with similar trends on others. MGL shows slightly higher time due to fine-grained calculations for assessing the violation degree of each data point against constraints, typically completing within seconds and can be further optimized via parallelism.

E. Repair evaluation (A4)

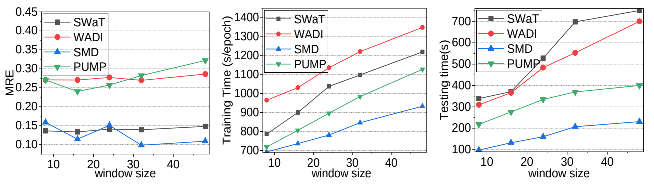
We report the performance of CAR and repair baselines in Tab. 6 and Fig. 8.

Effectiveness. Tab. 6 displays the MAE, MSE, and MRE on various datasets. It is clear that CAR demonstrates significant performance, achieving 10%, 7.5%, 12.75%, 6.42%, and 18.75% lower MRE than KNN, rGain, MPGRU, Brits, and GRIN. This stems from its two-stage bidirectional strategy—TCN for temporal dependencies and GCN for inter-variable relationships, ensuring context-aware and accurate repairs by fully leveraging bidirectional information.

Efficiency. As illustrated in Fig. 8, we evaluate the training and testing time of CAR and other repair baselines on SWaT dataset, with similar trends on others. KNN is excluded as it requires no training. Fig. 8(a) shows CAR has slightly higher training time due to its two-stage bidirectional repair strategy with TCN and GCN for modeling temporal and variable dependencies in forward and backward directions. Since training is offline, this overhead is acceptable. During testing, CAR demonstrates significant and competitive efficiency.

F. Parameter analysis (A5)

Fig. 9 investigates the performance of EDITOR with various window sizes on four datasets, with similar trends on others. As shown in Fig. 9(a), MRE remains relatively stable as the window size increases, demonstrating the robustness. Notably, the best MREs are achieved at window sizes between 16 and 32, depending on the dataset. However, training 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 effectiveness and efficiency, we set the window size to 32 as default.



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

Fig. 9. Effect of window sizes in EDITOR

Tab. 6
EFFECTIVENESS OF CAR ON DIFFERENT DATASETS

Methods	SWaT			WADI			SMD			PUMP			MSL		
	$MAE\downarrow$	$MSE\downarrow$	$MRE\downarrow$	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	0.007	0.004	1.301
rGain	0.417	0.138	0.173	0.323	0.103	0.283	0.046	0.005	0.103	0.280	0.064	0.291	0.037	0.022	1.105
MPGRU	0.482	0.235	0.199	0.474	0.190	0.417	0.069	0.010	0.145	0.323	0.079	0.330	0.037	0.022	1.097
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	0.004	0.003	1.135
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	0.035	0.021	1.021
CAR in EDITOR	0.075	0.025	0.120	0.049	0.016	0.187	0.010	0.0005	0.077	0.025	0.0025	0.083	0.003	0.002	0.864

Tab. 7
ABLATION STUDY OF EACH MODULE IN EDITOR

Module	SWaT			WADI			SMD			PUMP			MSL		
	$MAE\downarrow$	$MSE\downarrow$	$MRE\downarrow$	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	0.269	0.165	0.487	0.008	0.009	0.999
w/o MGL	0.172	0.100	0.324	0.199	0.120	0.673	0.124	0.029	0.700	0.299	0.155	0.541	0.008	0.009	0.901
w/o CAR	0.129	0.075	0.282	0.206	0.156	0.348	0.111	0.031	0.631	0.283	0.192	0.512	0.004	0.004	0.933
EDITOR	0.075	0.025	0.120	0.049	0.016	0.187	0.010	0.0005	0.077	0.025	0.0025	0.083	0.003	0.002	0.864

Tab. 8
ABLATION STUDY IN HSD

Component	SWaT			WADI			SMD			PUMP			MSL		
	$P\uparrow$	$R\uparrow$	$F1\uparrow$	P	R	$F1$	P	R	$F1$	P	R	$F1$	P	R	$F1$
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	0.432	0.999	0.603
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	0.509	0.886	0.648
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	0.979	0.988	0.984

