BIG DATA MINING AND ANALYTICS
ISSN 2096-0654 03/15 pp29–41
DOI: 10.26599/BDMA.2023.9020010
Volume 7, Number 1, March 2024

IoTDQ: An Industrial IoT Data Analysis Library for Apache IoTDB

Pengyu Chen*, Wendi He, Wenxuan Ma, Xiangdong Huang*, and Chen Wang

Abstract: There is a growing demand for time series data analysis in industry areas. Apache IoTDB is a time series database designed for the Internet of Things (IoT) with enhanced storage and I/O performance. With User-Defined Functions (UDF) provided, computation for time series can be executed on Apache IoTDB directly. To satisfy most of the common requirements in industrial time series analysis, we create a UDF library, IoTDQ, on Apache IoTDB. This library integrates stream computation functions on data quality analysis, data profiling, anomaly detection, data repairing, etc. IoTDQ enables users to conduct a wide range of analyses, such as monitoring, error diagnosis, equipment reliability analysis. It provides a framework for users to examine IoT time series with data quality problems. Experiments show that IoTDQ keeps the same level of performance compared to mainstream alternatives, and shortens I/O consumption for Apache IoTDB users.

Key words: industrial big data; data quality; data mining and analytics

1 Introduction

Internet of Things (IoT) is dedicated to connection and communication among machines. With the expansion of IoT systems and advancement in Internet transport technologies, there is a growing demand for industrial data collection, storage and processing. In this paper, we will introduce IoTDQ, a data processing library based on Apache IoTDB, designed for comprehensive data diagnosis in industrial areas.

1.1 Time series database and Apache IoTDB

For the time being, most of the time series databases, among which InfluxDB, Kdb+, and OpenTSDB are popular, are designed for scenarios of supervising servers or financial computation. However, an IoT system can create millions of data points per second, which could become a bottleneck for traditional time series databases. Apache IoTDB is an ideal time series database for industrial scenarios. Apache IoTDB supports flexible deployment and features low storage cost and I/O efficiency[1, 2]. With its excellent performance and semantic support, Apache IoTDB provides highly efficient functions from storage to queries.

Similar to InfluxDB and some other databases, Apache IoTDB[3, 4] allows utilizing User-Defined Function (UDF) when querying. Naturally, UDF for time series consumes data either by row or window, which usually corresponds to transformation functions and aggregation functions. With the help of Apache IoTDB UDF, we are able to construct a library for industrial data analysis usage.

1.2 Computation problems in time series from IoT

Models on traditional time series have been widely discussed in the past decades. These models cover stationary tests, forecast, imputation, and segmentation. These methods usually create a mathematical model with hypotheses such as Box-Jenkins modeling[5].

• Pengyu Chen, Wendi He, Wenxuan Ma, and Xiangdong Huang are with School of Software, Tsinghua University, Beijing 100084, China. E-mail: chenpy20@mails.tsinghua.edu.cn; hewd21@mails.tsinghua.edu.cn; mwx22@mails.tsinghua.edu.cn; huangxdong@tsinghua.edu.cn.
• Chen Wang is with National Engineering Research Center for Big Data Software (NERCBDS), Tsinghua University, Beijing 100084, China. E-mail: huangxdong@tsinghua.edu.cn.
* To whom correspondence should be addressed.

Manuscript received: 2022-08-30; revised: 2023-03-27; accepted: 2023-05-15

Recently, researches on time series analysis prefer to utilize machine learning to deal with complicated stochastic processes. For example, transformer[6], RNN[7], GNN[8,9], etc., are applied for time series forecast, anomaly detection, imputation, and other purposes.

Despite vast progress in time series model constructions, industrial IoT encounters rather fundamental problems. Meanwhile, Apache IoTDB has met the requirements of IoT data collection and storage, but these requirements have not been met in upstream data processing stage.

Different from classical time series data such as stock price or monthly sales, IoT sensors collect data in a somehow unstable way. Due to physical damage, transport delay, or nonstandard operations, data points do not necessarily line up with equal time intervals, and noises often show up in commonly seen electric signals. These are concluded as data quality problems. In other words, only if we solve these problems, it will be possible to apply time series models on downstream.

1.3 Designing requirements of IoTDQ

Based on these computation problems, we propose IoTDQ, a UDF library for Apache IoTDB, designed for IoT data processing. The primal requirement of constructing an industrial UDF library, namely IoTDQ, is to realize stream computing via Apache IoTDB UDF API. An Apache IoTDB UDF may consume data points either row by row or from a sliding window. Although it is theoretically possible to adopt all data points in a single window, streaming computing satisfies the computation problems of industrial big data[10]. Therefore, the main challenge of IoTDQ is to select, optimize, and integrate algorithms, which support an industrial analysis pipeline.

To construct such a pipeline, there are several types of functions to be collected. Primordial Apache IoTDB already offers a few fundamental aggression functions, including minimum, maximum, 1st order difference, and so on. IoTDQ thus should contain more statistic functions, which can generate a quick data profile. These functions do not aim to fix data quality problems in one step. Rather, these functions are practical for real-time system monitoring. They provide sketching profiles for each IoT device, which are economical ways to realize initial data diagnosis.

The second step is to satisfy the primary needs of IoT data. A typical case is to monitor the stability of the sensor and sketch out how often it goes wrong. Based on cases of two industrial scenes, i.e., wind turbines and vehicle engines, we draft a data quality paradigm for general IoT scenarios, which has been included in our library. Later we will show how data quality paradigm can help to infer faults in industrial production environments. Furthermore, we also take repairing quality problems into consideration. IoTDQ integrates novel restriction-based time series repairing algorithms. Via our library, users may make remedy for origin time series data of low data quality. The third step in the pipeline is to integrate some typical time series models into Apache IoTDB. It is usually the final step to utilize these models when processing a time series. By including these functions, we desire to make Apache IoTDB cover the full stage of IoT data analysis.

