CUR-Estimator: Towards reliable missing data imputation for aero-engine degradation process

Haoze Wu [a], Shisheng Zhong [a,b,c,*] , Minghang Zhao [b,c,*], Xuyun Fu [b,c], Yongjian Zhang [b,c], Song Fu [a]

[a] School of Mechatronics Engineering, Harbin Institute of Technology, Harbin 150001, China
[b] Department of Mechanical Engineering, Harbin Institute of Technology, Weihai 264209, China
[c] Weihai Key Laboratory of Intelligent Operation and Maintenance, Harbin Institute of Technology, Weihai 264209, China

Abstract: In the process of missing data imputation for aero-engine life cycle degradation time series, two primary challenges arise. First, due to variations in the duration of different flight missions, and the fact that the same flight phase may also vary in duration across different flights, the time intervals for collecting key samples are not always consistent. This variability increases the difficulty of evaluating the impact of individual flights on overall performance changes. Second, when using neural networks for imputing missing data, issues such as significant noise or extended periods of missing data may arise, leading to unreasonable imputation results. To address the challenges, this paper proposes a Constrained Unseen Recovery Estimator (CUR-Estimator) for imputing missing data in aero-engine life cycle degradation datasets. Firstly, the time interval information is encoded via a transformer-enhanced gate recurrent unit. The results are then combined with missing masks to adjust the hidden states and input weights for the missing segments, forming the Interval-Aware Temporal Imputation Network. Secondly, this paper uses statistical interpolation methods to constrain the imputation results of the neural network, limiting the range of imputation outcomes and thereby reducing the possibility of unreasonable outputs. As an example, the Piecewise Cubic Hermite Interpolating Polynomial is applied to constrain the Interval-Aware Temporal Imputation Network in handling time interval information. Finally, experiments were conducted using a simulation dataset and a real civil aero-engine dataset, which showed that the proposed method has high accuracy and strong stability.

Keywords: aero-engine, data imputing, gated recurrent unit, time interval

1 Introduction

The accurate collection of environmental conditions and machine status by sensors has promoted the implementation of Prognostics and Health Management (PHM) for complex equipment. By installing many sensors during the design phase, researchers can monitor the status of complex equipment during operation, providing a foundation for subsequent maintenance and support work. However, due to potential sensor malfunctions during operation, negligence during manual operation, and regulations regarding multi-departmental information sharing, the state data collected may be incomplete. This issue of incomplete data introduces challenges to subsequent PHM analysis.

There are various factors that can lead to data loss during the operation of aero-engines. First, some diagnostic procedures must be conducted under specific operating conditions. For instance, detecting cracks in engine blades typically requires the engine to be shut down, making it impossible to synchronously acquire certain signals under operational conditions [1]. Second, aero-engines often operate under extreme conditions, where some sensors may exceed

* Corresponding authors.
E-mail addresses: 21b908077@stu.hit.edu.cn (H. Wu), zhongss@hit.edu.cn (S. Zhong), zhaomh@hit.edu.cn (M. Zhao), fuxuyun@hit.edu.cn (X. Fu), zhangyj@hitwh.edu.cn (Y. Zhang), fusong@hit.edu.cn (S. Fu)

their measurement range, resulting in missing or distorted data due to over-threshold limitations [2]. Additionally, for signals that are long-term stable or change slowly, the sampling frequency is often reduced to lower acquisition costs. However, this practice may prevent the system from capturing critical abrupt changes in a timely manner, leading to intermittent data loss.

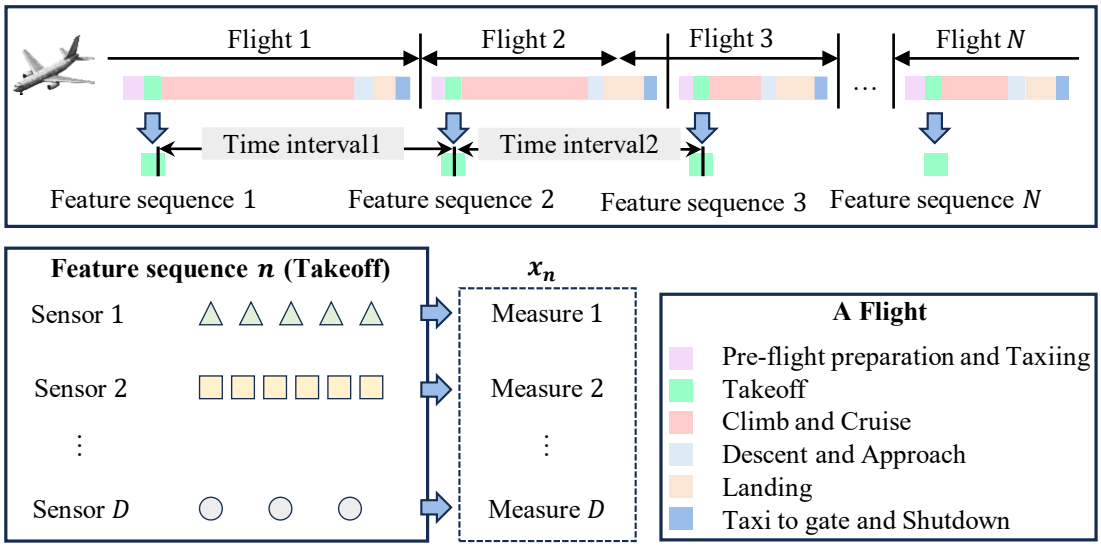

Fig. 1. Imputation data sources for aero-engine life cycle degradation.

The issue of data missing in aero-engines differs from the general problem of missing data. For general data missing problems, sensor data is typically collected at fixed time intervals. However, in aero-engine data analysis, due to the long duration of individual flights and the large volume of data generated, a common approach is to focus on key flight phases—such as takeoff and climb—and extract representative feature values from these segments for combined analysis. This strategy helps reduce data dimensionality while concentrating on critical operating conditions. Fig. 1 illustrates the data sources used for the aero-engine imputation task. The upper part of Fig. 1 depicts the complete usage cycle of an engine installed on an aircraft, from installation to removal. The lower right of Fig. 1 shows the phases of a typical flight. During each flight, the engine typically undergoes the full process, from pre-flight preparation and takeoff to final shutdown. Due to the variation in flight missions, the duration of each flight varies, resulting in different total engine running durations for each flight. At the same time, the duration of the same stage (such as takeoff or cruise) of two flights is generally different. This leads to a more complex relationship between the sampling of two adjacent flights (i.e. time interval1 and time interval2) and the changes in performance state between the two samples. Fig. 1 (bottom-left) illustrates the method for collecting key performance data from a single flight. In analyzing engine performance degradation, the sensor data sequence from a key stage (such as takeoff) is typically used as a sample. To account for discrepancies in sampling intervals between sensors, descriptive statistics are extracted from each sequence based on the specific characteristics of the sensors. These statistics represent the sensor's performance during the takeoff stage and, for the purpose of analysis, serve as a simplified representation of the engine's overall performance for the entire flight. The combination of descriptive statistics from multiple sensor sequences during a specific stage of a flight is used to represent the performance state $\boldsymbol{x_n}$ at a certain timestamp in the engine′s life cycle. The complete life cycle data for a single engine can then be represented as $\boldsymbol{X} = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3} \ldots, \boldsymbol{x_N}\}$. General time series imputation methods assume equal intervals between sampling times. However, applying such methods to $\boldsymbol{X}$ not only ignores the differences in sampling intervals between each feature sequence, but also overlooks the varying durations of different stages within a single flight.

There are various methods to process data containing missing values. While directly deleting the part of the data with missing values is the simplest and easiest approach, it is not applicable in all scenarios, particularly for time series data such as aero-engine operational data [3,4]. This is because the characteristics of multivariate time series data are reflected in the continuity of time and the correlation between different variables. If some data is deleted, this continuous relationship may be disrupted, leading to significant deviations in subsequent tasks. For aero-engine data, the absence of entire flight data may hinder subsequent anomaly detection, fault diagnosis, and life prediction tasks. Therefore, imputing missing data is considered a better approach for handling incomplete performance data of aero-engines.

Statistical methods for imputing missing values are the most used in industry due to their interpretability and ease of calculation. One of the most basic statistical imputation methods involves using descriptive statistics for imputing, such as forward imputing, backward imputing, mean imputation, and mode imputation. These methods are fast and can satisfy the requirement of eliminating missing values in the final dataset. However, they often neglect the interdependencies between temporal and feature dimensions, resulting in suboptimal imputation performance in many cases. Additionally, there are other statistical methods for imputation that are based on observed mathematical principles. For instance, the cubic spline interpolation method [5] constructs continuous cubic polynomial curves and constrains the data and derivatives of the curves at each point to be continuous, thus imputing the missing values. The special treatment of derivatives in the Piecewise Cubic Hermite Interpolating Polynomial (PCHIP) method [6] ensures stability in the imputation results and avoids excessive oscillations of the interpolation function. K-nearest neighbors (KNN) can also be applied to time series data, assuming that missing data is related to nearby values. Typically, the feature values of the nearest neighbors are used as the imputed values. Pan et al. [7]proposed a weighted gray KNN imputation method that incorporates the correlations between multiple sequences into the estimation process of the imputed values. Bashir & Wei [8] propose a method for data imputation based on vector autoregression to address challenges posed by dynamic data. Furthermore, when dealing with non-stationary processes, dynamic linear models can handle these processes through Kalman filtering and smoothing. This approach is also considered a fundamental method for data imputation [9]. However, these methods are constrained by the observable patterns, and the scope of patterns they can capture is relatively limited.

Autoencoders are also frequently employed for data imputation tasks [10], due to their strong ability to compress and reconstruct features. When applying autoencoders for imputation, the data is first preprocessed, and missing values are filled with zeros or statistical estimates from the sample data. Next, the autoencoder is trained: the processed data is input into the encoder for feature compression, and then the decoder restores the data. The autoencoder is optimized based on the reconstruction effect, observed by comparing the observed data with the decoder's output. Finally, the incomplete data is preprocessed and input into the autoencoder again, and the missing values in the decoder's output are regarded as the imputation results. Liu et al. [11] proposed a non-autoregressive multi-resolution interpolation method, based on an autoencoder model with a bidirectional encoder and multi-resolution decoder. This approach divides time series into different granularities for recursive decoding, and then uses a Transformer to extract information from both the temporal and feature dimensions. Mulyadi et al. [12] introduced an imputation method based on a Variational Autoencoder (VAE), which learns data distribution from both the temporal and feature dimensions, while also considering the uncertainty in the imputation results. In addition, Pan et al. [13] used median replacement and adaptive learning strategies to fill in missing data values in industrial data. However, tasks based on autoencoder methods typically only involve reconstruction errors, and autoencoder networks are not explicitly trained to impute missing data, thus their effectiveness in generating missing values is limited [14].

The diffusion model has also been applied to imputation for missing values in time series data. One of the most important concepts in this context is the score function, which is trained during the forward process of the diffusion model. By adding noise to the original data, the model learns the transformation law from the signal to the noise. Then, through the reverse process, guided by the score function, the initial noise is gradually converted back into the signal, yielding the imputation results. The primary difference between the diffusion model and other methods lies in the training process, where the score function is typically implemented using common machine learning architectures, such as Transformer networks. Tashiro et al. [15] proposed a Conditional Score-based Diffusion model for Imputation (CSDI), which uses an autoencoder to capture both temporal and feature dependencies, completing the imputation process. Zheng & Charoenphakdee [16] optimized the handling of tabular data based on CSDI, replacing the original temporal transformation layer with residual connections and multilayer perceptron (MLP) networks. However, this method has strict requirements regarding both the quantity and quality of the data. When the amount of data is small or the data quality is poor, the model may struggle to fully learn the distribution characteristics of the data. This issue is particularly pronounced in industrial scenarios, where data is often limited in both quantity and quality.

Transformer has an advantage in processing time series data, and thus, time series imputation methods based on Transformers have been extensively studied in recent years. Transformer can be used to build autoencoder structures, which are like autoencoder-based imputation methods. However, in these methods, the network used to capture patterns is replaced by a Transformer. The imputation process of this method consists of three main steps. First, a Transformer-based autoencoder structure is established. Then, the training set is used to perform the task of reconstructing the observed values. Finally, the trained autoencoder is applied to the test set, and the corresponding outputs are used as the imputed results. Suo et al. [17] used the self-attention mechanism to learn both global and local dependencies of multivariate time series, while employing multi-dimensional self-attention to capture cross-temporal and cross-feature correlations. Yıldız et al. [18] randomly masked some missing values in the training set and optimized the network by jointly reconstructing the observed values and imputing the missing ones. Du et al. [14] constructed two Diagonally-Masked Self-Attention blocks to capture deep latent patterns across both temporal and feature dimensions. However, this method does not account for varying time intervals in the data, treating all intervals as equal, and it also fails to consider the issue of imputing unreasonable values.

