Lebanese American University Repository (LAUR)

Post-print version/Author Accepted Manuscript

Publication metadata

Title: Data redundancy management for leaf-edges in connected environments

Author(s): Elio Mansour, Faisal Shahzad, Joe Tekli & Richard Chbeir

Journal: Computing

DOI/Link: https://doi.org/10.1007/s00607-021-01051-4

How to cite this post-print from LAUR:

Mansour, E., Shahzad, F., Tekli, J., & Chbeir, R. (2022). Data redundancy management for leaf-edges in connected environments. *Computing*, Doi: 10.1007/s00607-021-01051-4, http://hdl.handle.net/10725/15987

© Year 2022

This Open Access post-print is licensed under a Creative Commons Attribution-Non Commercial-No Derivatives (CC-BY-NC-ND 4.0)

This paper is posted at LAU Repository

For more information, please contact: archives@lau.edu.lb

Manuscript (inclusive Title Page)
Click here to access/download;Manuscript (inclusive Title Page);DRMF.pdf
Click here to view linked References

**Noname manuscript No.**
(will be inserted by the editor)

# Data Redundancy Management Framework for Connected Environments

**Elio Mansour · Faisal Shahzad · Joe Tekli · Richard Chbeir**

Received: date / Accepted: date

**Abstract** Major advances in the fields of Internet and Communication Technology (ICT), data modeling/processing, and sensing technology have rendered traditional environments (e.g., cities, buildings) more connected. Although the sensed data could be useful for various applications (e.g., event detection in cities, energy management in commercial buildings), it first requires pre-processing to clean various inconsistencies (e.g., anomalies, redundancies, missing values). In this work, we focus on managing data redundancies in connected environments. Existing approaches suffer from (i) disregarding edge data redundancies either at the edge or at the core of the network; (ii) disregarding sensor mobility and the dynamicity of the network; (iii) disregarding the limited resources of edge devices; (iv) disregarding network/infrastructure resources; and (v) disregarding data consumer needs/requirements when cleaning the data redundancies. To address these limitations, we propose here a new Data Redundancy Management Framework (DRMF) allowing to identify and remove data redundancies in connected environments at the device level. DRMF considers both static and mobile edge devices, and provides two algorithms for temporal and spatio-temporal redundancy detection. Once redundancies are identied, DRMF performs data deduplication taking into account the dynamic requirements of data consumers and device resources (e.g., processing, battery, memory). Experimental results highlight the performance and accuracy of our solution in detecting and eliminating edge data redundancies.

**Keywords** Connected Environments · Sensor Networks · Internet of Things (IoT) · Data Redundancy · Data Cleaning

E. Mansour, F. Shahzad, R. Chbeir
Univ. Pau & Pays Adour, E2S UPPA, LIUPPA, Anglet, 64600, France
E-mail: firstname.lastname@univ-pau.fr

J. Tekli
Lebanese American University, E.C.E Dept. 36 Byblos, Lebanon
E-mail: firstname.lastname@lau.edu.lb

## 1 Introduction

Recent advances in the fields of Internet and Communication Technology (ICT), sensing technology, sensor networks, and data processing have allowed traditional environments to become more and more connected [6]. These connected environments are typically defined as physical infrastructures (e.g., buildings, cities, grids) equipped with a sensor network capable of providing various recordings and observations from the real world. As a result, the aforementioned environments generate huge amounts of heterogeneous data that are useful for various data consumers (e.g., users, data stores, data processing services, other network devices) and high-level applications (e.g., traffic event detection in a city [17], energy management in a building [14], electrical storage in a smart grid [5]). Although the sensed data contains useful and valuable information, pre-processing is needed in most cases since the observations are in raw form and often suffer from various inconsistencies [8, 18] (e.g., redundancies, anomalies, and missing values). In this work, we focus on managing data redundancies in connected environments. Identifying and cleaning unnecessary data duplicates are beneficial for (i) querying edge data efficiently from edge devices and/or the network's database; (ii) querying spatial-temporal data efficiently from mobile edge devices; (iii) conserving the limited resources of edge devices (e.g., battery, memory, bandwidth); (iv) conserving the network's shared resources (e.g., central database, bandwidth) ; and (v) providing ready-to-use data collections for consumers based on their needs/requirements. Existing works (e.g., [1,4,7,10–13,16,19]) have targeted data redundancy in connected environments, however most of them suffer from the following limitations:

1. *Disregarding edge data redundancies at the edge/core:* one should be able to query the edge data from devices directly or from the network core (i.e., central database). Therefore, it is important to handle redundancies at the device level to improve querying data from the edge, and to prevent redundancies from reaching the core for improved database querying.
2. *Disregarding environment dynamicity:* dynamic environments include mobile devices/sensors in addition to static nodes. Considering mobile devices allows the detection of new redundancies generated by device mobility.
3. *Disregarding the limited resources of edge devices:* devices at the edge of the network often have limited resources (e.g., processing, memory, and power). Moreover, devices need to exchange data as well as push data to the core. Therefore, it is important not to deplete device resources by excessive communications and heavy processing of redundant data.
4. *Disregarding the network's shared resources:* the network provides several shared resources for all nodes (e.g., communication channels, storage repositories). Therefore, it is important not to deplete the network's resources by consuming unnecessary bandwidth and storage space for redundant data.
5. *Disregarding data consumer needs:* different data consumers (e.g., users, services, databases, devices) have different requirements for the requested data (e.g., specific deduplication ratio, data size, resource consumption

constraints for deduplication). Therefore, it is important to consider the aforementioned needs when removing redundancies to provide ready-to-use data sets for these consumers.