IoTDQ has now become a sub-project in Apache IoTDB[11], known as UDF-Library. As shown in Fig. 1, composed of six modules, IoTDQ provides enhanced computation functions based on Apache IoTDB UDF, and user may operate IoTDQ with IoTDB JDBC to accomplish data preprocess. The code is available at Github[12], and the document can be viewed on the official website of Apache IoTDB[13]. In this paper, we present all categories of IoTDQ divided by functions. We also summarize a few industry scenarios, with some examples of IoTDQ, to elaborate universality of

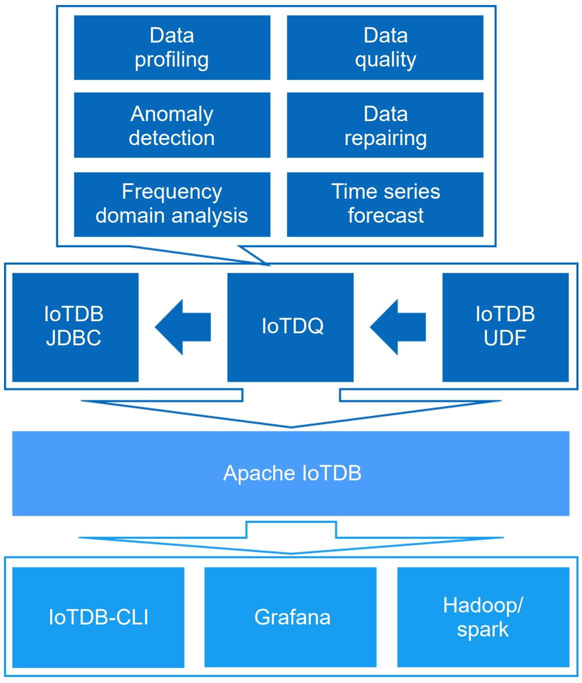


Fig. 1 IoTDQ in Apache IoTDB ecosystem.

this library.

2 Overview of IoTDQ

A suvery[14] points out that time series data mining involves three major questions, namely data representation, similarity measurement, and indexing method. A database system covers resolution for indexing method, and IoTDQ attempts to make up for the other two questions. In detail, concrete tasks to solve these questions include clustering, classification, segmentation, prediction, anomaly detection, motif discovery, and so on. As for industrial time series, these questions are tightly related. A full process of computing includes data quality diagnosis, data profiling, anomaly detection, repairing, and model fitting and forecast. IoTDQ tries to focus on most aspects mentioned above, and offers a pipeline for conventional IoT time series.

Figure 2 is a flowchart which shows how a user can use IoTDQ as a standard diagnosis tool. According to

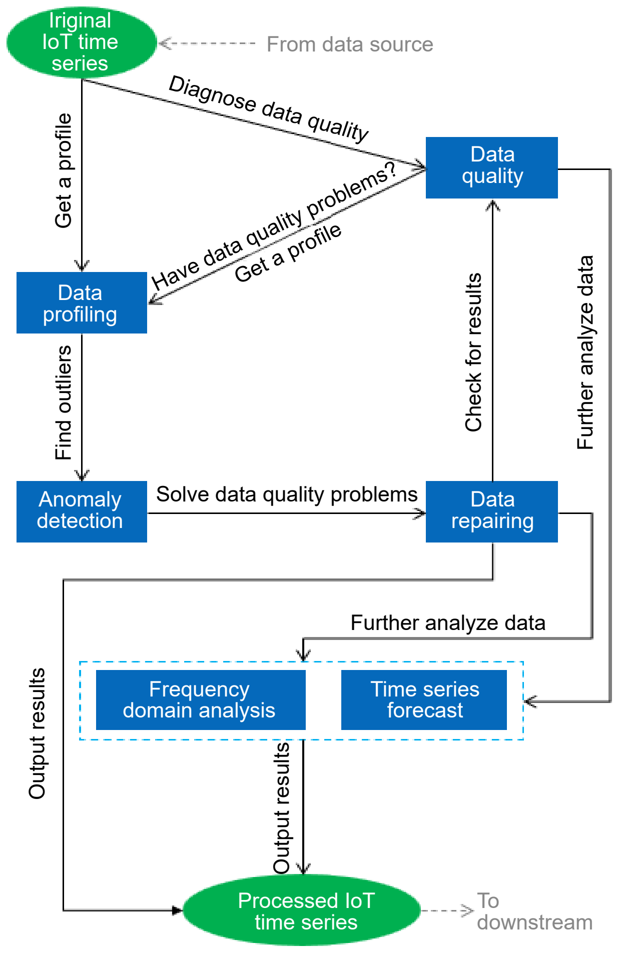


Fig. 2 Pipeline of IoT time series in IoTDQ.

the functions provided, six categories in IoTDQ compose the pipeline, namely data profiling, data quality, anomaly detection, data repairing, frequency domain analysis, and time series forecast. In this section, we will discuss the modification of algorithms in each category that meets the designing requirements mentioned in Section 1.3.

2.1 Data quality

Data quality problems were first summarized by Lee et al.[15] These problems in relational data have been clearly clarified[16], but IoT data quality problems behave differently from them. Data quality problems in IoT time series are caused by the instability of IoT sensors. A thermocouple sensor translates an electric signal to temperature, but jamming signals often appear, leading to abnormal signal fluctuation. Connections between sensor and receiver sometimes get lost, causing discontinuous periods in time series. Similar occasions occur repeatedly in industry scenarios. These become troubles when dirty and irregularly arranged raw data is undesired for downstream analysis. Most time series analysis methods, like decomposition and model regression, treat data with equal time intervals. Consequently, we need remedies for data quality problems.

Song and Zhang[17] proposed a framework for IoT data quality analysis, which includes three dimensions, namely validity, completeness, and consistency. Validity considers about data constraints; completeness measures the degree all data are observed; consistency evaluates how data are consistent with each other. However, this framework mainly focuses on relational data, and utilizes corresponding examination methods. Then the question lies on how to extend this framework to describe quality problems on IoT time series. We have brought a general data quality classification for overall quality examination on time series (see Table 1). We induct common data quality problems into four indices and twelve error types. Each index signs for a root cause of data quality problems, and ranges from 0 to 1.

Fang et al.[18] discussed on completeness, consistency, and timeliness. Here we summarize the definition and evaluations of these indices, and then introduce their corresponding stream computing UDFs.

Completeness describes the proportion of correctly collected data points, as shown in the following:

Table 1 Data quality issues.

Completeness error	Consistency error	Validity error	Timeliness error
Missing point	Overcrowded point	Value exceeding threshold	Latency
NaN value	Repeated point	Speed exceeding threshold	—
Error code representing NaN	—	Acceleration exceeding threshold	—
Power-off	—	Jerk exceeding threshold	—

Note: NaN means "not a number".

$$\text{Completeness} = 1 - \frac{N_{\text{null}} + N_{\text{special}} + N_{\text{miss}}}{N + N_{\text{miss}}} \tag{1}$$

where N_{null}, N_{special}, and N_{miss} are the numbers of null, special, and missing values in a series length of N.

Assuming that an IoT sensor creates a data point with a fixed time interval, we can infer on which timestamp there is a missing value. Thus completeness indicates the stability of data transport. Also, when the sensor is power-off, it leaves the series continuous blank points. In addition to the raw definition, power-off periods are excluded when using "Completeness". This UDF recognizes power-off by detecting missing of a long duration.

Consistency describes the cleanness of data points in refer to timestamps, as shown in the following:

$$\text{Consistency} = 1 - \frac{N_{\text{redundancy}}}{N} \tag{2}$$

where $N_{\text{redundancy}}$ is the number of redundant points.