Imputation tasks are often addressed using recurrent neural networks (RNNs). These methods first apply simple statistical techniques (such as mean imputation, forward filling, or cubic spline interpolation) to preliminarily fill missing values, and then use the filled complete dataset to train an RNN. During the training phase, the error between the reconstruction results of the RNN and the observed value without missing entries is calculated. This error is used as the loss function to optimize the RNN network parameters. The trained RNN network can provide a complete reconstruction based on the preliminarily filled data, and the missing parts in the reconstruction results can be used as the imputed values in this method. Che et al. [19] designed a decay mechanism and integrated it into the Gated Recurrent Unit (GRU), naming it GRU-D. This mechanism incorporates time intervals as input into the GRU to evaluate the importance of the observed values near missing data points. Cao et al. [20] proposed BRITS, which designs a new RNN structure based on GRU-D's decay mechanism. This structure gathers information from both past and future time steps of the sequence, addressing the issue of data dynamics. Yoon et al. [21] proposed an imputation method based on M-RNN, which, unlike Bi-RNN, extracts information across both the time and feature dimensions. Additionally, combining the RNN network structure with the GAN training process is another feasible imputation approach [4,22]. Kim and Chi [23] introduced the concept of Time Belief Memory,

which calculates the beliefs of certain observations based on the extent of missing data and the duration of the missingness, and then selects imputation methods based on these beliefs.

The current method, when applied to aero-engines, presents two main challenges. First, the performance of the current neural network is unstable when applied to actual industrial data, and sometimes generates imputed values that exceed the normal range, resulting in unreasonable values. Although these unreasonable values can be uniformly handled after imputation, this approach does not address the root issue that occurs during the imputation process itself. Instead, it merely replaces the unreasonable value with another imputed feature value. Secondly, the operational duration of aero-engines is not consistent across different flights, and various operating conditions with different durations accumulate during each flight. For civil aviation aircraft, a complete flight process mainly includes takeoff, climb, cruise, descent, and landing. In special cases, go-arounds, holding patterns, and engine ground testing may also be involved. These different states result in inconsistencies in operating time and performance degradation between two consecutive flights. The traditional imputation methods, which typically handle data with equal-interval sampling, are not well-suited for such condition.

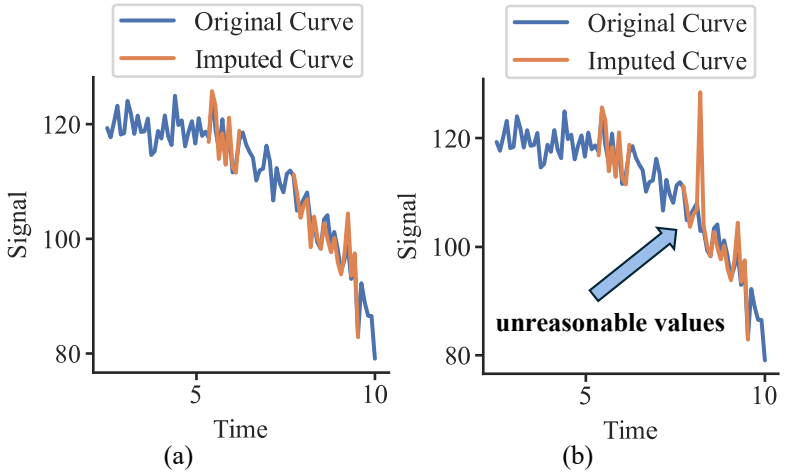

Fig. 2. A schematic illustration of imputation using a neural network-based method, which may produce unreasonable values. (a)The ideal result of neural network-based imputation, (b) the potential outcomes when neural networks are used to impute possible situations that may arise.

Owing to their complex nonlinear nature, neural network-based imputation methods generally demand high-quality and large-scale datasets. In the absence of such conditions, their performance may fall short of that achieved by traditional techniques [24]. However, in real-world industrial settings characterized by intricate operational environments and substantial noise, adversarial examples may unavoidably arise, potentially leading neural networks to generate unreliable or misleading outputs [25]. Fig. 2 is a schematic diagram illustrating the potential problem of unreasonable imputation values when applying a neural network method to actual industrial data. These small amounts of unreasonable imputation results eventually render the neural network impractical unreliable for real-world applications. The blue curve in the figure represents the original values, while the original values in the orange area are artificially masked to impute the missing data. The orange curve represents the imputed results in regions where values are missing. Fig. 2(a) shows the ideal imputation results produced by neural network methods. However, in some cases, as shown in Fig. 2(b), the performance of the neural network results in imputed values that deviate significantly from the actual distribution. This outcome is less reasonable than the results generated by simpler statistical methods, making the neural network-based approach unsuitable for practical applications. In contrast, statistical methods with mathematical constraints can sometimes better avoid such

unreasonable values.

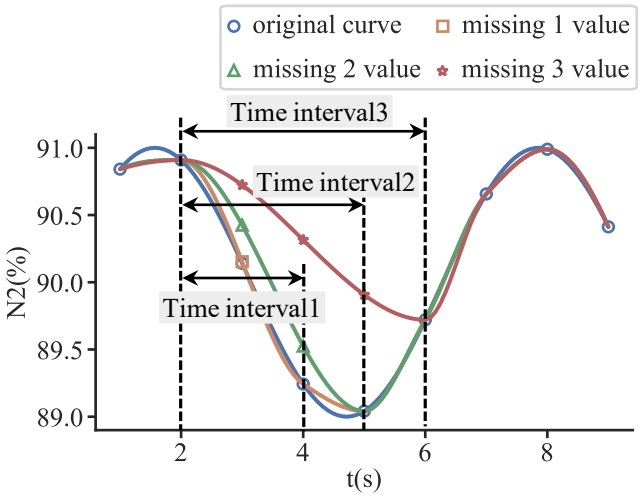

Fig. 3. Schematic interpolation results of the PCHIP method for illustrative cases with 1, 2, or 3 missing values.

Fig. 3 shows the imputation effect of PCHIP under different time intervals. In the figure, the circles represent the original data, while the squares, triangles, and stars represent the imputation results for missing 1, 2, and 3 data points, respectively. As can be seen from the figure, the longer the time interval is, the less accurate the imputation becomes. However, using statistical methods, when data is missing for a longer period, the imputed results tend to revert to the average distribution, reducing the likelihood of imputing values far from the data distribution. To prevent unreasonable results from the neural network, the imputation results of PCHIP can be integrated into the training process to constrain the network's learning and reduce the possibility of generating unreasonable values when the neural network performs imputation. Moreover, unlike laboratory simulations or experiments where ground truth is available, missing data in actual navigation operations cannot be directly validated. Therefore, experimental studies should be conducted under conditions of data incompleteness.

Based on the issues mentioned above, this paper proposes an unsupervised method named as Constrained Unseen Recovery Estimator, which is applied to the incomplete life cycle data of aero-engines to impute missing values. The main features of this method include two aspects. First, considering the issue of imputation being influenced by the value of the aero-engine under different time intervals, the time interval is used as part of the network input. A self-attention mechanism is employed to capture the nonlinear relationship between sampling intervals and performance changes, adjusting the influence of the hidden state of the GRU and the observed value at the current timestamp on the network. Second, considering that statistical imputation methods can constrain the range of imputation results, statistical methods are incorporated during network training to provide an alternative set of imputed values. In this context, “unseen” refers to the missing target values that the model is expected to impute during the data completion process. The deviation between the statistical imputation results and the network imputation results is included in the loss function to reduce the likelihood of unreasonable values produced by the neural network. The main contributions of this paper are as follows:

1. A novel imputation network structure is designed, where the time interval is converted into weights using a self-attention mechanism. This approach controls the influence of the current observation values and historical data on the imputed missing values in the time sequence.
2. In the backward propagation process, the loss function includes a measure of the deviation between the network's imputation results and the results of constraint component. This deviation influences the network parameters during training, helping to reduce the

likelihood of the network generating unreasonable values.

3. Experiments were conducted using Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) data and real civil aero-engine operational data. The results showed that the proposed method outperformed other advanced techniques discussed in this paper.

The remaining sections are structured as follows: Section 2 introduces the proposed method, Section 3 outlines the overall experimental process, followed by a detailed discussion of the experimental setup and results. Finally, Section 5 summarizes the key contributions and conclusions of this paper.

2 Proposed method

This section presents the implementation details of the proposed method, which consists of four parts. First, the network input and overall structure of the proposed method are introduced. Then, Sections 2.2 to 2.4 describe several key components involved in the overall structure, including Interval-aware Temporal Imputation Network (ITIN), constraint component, and a redesigned loss function.

2.1 Overall introduction

2.1.1 Input of the network

The input to the network primarily consists of incomplete data $\boldsymbol{X}$, missing masks $\boldsymbol{M}$, a time interval $\boldsymbol{\Delta}$, and absolute timestamps $\boldsymbol{s}$. The multivariate time series $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ represents incomplete data, where N is the length of the data along the temporal dimension, and D is the feature dimension of the data. The missing masks $\boldsymbol{M}$ marks the locations of missing values in the original data $\boldsymbol{X}$, and has the same shape as $\boldsymbol{X}$. For the data at time step t and dimension d, M_n^d can be expressed as

$$M_n^d = \begin{cases} 1 & \text{if } X_n^d \text{ is observed} \\ 0 & \text{if } X_n^d \text{ is missing} \end{cases} \tag{1}$$

The time interval, denoted as $\boldsymbol{\Delta}$, shares the same shape as $\boldsymbol{X}$. Suppose there is a sample sequence $\boldsymbol{X}$, as shown on the left side of Table 1, where $\boldsymbol{x}_1 \sim \boldsymbol{x}_6$ represent the split vectors of matrix $\boldsymbol{X}$ along the time dimension. The symbol "/" indicates missing data, while the presence of data is indicated by the corresponding value at that position. The absolute times for $\boldsymbol{x}_1 \sim \boldsymbol{x}_6$ are $\boldsymbol{s} = [0,3,7,12,18,20]$, and the corresponding time intervals $\boldsymbol{\Delta}$ are shown on the right side of Table 1. The time interval $\boldsymbol{\Delta}$ is calculated independently for each dimension. The rules for calculating the values at corresponding positions are as follows: Firstly, the time interval for the initial timestamp $\boldsymbol{\delta}_1$ is set to 0. Secondly, for subsequent timestamps, if the sample sequence $\boldsymbol{X}$ contains data in the corresponding dimension at the previous timestamp, the value of $\boldsymbol{\Delta}$ is computed as the difference between the current absolute time and the absolute time of the previous timestamp. Finally, if the data in the corresponding dimension is missing at the previous timestamp, the difference between the current and previous absolute times is calculated, and this difference is further increased by adding the time interval value of the previous timestamp. The computation of the time interval value for the d dimension at time t can be expressed as:

$$\Delta_n^d = \begin{cases} s_n - s_{n-1} + \Delta_{n-1}^d & \text{if } n > 1,\ X_{n-1}^d \text{ is missing} \\ s_n - s_{n-1} & \text{if } n > 1,\ X_{n-1}^d \text{ is observed} \\ 0 & \text{if } n = 1 \end{cases} \tag{2}$$

Among these, s_t represents the absolute time at moment t, while Δ_t^d denotes the time interval of dimension d at moment t.

Table 1 A sample of time intervals constructed based on the positions of missing data and the structure of

absolute time.

Sample timeseries $\boldsymbol{X}$					
x_1	x_2	x_3	x_4	x_5	x_6
47	/	73	/	6	77
96	/	/	/	29	54
84	85	43	/	/	72

Time intervals $\boldsymbol{\Delta}$					
δ_1	δ_2	δ_3	δ_4	δ_5	δ_6
0	3	7	5	11	2
0	3	7	12	18	2
0	3	4	5	11	13

In addition to the derivatives $\boldsymbol{M}$ and $\boldsymbol{\Delta}$ derived from $\boldsymbol{X}$, it is also necessary to mask a part of the observation values in the validation and test sets. This is done to assess the ability of neural networks to impute missing values, and to evaluate the imputation effect of different methods. The data that has been artificially masked is denoted as $\widehat{\boldsymbol{X}}$, and the positions of these data are indicated by indicator masks $\boldsymbol{I}$. The shape of the indicator masks is the same as that of the original data, where observed values are present. However, the positions that are masked and intended for imputation in actual use are represented by 1, while the remaining positions are represented by 0. The indicator mask $\boldsymbol{I}$ at flight n for dimension d is expressed as

$$I_n^d = \begin{cases} 1 & \text{if } \hat{X}_n^d \text{ is artificially masked} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