In this paper, we propose a new Data Redundancy Management Framework (DRMF) to identify and remove data redundancies in connected environments at the device level. DRMF considers both static and mobile sensing devices when identifying/detecting redundancies within the generated sensor observations. It provides two algorithms for redundancy detection: the first relies on the temporal feature (specifically designed for static/immobile devices), and the second relies on both temporal and spatial features (specifically designed for mobile devices). The algorithms' parameters are automatically tuned based on historical device data, in order to identify relevant redundancy partitions (e.g., groups of similar-enough data that are considered duplicates). Once redundancies are identified, DRMF proposes a data deduplication module that takes into account the requirements of data consumers, edge data redundancies at the edge and core of the network, the network dynamicity and device mobility, device/network resources (e.g., processing, battery, memory, bandwidth), and personalized ready-to-use data sets for data consumers. A summary description of DRMF was given in [13]. This paper adds: (i) an extended motivating scenario that addresses redundancy identification and cleaning; (ii) an extended discussion and evaluation of the existing approaches; (iii) a parameter tuning process for the redundancy identification algorithms; (iv) a data consumer-centric deduplication process; and (v) an extensive experimentation and evaluation of the performance and accuracy of our proposal.

The remainder of the paper is organized as follows. Section 2 presents a motivating scenario that highlights the needs and challenges of this work. Section 3 reviews existing redundancy management approaches. Then, Section 4 details our proposal. Section 5 describes the experimentation and evaluates the performance and accuracy of DRMF. Finally, Section 6 concludes this work and discusses future research directions.

## 2 Motivating Scenario

Consider the smart parking connected environment presented in Figure 1. Although this example does not summarize all data redundancy management issues in connected environments, we use it to highlight the motivation, specific needs, and challenges of this work. The smart parking is equipped with a variety of devices, some static (e.g., deployed in individual parking spots) and others mobile (e.g., smart phones, connected vehicles). Devices could embed one or more sensors, each sensing a specific observation. For the sake of brevity, we consider three observations in this example: (i) temperature for heating/air conditioning management; (ii) air quality for ventilation and pollution management; and (iii) occupancy for free parking spot detection. The aforementioned edge devices are also equipped with a local memory for temporary sensed data storage, a processor for query answering, and a network

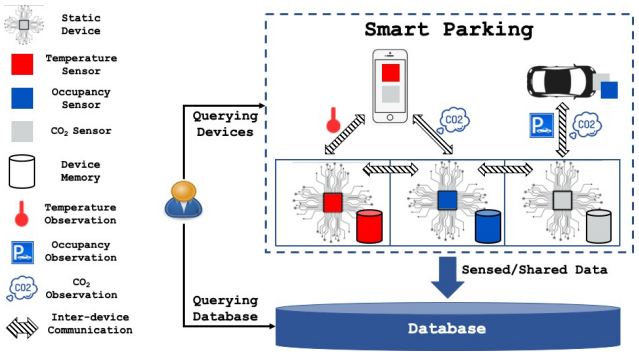



Fig. 1: The smart parking

interface for communicating with data consumers (e.g., database, users, services, other devices). All devices push data to the central database, and all data consumers can retrieve edge data directly from either devices or the database. The parking manager (i.e., user) needs to monitor the environment and handle specific events (e.g., overheating, pollution) using a set of resources (e.g., the network, data processing services, central database). To do so, he/she has the following needs:

- **Need 1.** Querying edge data from devices and the central database to monitor sensor breakdowns, anomalies, and local events. The retrieved data can be exploited by various data processing and mining services.
- **Need 2.** Querying edge data from mobile devices to handle the environment dynamicity, while considering mobility and spatial-temporal features.
- **Need 3.** Conserving the limited edge devices' resources to minimize unnecessary data processing, exchange, and local storage.
- **Need 4.** Conserving the network's shared resources to maintain the sustainability of the network by relieving communication channels and the central database from unnecessary data communication and storage.
- **Need 5.** Considering data consumer needs during deduplication, e.g.,:
  - *User needs:* the user might require a high deduplication ratio if he/she doesn't have enough resources to process huge amounts of data.
  - *Database needs:* the data storage strategy might require a specific deduplication ratio to store a specific amount of data from each device or zone of the network (e.g., 1 MB per device per day).
  - *Device needs:* the data requested by devices might require a specific deduplication in order to comply with resource-related limitations (e.g., remaining battery, remaining memory, processing load).
  - *Service needs:* different services might have different requirements for deduplication (e.g., services that provide a statistical overview of the data might benefit from recurring repetitions in the data allowing to detect certain behavioral patterns, while information retrieval or data indexing services might require full deduplication).