This index is concluded from the phenomenon that physical errors of IoT sensor or network cause the creation of unwanted extra points. Sometimes a data point is collected two times, or sometimes the sensor sends multiple points in a very short period. Time series databases always neglect overlapped points, so the function focuses on finding out overcrowded points.

Timeliness errors usually occur when there is a transport latency. The computation is defined in the following:

$$\text{Timeliness} = 1 - \frac{N_{\text{latency}}}{N} \tag{3}$$

where N_{latency} is the number of latent points.

Different from consistency errors, timeliness errors display as data point with incorrect timestamp.

These mentioned indices are integrated into IoTDQ as "Completeness", "Consistency", and "Timeliness". The fundamental methods consumes all data points to infer the standard timestamp interval by estimating the median of 1st order timestamp differences. IoTDQ also provides the stream computation variations, namely "StreamCompleteness", "StreamConsistency", and "StreamTimeliness". These UDFs infer standard time interval from the first sliding window. When consuming new data points, the self-adaptive functions detect whether the standard interval has changed.

Validity errors may come from either IoT devices or the measured system itself, as shown in the following:

$$\text{Validity} = 1 - \frac{\sum_i N_{\text{anomaly}_i}}{N} \tag{4}$$

where N_{anomaly_i} is the number of points exceeding threshold on i-th order difference. This index does not take timestamps into account.

In the general, validity measures the proportion of outliers according to system restrictions. In a usual case, a thermal sensor records temperature in a desired value range. By setting a threshold to the series, it is possible to calculate the number of illegal points. Sometimes the thresholds are set to seed, acceleration or even higher order of difference. And by default, through input series, we may infer a threshold by executing anomaly detection methods when the threshold is not given.

Among these four indices, completeness, consistency, and timeliness are calculated by finding out theoretically correct timestamps to decide the number of missing, redundant, and latent points. Suppose we have got a standard timestamp interval, the remaining question lies in matching each point from the original series to a reference point acquired by the interval. To construct a streaming algorithm, we calculate the difference of neighboring timestamps, and convert it to a type of error,

$$\text{error} = \begin{cases} \text{power-off}, \ 10\delta < \Delta t; \\ \text{missing}, \ 1.5\delta < \Delta t \leqslant 10\delta; \\ \text{latency}, \ 0.5\delta < \Delta t \leqslant 1.5\delta; \\ \text{redundancy}, \ 0 < \Delta t \leqslant 0.5\delta \end{cases} \tag{5}$$

which defines the recognition of error types. δ is the standard timestamp interval, and Δt is the 1st order difference of original timestamps. Note that power-off is not considered as an error, since no points are expected when equipment is offline. And the judgement of power-off here is comparatively rough. There is a case in Section 3.1, where a delicate method is introduced.

Now consider the situation where the user does not provide a standard time interval, which is often the case. If the function consumes all data points at once, then we may choose the mode or median of the 1st order timestamp interval as the standard interval. However, as mentioned in Section 1.3, we desire a streaming algorithm to deal with the original series. We focus on a sliding window to determine the standard interval. Take δ as undecided standard timestamp interval, t_i as timestamp of the i-th point from the original series, t_i' as theoretical timestamp of the i-th data point, then the standard interval is decided in the following:

$$\arg\min_{\delta}\left\{\sum_i \left(t_i, t_i'\right)^2 + n\delta^2\right\} \tag{6}$$

where n is the number of theoretical points which do not correspond to a point from original series. Figure 3 illustrates an example. To solve Formula (6), the functions search δ near the median of 1st order timestamp difference of the sliding window, and always suppose that the first point in the sliding window is on the correct timestamp. For each window, the function calculates three indices, and the overall indices are the arithmetic mean of all windows.

The calculation of validity is related to anomaly detection problems. Here we take a snap on how this function profiles a time series. This function uses Median Absolute Deviation (MAD) as a standard to assess validity errors from a series. in Section 2.2, we will further introduce this profiling function. In a word, MAD profiles the ranges in which values are distributed depart from the median, and provides a reference for a reasonable range of value, speed, and acceleration changes. In each sliding window, the function calculates MAD and then checks the value correctness of each point.

Based on the standards of error classification, we develop these four data quality functions to compute the indices. The results should be considered as an overall assessment on data cleanness. Users may conduct retrospective analysis of generated data, where data profiling functions usually help, and try to fix root problems intensively. In Section 3.2, we present an example of computing data quality indices.

2.2 Data profiling

Data profiling functions may be treated as beneficial supplements for primordial Apache IoTDB, which lacks statistics computation functions. In correspondence to data mining tasks, this class covers data representation, which aims to show fundamental properties or statistics of time series. This is the second step that IoT data enters. Whether there are data quality problems or not, a time series digest helps user infer basic problems alongside data quality indices. Table 2 lists these fundamental functions.

The question remains the same as other functions. Considering the large scale of data points in time series in industry scenarios, statistics functions require stream computing algorithms to satisfy execution efficiency. For example, the convenience of implementation takes precedence in Apache DataFu for Spark[19], a UDF library extended from Apache DataFu for Pig[20]. In reality, these statistics which support stream computation algorithms, such as variance, skewness, and integral average, can only make a rough sketch of time series, and somehow neglect special properties of time series. For statisticians and economists, complex models which fully consider self-correlation are frequently utilized. Therefore IoTDQ chooses to keep the comprehensive practicability of functions despite the possible high resource occupation.

As mentioned in Section 2.1, MAD is an effective index to describe data departure from median[21]. Shrivastava et al.[22] proposed Q-Digest method as a stream computing algorithm to calculate approximate quantiles. Similarly, Chen et al.[21] proposed TP-MAD,

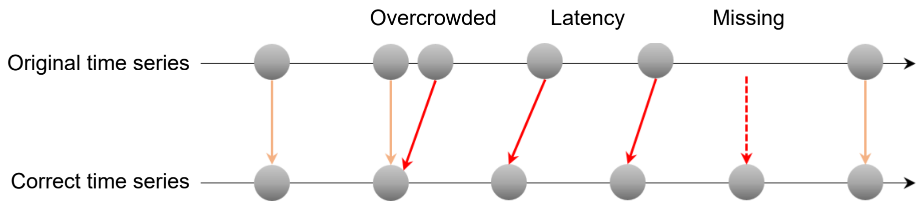


Fig. 3 Example of deciding standard interval.

Table 2 Data profiling functions.