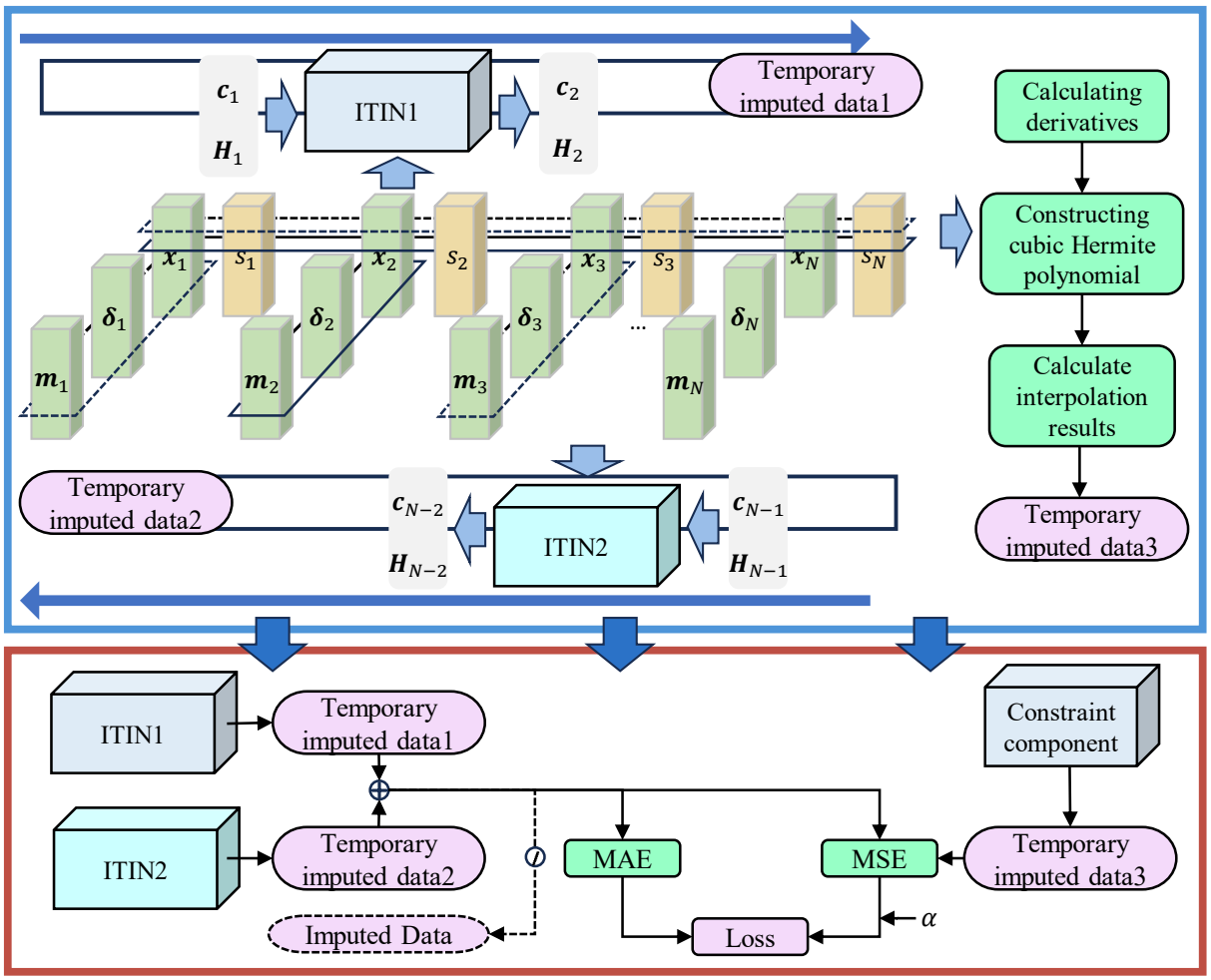

Fig. 4. The execution process of the basic module.

2.1.2 Main components of the method

Fig. 4 shows the main components of the proposed method and the process of utilizing the input data introduced earlier. In the upper part of Fig. 4, the inputs for each module are in the middle and consist of the four types mentioned previously. The two Interval-Aware Temporal

Imputation Networks, positioned above and below, capture patterns from both the forward and reverse directions in the data, each providing a temporary imputation result. On the right is the PCHIP unit, which applies statistical methods to impute the missing data. The light green rectangles in the middle of the upper part of Fig. 4 represent the inputs to the two Interval-Aware Temporal Imputation Networks. These inputs include the original data $\boldsymbol{X}$, the time interval $\boldsymbol{\Delta}$, and missing masks $\boldsymbol{M}$, which have been split by flight into individual components $\boldsymbol{x}_n, \boldsymbol{\delta}_n, \boldsymbol{m}_n$. Interval-Aware Temporal Imputation Network 1 (INIT1) processes the features at each timestamp sequentially, starting from the first timestamp and continuing to the last, representing the process of capturing temporal relationships in the data. The hidden layer outputs from the INIT structure $\boldsymbol{H}_t$ are passed to the next layer, and the outputs from each step, $\boldsymbol{c}_1 \sim \boldsymbol{c}_N$ are concatenated along the time dimension. The temporary imputed data 1 (i.e. $\widetilde{\boldsymbol{X}}_1$ can then be expressed as

$$\boldsymbol{C} = \mathrm{Concat}(\boldsymbol{c}_1, \boldsymbol{c}_2, \dots, \boldsymbol{c}_N) \tag{4}$$

$$\widetilde{\boldsymbol{X}}_1 = \boldsymbol{X} \odot \boldsymbol{M} + \boldsymbol{C} \odot (1 - \boldsymbol{M}) \tag{5}$$

$\boldsymbol{C}$ represent the matrix obtained by concatenating the outputs of each step in the GRU model. The function $\mathrm{Concat}(\cdot)$ denotes the concatenation of vectors into a matrix, and $\odot$ represents element-wise multiplication of matrices with the same size. The Interval-Aware Temporal Imputation Network2 (INIT2) operates in contrast to the first cell, where the time series is learned in reverse along the temporal dimension to capture the influence of future data on the imputation of past values. The imputation result of the Interval-Aware Temporal Imputation Network2 (INIT2) is represented as $\widetilde{\boldsymbol{X}}_2$.

The input to the PCHIP unit processing includes the original data $\boldsymbol{X}$ and the corresponding absolute time vectors $\boldsymbol{s}$ for each timestamp. The PCHIP unit processes each dimension of $\boldsymbol{X}$ separately. When processing the d-th dimension, the derivative is calculated at each point $(s_1, X_1^d), (s_1, X_2^d), \dots, (s_1, X_N^d)$. Then, the coefficients of the cubic functions are computed based on the derivatives and slopes. Finally, the Hermite polynomial, composed of multiple functions, is used to solve for the corresponding value X_n^d at the missing data position s_n. Once all missing values are imputed, the result $\widetilde{\boldsymbol{X}}_3$ is obtained.

The lower half of Fig. 4 illustrates the training and validation process of the module in the upper half. The loss function is constructed from the temporary imputation results and the error measurements of the reconstruction from the upper half, as well as the imputed results and error measurements from the PCHIP. This loss function is used to train the two Interval-Aware Temporal Imputation Networks. Once both networks are trained, the result is obtained by averaging the imputed results from the two networks, as indicated by the dashed line labeled "Imputed Data" in the figure.

2.2 Interval-aware temporal imputation network

The Interval-Aware Temporal Imputation Network is a model based on GRU architecture. A typical feature of GRU is its stepwise processing of each timestamp. Essentially, it assumes that the data at the current timestamp is regressed based on the data from previously processed timestamps. At time t, the original input includes the vectors $\boldsymbol{x}_n, \boldsymbol{\delta}_n, \boldsymbol{m}_n$, which represent the original data, the time interval, and the missing mask at flight n, respectively.

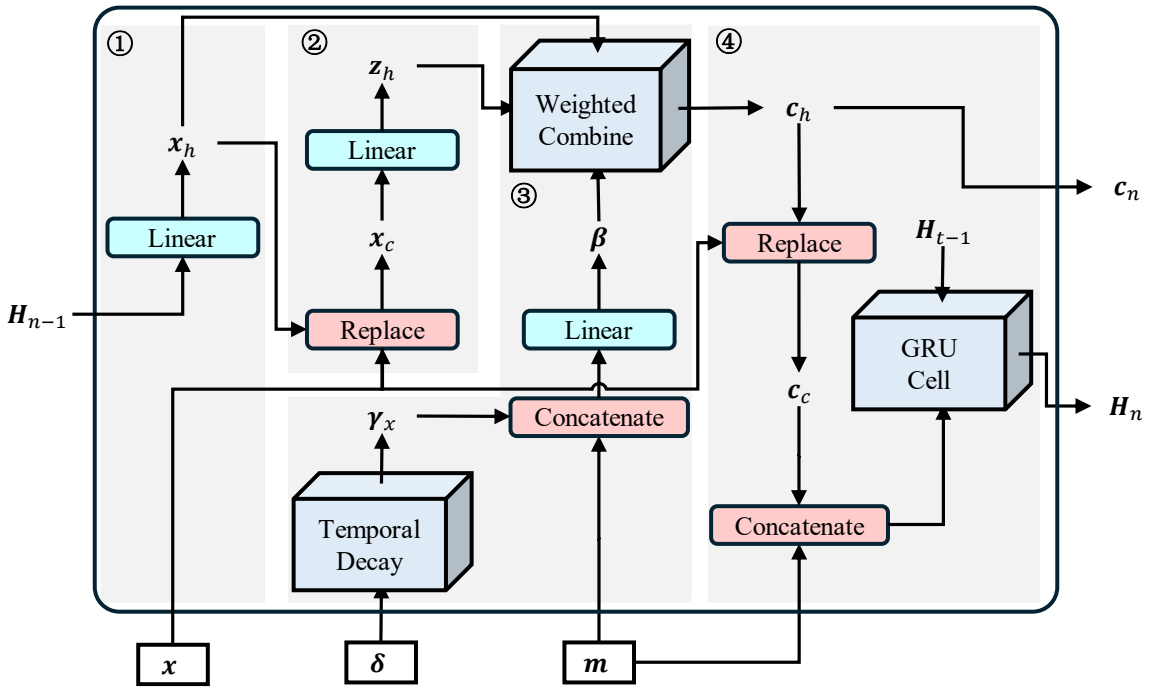

Fig. 5. Internal structure of interval-aware temporal imputation network.

Fig. 5 illustrates the data processing method for all feature dimensions at each flight key data. In addition to the original input at each timestamp, it also incorporates information passed from historical timestamps, where $\boldsymbol{H}_{n-1}$ represents the hidden features passed from the previous timestamp. When processing the first timestamp, the historical timestamp information is initialized as a zero matrix. The following outlines the specific steps of the data processing for each timestamp.

Step ①: The hidden variables from the previous layer are transformed through a linear layer, resulting in $\boldsymbol{x}_h$, which has the same dimensions and size as the input $\boldsymbol{x}$.

Step ②: The signal attenuation of the hidden layer from the previous layer is replaced by the existing values in $\boldsymbol{x}$, reconstructing the data at the corresponding positions of $\boldsymbol{x}_h$ to form the input $\boldsymbol{x}_c$. Then, $\boldsymbol{x}_c$ is passed through a linear layer transformation to obtain the network's output $\boldsymbol{z}_h$. This step can be represented as follows:

$$\boldsymbol{x}_c = \boldsymbol{x} \odot \boldsymbol{m} + \boldsymbol{x}_h \odot (1 - \boldsymbol{m}) \tag{6}$$

$$\boldsymbol{z}_h = \text{Linear}(\boldsymbol{x}_c) \tag{7}$$

Step ③: We generate the weight $\boldsymbol{\gamma}_x$ based on the current time interval $\boldsymbol{\delta}$, which is used to balance the importance of the input $\boldsymbol{z}_h$ and the historical information $\boldsymbol{x}_h$. Then, $\boldsymbol{\gamma}_x$ is concatenated with the missing mask $\boldsymbol{m}$, and this concatenated matrix is transformed through a linear transformation followed by a Sigmoid activation function to match the same size as $\boldsymbol{x}_h$ and $\boldsymbol{z}_h$, represented as $\boldsymbol{\beta}$. $\boldsymbol{\beta}$ is used to merge the features from Step ① and Step ②. The merging formula is as follows:

$$\boldsymbol{c}_h = \boldsymbol{\beta} \cdot \boldsymbol{x}_h + (1 - \boldsymbol{\beta}) \cdot \boldsymbol{z}_h \tag{8}$$

Step ④: The observation values in $\boldsymbol{c}_h$ from Step ③ are replaced, and the result is denoted as $\boldsymbol{c}_c$. Then, $\boldsymbol{c}_c$ is concatenated with the missing mask $\boldsymbol{m}$ along the feature dimension to form the input $\boldsymbol{X}_i$ for the GRU. The previous layer's hidden state $\boldsymbol{H}_{n-1}$ is fed into the GRU to obtain the current timestamp's hidden state $\boldsymbol{H}_n$.

Next, we will introduce time decay and the GRU Cell in detail.

1) **Temporal Decay**

The process of generating time decay factors can be used to calculate the time decay factor $\boldsymbol{\gamma}_{\boldsymbol{x}}$ based on the time interval $\boldsymbol{\delta}$. By incorporating the time decay factor into the update process,

the model adjusts feature weights according to the time interval $\boldsymbol{\delta}$ between the current and previous timestamps $\boldsymbol{x}$, increasing the weight of features closer to the current timestamp and decreasing the weight of more distant features [23]. In this paper, a self-attention mechanism is employed to enhance the model's sensitivity to time interval and improve the balance between hidden states and input weights in GRU. The structure is shown in Fig. 6.