As a result, static and mobile sensors in the smart parking environment will be producing large amounts of data, exchanging some of them among each other, and sending them periodically to the database even if no significant changes occur in the sensed data. For instance, if a vehicle is parked for hours in the same spot, redundant occupancy data is sent during the whole occupancy time period. Similarly, redundant air quality and temperature data will be periodically produced, exchanged, and stored even when the parking is not witnessing any activity (e.g., car movements, people entering). Therefore, a redundancy identification and cleaning framework is required to cover the aforementioned needs, and the following challenges need to be addressed:

- **Challenge 1.** How to identify and clean redundancies at the edge and how to prevent them from reaching the network's database (cf. Need 1)?
- **Challenge 2.** How to consider spatial-temporal features when identifying redundancies to cope with dynamic device mobility (cf. Need 2)?
- **Challenge 3.** How to prevent unnecessary data processing, exchange, and storage on devices in order to preserve their resources (cf. Need 3)?
- **Challenge 4.** How to prevent unnecessary data transmission and storage in the database to preserve the network's shared resources (cf. Need 4)?
- **Challenge 5.** How to provide a (personalized) consumer-based deduplication to provide ready-to-use data sets for consumers (cf. Need 5)?

In this work, we tackle the data redundancy problem at the device level in order to address the aforementioned challenges. Before detailing the proposal, we first present and compare existing approaches for data redundancy management in sensor networks.

## 3 Related Works

To compare existing approaches, we propose here the following criteria based on the previously discussed needs and challenges:

- **Criterion 1.** Considering edge redundancies at the edge and core: stating if the approach handles data redundancy at the source (device level) and if it prevents redundancies from reaching the database. This enables efficient querying on the edge and core of the network (cf. Need 1).
- **Criterion 2.** Considering device mobility: specifying if the approach considers dynamic redundancies due to device mobility (cf. Need 2).
- **Criterion 3.** Considering device resources: specifying if the approach considers the limited resources of edge devices when processing, storing, and exchanging data (cf. Need 3).
- **Criterion 4.** Considering network resources: indicating if the approach considers the network's shared resources when transmitting and permanently storing data (cf. Need 4).
- **Criterion 5.** Considering consumer-centric data deduplication: specifying if the approach considers data consumer needs and requirements when removing data redundancies (cf. Need 5).

In the following, we review related data redundancy management works in connected environments [1,4,7,10–13,16,19]. Then, we present a comparison of these approaches based on the aforementioned criteria.

### 3.1 Existing Approaches

In [4], the authors focus on the spatial distribution of sensors in the environment, and how it can be managed in order to prevent redundancies. To do so, a graph of nodes and detected events is constructed from raw sensory data to identify nodes producing redundant data. Next, these so-called "redundant" nodes are either relocated or put into sleep mode using a circle packing technique to enhance coverage while minimizing energy usage during relocation. This work only handles redundancy from a sensor deployment perspective (i.e., avoiding deploying sensors that provide the same type of data in the same area). Therefore, the emphasis is on detecting redundant sensor nodes and not on the data itself. Moreover, the proposal does not consider sensor mobility (cf. Criterion 2). In [11], the authors present a data reduction scheme for Internet of Things (IoT) using data filtering and fusion. Their approach handles redundancies at the device layer before forwarding non-redundant data to sink nodes. Redundancy detection is solely based on data value deviations. Although, this work handles redundancies at the edge of the network, it does not cover redundant data from mobile devices (cf. Criterion 2). Therefore, specific spatial-temporal redundancies at device level are not handled. Moreover, redundancy cleaning does not consider data consumer needs (cf. Criterion 5). In [16], the authors address data redundancies at the core of the network using a supervised machine learning solution based on Support Vector Machine (SVM). They build an aggregation tree for the given size of the network and then apply SVM to recognize data redundancies. In this work, the authors target temporal and spatial redundancies once the data is consolidated in a central node, which provides a redundancy-free data repository that could be mined using advanced data processing techniques. However, redundancies are not handled at the device level, and data exchange between devices at the edge remains costly due to unnecessary communications. Moreover, the authors do not consider spatial-temporal redundancies generated by mobile devices (cf. Criterion 2-4). Although the central database contains redundancy-free data, the deduplication process is not customized to fit data consumer needs (cf. Criterion 5). In [19], the authors present a data deduplication technique in a healthcare-based IoT environment. They propose a Controlled Window-size based Chunking Algorithm (CWCA) to identify cut-points in sensor data distributions. The data deduplication is applied at the collector node (i.e., at the core and does not consider data consumer needs). More recently, the authors in [12] propose a data redundancy elimination technique using an unsupervised learning approach based on data clustering. The authors suggest clustering the edge nodes based on their produced sensory data in order to aggregate identical data to eliminate redundancies, before storing the data in