Function	Introduction	Function	Introduction
ACF	Compute auto-correlation factor	PACF	Compute partial auto-correlation factor
Distinct	Show distinct values	Percentile	Find value on appointed percentile
Histogram	Construct a histogram with appointed buckets	Period	Calculate length of period in periodic data
Integral	Compute trapezoidal integral	QLB	Compute p-value of Ljung-Box Q statistics
IntegralAvg	Compute integral mean	Sample	Do data sampling
Median	Compute median	Skew	Compute skewness
MAD	Compute mean absolute deviation	Spread	Compute spread
MinMax	Do min-max uniformization	Stddev	Compute standard deviation
Mode	Compute mode	ZScore	Do z-score normalization
MvAvg	Compute moving average		

an stream computing method to calculate MAD with provided bounded error with bucket skipping improvement. IoTDQ adopts this algorithm, and in Section 4 there is a comparison between the accurate method and TP-MAD.

2.3 Anomaly detection

Anomaly detection in times series, according to targeted outlier types, can be classified as point detection, subsequence detection and series detection[23]. For series detection, functions in data matching (see Section 2.6) and in data profiling reduce the dimension of the original series, and make it possible to execute clustering, by which the problem turns into a point detection problem. This section mainly discusses the other two situations.

As for point detection, fundamental algorithms rely on basic statistics like standard deviation and InterQuartile Range (IQR). These models detect outliers by regression and judging by the distribution of error. For example, k-sigma method hypothesizes normal distribution, and utilizes standard deviation to assist detection. Chen et al.[21] studied on the stability of standard deviation of data points in a sliding window on IoT time series, and it turned out that standard deviation fluctuates when the data window slides, because very few outliers contribute greatly to it. Except for some extreme cases, MAD fluctuates much milder than standard deviation in a sliding window. At the price of a higher degree of calculation complexity, detection criteria replaced with MAD provides higher accuracy especially when there are extreme outliers. The same criteria can be applied on higher orders of difference of data, according to physics principles of the series.

Despite those based on distribution, there are also means of detection based on distance and density. However, these methods, among which DBSCAN[24] is one of the most commonly used, usually require to iterate over all historical data points in stream data. Again, this is intolerable in large scale of data stream. Yang et al.[25] first proposed neighbor-based detection, which means the degree outlier is mainly based on neighbors in a sliding window, and thus feasible for large scale of data. There are already much practice on real-time outlier detection of time series, such as Abstract-C[25], MCOD[26], LEAP[27], and NETS[28]. Most of these implementations focus on the change of outlier status when there is a shift on the overall data, or in other words, a concept drift. Another key common character of these methods is that they still keep most of information of past sliding windows to make it more precise for calculating distance or density. This idea is mainly inherited from DBSCAN or other classical algorithms.

The challenge in questions of this type still lies in the balance of the precision of calculation and cost of time. A typical improvement is to use grids to conduct rough and faster evaluation. Methods like CPOD[29] or DORC[30] follow this idea. In IoTDQ, we provide “GridDetect” and “SphereDetect”, which drop more information and ignore the degree of outlier that changes because of concept drift. We recommend to fit a new model when a concept drift occurs, instead of one function to solve all anomaly problems. The core concept of our method is migration of collision detection to anomaly detection on multivariate time series. When initializing, we convert points of high density or low distance into a solid object in space, and then partition the space by grid, spherical bounding volume hierarchy tree, or k-dimensional tree. In “GridDetect”, nearby points are converted to an object composed by a series of neighboring hyper-cubes, and in “SphereDetect”, a series of hyper-spheres. On arrival of new points from a sliding window, the function checks if they “collide” with existing objects. If they do, update the border of each object; if they do not, mark them as outlier candidates and decide whether to treat these points as outlier and discard them all after data in next window are all processed. Objects originated from few points are also marked as candidates to discard. When using grid, the algorithm hashes every grid to improve storage efficiency, which is also widely used. A spherical bounding volume hierarchy tree used by “SphereDetect” also supports merging spheres based on the principle of the tree. Actually, the two functions are based on clustering, but with higher space partition efficiency. Precise information on space coordinates of past points are discarded, thus calculation efficiency and memory consumption are improved.

Based on concept of neighbor-based detection, IoTDQ also provides “TwoSidedFilter”, which filters concept drifts by calculating degree of outlier in a sliding window. There is a usage exhibited in Section 3.1.

Another special type of point anomaly in IoT time series is timestamp error. Section 2.1 has discussed about these errors, but it is unnecessary to consider all kind of timestamp errors. In some cases, where missing points are most frequently occurred errors, detection could be much easier. IoTDQ provides “MissDetect”, a stream algorithm for this situation. “MissDetect” conducts linear regression on timestamps, and pops out anomaly if regression coefficient goes below a threshold when receiving a new point. This method avoids calculating a standard timestamp interval, which means it reduces time complexity comparing to “Completeness”. Remember that overlap or latency may still occur in this situation, so calculating a standard interval still costs.

Subsequence anomaly detection can be treated as detection on multivariate data. Based on this transformation, IoTDQ adopts LOF, a classical algorithm for multivariate data. We may input a multivariate time series to conduct anomaly detection based on density, or either treat continuous data in a sliding window as a multivariate point in a univariate time series.

2.4 Data repairing

To make time series suitable for downstream analysis, for example, machine learning, we develop data repairing functions. These functions try to minimize data quality problems while reserving original information as much as possible. For example, we utilize SCREEN[31] to deal with validity errors on speed. SCREEN is a repair algorithm based on dynamic programming with value restrictions to obtain minimum value change designed for univariate data. We modified it to repair data window by window for less time cost. To further improve performance on large scale of data, Zhang et al.[32] proposed LsGreedy, which migrates to greedy algorithm. UDF ValueRepair provides both algorithms.

Another type of IoT error occurs in timestamps, as mentioned in Section 2.1. Since most downstream processing algorithms assume all data are collected with equal time intervals, it is necessary to deal with them in advance. If the errors can be inducted into data quality problems mentioned before, user may choose “TimestampRepair” to generate ideal series. Otherwise, sampling, interpolation, or regression UDFs, such as “Sample”, “Spline”, and “Seasonal AutoRegressive Integrated Moving Average” (SARIMA) are available.

Based on anomaly detection functions “GridDetect” and “SphereDetect”, there are also corresponding methods “GridRepair” and “SphereRepair”, which repair data to the nearest border of grid or hyper-sphere. By the time when this paper is written, Apache IoTDB has not provided UDF for multivariate output. Thus the corresponding methods are temporarily not implemented in IoTDQ. This leaves an aim in the future roadmap.

In IoTDQ, repairing function also includes missing value imputation. Imputation methods which rely on restrictions from multiple series and consume larger computing resources are not widely used in industrial scenarios. At present, IoTDQ provides imputation and repair based on regression models. in Section 3.2, there is an example on how to use data quality analysis in reality.