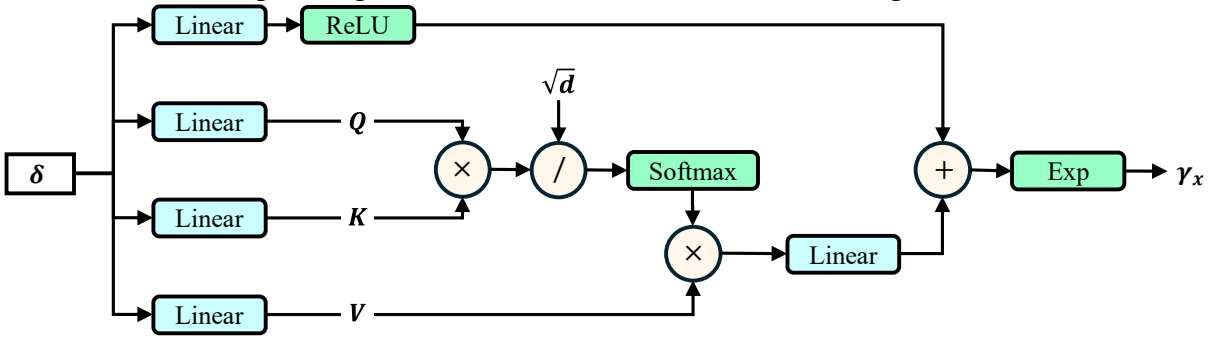

Fig. 6 Self-attention mechanism is employed to compute $\boldsymbol{\gamma_x}$.

The overall input of Fig. 6 is $\boldsymbol{\delta}$, the time interval for the multi-sensor data at each timestamp, and the output is the time decay factor $\boldsymbol{\gamma_x}$. Fig. 6 consists of two parts. The left side represents a linear layer used to generate a temporary $\boldsymbol{\gamma}$. The calculation method is as follows:

$$\boldsymbol{\gamma} = \mathrm{ReLU}(\boldsymbol{\delta W}_d \odot \boldsymbol{I} + \boldsymbol{b}_d) \tag{9}$$

where $\boldsymbol{W}_d$ and $\boldsymbol{b}_d$ are the weights and biases of the network that provide the decay factor, $\boldsymbol{I}$ is the identity mask used to retain the information of the time intervals, and $\mathrm{ReLU}(\cdot)$ is the activation function. The right side of Fig. 6 implements a self-attention mechanism. First, the time interval $\boldsymbol{\delta}$ is passed through three consecutive linear layers, whose outputs serve as Query, Key, and Value, respectively. The dot product between Q and K is then calculated and divided by a scaling factor $\sqrt{d}$ to prevent the results from being overly large. After normalization through $\mathrm{Softmax}(\cdot)$, the resulting values are used as the weight parameters of the self-attention mechanism. These weights are then used to compute the dot product with V, followed by a linear transformation to match the dimensionality of the output from the linear layer. Finally, the results from both sides are combined, and the output is adjusted using an exponential function to regulate the distribution range, forming the final time decay factor $\boldsymbol{\gamma_x}$.

2) **GRU Cell**

In the gated recurrent unit, the hidden state $\boldsymbol{H}_{n-1}$ and input $\boldsymbol{X}_i$ undergo further processing. The GRU contains two gates: the reset gate $\boldsymbol{R}_n$ and the update gate $\boldsymbol{Z}_n$. The role of the reset gate is to compute the candidate hidden state $\widetilde{\boldsymbol{H}}_n$, while the update gate adjusts the weights between the candidate hidden state $\widetilde{\boldsymbol{H}}_n$ and the previous hidden state $\boldsymbol{H}_{n-1}$. Both gates are structured as fully connected layers, each with a Sigmoid activation function, expressed as:

$$\boldsymbol{R}_n = \mathrm{Sigmoid}(\boldsymbol{X}_i\boldsymbol{W}_{xr} + \boldsymbol{H}_{n-1}\boldsymbol{W}_{hr} + \boldsymbol{b}_r) \tag{10}$$

$$\boldsymbol{Z}_n = \mathrm{Sigmoid}(\boldsymbol{X}_i\boldsymbol{W}_{xz} + \boldsymbol{H}_{n-1}\boldsymbol{W}_{hz} + \boldsymbol{b}_z) \tag{11}$$

where $\boldsymbol{W}_{xr}, \boldsymbol{W}_{hr}, \boldsymbol{b}_r$ are the network parameters of the reset gate, and $\boldsymbol{W}_{xz}, \boldsymbol{W}_{hz}, \boldsymbol{b}_z$ are the parameters of the update gate. The reset gate is then combined with the hidden state to compute the candidate hidden state $\widetilde{\boldsymbol{H}}_n$, as follows:

$$\widetilde{\boldsymbol{H}}_n = \tanh(\boldsymbol{X}_i\boldsymbol{W}_{xh} + (\boldsymbol{R}_n \odot \boldsymbol{H}_{n-1})\boldsymbol{W}_{hh} + \boldsymbol{b}_h) \tag{12}$$

where $\boldsymbol{W}_{xh}, \boldsymbol{W}_{hh}, \boldsymbol{b}_h$ are the network parameters used to compute the candidate hidden state, and $\odot$ denotes the element-wise multiplication operator. Finally, the output hidden state of the GRU is computed as:

$$\boldsymbol{H}_n = \boldsymbol{Z}_n \odot \boldsymbol{H}_{n-1} + (1 - \boldsymbol{Z}_n) \odot \widetilde{\boldsymbol{H}}_n \tag{13}$$

2.3 Constraint component

The constraint component refers to the application of an interpolation method as a supplement to the imputation neural network, designed to constrain the range of generated results during the imputation process. In this paper, the PCHIP is employed as a case study for the experiments. PCHIP is used for smooth interpolation of data. The implementation of this algorithm involves constructing a cubic polynomial in the interval between every two observation points and ensuring that these polynomials share the same function value and derivative at the observation points. The algorithm consists of four steps: First, split the original input data into incomplete sequences; second, calculate the derivative at each existing data point to ensure consistency of derivatives on both sides of the data points; third, construct cubic Hermite polynomials in each interval, ensuring that each value corresponds to the same function value and derivative at each position; finally, use the cubic polynomials to compute the interpolated results at any given position.

1) Constructing Input Data

Using the original data and time interval data $\boldsymbol{X}, \boldsymbol{s}$ provided above as input, compute the corresponding PCHIP interpolation results for each dimension to obtain a new set of complete values. For the data of the d-th dimension, a set of incomplete data $\boldsymbol{X}^d$ is first given, which is then combined sequentially with the real time intervals. This results in the PCHIP input as $(s_1, X_1^d), (s_2, X_2^d), \ldots, (s_N, X_N^d)$.

2) Calculating Derivatives

The derivatives of $(\boldsymbol{s}_1, \boldsymbol{X}_1^d), (\boldsymbol{s}_2, \boldsymbol{X}_2^d), \ldots, (\boldsymbol{s}_N, \boldsymbol{X}_N^d)$ are computed. While the numerical derivatives can be directly calculated using $d = \Delta y / \Delta x$, this method cannot ensure smoothness and monotonicity in the results when applying the PCHIP interpolation method. Therefore, the following procedure is employed in PCHIP to determine the derivative for any interval k (i.e., between k and $k+1$).

First, the slope m_k is calculated. Let $h_k = \boldsymbol{s}_{k+1} - \boldsymbol{s}_k$, then the slope at s_k is given by $m_k = (\boldsymbol{X}_{k+1}^d - \boldsymbol{X}_k^d)/h_k$. Next, d_k is computed. For $k \neq 1$ and $k \neq N$, if $m_k = 0$, $m_{k-1} = 0$, or if $\mathrm{SGN}(m_k) \neq \mathrm{SGN}(m_{k-1})$, then d_k is set to 0; otherwise, d_k is calculated by weighted harmonic averaging of m_k and m_{k-1} based on h_k and h_{k-1} as follows:

$$w_1 = 2h_k + h_{k-1} \tag{14}$$

$$w_2 = h_k + 2h_{k-1} \tag{15}$$

$$1/d_k = 1/(w_1 + w_2) * (w_1/m_k + w_2/m_{k-1}) \tag{16}$$

$\mathrm{SGN}(\cdot)$ denotes the sign function, which returns values $-1, 0, 1$.

When $k = 1$ or $k = N$, the boundary derivative is determined using a one-sided three-point derivative estimate. At $k = 1$, d_k is computed using $h_k, h_{k+1}, m_k, m_{k+1}$. First, a preliminary estimate of d_k is obtained:

$$d_k = \mathrm{G}(h_k, h_{k+1}, m_k, m_{k+1}) = \big((2 \times h_k + h_{k+1}) \times m_k\big)/(h_k + h_{k+1}) \tag{17}$$

To ensure that the derivative does not violate the shape or monotonicity of the interpolation function, the derivative may need to be adjusted under certain conditions. First, it is verified whether $\mathrm{SGN}(d_k) = \mathrm{SGN}(m_k)$. If not, d_k is set to 0; otherwise, a further check is performed to determine if $\mathrm{SGN}(m_k) \neq \mathrm{SGN}(m_{k+1})$ and $|d_k| > 3 \times |m_k|$. If these conditions hold, d_k is set to $3 \times m_k$. When $k = T$, the derivative is calculated as $d_k = G(h_{k-1}, h_{k-2}, m_{k-1}, m_{k-2})$.

This process yields the derivative at each point, and the sequence of derivatives is formed based on the timestamps of the observations, represented by $\boldsymbol{d}$.

3) Constructing cubic Hermite polynomial

First, the interpolation parameters are calculated, considering the relationship between the derivative and the slope.

$$\mathrm{t_k} = (d_k + d_{k+1} - 2 \times m_k)/h_k \tag{18}$$

The Hermite interpolation method also uses cubic polynomials for interpolation. For any interval $[x_i, x_{i+1}]$, the polynomial has the form:

$$F_k(x) = p_k(x - x_i)^3 + q_k(x - x_i)^2 + r_k(x - x_i) + s_k \tag{19}$$

The calculation method is $p_k = t_k/h_k,\ q_k = (m_k - d_k)/h_k - \mathrm{t},\ r_k = d_k,\ s_k = y_k$.

4) Calculating interpolation results

Finally, based on the provided location $x_p \in [x_k, x_{k+1}]$, select the corresponding polynomial $F_k(x)$, and use the provided x_p to calculate the interpolation result.

2.4 Design of loss function

In the traditional RNN-based imputation method training process, the reconstruction results and observed values are typically used to calculate the loss function. The loss function for the training process of the proposed model in this article is composed of four main components: the reconstruction errors of the two Interval-Aware Temporal Imputation Networks, the imputation consistency error between these two networks, and the error between the average imputed results of the two networks and the imputed result obtained using the PCHIP method. After the parameters of both models have been trained, the test data is input into the models to obtain two sets of reconstruction results from both forward and backward passes. These reconstruction results are averaged to produce the final imputed result.

The first part is the reconstruction errors of the two Interval-Aware Temporal Imputation Networks. For each model, the reconstruction error consists of three terms, which are the MAE between $\boldsymbol{x}_h, \boldsymbol{z}_h, \boldsymbol{c}_h$, and $\boldsymbol{x}$. The loss needs to be calculated at each timestamp, and after all timestamps have been computed, the reconstruction loss can be accumulated. The loss at each timestamp is expressed as:

$$\mathcal{L}_r' = [(\boldsymbol{x}_h - \boldsymbol{x}) + (\boldsymbol{z}_h - \boldsymbol{x}) + (\boldsymbol{c}_h - \boldsymbol{x})] \cdot \boldsymbol{m}/\mathrm{SUM}(\boldsymbol{m}) \tag{20}$$

$\mathrm{SUM}(\cdot)$ denotes the summation of all elements in the matrix. The sum of reconstruction errors across all timestamps serves as the final loss function for the unilateral RNN network, denoted as $\mathcal{L}_r$, with $\mathcal{L}_{rf}$ representing the forward imputation network and $\mathcal{L}_{rb}$ the backward imputation network.

The second part is the imputation consistency error between the two Interval-Aware Temporal Imputation Networks. Since this error tends to be large, it needs to be scaled down to 10% of its original value. The calculation can be expressed as:

$$\mathcal{L}_c = \left|\tilde{X}_1 - \tilde{X}_2\right|/(\mathrm{T} \times \mathrm{D}) \times 0.1 \tag{21}$$

The final part is the error between the imputed results and the PCHIP imputed results, which is represented as:

$$\mathcal{L}_p = \left(\widetilde{\boldsymbol{X}}_1 - \widetilde{\boldsymbol{X}}_3\right)^2/\mathrm{SUM}(1 - \boldsymbol{M}) \tag{22}$$

Thus, the total loss function for training the network can be expressed as:

$$\mathcal{L} = \mathcal{L}_c + \mathcal{L}_{rf} + \mathcal{L}_{rb} + \mathcal{L}_p \tag{23}$$

3 Overall workflow of the method

Fig. 7 illustrates the complete experimental process, which is mainly divided into four parts: the acquisition of the original data, data pre-processing, network training, and application. In the figure, the overall process is evaluated using a real aero-engine dataset to introduce the method.