the cloud. However, the works in [12,19] do not consider device mobility and spatial-temporal redundancies (cf. Criterion 2). The authors in [1] propose an approach for cleansing indoor RFID data. They focus on two tasks: (i) temporal redundancy elimination; and (ii) spatial ambiguity reduction (undetermined whereabouts of an object). To detect temporal redundancy, they look for temporal intervals where values do not change. Although, this work handles data redundancies at device level, it does not consider the dynamicity of the environment and it disregards device mobility (cf. Criterion 2). Moreover, a straightforward aggregation of duplicates is done to clean redundancies without considering data consumer needs (cf. Criterion 5). In [7], the authors propose an approach that filters data at the sensor level in Periodic Sensor Networks (PSNs) using the Pearson coefficient metric. Then, another filtering process is triggered at the sink node to remove redundancies even more. However, the approach only considers static devices in order to detect redundant data (cf. Criterion 2). Moreover, the redundancy removal focuses on reducing the data size in order to minimize energy consumption in the network. Also, the deduplication cannot be customized based on individual data consumer needs (cf. Criterion 5). In [10], the authors propose a technique denoted DiDAMoK (Distributed Data Aggregation based Modified K-Means) for detecting sensor data redundancies in an IoT environment. This approach detects clusters of redundant data within a time period. However, the authors do not consider spatial-temporal redundancies produced by mobile sensors (cf. Criterion 2). Furthermore, the redundancy removal process consists of summarizing each cluster of redundancies at the device level into one representative sensor observation prior to data transmission.

### 3.2 Comparison Summary

The main characteristics of existing solutions are summarized in Table 1. To sum up, none of the aforementioned works covers all the required criteria. Some approaches focus on handling redundancies at the core of the network, thus neglecting the impact of redundancies on the edge devices where resources are often limited (e.g., power, processing, and memory). Last but not least, none of the mentioned works provides a consumer-centric data deduplication solution.

Table 1: Related Works Comparison

| **Approach** | **Criterion 1** *Edge & Core Consideration* | **Criterion 2** *Dynamicity Consideration* | **Criterion 3** *Device Resources Consideration* | **Criterion 4** *Network Resources Consideration* | **Criterion 5** *Personalized Cleaning* |
|---|---|---|---|---|---|
| Baba A.I. et al. [1] | ✓ | × | ✓ | ✓ | × |
| Chowdhury S. et al. [4] | × | × | × | × | × |
| Harb H. et al. [7] | ✓ | × | ✓ | ✓ | × |
| Idrees A.K. et al. [10] | ✓ | × | ✓ | ✓ | × |
| Ismael W.M. et al. [11] | ✓ | × | ✓ | ✓ | × |
| Li S. et al. [12] | ✓ | × | ✓ | ✓ | × |
| Patil P. et al. [16] | × | × | × | × | × |
| Ullah A. et al. [19] | × | × | × | × | × |
| **Our approach DRMF** | ✓ | ✓ | ✓ | ✓ | ✓ |

## 4 Proposal: DRMF

To address the aforementioned limitations, we extend DRMF, a new Data Redundancy Management Framework that provides: (i) static and mobile device handling; (ii) temporal and spatial-temporal data handling; (iii) flexible deduplication to consider data consumer needs; (iv) automatic tuning of the redundancy identification algorithms based on historical data; and (v) edge data deduplication considering the dynamic requirements of device resources.

The overall architecture of DRMF is depicted in Figure 2. The bottom layer, denoted Edge Device Data, represents the edge device local memory where sensor observations are stored. The second layer, denoted Device Level Redundancy Management, groups the set of modules that identify and clean redundancies in the device. More specifically, redundancy management consists of four main modules: (i) **Datatype Filtering** which separates the input data into type-based data collections; (ii) **Redundancy Detection** which detects temporal (for static devices) or spatial-temporal (for mobile devices) redundancies from each data collection; (iii) **Redundancy Detection Tuning** which auto-configures the redundancy detection algorithms based on historical data; and (iv) **Redundancy Cleaning** which deduplicates redundant data based on consumer needs and evaluates the accuracy of redundancy removal. The third layer, denoted Network Orchestrator, represents a middleware tasked with bridging data consumers (e.g., users, services, databases, devices) and producers (e.g., devices). We consider here that devices are prosumers (i.e., producers when they sense and share data, and consumers when they request data from other devices). The orchestrator allows consumers to make data consumption subscriptions in order to define the required data, deduplication perferences, and other features (e.g., data request frequency). We specifically focus on how the data consumption subscriptions are used to deduplicate data based on consumer needs. Note that we report the subscription generation protocol (covering consumer-orchestrator communication and preference exchange) for a dedicated future work. In the following subsections, we start by describing the nature of sensory data in a dynamic environment. Then, we detail the redundancy management process at device level.