Tab. 9
ABLATION STUDY IN MGL

Component	SWaT			WADI			SMD			PUMP			MSL		
	$P\uparrow$	$R\uparrow$	$F1\uparrow$	P	R	$F1$	P	R	$F1$	P	R	$F1$	P	R	$F1$
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	0.830	0.102	0.182
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	0.919	0.100	0.180
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	0.874	0.312	0.460
MGL 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	0.759	0.969	0.851

G. Ablation Study (A6)

This experiment first conducts ablation studies on the three modules (HSD, MGL and CAR) of EDITOR (Tab. 7), followed by ablations on key components in each module (Tabs. 8-10).

EDITOR. Tab. 7 reports MAE, MSE, and MRE on various datasets when HSD, MGL, or CAR is excluded from EDITOR and replaced by the best baseline method in the repair subtask (as described in Section VI-E). The full EDITOR with all designed modules achieves the best performance, while removing or replacing any single module leads to notable performance degradation across all metrics. Specifically, excluding HSD increases MRE by up to 0.58, mainly due to missed low-deviation errors, which are difficult to detect without explicitly enhancing difference patterns. Removing MGL increases it by 0.62, indicating that coarse window-level correction tends to introduce excessive modifications to originally clean data, especially when errors are sparse within a window. And replacing CAR increases it by 0.55, suggesting that the model struggles to preserve contextual consistency when repairing errors under complex temporal and variable dependencies. Other metrics show similar trends. These results showcase that each module in EDITOR plays a vital role: HSD captures complex relationships and detects windows containing varying-magnitude errors, MGL localizes multi-granularity errors and prevents unnecessary whole window modifications, and CAR repairs localized errors by maintaining contextual consistency. Among these, MGL is the most critical, as its removal incurs extensive alterations to the originally correct data, resulting in deuterogenic errors as discussed in §II-A. Below, we further remove key components from each module to verify the necessity of their design.

HSD. Tab. 8 displays P, R and $F1$ when dual attention or difference-enhanced technique in HSD is omitted. The exclusion of either leads to declines in P, R and $F1$ across all datasets, hindering HSD's ability to focus on critical relationships in MTS and amplify discrepancies, reducing its effectiveness in identifying windows with high- and low-deviation errors.

MGL. As shown in Tab. 9, removing the constraint conditions g_v, g_t, or AEA from MGL results in lower P, R, and $F1$. This is because relying solely on a constraint condition based on a single relationship is insufficient to capture variable and temporal distortions necessary for finding multi-granularity errors, while neglecting the AEA increases omissions by failing to maximize the violation distance of errors.

CAR. Tab. 10 indicates that the MAE, MSE and MRE deteriorate significantly when either the TCN, GCN or bidirection component is excluded from CAR, highlighting the importance of leveraging the bidirectional temporal and variable relationships in MTS to achieve accurate repair while effectively preventing variable and temporal distortions.

H. Downstream Tasks (A7)

To demonstrate the cleaning effectiveness, we compare EDITOR with baseline methods by applying their cleaned data

Tab. 10
ABLATION STUDY IN CAR

Component	SWaT			WADI			SMD			PUMP			MSL		
	$MAE\downarrow$	$MSE\downarrow$	$MRE\downarrow$	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	0.004	0.004	0.917
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	0.0039	0.0027	0.872
w/o Bidirection	0.088	0.028	0.141	0.051	0.019	0.202	0.010	0.001	0.080	0.093	0.020	0.326	0.004	0.004	0.900
CAR in EDITOR	0.075	0.025	0.12	0.049	0.016	0.187	0.010	0.0005	0.077	0.025	0.0025	0.083	0.003	0.002	0.864

Tab. 11
EVALUATION ON DOWNSTREAM REGRESSION TASK (MSE ↓)