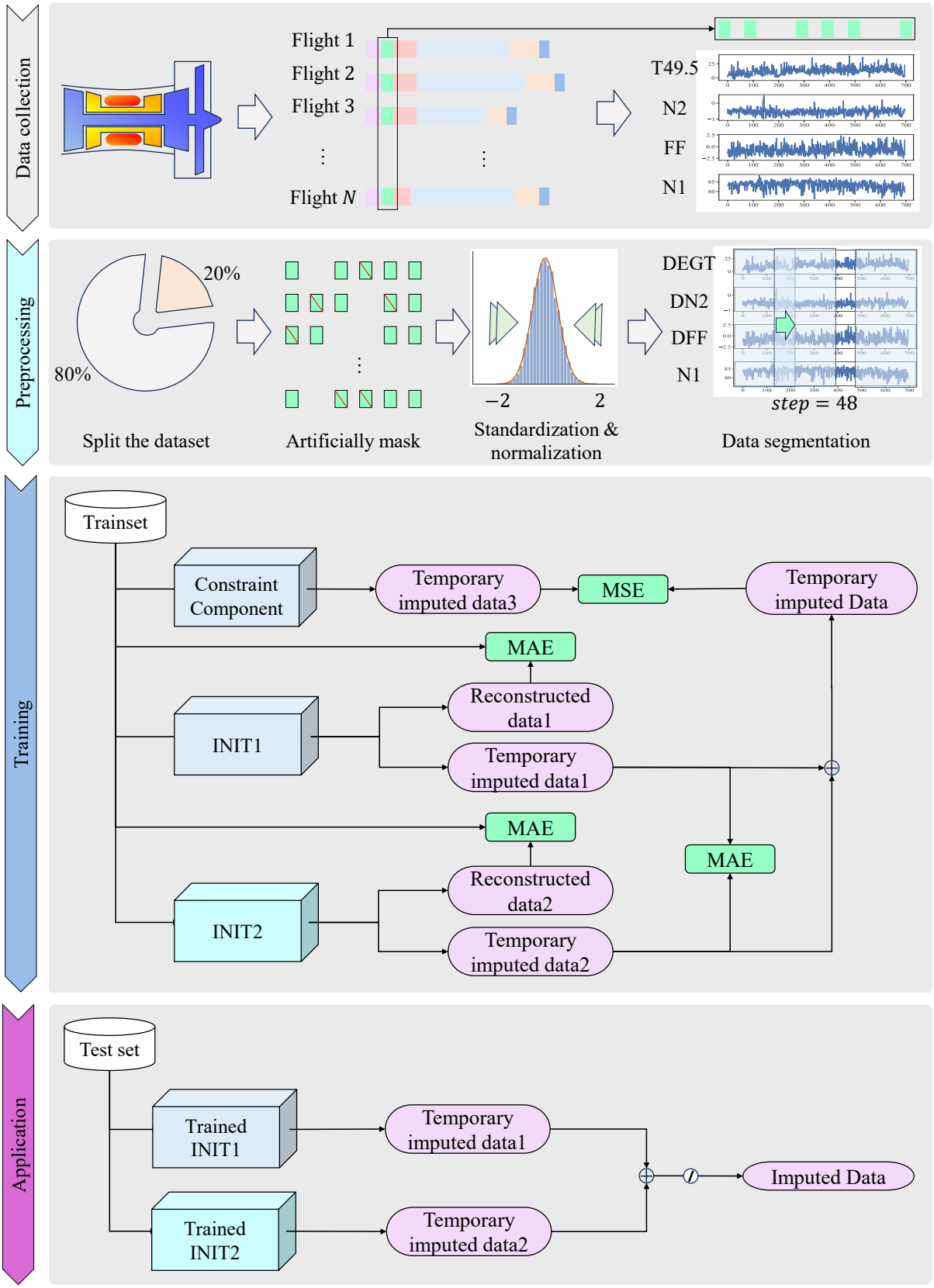

Fig. 7 Implementation process of the method.

The first step is data acquisition, which involves extracting relevant data from engine operation records to form the dataset. Two datasets are used in this experiment: the C-MAPSS dataset and real civil aero-engine data. Both datasets share the same format, consisting of

multiple sets of unequal-length multivariate time series data with missing values. The data is initially collected by sensors installed on the aero-engine and recorded by the aircraft's Quick Access Recorder, which captures the complete data from all stages. Critical data from key stages is transmitted in real-time to the ground via the ACARS communication system. Since the data for each flight forms a lengthy record covering multiple stages, a key operational state that reflects engine performance is first selected. Features are then extracted from this state to generate timestamped data. Subsequently, all records from the same engine are chronologically stitched together, resulting in a comprehensive dataset representing each engine's operational status.

The next phase is data preprocessing. The first step is to split the dataset by engine number, separating different engines into distinct subsets. 20% of the entire dataset is used as the test set, and within the training set, 20% is set aside as the validation set. In the second step, 10% of the data is artificially masked to assess the performance of the model and test the imputation effects of different methods. The third step involves concatenating the original data sequences according to the training, validation, and test sets. Data normalization is applied based on the training set to minimize discrepancies across feature dimensions, thereby improving the neural network's imputation performance. The fourth step is to apply a sliding window to segment the unequal-length sequences into equal-length segments, which facilitates their input into the neural network.

The third step involves training the imputation network. During the training process, the incomplete training set is fed into two Interval-Aware Temporal Imputation Networks. These networks generate reconstructed data and temporary imputed data. The MAE between the reconstructed data and the original data is used as part of the loss function. The consistency of the imputed results from both networks is also included in the loss function as an additional MAE term. After averaging the imputed results from both networks, this average serves as the temporary imputed result during training. The incomplete data is then input into a PCHIP component, generating further temporary imputed data. The MSE between the temporary imputed data and output of the PCHIP component is also integrated into the loss function, which optimizes the parameters of both ITIN.

The final step is applying the trained imputation networks. The test data is input into the two trained Interval-aware Temporal Imputation Networks to obtain both reconstructed data and temporary imputed data. The average of the two networks' temporary imputation results is taken as the final imputed result for the test set. Lastly, the difference between the imputed test set results and the original data is measured, using metrics such as MSE and MAE to evaluate the performance. The calculation methods are

$$\mathrm{MSE}\left(\widetilde{\boldsymbol{X}}, \boldsymbol{X}, \boldsymbol{I}\right) = \frac{\sum_{d=1}^{D}\sum_{n=1}^{N}\left(\tilde{X}_n^d - X_n^d\right)^2 \cdot I_n^d}{\sum_{d=1}^{D}\sum_{n=1}^{N} I_n^d} \tag{24}$$

$$\mathrm{MAE}\left(\widetilde{\boldsymbol{X}}, \boldsymbol{X}, \boldsymbol{I}\right) = \frac{\sum_{d=1}^{D}\sum_{n=1}^{N}\left|\tilde{X}_n^d - X_n^d\right| \cdot I_n^d}{\sum_{d=1}^{D}\sum_{n=1}^{N} I_n^d} \tag{25}$$

$\widetilde{\boldsymbol{X}}$ represents the output of the imputation network, $\boldsymbol{X}$ represents the original incomplete data, and $\boldsymbol{I}$ represents the indicating mask matrix.

4 Experimental results

The following section provides the experimental design and analysis of the results. It consists of three main parts: the first part introduces the datasets and experimental platforms; the second part presents the imputation results and analysis of the C-MAPSS dataset, while the third part focuses on the imputation results and analysis of the real aero-engine dataset.

4.1 Experimental setup

4.1.1 Dataset introduction

1) C-MAPSS Dataset

The C-MAPSS dataset is a publicly available dataset generated by the C-MAPSS simulator to simulate the degradation trends of turbofan engines. In this paper, we mainly use the results of simulated data to carry out experiments. The details of dataset preparation can be found in [26]. This dataset consists of four subsets, namely FD001 through FD004. The dataset details are provided in Table 2. Each dataset contains engine numbers, cycles, three operating conditions, and data from 21 sensors.

Table 2 C-MAPSS Dataset Details.

Subset	Operation conditions	Fault modes	Engines in training set	Engines in test set
FD001	1	1	100	100
FD002	6	1	260	259
FD003	1	2	100	100
FD004	6	2	249	248

To avoid the impact of different operating conditions, FD001 and FD003 were selected for the experiments. Additionally, to mitigate the influence of data less correlated with changes in engine status on the experimental results, Pearson correlation coefficients were used to eliminate sensor data with low correlation to the experimental outcomes. The remaining data is presented in Table 3. The original missing rate of the experimental data was set at 10%. On this basis, an additional 10% missing data was introduced for training and validation of the network's performance in handling missing values.

Table 3 Details of Sensor Data Used in the Experiment.

Field name	Description	Unit
T24	Total temperature at LPC outlet	°R
T30	Total temperature at HPC outlet	°R
T50	Total temperature at LPT outlet	°R
P2	Total pressure at HPC outlet	psia
Nf	Physical fan speed	rpm
Nc	Physical core speed	rpm
Ps30	Static pressure at HPC outlet	psia
phi	Ratio of fuel flow to Ps30	pps/psia
NRf	Corrected fan speed	rpm
NRc	Corrected core speed	rpm
BPR	Bypass Ratio	-
htBleed	Bleed Enthalpy	-
W31	HPT coolant bleed	lbm/s
W32	LPT coolant bleed	lbm/s

2) Real Civil Aero-Engine Dataset

The real aero-engine dataset originates from the operational data of a turbofan engine from an Asian airline. The basic structure diagram of the turbofan engine is shown in Fig. 8, which mainly includes the high-pressure turbine (HPT), low-pressure turbine (LPT), combustion chamber (CC), high-pressure compressor (HPC), low-pressure compressor (LPC), and fans. The data used in this paper contains four dimensions: N_1 represents the low-pressure rotor speed, N_2 represents the high-pressure rotor speed, FF represents fuel flow, and the thermocouple on the T49.5-line harness component can be used to collect the engine exhaust gas temperature (EGT).

The actual data contains faulty data that may affect the patterns in the dataset. Since this paper does not involve fault diagnosis, to avoid the impact of faulty data, any samples within

10 instances related to the failure are excluded from the dataset. The default missing rate and the percentage of artificially masked data in this dataset are consistent with the C-MAPSS dataset, both of which are 10%.

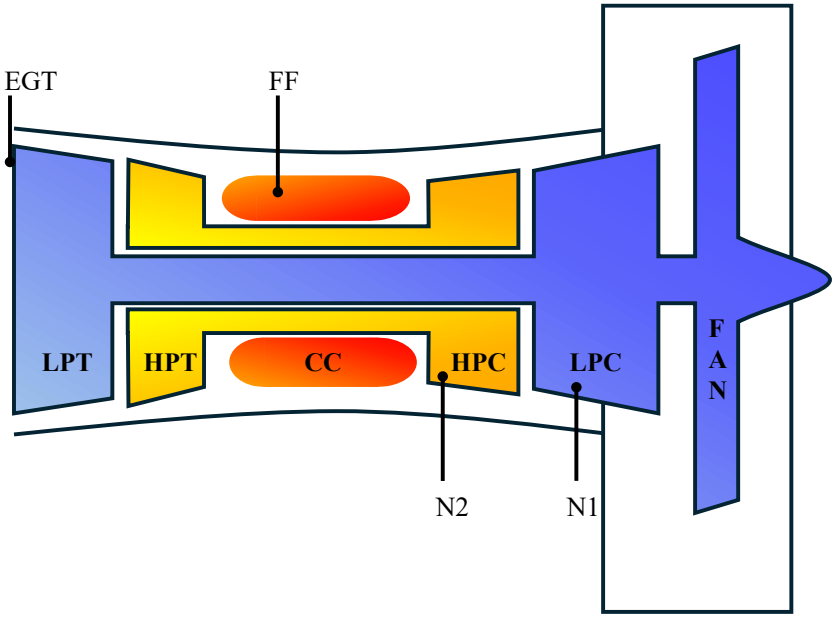

Fig. 8. Basic structure of turbofan engine.

4.1.2 Experimental environment and setup

The experiment was conducted using Python version 3.10 and PyTorch version 2.1.1. The machine's CPU is an Intel Core i7-9700K, and it has 16 GB of DDR4 memory. In this experiment, the step size for both datasets is set to 48, with the C-MAPSS dataset having 17 features and the real civil aero-engine dataset having 4 features. The GRU's hidden layer size is set to 128, and the mini-batch size is 32. The training process lasts for up to 30 epochs, but early stopping is applied if the loss does not decrease for 3 consecutive epochs. The Adam optimizer is used with an initial learning rate of 0.001.

4.2 Analysis of experimental results on the C-MAPSS dataset

4.2.1 Selection of effective weight coefficients

Since the value of the proposed method's parameter α is yet to be determined, it is necessary to choose an optimal α before conducting comparative experiments. A grid search method is employed, where α is explored within the range $[0,6)$ with a step size of 0.6. For each α, ten experiments are conducted, and the average and median values are recorded. The goal of the proposed method is to improve reliability in real industrial scenarios while achieving better imputation results. Therefore, when selecting hyperparameters, this paper places greater emphasis on the evaluation results of MSE.

Table 4 and Table 5 present the results of the grid search method, where ten experiments were conducted for each of the aforementioned α values. The tables report the MSE and MAE results, along with their corresponding average and median values. From Table 4, it can be observed that the average and median MSE values for $\alpha = 0$ across the ten experiments are 0.0730 and 0.0738, respectively. For all $\alpha > 0$, the experimental errors are lower than these two values, indicating that when the PCHIP unit is active, the experiments yield better results. The optimal average and median MSE values were achieved when $\alpha = 2.4$. Similarly, Table 5 shows the MAE results, where the smallest average and median values, 0.1949 and 0.1951

respectively, were also achieved at $\alpha = 2.4$. Based on the results from both tables, $\alpha = 2.4$ was selected as the hyperparameter for subsequent experiments.

Table 4 Relationship between α and MSE on the C-MAPSS dataset.