2.5 Frequency domain analysis and time series forecast

These two classes integrate renowned functional supplement for Apache IoTDB. Frequency domain analysis is a common demand of those who use Apache IoTDB to store electric signals. It also refers to the question of data representation. Since most IoT sensors detect electric signals directly or indirectly, approaches on electric signals should also be applied on IoT time series. There are multiple frequently-used frequency domain analysis functions, including discrete Fast Fourier Transformation (FFT), Discrete Wavelet Transformation (DWT), filters, convolutions, and Short-Time Fourier Transformation (STFT). Frequency domain analysis is also helpful for periodic analysis. DWT and many other transformations are also widely used in denoising.

You may find out that there are also functions prepared for classic time series fitting and prediction models. These functions are also concluded into data profiling class, because parameters or coefficients of regression and fitting models can also be treated as characteristic of given time series. For example, SARIMA model contains four coefficients to describe seasonal changes, auto-regression level, moving average level, and stationary order.

In order to make it convenient for users who are used to use Matlab or R functions to fit a time series model, as well as to modify Apache IoTDB in order to make it qualified for multi-sourced data besides sole IoT data, IoTDQ provides these classical models available for forecast. These renowned functions are time-tested and not resource-consuming. They would satisfy most elementary analysis demands. The most significant advantage of utilizing Apache IoTDB UDFs is that fitting queries can be executed on the device where the data are stored with less necessary software. The minimal configurations make it easier to finish data modeling on production environments, where operators do not expect to install professional data analysis kits. This also avoids extra data Extraction, Transformation, and Loading (ETL).

2.6 Data matching

Functions in this class focus on similarity measurement of time series. As we all know, time series data requires special metrics to measure similarity. Our approach on common measuring functions, which construct the data matching module, makes it convenient to query similar series on an IoTDB user client. In industry scenarios, subsequences matching is used for both fault detection and forecast. Comparing recent data to a set of data composed of historical error to alarm is a typical method on system monitoring.

3 Case Study

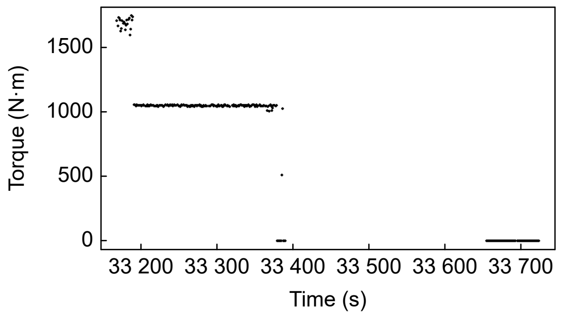
In this section, we present some realistic applications of IoTDQ. These examples come from Apache IoTDB users. We may have a glimpse of users’demands in these cases, where some unimpressive functions play a role.

3.1 Equipment sustainability analysis

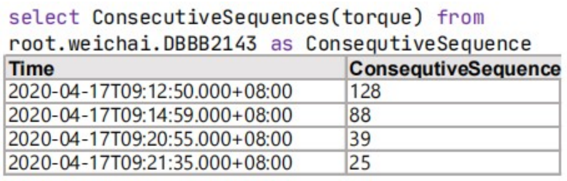
In this example, a vehicle torque sensor records data when the equipment is on. The equipment is available 24 hours a day, but only runs when there is staff working on it. We may want to evaluate the total running time of the equipment with sensor records, in order to get informed when the lifespan of the equipment comes to an end.

A simple way is to compute the total sustaining time of all periods with continuous data points. However, IoT sensors are not as stable as we expect. In this case, the sensor creates noise signals very soon after the equipment shuts down, when the power is already off. Figure 4 shows a typical series collected within one day.

To make a precise evaluation, we may first use



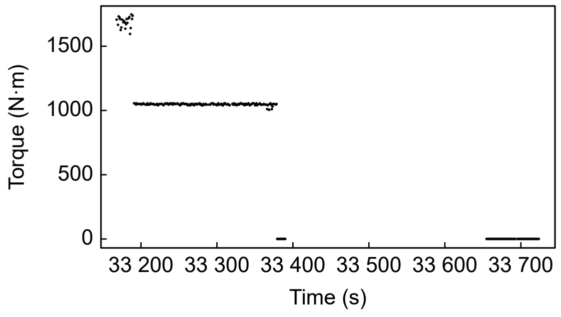
(a) Original series



```
select ConsecutiveSequences(torque) from
root.weichai.DBBB2143 as ConsequtiveSequence
```

Time	ConsequtiveSequence
2020-04-17T09:12:50.000+08:00	128
2020-04-17T09:14:59.000+08:00	88
2020-04-17T09:20:55.000+08:00	39
2020-04-17T09:21:35.000+08:00	25

(b) Power-off noise excluded series



(c) Cleaned series with two-sided window filter

Fig. 4 Equipment sustainability analysis on torque.

"ConsequtiveSequences" to filter noise points during the power-off period, as shown in Fig. 4b. The next step is to locate the accurate shutting-down moment at the end of consecutive sequences. We select 1st order difference of timestamp, and then apply "KSigma" to locate the shutting-down periods, which lead to a long timestamp interval recognized by *k*-sigma anomaly detection rule. "KSigma" can be replaced with MAD along with "Range", corresponding to detection rules of *k* times of MAD deviation from median. IQR is also a choice by detecting outliers lying over 1.5 times IQR deviation from upper and lower quartiles.

During the shutting-down period, data collected from the sensor jump to zero, and then bounce back shortly. We may use "TwoSidedFilter" to implement a two-sided window detection, which creates two windows before and after a specific point, and judge if it could be removed to smoothen the original series. If the answer is positive, it indicates that the specific point is an anomaly during the shutting-down period, and thus should not be counted into equipment running time. (see Fig. 4c)

3.2 Sensor stability analysis

In Section 2.1, we introduce 4 indices of data quality and the classification of data quality problems. In this industrial occasion, a few time series are generated from a wind power plant. We apply "Completeness", "Consistency", Validity", and "Timeliness" to every device, and group them by month or day, then we can have an overall view of the sensor stability. Furthermore, we can examine questionable series to locate concrete sensor errors. Figure 5 shows an example of data quality indices computing and visualization of data quality across the year. Advanced analysis shows that a few sensors are not connected correctly to IoT. "TimestampRepair" and "ValueRepair" will help if we want to fix the problems.

3.3 Fault monitoring

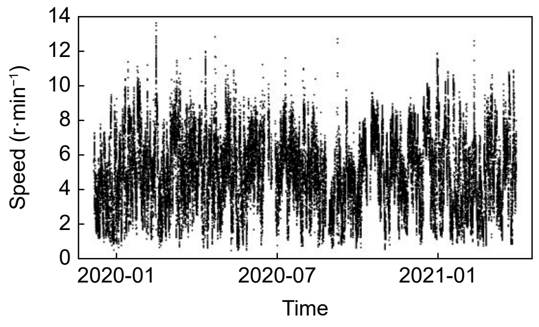
In most cases, fault detection is the primal demand in time series analysis of industry. A simple way to monitor the correctness of a system is to preset a threshold. When the sensor record exceeds the threshold, the system triggers the alarm. "Range" is a simple implementation for this situation.