Methods	SWaT	WADI	SMD	PUMP	MSL
Clean	0.831	0.489	0.706	0.872	0.476
Dirty	0.931	0.961	0.836	0.926	0.715
Vari	0.936	0.955	0.808	0.900	0.582
EWMA	0.935	0.980	0.856	0.916	0.768
Clean4MTS	0.901	0.906	0.829	0.911	0.699
CO	0.887	0.495	0.855	0.924	0.501
EDITOR	0.859	0.493	0.770	0.875	0.489

Tab. 12
EVALUATION ON DOWNSTREAM CLUSTERING TASK (SS ↑)

Methods	SWaT	WADI	SMD	PUMP	MSL
Clean	0.955	0.992	0.545	0.410	0.995
Dirty	0.623	0.203	0.269	0.189	0.590
Vari	0.468	0.151	0.313	0.394	0.745
EWMA	0.275	0.100	0.376	0.401	0.638
Clean4MTS	0.540	0.131	0.302	0.372	0.701
CO	0.952	0.170	0.314	0.399	0.967
EDITOR	0.953	0.947	0.400	0.431	0.989

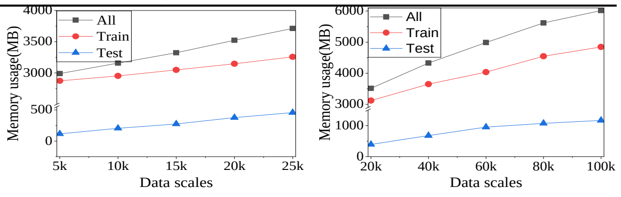
into downstream regression and clustering tasks. “Clean” and “Dirty” refer to the performance on the original clean and erroneous dataset (§VI-A2). For regression, datasets are split into training and testing sets, an autoencoder is trained to generate the testing data, with MSE (§VI-A) as the metric. For clustering, K-means is applied to datasets, with SS as the metric. As shown in Tab. 11 and Tab. 12, EDITOR consistently outperforms other methods and comparable to “Clean” performance, owing to its effective handling of errors with varying magnitudes and granularities through modeling complex temporal and inter-variable dependencies.

I. Analysis of AEA in MGL and its simple alternative (A8)

We compare MGL with AEA against the AEA alternative, a basic constraint-and-probability approach [4, 12] that computes an error probability for each point based on its violation distance distribution and selects the top-k points with the highest probabilities as identified errors. As shown in Tab. 13, AEA consistently outperforms the alternative across datasets and metrics. The alternative fails to effectively localize multi-granularity errors in the complex decision space induced by coupled temporal and variable dimensions, relying on probabilistic selection rather than objective-driven optimization. In contrast, AEA refines error localization via constraint-guided optimization, where its superiority stems from both the constraints and the optimization process.

Tab. 13
IMPACT OF REPLACING AEA IN MGL WITH THE SIMPLE ALTERNATIVE

Dataset	Components	$P\uparrow$	$R\uparrow$	$F1\uparrow$
SWaT	AEA alternative	0.360	0.077	0.127
	AEA	0.738	0.780	0.758
WADI	AEA alternative	0.552	0.534	0.543
	AEA	0.622	0.804	0.701
SMD	AEA alternative	0.858	0.505	0.636
	AEA	0.876	0.855	0.865
PUMP	AEA alternative	0.238	0.323	0.274
	AEA	0.801	0.642	0.713
MSL	AEA alternative	0.886	0.035	0.067
	AEA	0.759	0.969	0.851



(a) On SMD dataset (b) On PUMP dataset

Fig. 10. Scalability of EDITOR

J. Analysis of potential error propagation in TCN (A9)

We evaluate the impact of error propagation in the TCN by replacing the uncorrected points in the original W_o with mean values or weighted averages based on temporal distance (refer to [4]), as shown in Tab. 14. On the SMD, PUMP, and WADI datasets, replacing uncorrected points does not consistently improve the performance of the TCN compared to the original ”dirty” inputs. Specifically, on SMD, both mean and weighted average replacements perform well; on PUMP, these methods remain effective but show limited improvement; whereas on WADI, neither mean nor weighted average replacements are effective. These results indicate that preprocessing W_o may alleviate error propagation to some extent. However, the effectiveness of the optimal imputation method is data-dependent and suboptimal strategies may introduce biases more severe than the original errors, warranting further study.