Test	The value of α									
	0	0.6	1.2	1.8	2.4	3	3.6	4.2	4.8	5.4
Trial 1	0.0675	0.0683	0.0653	0.0657	0.0662	0.0640	0.0642	0.0645	0.0648	0.0651
Trial 2	0.0743	0.0748	0.0742	0.0744	0.0728	0.0717	0.0719	0.0721	0.0724	0.0728
Trial 3	0.0739	0.0746	0.0734	0.0742	0.0723	0.0716	0.0718	0.0720	0.0721	0.0722
Trial 4	0.0726	0.0727	0.0754	0.0740	0.0708	0.0733	0.0737	0.0741	0.0745	0.0749
Trial 5	0.0745	0.0746	0.0737	0.0721	0.0716	0.0735	0.0738	0.0741	0.0745	0.0726
Trial 6	0.0714	0.0718	0.0716	0.0724	0.0711	0.0716	0.0719	0.0721	0.0722	0.0744
Trial 7	0.0733	0.0745	0.0728	0.0754	0.0720	0.0722	0.0725	0.0727	0.0730	0.0732
Trial 8	0.0737	0.0742	0.0720	0.0760	0.0746	0.0747	0.0748	0.0748	0.0748	0.0749
Trial 9	0.0748	0.0751	0.0729	0.0734	0.0730	0.0731	0.0733	0.0736	0.0739	0.0741
Trial 10	0.0739	0.0745	0.0725	0.0729	0.0726	0.0728	0.0730	0.0733	0.0737	0.0739
Average	0.0730	0.0735	0.0724	0.0731	**0.0717**	0.0719	0.0721	0.0723	0.0726	0.0728
Median	0.0738	0.0745	0.0729	0.0737	**0.0722**	0.0725	0.0728	0.0730	0.0734	0.0736

Table 5 Relationship between α and MAE on the C-MAPSS dataset.

Test	The value of α									
	0	0.6	1.2	1.8	2.4	3	3.6	4.2	4.8	5.4
Trial 1	0.1904	0.1929	0.1868	0.1879	0.1890	0.1860	0.1861	0.1867	0.1873	0.1877
Trial 2	0.1970	0.1985	0.1968	0.1970	0.1965	0.1941	0.1946	0.1952	0.1960	0.1969
Trial 3	0.1996	0.2022	0.1978	0.1998	0.1953	0.1940	0.1943	0.1946	0.1950	0.1953
Trial 4	0.1939	0.1952	0.2005	0.1986	0.1933	0.1974	0.1983	0.1995	0.2004	0.2012
Trial 5	0.1957	0.1965	0.1977	0.1949	0.1942	0.1974	0.1981	0.1988	0.1995	0.1957
Trial 6	0.1924	0.1936	0.1928	0.1962	0.1935	0.1947	0.1955	0.1958	0.1960	0.1982
Trial 7	0.1962	0.1998	0.1957	0.2018	0.1949	0.1950	0.1954	0.1959	0.1964	0.1967
Trial 8	0.1966	0.1980	0.1942	0.2037	0.2004	0.2007	0.2004	0.2002	0.2002	0.2004
Trial 9	0.1986	0.2011	0.1955	0.1964	0.1958	0.1961	0.1964	0.1970	0.1977	0.1983
Trial 10	0.1978	0.1996	0.1957	0.1965	0.1962	0.1963	0.1967	0.1976	0.1985	0.1993
Average	0.1958	0.1977	0.1954	0.1973	**0.1949**	0.1952	0.1956	0.1961	0.1967	0.1970
Median	0.1964	0.1983	0.1957	0.1968	**0.1951**	0.1956	0.1960	0.1965	0.1971	0.1976

4.2.2 Comparison of different methods

To evaluate the effectiveness of the proposed method CUR-Estimator, comparative experiments were conducted against recent methods such as CSDI, GP-VAE, and BRITS. The results from ten repeated experiments are shown in Fig. 9. As illustrated in Fig. 9 (a), CSDI exhibits a relatively high overall MSE value. Similarly, Fig. 9 (b) shows that GP-VAE has a larger value under the MAE evaluation metric. Across the 10 experiments, CUR-Estimator outperforms the other three methods in terms of both MSE and MAE.

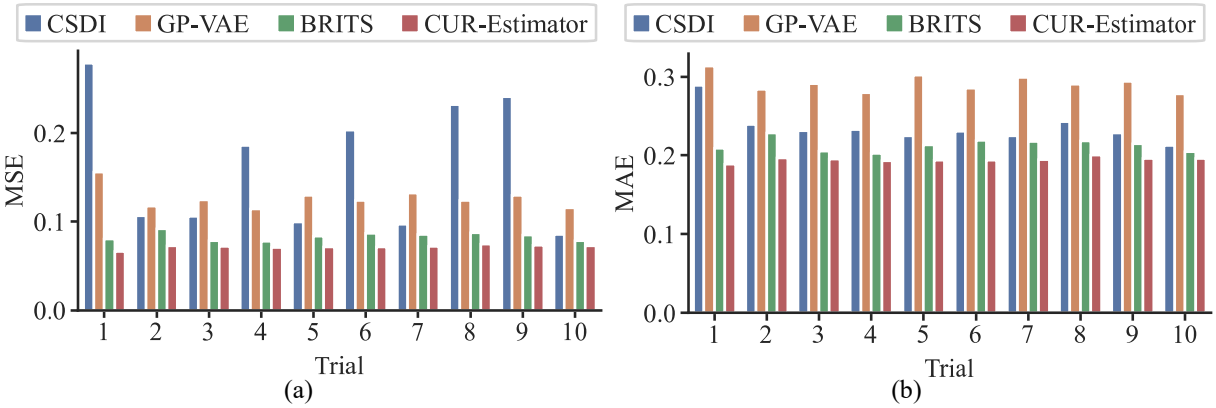

Fig. 9 Results of ten experiments on the C-MAPSS dataset using different methods, (a)MSE, (b)MAE.

Table 6 and Table 7 show the average, standard deviation, and median of MSE and MAE from ten experiments. From Table 6, it can be observed that the proposed method yields the smallest average MSE and MAE values, which are 0.0717 and 0.1949, respectively. Additionally, the standard deviations of MSE and MAE from ten repeated experiments are also small, at 0.0022 and 0.0029, respectively. This indicates that CUR-Estimator performs better in overall imputation, with more stable results across multiple experiments. From Table 7, we can see that CUR-Estimator achieves MSE and MAE values of 0.0722 and 0.1951, respectively, which are better than several other methods in this paper. This suggests that CUR-Estimator performs more consistently in most cases.

Table 6 Mean results on the C-MAPSS dataset.

Average±STD	Method			
	CSDI	GP-VAE	BRITS	CUR-Estimator
MSE	0.1635 ± 0.0687	0.1266 ± 0.0114	0.0800 ± 0.0048	**0.0717 ± 0.0022**
MAE	0.2357 ± 0.0195	0.2917 ± 0.0104	0.2088 ± 0.0090	**0.1949 ± 0.0029**

Table 7 Median results on the C-MAPSS dataset.

Median	Method			
	CSDI	GP-VAE	BRITS	CUR-Estimator
MSE	0.1462	0.1241	0.0841	**0.0722**
MAE	0.2310	0.2908	0.2141	**0.1951**

4.2.3 Hypothesis validity experiment

The principle of CUR-Estimator is to minimize the likelihood of generating unreasonable values by reducing the deviation between the imputed values and the actual data distribution. To verify this, this paper compares the deviation between the imputed and actual values with and without statistical constraints. For the sake of comparison, BRITS, which performs better among the baseline methods, is selected as the control method. Table 8 shows the results of ten experiments, including the average values of the differences between the imputed and actual values, as well as the deviation ranges for both BRITS and the proposed method. The left side of the table displays the left and right boundaries for BRITS, while the right side shows the boundaries for CUR-Estimator. As seen in Table 8, the average left boundary of the difference between the imputed and actual values over ten experiments for BRITS is -1.4613, while the right boundary is 1.9997, yielding an average range of 3.4610. In comparison, CUR-Estimator has left and right boundaries of -1.2999 and 1.8036, respectively, with a range of

3.1035. Therefore, CUR-Estimator produces smaller absolute boundary values than BRITS. The experimental results illustrate that the deviation range between the imputed and actual values for CUR-Estimator is smaller.

Table 8 Boundaries of the deviation for BRITS and CUR-Estimator on the C-MAPSS dataset.

Test	BRITS			CUR-Estimator		
	Left edge	Right edge	Range	Left edge	Right edge	Range
Trial 1	-1.5047	2.1095	3.6142	-1.1449	1.7560	2.9009
Trial 2	-1.5137	2.0003	3.5140	-1.3318	1.7829	3.1147
Trial 3	-1.3639	1.9172	3.2811	-1.2485	1.8373	3.0858
Trial 4	-1.5285	1.9499	3.4784	-1.3273	1.8043	3.1316
Trial 5	-1.4190	2.0620	3.4810	-1.3899	1.8419	3.2318
Trial 6	-1.4870	2.0158	3.5028	-1.2931	1.8028	3.0959
Trial 7	-1.3814	2.0377	3.4191	-1.2886	1.8266	3.1152
Trial 8	-1.5142	2.0504	3.5646	-1.3507	1.8064	3.1571
Trial 9	-1.4672	1.9147	3.3819	-1.3490	1.7912	3.1402
Trial 10	-1.4331	1.9396	3.3727	-1.2752	1.7867	3.0619
Average	-1.4613	1.9997	3.4610	**-1.2999**	**1.8036**	**3.1035**

4.2.4 Experimental results under different SNRs

Fig. 10 compares the experimental results of CUR-Estimator and BRITS under different signal-to-noise ratios (SNRs). Fig. 10(a)shows the comparison of MSE at various SNRs. It can be observed from the figure that when the SNR is 10dB, the MSE of both methods exceeds 0.1, whereas at other SNRs, the MSE remains below 0.1. However, across various SNRs, CUR-Estimator consistently achieves a lower MSE than BRITS, and the standard deviation across ten experiments is smaller for CUR-Estimator. The MAE comparison shown in Fig. 10(b) indicates that CUR-Estimator also achieves lower errors than BRITS across multiple SNRs, and the standard deviation of CUR-Estimator under different noise conditions is significantly smaller than that of BRITS. These results show that CUR-Estimator performs better and is more stable across different levels of noise.

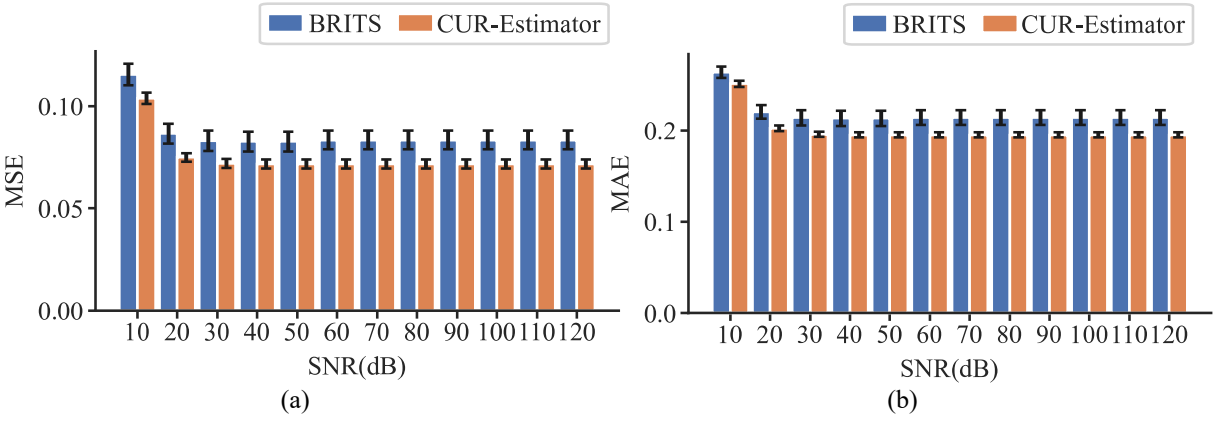

Fig. 10. Bar charts and standard deviations of MSE and MAE under different SNRs.

4.3 Analysis of experimental results on real civil aero-engine dataset

4.3.1 Selection of effective weight coefficients

Considering that different datasets should use different hyperparameters, during the process of conducting experiments based on real aero-engine data, the step size was set to 0.6 under the condition $\alpha \in [0,6)$. For each α, ten experiments were conducted, and both the average and the median values were recorded. The MSE and MAE results of the experiments

are shown in Table 9 and Table 10. As seen in Table 9, the smallest average MSE of 0.2590 was achieved at $\alpha = 0.6$, while the smallest median MSE of 0.2456 was achieved at $\alpha = 1.2$. Similarly, as shown in Table 10, the smallest average MAE of 0.3951 was achieved at $\alpha = 0.6$, and the smallest median MAE of 0.3857 was achieved at $\alpha = 1.2$. Since this paper focuses on the rationality of the imputed results, it prioritizes minimizing overall losses. Therefore, $\alpha = 0.6$ is selected as the optimal hyperparameter for the real aero-engine dataset.

Table 9 Relationship between α and MSE on the civil aero-engine dataset.