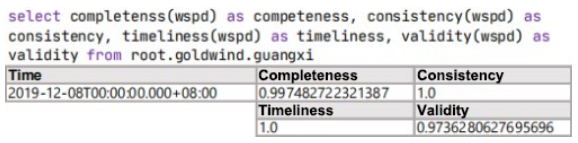
For some situations, like working temperature or voltage monitoring, it is easy to preset a threshold. But sometimes technicians do not really grasp priori knowledge on threshold. Under strict Gaussian distribution, anomalies are distributed over *k* times of standard deviation departure from mean, and this leads to *k*-sigma anomaly detection rule (usually *k* is 3 or above). Website maintainers, however, may expect Zipf distribution[33]. In general cases, it is hard to conclude a standard distribution for anomaly detection. Hence we recommend MAD to solve the problem. It has been found that median absolute deviation fluctuates in a narrower range in comparison to standard deviation, thus performs better in outlier detection. Figure 6a shows an example.

In some special cases, like stock price, or in other scenarios where data appear to be autoregressive, users can apply classical time series models like ARMA (Box-Jenkins)[5]. Method based on model regression is also a feasible way to detect sudden changes. Figure 6b is an example on weather data, which shows periodic changes across the year.

4 Experiment

In this section, we compare computations on massive industrial time series data with IoTDQ UDFs with that use other libraries. As mentioned in Section 1, Apache

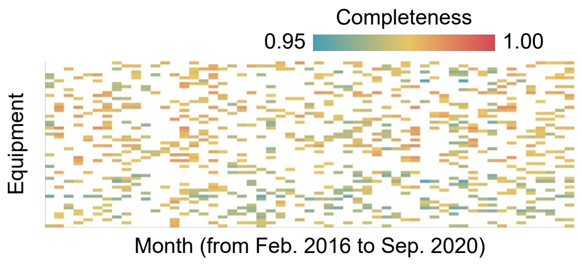




```
select completenss(wspd) as competeness, consistency(wspd) as
consistency, timeliness(wspd) as timeliness, validity(wspd) as
validity from root.goldwind.guangxi
```

Time	Completeness	Consistency
2019-12-08T00:00:00.000+08:00	0.997482722321387	1.0
	Timeliness	**Validity**
	1.0	0.9736280627695696

(a) Example of data quality computation in sliding windows



(b) Visualization of data quality by sensor by month

Fig. 5 Sensor stability analysis from a wind power plant.

IoTDB UDF boasts close connection to data storage engine, I/O consumption is greatly decreased. Based on different usages, we divide our experiment into two parts. The first part focuses on most widely-used data profiling functions with possible python substitutions. In Section 4.1, we compare the process efficiency between common data profiling tasks on IoTDQ and Apache IoTDB Python interface. Further in Section 4.2, we carefully choose some originally designed UDF functions in IoTDQ and provide same independent versions in Java which connect Apache IoTDB Java interface. It will be more persuasive to reveal the advantage of utilizing Apache IoTDB UDF.

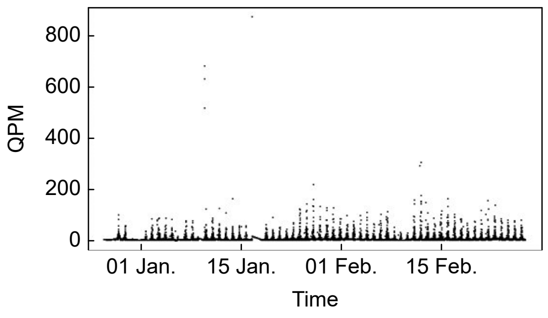
4.1 Common data profiling functions with IoTDQ and python interface

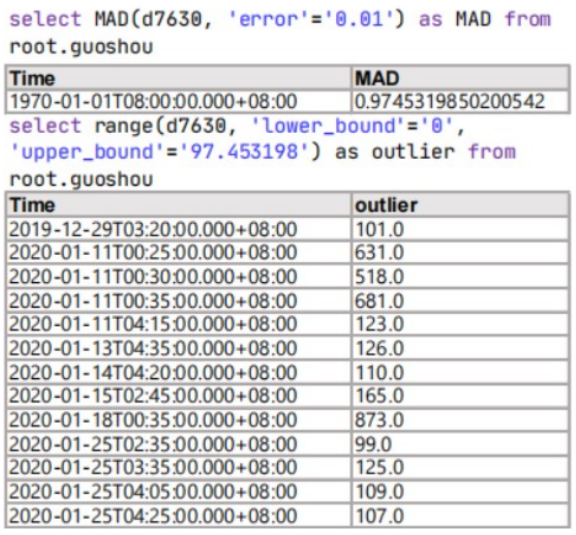
Without using a platform offered by database, users may prefer to adopt Python libraries for data profiling. IoTDQ data profiling section provides calculating standard deviation, counting distinct values, generating histogram, calculating median, calculating percentile number, calculating quantile number, and calculating mean absolute deviation. Most UDFs provided here use a stream calculation algorithm, making it fairer for comparison.

The experiments are run on a machine with Intel Core 8 CPU (2.3 GHz) and 16 GB of memory, with Apache IoTDB v1.0.1 installed. Time consumption results are listed in Table 3.

In our test, we generate 10 time series with 1 million data points each. The time costs listed in Table 3 are the values of algorithmic mean on time costs of each series. According to convention, “np” is abbreviation for library Numpy, and “ df ” is for library Pandas.DataFrame.

As seen Table 3, comparing to using Python interface, IoTDQ greatly saves read time.





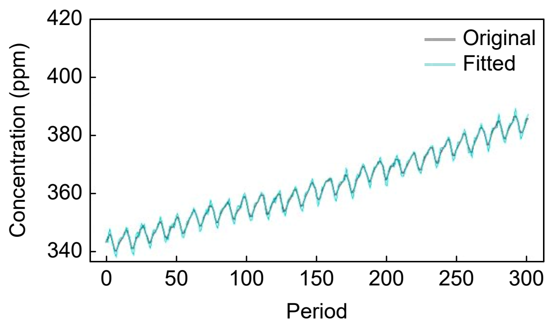
```
select MAD(d7630, 'error'='0.01') as MAD from
root.guoshou
```

Time	MAD
1970-01-01T08:00:00.000+08:00	0.9745319850200542

```
select range(d7630, 'lower_bound'='0',
'upper_bound'='97.453198') as outlier from
root.guoshou
```

Time	outlier
2019-12-29T03:20:00.000+08:00	101.0
2020-01-11T00:25:00.000+08:00	631.0
2020-01-11T00:30:00.000+08:00	518.0
2020-01-11T00:35:00.000+08:00	681.0
2020-01-11T04:15:00.000+08:00	123.0
2020-01-13T04:35:00.000+08:00	126.0
2020-01-14T04:20:00.000+08:00	110.0
2020-01-15T02:45:00.000+08:00	165.0
2020-01-18T00:35:00.000+08:00	873.0
2020-01-25T02:35:00.000+08:00	99.0
2020-01-25T03:35:00.000+08:00	125.0
2020-01-25T04:05:00.000+08:00	109.0
2020-01-25T04:25:00.000+08:00	107.0

(a) Anomaly detection based on *k*-MAD rule



(b) Fault monitoring with forecast methods

Fig. 6 Fault detection in time series. ppm represents parts per million.