### 4.1 Sensory Data

Connected environments contain diverse devices each embedding one or more sensors that provide data from the real world. Static devices are immobile, therefore the data generated by such devices could be redundant temporally. However, mobile devices produce data while moving around the environment, which potentially generates spatial-temporal redundancies. In the following, we provide a set of formal definitions that allow us to describe data items following both temporal and spatial dimensions (cf. Criterion 2).

**Definition 1 (Data Items)** We formally define a data item $d$ as a 5-tuple:

$$d : \langle a, v, t, l, s \rangle \quad where: \tag{1}$$

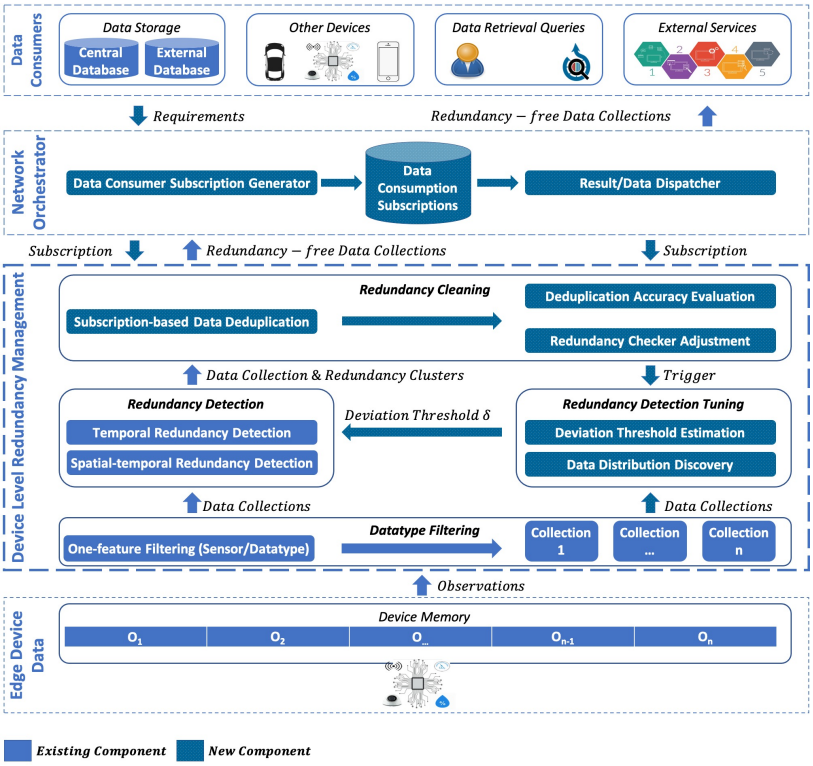



Fig. 2: The extended DRMF Architecture

- $a$ is the data attribute,
- $v$ is the data value,
- $t$ is the creation temporal stamp of $d$ (cf. Definition 2),
- $l$ is the creation location stamp of $d$ (cf. Definition 3),
- $s$ is the data source that produced/created $d$ ■

**Definition 2 (Temporal Stamp Definition)** A temporal stamp $t$ designates a single discrete temporal value formally defined as a 2-tuple:

$$t = \langle format, value \rangle \quad where: \tag{2}$$

- $format$ is a string indicating the format of the date-time value of t (e.g., "dd-MM-yyyy hh:mm:ss"),
- $value$ is the timestamp value (e.g., 10-11-2020 15:34:23 following the sample time format mentioned above) ■

**Definition 3 (Location Stamp Definition)** A location stamp $l$ is s a discrete and instantaneous location value defined as a 2-tuple:

$$l = \langle format, value \rangle \quad where: \tag{3}$$

– *format* is the location referential format following which the location stamp value will be represented (e.g., default GPS, or Cartesian, Spherical, Cylindrical),
– *value* = $\langle x, y, z \rangle$ is a discrete and instantaneous value, where x, y, and z designate individual coordinate values (the coordinates can be translated into the referential of choice following the designated format) ■

Table 2 shows an excerpt of the data produced by a device having two sensors S1, and S2 that produce $CO_2$ and temperature observations respectively.