Test	The value of α									
	0	0.6	1.2	1.8	2.4	3	3.6	4.2	4.8	5.4
Trial 1	0.4331	0.3647	0.3311	0.3020	0.4620	0.4576	0.4559	0.4555	0.4559	0.4569
Trial 2	0.2838	0.2440	0.2103	0.2707	0.3012	0.3813	0.3725	0.3661	0.3613	0.3577
Trial 3	0.2717	0.2760	0.2271	0.3624	0.2084	0.3382	0.3353	0.3330	0.3313	0.3299
Trial 4	0.2232	0.2549	0.2289	0.3641	0.3610	0.3613	0.3569	0.3540	0.3519	0.3504
Trial 5	0.1687	0.3330	0.2482	0.3728	0.3383	0.3508	0.3486	0.3470	0.3457	0.3447
Trial 6	0.2310	0.3000	0.3400	0.2833	0.1981	0.3448	0.3417	0.3395	0.3380	0.3368
Trial 7	0.2592	0.2680	0.2383	0.3453	0.2885	0.2728	0.2631	0.2562	0.2515	0.2484
Trial 8	0.2894	0.2089	0.4103	0.3356	0.3450	0.3478	0.3464	0.3449	0.3439	0.3429
Trial 9	0.3280	0.1588	0.2429	0.3597	0.3518	0.3959	0.3904	0.3863	0.3831	0.3805
Trial 10	0.2606	0.1820	0.3840	0.3549	0.3157	0.3444	0.3413	0.3391	0.3373	0.3358
Average	0.2749	**0.2590**	0.2861	0.3351	0.3170	0.3595	0.3552	0.3522	0.3500	0.3484
Median	0.2662	0.2615	**0.2456**	0.3501	0.3270	0.3493	0.3475	0.3460	0.3448	0.3438

Table 10 Relationship between α and MAE on the civil aero-engine dataset.

Test	The value of α									
	0	0.6	1.2	1.8	2.4	3	3.6	4.2	4.8	5.4
Trial 1	0.5141	0.4702	0.4389	0.4075	0.5284	0.5183	0.5110	0.5054	0.5012	0.4981
Trial 2	0.4161	0.3883	0.3428	0.3967	0.4181	0.4724	0.4611	0.4529	0.4468	0.4423
Trial 3	0.4106	0.4304	0.3818	0.4710	0.3471	0.4430	0.4382	0.4345	0.4317	0.4294
Trial 4	0.3740	0.3985	0.3713	0.4813	0.4682	0.4633	0.4577	0.4537	0.4509	0.4488
Trial 5	0.3189	0.4363	0.3683	0.4806	0.4559	0.4689	0.4660	0.4638	0.4620	0.4606
Trial 6	0.3624	0.4260	0.4659	0.4074	0.3347	0.4478	0.4422	0.4380	0.4348	0.4321
Trial 7	0.4212	0.4217	0.3844	0.4517	0.3975	0.3918	0.3803	0.3721	0.3665	0.3627
Trial 8	0.4192	0.3607	0.5105	0.4465	0.4559	0.4500	0.4479	0.4461	0.4448	0.4436
Trial 9	0.4666	0.2965	0.3870	0.4756	0.4590	0.4774	0.4724	0.4685	0.4655	0.4629
Trial 10	0.4023	0.3222	0.5003	0.4790	0.4240	0.4619	0.4582	0.4553	0.4529	0.4508
Average	0.4105	**0.3951**	0.4151	0.4497	0.4289	0.4595	0.4535	0.4490	0.4457	0.4431
Median	0.4134	0.4101	**0.3857**	0.4614	0.4400	0.4626	0.4580	0.4533	0.4489	0.4462

4.3.2 Comparison of different methods

Fig. 11 compares the experimental performance of four methods on the real civil aero-engine dataset. As shown in the figure, both the MSE and MAE of the CSDI method are generally the highest among the four. This may be due to the limited size of the aero-engine dataset, which causes CSDI to fail to fully converge due to its inherent method limitations. In most experimental results, the MSE and MAE of CUR-Estimator are lower than those of GP-VAE and BRITS. However, in the third experiment, the MAE of CUR-Estimator exceeds that of CSDI, and in the sixth experiment, both the MSE and MAE of CUR-Estimator are higher than those of BRITS. A possible explanation is that the samples selected for training in these experiments may not adequately represent the general degradation patterns of engines.

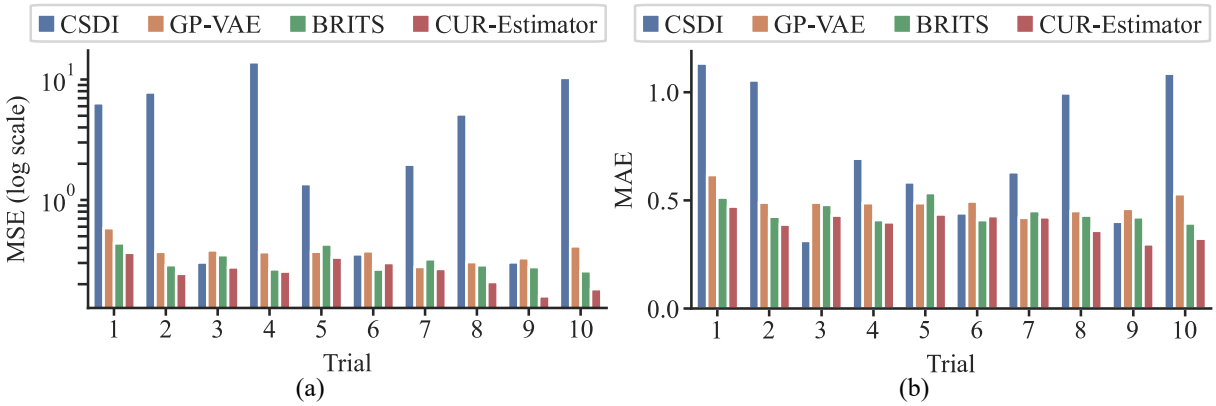

Fig. 11. Results of ten experiments with different methods on the civil aero-engine dataset, (a)MSE, (b)MAE.

Table 11 and Table 12 present the average values, standard deviations, and medians of the MSE and MAE errors for four methods over ten repeated experiments. From Table 11, it can be observed that the average MSE and MAE for CUR-Estimator across the ten experiments are 0.2590 and 0.3951, respectively, while the standard deviations are 0.0612 and 0.0515. In comparison, the average MSE and MAE for BRITS are 0.3174 and 0.4464, with standard deviations of 0.0631 and 0.0455. CUR-Estimator shows a slightly larger MAE standard deviation than BRITS, which may be due to higher noise in the dataset compared to the C-MAPSS dataset, leading to greater variability in CUR-Estimator's results across multiple experiments. As shown in Table 12, the median values of the MSE and MAE for CUR-Estimator are also the lowest among all the comparison methods, being 0.2615 and 0.4101, respectively.

Table 11 Mean results on the civil aero-engine dataset.

Average±STD	Method			
	CSDI	GP-VAE	BRITS	CUR-Estimator
MSE	4.7831 ± 4.5311	0.3780 ± 0.0784	0.3174 ± 0.0631	**0.2590 ± 0.0612**
MAE	0.7326 ± 0.2935	0.4928 ± 0.0502	0.4464 ± 0.0455	**0.3951 ± 0.0515**

Table 12 Median results on the civil aero-engine dataset.

Median	Method			
	CSDI	GP-VAE	BRITS	CUR-Estimator
MSE	3.5516	0.3721	0.2876	**0.2615**
MAE	0.6616	0.4886	0.4274	**0.4101**

To compare the performance of different methods under varying gap sizes, MAE is used as the evaluation metric to analyze how the imputation accuracy changes as the size of the missing interval increases. The results of the first test set from 10 repeated experiments are selected for analysis, and the performance under different gap sizes is statistically summarized, as shown in Table 13. As observed in the table, with the exception of the CSDI method, which exhibits generally unsatisfactory performance, all other methods show increased errors when the number of consecutive missing points increases from 1 to 3. The differences among the three remaining methods are compared in terms of the relative increase in MAE. Specifically, the GP-VAE method shows an 11.1% increase in error, the BRITS method increases by 13.0%, and the CUR-Estimator method shows only a 7.1% increase. These results indicate that the CUR-Estimator method exhibits the smallest increase in error among the compared methods. Its error increase is 36% lower than that of GP-VAE and 45% lower than that of BRITS, suggesting that the CUR-Estimator method maintains better imputation performance as the size of missing intervals grows.

Table 13 Error on different gaps on the civil aero-engine dataset.

Gaps	Method			
	CSDI	GP-VAE	BRITS	CUR-Estimator
1	1.2343	0.6237	0.5207	0.4459
2	1.3027	0.6066	0.4878	0.3869
3	0.6542	0.6927	0.5885	0.4777

Although the training process of the CUR-Estimator is relatively complex, its inference efficiency is comparable to that of other methods. Under the experimental environment described in Section 4.1.2, the inference time was repeatedly evaluated to compare the computational costs of different methods, as summarized in Table 14. As shown in the table, GP-VAE achieves the fastest inference among the four methods, while CSDI incurs the highest computational cost. The inference time of the proposed CUR-Estimator is similar to that of BRITS, demonstrating a level of efficiency that is acceptable for practical applications.

Table 14 Inference time comparison of different methods

Method	Time (s)
CSDI	186.44
GP-VAE	2.04
BRITS	34.63
CUR-Estimator	33.97

4.3.3 Hypothesis validity experiment

To verify whether CUR-Estimator can reduce the likelihood of generating unreasonable values, it is necessary to compare CUR-Estimator with other advanced methods. For convenience, the experiment directly uses BRITS as a control, and the deviation between the imputed results and the actual values is shown in Table 15. Like the experiments conducted on the C-MAPSS dataset, the difference between the imputed values and the actual values from different methods are recorded in the table, documenting the left bound, right bound, and the range of deviation across multiple experiments. From Table 15, it can be observed that the absolute values of the average left bound, and the average right bound in ten experiments of CUR-Estimator are 1.6312 and 2.0151, respectively, which are smaller than those of BRITS (1.6418 and 2.1647). The overall average deviation range of CUR-Estimator is also smaller.

Table 15 Boundaries of the deviation for BRITS and CUR-Estimator on the civil aero-engine dataset.

Test	BRITS			CUR-Estimator		
	Left edge	Right edge	Range	Left edge	Right edge	Range
Trial 1	-1.7291	2.2351	3.9642	-1.6312	2.0151	3.6463
Trial 2	-1.8286	1.9011	3.7298	-2.0229	1.5835	3.6064
Trial 3	-1.4038	2.2346	3.6384	-1.4794	2.0418	3.5212
Trial 4	-1.6045	1.9016	3.5062	-2.0582	1.8636	3.9218
Trial 5	-1.2553	2.7434	3.9987	-1.5672	2.5875	4.1547
Trial 6	-1.5263	2.2951	3.8215	-1.7809	2.2164	3.9973
Trial 7	-1.6050	2.1532	3.7581	-1.5479	1.7211	3.2690
Trial 8	-1.6986	2.2008	3.8994	-1.7278	1.8016	3.5294
Trial 9	-2.0173	1.9392	3.9565	-1.6052	1.7229	3.3281
Trial 10	-1.7496	2.0425	3.7921	-1.9704	1.7533	3.7237
Average	-1.6418	2.1647	3.8065	-1.6312	2.0151	3.6463

4.3.4 Experimental results under different SNRs

Fig. 12 shows the MSE and MAE data of the imputed values and actual values from real civil aero-engine data under different SNRs. As can be seen from the figure, although the standard deviation between the two methods is close under different SNRs, CUR-Estimator

consistently exhibits smaller errors compared to BRITS under the same SNR.

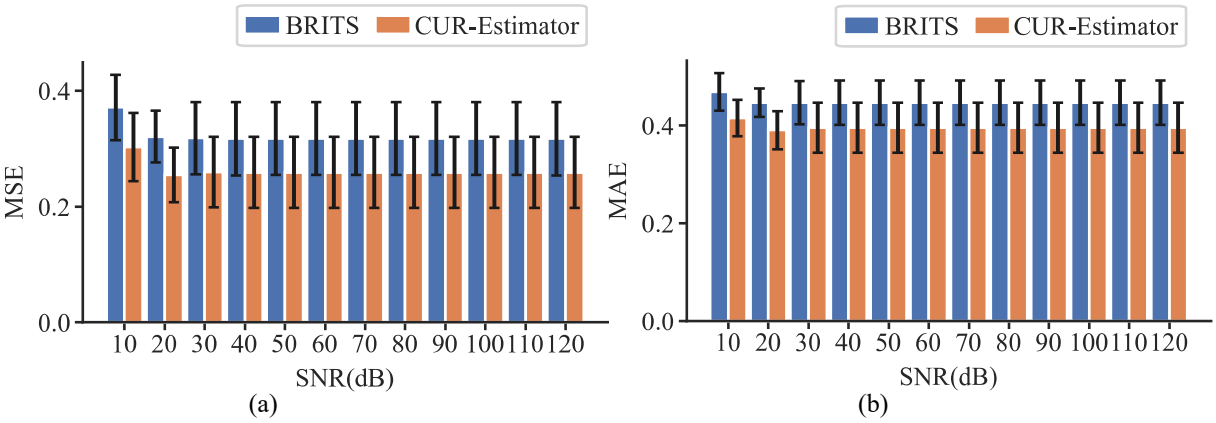

Fig. 12. Bar charts and standard deviations of MSE and MAE under different SNRs on the civil aero-engine dataset.