Table 3 Time cost comparison between IoTDQ and Python.

IoTDQ		Python				
IoTDQ UDF	Total time cost (s)	Python function	Total time cost (s)	Reading time (s)	Function time (s)	Writing time (s)
Stddev	0.1877	np.std	1.0410	1.0148	0.0016	0.0245
Distinct	0.4402	np.unique	0.8818	0.8128	0.0496	0.0194
Histogram	0.1899	np.histogram	0.8715	0.8494	0.0107	0.0114
Median	0.2302	np.median	0.7584	0.7406	0.0076	0.0101
Percentile	0.3386	np.percentile	0.7484	0.7241	0.0126	0.0117
Quantile	0.0064	np.quantile	0.7447	0.7236	0.0125	0.0086
MAD	0.2091	df.mad	1.0541	1.0382	0.0074	0.0084

Theoretically, read time with Python interface keeps same for different functions, since all experiments read same data from same database. Fluctuation in reality may be caused by memory allocation or response from database server. All functions included in the experiment are aggregation functions, which means theoretically they should also share same write time. If we take similar causes of fluctuation into consideration, this is reflected in the result.

The comparison reasonably reveals the advantage of IoTDQ on time cost. By consuming and discarding time series data directly and immediately, IoTDQ is more competent for real-time computation tasks in IoT.

4.2 Original IoTDQ functions with IoTDQ and Java interface

In other sections, however, original designed algorithms are provided, which means it is impossible to find a control functions group. Thus we modified IoTDQ UDFs to independent Java versions to test the difference with Java platform.

The experiments are run on a machine with Intel Core 8 CPU (2.3 GHz) and 16 GB of memory, with Apache IoTDB v1.0.1 installed. Time consumption results are listed in Table 4. We generate another 10 time series with 1 million data points each, with data quality problems such as missing data and outliers. The time costs listed in Table 4 are the values of algorithmic mean on time cost of each series.

Apache IoTDB is fully developed on Java, so reading data transports faster with Java interface than with Python. Algorithms and environments are almost completely same in the comparison. One difference here is that UDFs run on a Java Virtual Machine (JVM) created by Apache IoTDB, while Java functions run on another one. When testing ordinary anomaly detection functions, both methods show similar behaviors. While for some functions, too many errors are detected and a vast number of writing requests are made, causing a rapid increase on time cost. The results show that UDF provides better stability for IoT computation.

Table 4 Time cost comparison between IoTDQ and Java.

Function	UDF time cost (s)	Java time cost (s)
Completeness	0.5203	1.0222
Consistency	0.5178	1.0285
Timeliness	0.5522	1.0802
Validity	0.6804	1.0157
KSigma	0.4794	0.5574
IQR	0.2551	0.1854
LOF	0.2343	0.1673
Range	0.3640	0.2505
TwoSidedFilter	1.0289	30.6036
MissDetect	1.0350	78.2877
TimestampRepair	2.1743	93.5875
ValueRepair	1.8023	97.0783

5 Related Work

Our work is related to almost all aspects of problems in time series analysis, and there are lots of novel progress each year. The major idea of this paper inherits from IoT time series diagnosis, so in this section we will cover most relevant works in this field. Without Apache IoTDB, researchers tried to construct a diagnosis system with full pipeline, from data storage backend to Graphical User Interface (GUI). Huang et al.[34] designed TsOutlier to implement outlier detection in sliding windows of IoT time series. TsOutlier realizes detection based on distribution and model regression. Computation is realized with Apache Spark and Apache Flink. The GUI could exhibit a digest of series and data quality distribution across all windows. Liu et al.[35] upgraded TsOutlier to TsClean, adding Apache IoTDB into the data source engine and adopting novel anomaly detection and repair algorithms into the system. The GUI provided by TsClean was also enhanced with detailed figures based on vue.js, a Javascript-based framework for front end design.

The overall data quality diagnosis methods in IoT time series are introduced in Section 2.1.

6 Conclusion

In this paper, we introduce IoTDQ, a UDF library designed for Apache IoTDB. This library not only provides fundamental statistical computation, but also focuses on realistic industrial requirements. IoTDQ covers functions from anomaly detection to frequency domain analysis. Characteristic data quality functions make a standard for time series quality evaluation. IoTDQ could successfully solve primal industrial issues, and also builds a bridge for downstream data analysis.

References

[1] X. Huang, J. Wang, R. K. Wong, J. Zhang, and C. Wang, PISA: An index for aggregating big time series data, in

Proc. 25th ACM Int. Conf. Information and Knowledge Management, Indianapolis, IN, USA, 2016, pp. 979–988.

[2] J. Qiao, X. Huang, J. Wang, and R. K. Wong, Dual-PISA: An index for aggregation operations on time series data, *Inf. Syst.*, vol. 87, p. 101427, 2020.

[3] Apache Software Foundation, Apache IoTDB, http://iotdb.apache.org/, 2022.

[4] C. Wang, J. Qiao, X. Huang, S. Song, H. Hou, T. Jiang, L. Rui, J. Wang, and J. Sun, Apache IoTDB: A time series database for IoT applications, *Proc. ACM Manag. Data*, vol. 1, no. 2, p. 195, 2023.

[5] G. Box and G. M. Jenkins, Time series analysis forecasting and control, Journal of Time Series Analysis, doi: 10.2307/1912100, 1970.

[6] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin, FEDformer: Frequency enhanced decomposed transformer for long-term series forecasting, in *Proc. 39th Int. Conf. Machine Learning*, Baltimore, MD, USA, 2022, pp. 27268–27286.

[7] W. Chen, W. Wang, B. Peng, Q. Wen, T. Zhou, and L. Sun, Learning to rotate: Quaternion transformer for complicated periodical time series forecasting, in *Proc. 28th ACM SIGKDD Conf. Knowledge Discovery and Data Mining*, Washington, DC, USA, 2022, pp. 146–156.

[8] M. Schirmer, M. Eltayeb, S. Lessmann, and M. Rudolph, Modeling irregular time series with continuous recurrent units, in *Proc. 39th Int. Conf. Machine Learning*, Baltimore, MD, USA, 2022, pp. 19388–19405.