Table 2: Sample Data Items

| a | v | t | | l | | | | s |
|---|---|---|---|---|---|---|---|---|
| | | format | value | format | value | | | |
| | | | | | x | y | z | |
| $CO_2$ | 98 | dd/mm/yyyy hh:mm:ss | 10/02/2019 10:00:00 | cartesian | 8 | 12 | 8 | S1 |
| $CO_2$ | 109 | dd/mm/yyyy hh:mm:ss | 10/02/2019 10:02:00 | cartesian | 6 | 8 | 6 | S1 |
| $CO_2$ | 110 | dd/mm/yyyy hh:mm:ss | 10/02/2019 10:04:00 | cartesian | 2 | 4 | 8 | S1 |
| $CO_2$ | 111 | dd/mm/yyyy hh:mm:ss | 10/02/2019 10:06:00 | cartesian | 4 | 6 | 4 | S1 |
| Temperature | 22 | dd/mm/yyyy hh:mm:ss | 10/02/2019 10:08:00 | cartesian | 6 | 4 | 8 | S2 |

## 4.2 Datatype Filtering

Since the device can embed various sensors, its internal memory might store different datatypes (i.e., different data attributes or features such as CO2 and temperature in Table 2). Therefore, in order to detect redundancies in the data stored locally on the device, we start by filtering the data into collections having the same attributes (or datatypes), using DRMF's filtering module. In the following subsection, we show how to detect redundancies within each data collection (cf. Figure 2). To illustrate the attribute filtering process, the data shown in Table 2 produces two distinct data collections: the first for $CO_2$ data (first four tuples - cf. Table 3); and the second for temperature data containing the last tuple.

Table 3: $\boldsymbol{CO_2}$ Data Collection

| a | v | t | | l | | | | s |
|---|---|---|---|---|---|---|---|---|
| | | format | value | format | value | | | |
| | | | | | x | y | z | |
| $CO_2$ | 98 | dd/mm/yyyy hh:mm:ss | 10/02/2019 10:00:00 | cartesian | 8 | 12 | 8 | S1 |
| $CO_2$ | 109 | dd/mm/yyyy hh:mm:ss | 10/02/2019 10:02:00 | cartesian | 6 | 8 | 6 | S1 |
| $CO_2$ | 110 | dd/mm/yyyy hh:mm:ss | 10/02/2019 10:04:00 | cartesian | 2 | 4 | 8 | S1 |
| $CO_2$ | 111 | dd/mm/yyyy hh:mm:ss | 10/02/2019 10:06:00 | cartesian | 4 | 6 | 4 | S1 |

### 4.3 Redundancy Detection & Algorithm Tuning

In this step, the redundancy checker applies our redundancy detection algorithms over the sensor's locally stored data. More specifically, the aforementioned algorithms cluster the data based on the deviation of the data item values, while also considering the temporal, or spatial-temporal spread (or coverage) of the clusters (i.e., sets of redundant data). We adopt an unsupervised cluster-based approach for two main reasons: (i) to avoid applying supervised learning which requires training time and computation power on the edge where resources are limited; and (ii) since training data for supervised learning algorithms might not be available at the device level. We provide two redundancy checking algorithms: one for temporal redundancy detection, specifically designed to handle data from static devices; and another for spatial-temporal redundancy detection, specifically designed to handle data from mobile devices. A temporal redundancy (cf. Definition 4) represents a cluster of redundant values spanning over a specific time coverage (cf. Definition 5). Similarly, a spatial-temporal redundancy (cf. Definition 6) is defined as a cluster of redundant values spanning over a specific time coverage and spatial coverage (cf. Definition 7).

**Definition 4 (Temporal Redundancy)** A temporal redundancy $tr$ is defined as a 2-tuple:

$$tr : (coverage_t, D) \ where: \tag{4}$$

- $coverage_t$ is the temporal coverage during which the data is temporally redundant
- $D = \bigcup_{j=0}^{z} d_j$ is a cluster of redundant data items where:
  - $\forall\ d_j \in D,\ d_j.t \in coverage_t.\delta_t$
  - $\forall\ d_{j1}, d_{j2} \in D,\ d_{j1}.a = d_{j2}.a$
  - $\forall\ k \in \mathbb{N}^+,\ d_k.v = d_{centroid}.v \pm \delta_v \quad where:$
    - $d_{centroid}.v$ is the centroid value of all data items in $D$
    - $\delta_v$ is an acceptable deviation threshold

*Remark 1* The threshold $\delta_v$ is calculated based on the data distribution within the redundant data set $D$. ■

**Definition 5 (Temporal Coverage Definition)** A temporal coverage $coverage_t$ is a time interval consisting of an ordered collection of temporal stamps enclosed within a start and an end stamp, describing the temporal coverage of a sensor observation (e.g., video feed) or a group of observations (e.g., scalar measurements, images). Formally, it is defined as a 2-tuple:

$$coverage_t = \langle \delta_t, g_t \rangle \quad where: \tag{5}$$

- $\delta_t = [t_s, t_e]$ is a temporal interval where:
  - $t_s < t_e$ is the start temporal stamp
  - $t_e$ is the end temporal stamp
- $g_t$ is a temporal granularity or unit of the temporal coverage (e.g., millisecond, second, minute, etc.) ■