4.4 Generalization experiments for different methods

To verify the applicability of the CUR-Estimator to other types of complex equipment, a generalization experiment was conducted using a wind power dataset. The dataset is publicly available and consists of real-world wind power generation data collected by a wind power operating company. It contains approximately 300,000 records from 10 different wind farms.

For each wind farm, the training data were subjected to 10 repeated experiments using all baseline methods discussed in this paper. The hyperparameter settings remained consistent with those used in previous sections. The visualized results of the 10 experiments are shown in Fig. 13. As observed from the figure, unlike the results on the aero-engine dataset, the CSDI method shows better performance than GP-VAE on the wind power dataset. In contrast, the GP-VAE method exhibits relatively large errors under both evaluation metrics. Regarding MSE, BRITS performs better than CUR-Estimator in some runs; however, in the first experiment, BRITS yielded a notably poor result. When considering the corresponding MAE, this suggests that BRITS produced a small number of large deviations during imputation. In terms of MAE, the proposed CUR-Estimator outperforms the baseline methods in the majority of the experiments.

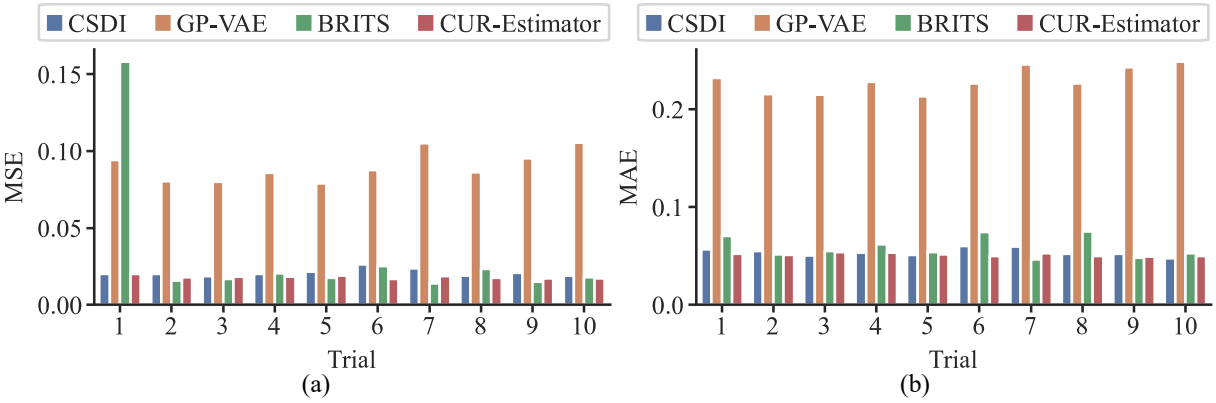

Fig. 13. Results of ten experiments on the wind power dataset using different methods, (a)MSE, (b)MAE.

Table 16 and Table 17 present the average, standard deviation, and median values across the ten repeated experiments. As shown in Table 16, GP-VAE performs the worst on this dataset, with the highest average MSE and MAE values of 0.0899 and 0.2292, respectively. BRITS ranks second, with average values of 0.0324 for MSE and 0.0589 for MAE. The proposed CUR-

Estimator achieves the best performance, with the lowest average MSE and MAE of 0.0181 and 0.0514, respectively. It is also noteworthy that CUR-Estimator shows the smallest standard deviations among all methods—0.0009 for MSE and 0.0016 for MAE—indicating its stability across repeated runs. In terms of median values, BRITS achieves the lowest MSE median of 0.0177, which aligns with the visual results shown in Fig. 13(a). The CUR-Estimator follows closely with a median MSE of 0.0182, only 0.0005 higher than BRITS. However, for the MAE metric, BRITS yields a median of 0.0546, which is even worse than that of CSDI (0.0527). In contrast, the CUR-Estimator achieves the lowest MAE median of 0.0514.

Table 16 Mean results on the wind power dataset.

Average±STD	Method			
	CSDI	GP-VAE	BRITS	CUR-Estimator
MSE	0.0209 ± 0.0023	0.0899 ± 0.0093	0.0324 ± 0.0421	**0.0181 ± 0.0009**
MAE	0.0538 ± 0.0038	0.2292 ± 0.0123	0.0589 ± 0.0103	**0.0514 ± 0.0016**

Table 17 Median results on the wind power dataset.

Median	Method			
	CSDI	GP-VAE	BRITS	CUR-Estimator
MSE	0.0200	0.0869	**0.0177**	0.0182
MAE	0.0527	0.2272	0.0546	**0.0514**

Based on the above analysis, it can be concluded that the proposed CUR-Estimator outperforms the baseline methods in terms of missing data imputation on the wind power dataset and shows strong stability across 10 repeated experiments. These results effectively validate the applicability of the proposed method not only to aero-engine data but also to degradation data from other types of complex equipment.

5 Conclusion

This paper proposes a Constrained Unseen Recovery Estimator (CUR-Estimator) for aero-engine performance data. The network not only considers the impact of the engine's performance degradation at different stages and over different durations during each flight but also effectively imputes in missing parts of the engine performance data. Additionally, to address the issue of unreasonable values generated by neural networks due to their lack of interpretability, PCHIP is used to constrain the range of imputed results during the training phase, reducing the possibility of ITIN generating unreasonable values. Experiments are conducted using the C-MAPSS dataset and real civil aero-engine data. The experimental results show that the proposed method achieves better results compared to the current state-of-the-art approaches. Moreover, the paper discusses the hyperparameter settings of the proposed method and evaluates its stability on datasets under different noise conditions. The results show that the stability of CUR-Estimator surpasses that of advanced RNN-based methods on the two datasets.

Considering that the challenges associated with imputing data under different interval sizes tend to exhibit similar characteristics across various domains, a generalization experiment was conducted on a wind power dataset. The results show that the proposed method is not only applicable to temporal sequence modeling of missing data in aero-engines, but also effective for imputing data in other types of complex equipment. In addition, the use of statistical methods to constrain the neural network paradigm can be extended to combinations of other neural networks and statistical approaches.

In future work, we will further investigate the potential issue of distributional shifts under non-identically distributed conditions, in order to enhance the adaptability of the proposed method to diverse application scenarios.

6 Acknowledgement

This work was supported by National Key R&D Program of China (2023YFB4302400), National Natural Foundation of China (No. 92360308).

7 Reference

[1] N. Chen, Y. Sun, Z. Wang, C. Peng, Correction and Fitting Civil Aviation Flight Data EGT Based on RPM: Polynomial Least Squares Analysis, Applied Sciences 12 (2022) 2545. https://doi.org/10.3390/app12052545.

[2] X. Fang, R. Zhou, N. Gebraeel, An adaptive functional regression-based prognostic model for applications with missing data, Reliability Engineering & System Safety 133 (2015) 266–274. https://doi.org/10.1016/j.ress.2014.08.013.

[3] I. Pratama, A.E. Permanasari, I. Ardiyanto, R. Indrayani, A review of missing values handling methods on time-series data, in: 2016 International Conference on Information Technology Systems and Innovation (ICITSI), 2016: pp. 1–6. https://doi.org/10.1109/ICITSI.2016.7858189.

[4] Y. Luo, X. Cai, Y. Zhang, J. Xu, X. Yuan, Multivariate Time Series Imputation with Generative Adversarial Networks, in: Advances in Neural Information Processing Systems, Curran Associates, Inc., 2018.

[5] I.J. Schoenberg, Contributions to the problem of approximation of equidistant data by analytic functions. Part A. On the problem of smoothing or graduation. A first class of analytic approximation formulae, Quart. Appl. Math. 4 (1946) 45–99. https://doi.org/10.1090/qam/15914.

[6] F.N. Fritsch, R.E. Carlson, Monotone Piecewise Cubic Interpolation, SIAM Journal on Numerical Analysis 17 (1980) 238–246.

[7] R. Pan, T. Yang, J. Cao, K. Lu, Z. Zhang, Missing data imputation by K nearest neighbours based on grey relational structure and mutual information, Applied Intelligence 43 (2015). https://doi.org/10.1007/s10489-015-0666-x.

[8] Bashir F., Wei H.-L., Handling missing data in multivariate time series using a vector autoregressive model-imputation (VAR-IM) algorithm, Neurocomputing 276 (2018) 23–30. https://doi.org/10.1016/j.neucom.2017.03.097.

[9] J.-P. Montillet, M.S. Bos, eds., Geodetic Time Series Analysis in Earth Sciences, Springer International Publishing, Cham, 2020. https://doi.org/10.1007/978-3-030-21718-1.

[10] R.C. Pereira, M.S. Santos, P.P. Rodrigues, P.H. Abreu, Reviewing Autoencoders for Missing Data Imputation: Technical Trends, Applications and Outcomes, Journal of Artificial Intelligence Research 69 (2020) 1255–1285. https://doi.org/10.1613/jair.1.12312.

[11] Y. Liu, R. Yu, S. Zheng, E. Zhan, Y. Yue, NAOMI: Non-Autoregressive Multiresolution Sequence Imputation, in: Advances in Neural Information Processing Systems, Curran Associates, Inc., 2019.

[12] A.W. Mulyadi, E. Jun, H.-I. Suk, Uncertainty-Aware Variational-Recurrent Imputation Network for Clinical Time Series, IEEE Transactions on Cybernetics 52 (2022) 9684–9694. https://doi.org/10.1109/TCYB.2021.3053599.

[13] Z. Pan, Y. Wang, K. Wang, H. Chen, C. Yang, W. Gui, Imputation of Missing Values in Time Series Using an Adaptive-Learned Median-Filled Deep Autoencoder, IEEE Transactions on Cybernetics 53 (2023) 695–706. https://doi.org/10.1109/TCYB.2022.3167995.

[14] W. Du, D. Côté, Y. Liu, SAITS: Self-attention-based imputation for time series, Expert Systems with Applications 219 (2023) 119619. https://doi.org/10.1016/j.eswa.2023.119619.

[15] Y. Tashiro, J. Song, Y. Song, S. Ermon, CSDI: Conditional Score-based Diffusion Models

for Probabilistic Time Series Imputation, in: Advances in Neural Information Processing Systems, 2021: pp. 24804–24816.
[16] S. Zheng, N. Charoenphakdee, Diffusion models for missing value imputation in tabular data, (2023). https://doi.org/10.48550/arXiv.2210.17128.
[17] Q. Suo, W. Zhong, G. Xun, J. Sun, C. Chen, A. Zhang, GLIMA: Global and Local Time Series Imputation with Multi-directional Attention Learning, in: 2020 IEEE International Conference on Big Data (Big Data), 2020: pp. 798–807. https://doi.org/10.1109/BigData50022.2020.9378408.
[18] A.Y. Yıldız, E. Koç, A. Koç, Multivariate Time Series Imputation With Transformers, IEEE Signal Processing Letters 29 (2022) 2517–2521. https://doi.org/10.1109/LSP.2022.3224880.
[19] Z. Che, S. Purushotham, K. Cho, D. Sontag, Y. Liu, Recurrent Neural Networks for Multivariate Time Series with Missing Values, Scientific Reports 8 (2018) 6085. https://doi.org/10.1038/s41598-018-24271-9.
[20] W. Cao, D. Wang, J. Li, H. Zhou, L. Li, Y. Li, BRITS: Bidirectional Recurrent Imputation for Time Series, in: Advances in Neural Information Processing Systems, 2018.
[21] J. Yoon, W.R. Zame, M. van der Schaar, Estimating Missing Data in Temporal Data Streams Using Multi-Directional Recurrent Neural Networks, IEEE Transactions on Biomedical Engineering 66 (2019) 1477–1490. https://doi.org/10.1109/TBME.2018.2874712.
[22] Y. Luo, Y. Zhang, X. Cai, X. Yuan, E^2GAN: End-to-End Generative Adversarial Network for Multivariate Time Series Imputation, in: Proceedings of the 28th International Joint Conference on Artificial Intelligence, 2019: pp. 3094–3100.
[23] Y.J. Kim, M. Chi, Temporal Belief Memory: Imputing Missing Data during RNN Training, in: International Joint Conference on Artificial Intelligence, 2018: pp. 2326–2332. https://www.ijcai.org/proceedings/2018/322.
[24] Y. Sun, J. Li, Y. Xu, T. Zhang, X. Wang, Deep learning versus conventional methods for missing data imputation: A review and comparative study, Expert Systems with Applications 227 (2023) 120201. https://doi.org/10.1016/j.eswa.2023.120201.
[25] K. Cao, M. Liu, H. Su, J. Wu, J. Zhu, S. Liu, Analyzing the Noise Robustness of Deep Neural Networks, IEEE Transactions on Visualization and Computer Graphics 27 (2021) 3289–3304. https://doi.org/10.1109/TVCG.2020.2969185.
[26] A. Saxena, K. Goebel, D. Simon, N. Eklund, Damage propagation modeling for aircraft engine run-to-failure simulation, in: 2008 International Conference on Prognostics and Health Management, 2008: pp. 1–9. https://doi.org/10.1109/PHM.2008.4711414.