[9] Z. Shao, Z. Zhang, F. Wang, and Y. Xu, Pre-training enhanced spatial-temporal graph neural network for multivariate time series forecasting, in *Proc. 28th ACM SIGKDD Conf. Knowledge Discovery and Data Mining*, Washington, DC, USA, 2022, pp. 1567–1577.

[10] T. Kolajo, O. Daramola, and A. Adebiyi, Big data stream analysis: A systematic literature review, *J. Big Data*, vol. 6, no. 1, p. 47, 2019.

[11] Apache software foundation, IoTDQ, https://incubator.apache.org/ip-clearance/iotdb-udf-library.html, 2022.

[12] Apache software foundation, IoTDQ code, https://github.com/apache/iotdb/tree/master/library-udf, 2022

[13] Apache software foundation, IoTDQ document, https://iotdb.apache.org/UserGuide/Master/UDF-Library/Quick-Start.html, 2022.

[14] P. Esling and C. Agón, Time-series data mining, *ACM Comput. Surv.*, vol. 45, no. 1, p. 12, 2012.

[15] Y. W. Lee, L. L. Pipino, J. D. Funk, and R. Y. Wang, *Journey to Data Quality*. Cambridge, MA, USA: MIT Press, 2006.

[16] ISO 8000-1:2022 data quality - part 1: Overview. https://www.iso.org/standard/81745.html, 2022.

[17] S. Song and A. Zhang, IoT data quality, in *Proc. 29th ACM Int. Conf. Information and Knowledge Management*, Virtual Event, 2020, pp. 3517–3518.

[18] C. Fang, S. Song, and Y. Mei, On repairing timestamps for regular interval time series, *Proc. VLDB Endow.*, vol. 15, no. 9, pp. 1848–1860, 2022.

[19] Apache software foundation, Apache DataFu spark, https://datafu.apache.org/docs/spark/getting-started.html, 2022.

[20] Apache software foundation, Apache DataFu pig, https://datafu.apache.org/docs/datafu/getting-started.html, 2022.

[21] Z. Chen, S. Song, Z. Wei, J. Fang, and J. Long, Approximating median absolute deviation with bounded error, *Proc. VLDB Endow.*, vol. 14, no. 11, pp. 2114–2126, 2021.

[22] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri, Medians and beyond: New aggregation techniques for sensor networks, in *Proc. 2nd Int. Conf. Embedded Networked Sensor Systems*, Baltimore, MD, USA, 2004, pp. 239–249.

[23] C. Leys, C. Ley, O. Klein, P. Bernard, and L. Licata, Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median, *J. Experim. Soc. Psychol.*, vol. 49, no. 4, pp. 764–766, 2013.

[24] M. Ester, H. P. Kriegel, J. Sander, and X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in *Proc. 2nd Int. Conf. Knowledge Discovery and Data Mining*, Portland, OR, USA, 1996, pp. 226–231.

[25] D. Yang, E. A. Rundensteiner, and M. O. Ward, Neighbor-based pattern detection for windows over streaming data, in *Proc. 12th Int. Conf. Extending Database Technology*: *Advances in Database Technology*, Saint Petersburg, Russia, 2009, pp. 529–540.

[26] M. Kontaki, A. Gounaris, A. N. Papadopoulos, K. Tsichlas, and Y. Manolopoulos, Continuous monitoring of distance-based outliers over data streams, in *Proc. 27th Int. Conf. Data Engineering*, Hannover, Germany, 2011, pp. 135–146.

[27] L. Cao, D. Yang, Q. Wang, Y. Yu, J. Wang, and E. A. Rundensteiner, Scalable distance-based outlier detection over high-volume data streams, in *Proc. 2014 IEEE 30th Int. Conf. on Data Engineering*, Chicago, IL, USA, 2014, pp. 76–87.

[28] S. Yoon, J. G. Lee, and B. S. Lee, NETS: Extremely fast outlier detection from a data stream via set-based processing, *Proc. VLDB Endow.*, vol. 12, no. 11, pp. 1303–1315, 2019.

[29] L. Tran, M. Y. Mun, and C. Shahabi, Real-time distance-based outlier detection in data streams, *Proc. VLDB Endow.*, vol. 14, no. 2, pp. 141–153, 2020.

[30] S. Song, C. Li, and X. Zhang, Turn waste into wealth: On simultaneous clustering and cleaning over dirty data, in *Proc. 21st ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, Sydney, Australia, 2015, pp. 1115–1124.

[31] S. Song, A. Zhang, J. Wang, and P. S. Yu, SCREEN: Stream data cleaning under speed constraints, in *Proc. 2015 ACM SIGMOD Int. Conf. Management of Data*, Melbourne, Australia, 2015, pp. 827–841.

[32] A. Zhang, S. Song, and J. Wang, Sequential data cleaning: A statistical approach, in *Proc. 2016 Int. Conf.*

Management of Data, San Francisco, CA, USA, 2016, pp. 909–924.

[33] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, Web caching and Zipf-like distributions: Evidence and implications, in *Proc. IEEE INFOCOM '99. Conf. Computer Communications. Eighteenth Annu. Joint Conf. IEEE Computer and Communications Societies*, New York, NY, USA, 1999, pp. 126–134.

[34] R. Huang, Z. Chen, Z. Liu, S. Song, and J. Wang, TsOutlier: Explaining outliers with uniform profiles over IoT data, in *Proc. 2019 IEEE Int. Conf. Big Data*, Los Angeles, CA, USA, 2019, pp. 2024–2027.

[35] Z. Liu, Y. Zhang, R. Huang, Z. Chen, S. Song, and J. Wang, EXPERIENCE: Algorithms and case study for explaining repairs with uniform profiles over IoT data, *J. Data Inf. Qual.*, vol. 13, no. 3, p. 18, 2021.

Pengyu Chen recieved the BEng degree from Tsinghua University, China in 2020. He is a master student at School of Software, Tsinghua University, China. His main research interest is industrial big data management.

Wendi He received the BEng degree from Shanghai Jiao Tong University, China in 2021. He is currently a master student at School of Software, Tsinghua University, China. His main research interest is IIoT data management.

Chen Wang received the BEng and MEng degrees from Fudan University, China in 2003 and 2006, respectively. He is currently the chief engineer at National Engineering Research Center for Big Data Software (NERCBDS), Tsinghua University, China. His research interests include database technology, stream computing, and big data systems.

Wenxuan Ma recieved the BEng degree from Tsinghua University, China in 2022. He is currently a master student at School of Software, Tsinghua University, China. His main research interest is IoT data management.

Xiangdong Huang received the BEng degree from Chongqing University, China in 2012, and the PhD degree from Tsinghua University, China in 2017. He is currently an associate professor at School of Software, Tsinghua University, China. His research interests include big data storage system and time series data management.