Tab. 14
IMPACT OF POTENTIAL ERROR PROPAGATION IN TCN

Forms	SMD			PUMP			WADI		
	MAE (↓)	MSE (↓)	MRE (↓)	MAE (↓)	MSE (↓)	MRE (↓)	MAE (↓)	MSE (↓)	MRE (↓)
Mean	0.0165	0.0012	0.1197	0.1259	0.0563	0.4000	0.224	0.124	0.773
Weight	0.0172	0.0013	0.1263	0.1211	0.0546	0.3899	0.228	0.101	0.786
Original input	0.0189	0.0014	0.1405	0.1561	0.0873	0.4002	0.099	0.041	0.359

K. Memory footprint in EDITOR *and its scalability (A10)*

We analyze the memory footprint of EDITOR and its modules, as well as its scalability on long-horizon MTS in Tab. 15 and Fig. 10. As shown in Tab. 15, on SMD and PUMP datasets, EDITOR’s memory usage during training is dominated by HSD and CAR modules, which together account for about 80% of total EDITOR’s memory usage on average; similar trends hold on other datasets. During inference, EDITOR’s memory usage is determined by HSD and is reduced overall. Fig. 10 shows that as sequence length increases by 5k or 20k points per step (depending on the dataset), EDITOR’s memory usage grows nearly linearly, while inference scales much slower than training, enabling efficient handling of long-horizon MTS.

Tab. 15
MEMORY FOOTPRINT IN EDITOR AND ITS MODULES (MB ↓)

Forms	SMD				PUMP			
	HSD	MGL	CAR	EDITOR	HSD	MGL	CAR	EDITOR
All	3539.36	21.93	266.57	3827.86	5699.16	53.27	1043.35	6795.78
Train	2945.72	–	219.94	3165.66	4522.25	–	915.51	5437.76
Test	593.64	21.93	46.63	662.20	1176.91	53.27	127.84	1358.02

VII. Related work

Classical cleaning methods. Classical MTS cleaning methods can be broadly divided into four categories: *Smoothing-based*, *Arithmetic-based*, *Constraint-based*, and *Statistical* strategies. *Smoothing-based* methods compute (exponentially) weighted averages of neighboring points to capture short-term dependencies for cleaning errors, e.g., MA [45] and EWMA [22]. *Arithmetic-based* ones use the arithmetic expressions derived from temporal and variable dimensions to detect and repair MTS. For example, ARIMA [46] applies linear regression to capture short-term dependencies between current and past observations. CR [6] models linear arithmetic relationships among variables. Clean4MTS [21] identifies errors via linear arithmetic relationships between variables and repairs them using short-term temporal dependencies. MTSClean and MTSClean-soft [7] exploit short-term and linear relationships across temporal and variable dimensions to clean errors. *Constraint-based* methods rely on speed, acceleration [4], multi-speed [10], or variance [11] for MTS cleaning, effectively capturing short-term temporal dependencies extracted from historical data (often represented by maximum and minimum constraints) to detect and repair errors. *Statistical* methods [12, 13] aim to maximize the probability of speed changes in temporal dimension during the cleaning process.

Classical methods are lightweight [14] but struggle to jointly model complex temporal and variable dependencies in MTS, limiting their effectiveness in complex MTS scenarios [14, 16]. Moreover, they often fail to handle varying-magnitude errors.