**Definition 6 (Spatial-Temporal Redundancy)** A spatial-temporal redundancy *str* is defined as a 3-tuple:

$$str : (coverage_t, coverage_l, D) \ \ where: \tag{6}$$

- $coverage_t$ is the temporal coverage
- $coverage_l$ is the location coverage
- $D = \bigcup_{j=0}^{z} d_j$ is a cluster of redundant data where:
  - $\forall\ d_j \in D,\ d_j.t \in coverage_t.\delta_t$
  - $\forall\ d_j \in D,\ d_j.l \in coverage_l.\delta_l$
  - $\forall\ d_{j1}, d_{j2} \in D,\ d_{j1}.a = d_{j2}.a$
  - $\forall\ k \in \mathbb{N}^+,\ d_k.v = d_{centroid}.v \pm \delta_v \quad where:$
    - $d_{centroid}.v$ is the centroid value of all data items in $D$
    - $\delta_v$ is an acceptable deviation threshold ■

**Definition 7 (Location Coverage Definition)** A location coverage $coverage_l$ is the set of spatial stamps designating the surface coverage in which a sensor observation is created (e.g., area in which a video stream or a bunch of mobile measurements are recorded). Formally, it is defined as a 2-tuple:

$$coverage_l = (\delta_l, g_l) \quad where: \tag{7}$$

- $\delta_l = \langle shape, L \rangle$ defines the area of the location coverage where:
  - $L = \bigcup_{i=0}^{n} l_i \forall i \in \mathbb{N}$ is a set of location stamps
  - *shape* is a mathematical abstraction used to describe the location coverage, as a continuous coverage area (e.g., rectangle, circle), or non-continuous coverage area (e.g., disk, path, polygon, random)
- $g_l$ is the location granularity or unit of the location coverage (e.g., millimeter, centimeter, meter).

*Remark 2* The shape of a location coverage depends on the sensors and the environment where they are deployed. For instance, the shape could be lines (for mobile sensor tracking), continuous rectangles (e.g., in an office), or non-continuous disks or random shapes (e.g., in a forest excluding lakes). ■

The **Redundancy Detection** module (cf. Figure 2) consists of two clustering algorithms for the detection of temporal redundancies from static devices (cf. Algorithm 1), and the detection of spatial-temporal redundancies from mobile devices (cf. Algorithm 2). The generated clusters contain redundant data based on value similarity. In addition, the temporal and spatial-temporal coverage of each cluster is calculated to keep track of the temporal and spatial spread of each redundancy.

Algorithm 1 groups the data into clusters of temporally redundant data. It takes a data collection $C$ as input, and produces a set $TR$ of temporal redundancies (clusters) as output. First, the algorithm sorts all data items in the input collection by ascending time. Then, for each data item, the algorithm checks if a cluster already exists. If not, a new cluster is created with the current data item added as its centroid (lines 3-6). However, if a cluster

**Algorithm 1:** Temporal Redundancy Checker

```
   Input  : C
   Output: TR
   Parameters: δ_v, g_t
   Local Variables: SC, cov_t, centroid, D, min_t, max_t, δ_t
 1 Initialize TR ← ∅
 2 SC ← sort_t(C) foreach data item d_i ∈ SC do
 3     if (∄ cluster of redundant data D) then
 4         Create new cluster D
 5         Initialize centroid ← d_i.v
 6     else
 7         if (Absolute difference |d_i.v − centroid| ≤ δ_v) then
 8             Add data item to cluster D ← d_i
 9             Update centroid ← Avg(all d_i.v ∈ D)
10         else
11             Identify temporal interval δ_t ← [min_t, max_t] of D
12             Compute temporal coverage w.r.t. time unit cov_t ← (g_t, δ_t)
13             Add new temporal redundancy TR ← (cov_t, D)
14             Flush out cluster D
15         end
16     end
17 end
18 Return TR
```

already exists, the algorithm checks if the current data item belongs to the aforementioned cluster. This is done by measuring the distance between the data item and the cluster centroid values and comparing it to a deviation threshold $\delta_v$ (line 8). If the current data item belongs to the cluster, a new centroid is computed and the algorithm checks the next value in the collection (lines 9-10). This step is repeated until the algorithm finds a value that does not belong to the cluster. In this case, the temporal coverage of the cluster is calculated (lines 12-13), the cluster (i.e., temporal redundancy) is added to the output list (line 14), and the variable cluster content ($D$) is reset (line 15) in order to generate a new cluster and look for other redundancies.

Similarly, Algorithm 2 takes a data collection as input and generates a set of clusters as output, where each cluster represents a spatial-temporal redundancy. The clustering principles are the same in both algorithms. However, the spatial-temporal redundancy checker calculates the spatial coverage for each redundancy (i.e., cluster) in addition to the temporal coverage. This entails keeping track of data location stamps in each cluster (line 11) and calculating the characteristics of the coverage area (lines 13, 15, and 17). Note that both clustering algorithms calculate the temporal and spatial-temporal coverage of each cluster respectively, in order to keep track of the temporal and spatial spread of each redundancy.