Deep learning-based methods. Recently, deep learning-based anomaly detection methods have been adapted for MTS cleaning [18, 23, 47], leveraging complex MTS dependencies via sliding windows and reconstruction-based strategies to achieve efficient cleaning performance. Several methods are inefficient to simultaneously model both temporal and variable dependencies. NumentaHTM [48] exploits long-term temporal dependencies but not inter-variable relationships. TripleES [47] captures short-term temporal and partial linear inter-variable relationships, yet struggles with nonlinear ones. DCdetector [49] is effective in capturing short- and partial long-term temporal dependencies but lacks explicit inter-variable modeling. Nominality [35] models temporal dependencies according to error types, without inter-variable relationships. USAD [23] captures temporal dependencies but handles nonlinear inter-variable relations poorly. TranAD [18] captures temporal dependencies and attempts inter-variable modeling. RobustTAD [50] combines time-series decomposition with a generic U-Net only for temporal dependencies. DiffAD [51] integrates a structured state-space sequence model into the U-Net of diffusion model to enhance temporal modeling, but neither explicitly models inter-variable relationships. Moreover, these methods tend to identify entire windows as erroneous, rather than specific multi-granularity errors, modifying correct data and introducing induced errors [14, 16]. Additionally, they often rely on fixed thresholds to determine whether the reconstruction data deviate from the input, making them ineffective for windows with errors of varying magnitudes. For example, DCdetector [49] uses prior thresholds, while Nominality [35] applies predefined thresholds. Furthermore, imputation-based methods—GRIN [38], MPGRU [36], rGain [42, 44], Brits [37], BiTGraph [52], and ModernTCN [53]—use DL models (e.g., TCN or GCN) to model unidirectionally temporal or variable dependencies for missing value imputation, but typically neglect other error types, failing to achieve efficient error detection and repair.

Conversely, EDITOR leverages complex temporal and variable relationships to clean errors of varying granularities and magnitudes via Detect-Localize-Repair.

VIII. Conclusion

This paper proposes EDITOR, a multi-resolution MTS cleaning framework that leverages complex temporal and variable dependencies for errors with varying granularities and magnitudes. EDITOR is an efficient Detect-Localize-Repair pipline, consisting of high-sensitivity detection, multi-granularity localization, and context-aware repair to facilitate MTS cleaning. Comprehensive experiments show that EDITOR significantly outperform baselines and improve the performance for downstream analysis tasks. Future work aims to improve the scalability for large-scale, high-dimensional MTS data.

IX. AI-Generated Content Acknowledgement

The authors confirm that no AI-generated content was used in the creation of the technical content of this paper. Only revision and proofreading tools were employed to improve grammar and readability, without altering the technical content, ideas, or conclusions.

References

[1] R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley, "Deep learning for healthcare: review, opportunities and challenges," *Briefings in bioinformatics*, vol. 19, no. 6, pp. 1236–1246, 2018.

[2] S. J. Plathottam, A. Rzonca, R. Lakhnori, and C. O. Iloeje, "A review of artificial intelligence applications in manufacturing operations," *Journal of Advanced Manufacturing and Processing*, vol. 5, no. 3, p. e10159, 2023.

[3] K. L. Siau, F. F. H. Nah, Y. Qian, B. L. Eschenbrenner, and L. Chen, "Artificial intelligence in financial technology," in *15th China Summer Workshop on Information Management (CSWIM 2022)*, 2022.

[4] S. Song, A. Zhang, J. Wang, and P. S. Yu, "Screen: Stream data cleaning under speed constraints," in *SIGMOD*, 2015, pp. 827–841.

[5] S. Galhotra, A. Fariha, R. Lourenço, J. Freire, A. Meliou, and D. Srivastava, "Dataprism: Exposing disconnect between data and systems," in *SIGMOD*, 2022, pp. 217–231.

[6] A. Fariha, A. Tiwari, A. Radhakrishna, S. Gulwani, and A. Meliou, "Conformance constraint discovery: Measuring trust in data-driven systems," in *Proceedings of the*

2021 International Conference on Management of Data, 2021, pp. 499–512.

[7] X. Ding, Y. Song, H. Wang, C. Wang, and D. Yang, "Mtsclean: Efficient constraint-based cleaning for multi-dimensional time series data," *Proc. VLDB Endow.*, vol. 17, no. 13, pp. 4840–4852, 2024.

[8] A. Zhang, S. Deng, D. Cui, Y. Yuan, and G. Wang, "An experimental evaluation of anomaly detection in time series," *Proceedings of the VLDB Endowment*, vol. 17, no. 3, pp. 483–496, 2023.

[9] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2828–2837.

[10] S. Song, F. Gao, A. Zhang, J. Wang, and P. S. Yu, "Stream data cleaning under speed and acceleration constraints," *ACM Transactions on Database Systems (TODS)*, vol. 46, no. 3, pp. 1–44, 2021.

[11] W. Yin, T. Yue, H. Wang, Y. Huang, and Y. Li, "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, 2018, pp. 108–113.

[12] A. Zhang, S. Song, and J. Wang, "Sequential data cleaning: A statistical approach," in *Proceedings of the 2016 International Conference on Management of Data*, 2016, pp. 909–924.

[13] H. Wang, A. Zhang, S. Song, and J. Wang, "Streaming data cleaning based on speed change," *The VLDB Journal*, vol. 33, no. 1, pp. 1–24, 2024.

[14] X. Han, H. Xiong, Z. He, P. Wang, C. Wang, and X. S. Wang, "Akane: Perplexity-guided time series data cleaning," *SIGMOD*, vol. 2, no. 3, pp. 1–26, 2024.

[15] A. Zhang, Z. Wu, Y. Gong, Y. Yuan, and G. Wang, "Multivariate time series cleaning under speed constraints," *Proceedings of the ACM on Management of Data*, vol. 2, no. 6, pp. 1–26, 2024.

[16] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: identifying density-based local outliers," in *SIGMOD*, 2000, pp. 93–104.

[17] G. Angloher, S. Banik, D. Bartolot, G. Benato, A. Bento, A. Bertolini, R. Breier, C. Bucci, J. Burkhart, L. Canonica ***et al.***, "Towards an automated data cleaning with deep learning in cresst," *The European Physical Journal Plus*, vol. 138, no. 1, pp. 1–11, 2023.

[18] S. Tuli, G. Casale, and N. R. Jennings, "Tranad: Deep transformer networks for anomaly detection in multivariate time series data," *arXiv preprint arXiv:2201.07284*, 2022.

[19] A. Deng and B. Hooi, "Graph neural network-based anomaly detection in multivariate time series," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 5, 2021, pp. 4027–4035.

[20] F. Gao, S. Song, and J. Wang, "Time series data cleaning under multi-speed constraints." *Int. J. Softw. Informatics*, vol. 11, no. 1, pp. 29–54, 2021.

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

[22] R. Fried and A. C. George, "Exponential and holt-winters smoothing." 2011.

[23] J. Audibert, P. Michiardi, F. Guyard, S. Marti, and M. A. Zuluaga, "Usad: Unsupervised anomaly detection on multivariate time series," in *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, 2020, pp. 3395–3404.

[24] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, and H. Chen, "Deep autoencoding gaussian mixture model for unsupervised anomaly detection," in *International conference on learning representations*, 2018.

[25] J. Goh, S. Adepu, K. N. Junejo, and A. Mathur, "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, 2017, pp. 88–99.

[26] K. Berahmand, F. Daneshfar, E. S. Salehi, Y. Li, and Y. Xu, "Autoencoders and their applications in machine learning: a survey," *Artificial Intelligence Review*, vol. 57, no. 2, p. 28, 2024.

[27] Y. Li, S. Gao, and W. Zhao, "Transformer-based autoencoder with combined multi-head-attention for industrial soft-sensor modeling," *Engineering Applications of Artificial Intelligence*, vol. 159, p. 111681, 2025.

[28] K. Pawar and V. Z. Attar, "Assessment of autoencoder architectures for data representation," *Deep learning: concepts and architectures*, pp. 101–132, 2020.

[29] O. Ronneberger, P. Fischer, and T. Brox, "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, 2015, pp. 234–241.

[30] K. Trebing, T. Sta̐czyk, and S. Mehrkanoon, "Smaat-unet: Precipitation nowcasting using a small attention-unet architecture," *Pattern Recognition Letters*, vol. 145, pp. 178–186, 2021.

[31] G. Sun, Y. Pan, W. Kong, Z. Xu, J. Ma, T. Racharak, L.-M. Nguyen, and J. Xin, "Da-transunet: integrating spatial and channel dual attention with transformer u-net for medical image segmentation," *Frontiers in Bioengineering and Biotechnology*, vol. 12, p. 1398237, 2024.

[32] A. Lin, B. Chen, J. Xu, Z. Zhang, G. Lu, and D. Zhang, "Ds-transunet: Dual swin transformer u-net for medical image segmentation," *IEEE Transactions on Instrumentation and Measurement*, vol. 71, pp. 1–15, 2022.

[33] W. Liu, Y. Sun, and Q. Ji, "Mdan-unet: multi-scale and dual attention enhanced nested u-net architecture for

segmentation of optical coherence tomography images," *Algorithms*, vol. 13, no. 3, p. 60, 2020.

[34] T. Wong and Z. Luo, "Recurrent auto-encoder model for large-scale industrial sensor signal analysis," in *Engineering Applications of Neural Networks*. Springer, 2018, pp. 203–216.

[35] C.-Y. A. Lai, F.-K. Sun, Z. Gao, J. H. Lang, and D. Boning, "Nominality score conditioned time series anomaly detection by point/sequential reconstruction," *Advances in Neural Information Processing Systems*, vol. 36, pp. 76 637–76 655, 2023.

[36] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," *arXiv preprint arXiv:1707.01926*, 2017.

[37] W. Cao, D. Wang, J. Li, H. Zhou, L. Li, and Y. Li, "Brits: Bidirectional recurrent imputation for time series," *Advances in neural information processing systems*, vol. 31, 2018.

[38] A. Cini, I. Marisca, and C. Alippi, "Filling the g_ap_s: Multivariate time series imputation by graph neural networks," *arXiv preprint arXiv:2108.00298*, 2022.

[39] Y. Wang, Z. Liu, D. Hu, and M. Zhang, "Multivariate time series prediction based on optimized temporal convolutional networks with stacked auto-encoders," in *Asian Conference on Machine Learning*. PMLR, 2019, pp. 157–172.

[40] C. M. Ahmed, V. R. Palleti, and A. P. Mathur, "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*, 2017, pp. 25–28.

[41] H. K, C. V, and L. C, "Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018, pp. 387–395.

[42] X. Miao, Y. Wu, J. Wang, Y. Gao, X. Mao, and J. Yin, "Generative semi-supervised learning for multivariate time series imputation," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 10, 2021, pp. 8983–8991.

[43] C. K. R, "Knnimputer: A robust way to impute missing values (using scikit-learn)," 2020.

[44] J. Yoon, J. Jordon, and M. Schaar, "Gain: Missing data imputation using generative adversarial nets," in *International conference on machine learning*. PMLR, 2018, pp. 5689–5698.

[45] D. R. Brillinger, *Time series: data analysis and theory*. SIAM, 2001.

[46] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[47] A. Aboode, "Anomaly detection in time series data based on holt-winters method," 2018.

[48] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, "Unsupervised real-time anomaly detection for streaming data," *Neurocomputing*, vol. 262, pp. 134–147, 2017.

[49] Y. Yang, C. Zhang, T. Zhou, Q. Wen, and L. Sun, "Dcdetector: Dual attention contrastive representation learning for time series anomaly detection," in *Proceedings of the 29th ACM SIGKDD conference on knowledge discovery and data mining*, 2023, pp. 3033–3045.

[50] J. Gao, X. Song, Q. Wen, P. Wang, L. Sun, and H. Xu, "Robusttad: Robust time series anomaly detection via decomposition and convolutional neural networks (2020)," *arXiv preprint arXiv:2002.09545*, 2002.

[51] C. Xiao, Z. Gou, W. Tai, K. Zhang, and F. Zhou, "Imputation-based time-series anomaly detection with conditional weight-incremental diffusion models," in *Proceedings of the 29th ACM SIGKDD conference on knowledge discovery and data mining*, 2023, pp. 2742–2751.

[52] X. Chen, X. Li, B. Liu, and Z. Li, "Biased temporal convolution graph network for time series forecasting with missing values," in *The Twelfth International Conference on Learning Representations*, 2023.

[53] D. Luo and X. Wang, "Moderntcn: A modern pure convolution structure for general time series analysis," in *The twelfth international conference on learning representations*, 2024, pp. 1–43.