To illustrate the temporal redundancy detection process, consider the $CO_2$ data collection presented in Table 3. If we apply the temporal redundancy detection algorithm with a deviation threshold $\delta_v = 3$, we detect one temporal redundancy (containing values 109, 110, and 111) spanning over a temporal coverage of 4 minutes (from 10/02/2019 10:02:00 till 10/02/2019 10:06:00).

*Algorithm Tuning.* The aforementioned algorithms cluster redundant data within collections based on the value deviation threshold $\delta_v$ which can be adjusted per device. However, the set value will affect the accuracy of the deduplication. Therefore, we propose here the **Redundancy Detection Tuning** component (cf. Figure 2) to automatically set, and re-adjust if necessary, the

**Algorithm 2:** Spatial-Temporal Redundancy Checker

```
   Input  : C
   Output: STR
   Parameters: δ_v, g_t, g_l
   Local Variables: SC, cov_t, cov_l, centroid, D, min_t, max_t, L, δ_t, δ_l, shape
 1 Initialize STR ← ∅
 2 SC ← sort_t(C) foreach data item d_i ∈ SC do
 3     if (∄ cluster of redundant data D) then
 4         Create new cluster D
 5         Initialize centroid ← d_i.v
 6     else
 7         if (Absolute difference |d_i.v − centroid| ≤ δ_v) then
 8             Add data item to cluster D ← d_i
 9             Update centroid ← Avg(all d_i.v ∈ D)
10             Add data item location to L ← d_i.l
11         else
12             Identify shape ← getShape(L)
13             Identify temporal interval δ_t ← [min_t, max_t] of D
14             Identify location area δ_l ← (shape, L) of D
15             Compute temporal coverage w.r.t. time unit cov_t ← (g_t, δ_t)
16             Compute location coverage w.r.t. location unit cov_l ← (g_l, δ_l)
17             Add new spatio − temporal redundancy STR ← (cov_t, cov_l, D)
18             Flush out cluster D
19         end
20     end
21 end
22 Return STR
```

deviation threshold (algorithms' parameter) based on historical data. To do so, the filtered data collections are sent to the data distribution discovery module that identifies the distribution of the sensor observation values using tests such as Chi-Squared and Kolmogrov-Smirnov. Once a distribution is identified (e.g., Normal, Gamma, Beta, Exponential, Weibull, Bernoulli), we estimate the deviation threshold $\delta_v$ using one of the following techniques: (i) MAD: Mean Absolute Deviation; (ii) Z-score; and (iii) IQR: Interquartile Range. The estimated threshold is used for a specific type of collection (e.g., temperature) when identifying temporal or spatial-temporal redundancies. In order to avoid repeating this process at each run, we keep using the same threshold until the accuracy of the deduplication drops below a specific acceptable level. Only then, the **Redundancy Detection Tuning** component is triggered again and the threshold is re-estimated and adjusted accordingly. To do so, we evaluate deduplication accuracy using the Jaccard similarity measure applied in the deduplication accuracy evaluation module, and compare the resulting accuracy to a configurable acceptable level in the redundancy checker adjustment module, in order to decide if the tuning should be triggered.

### 4.4 Redundancy Cleaning

Once redundancies are identified, the deduplication (i.e., redundancy cleaning) process occurs. We propose two cleaning modes: (i) the auto-clean mode fully summarizes a cluster of redundancies into one representative data item using the mean or median values; and (ii) the consumer-centric mode which considers data consumer subscirptions that describe the deduplication requirements/conditions when removing redundancies. Following the same example illustrated in Figure 3, the identified temporal redundancies could be removed using the auto-clean median method (cf. Figure 4).

Table 4: Auto-clean | Redundancy-free $CO_2$ Data Collection

| a | v | t | | l | | | | s |
|---|---|---|---|---|---|---|---|---|
| | | format | value | format | value | | | |
| | | | | | x | y | z | |
| $CO_2$ | 98 | dd/mm/yyyy hh:mm:ss | 10/02/2019 10:00:00 | cartesian | 8 | 12 | 8 | S1 |
| $CO_2$ | 110 | dd/mm/yyyy hh:mm:ss | 10/02/2019 10:04:00 | cartesian | 4 | 6 | 6 | S1 |

However, data consumers might have different needs for redundancy removal. For instance, a database could require a specific amount of data from each device per day (thus impacting the deduplication ratio). A user could have different requirements based on his/her available resources (e.g., high deduplication ratio if resources are low). Similarly, devices and services consuming data might have specifications for the redundancy removal process. In order to consider these needs (cf. Needs - Section 2), data consumers engage the network orchestrator (cf. Figure 2) to make data consumption subscriptions that detail their requirements. More formally:

**Definition 8 (Data Consumption Subscription Definition)** A data consumption subscription $dcs$ is defined as a 2-tuple:

$$dcs = (\delta_l, g_l) \quad where: \tag{8}$$

- $consumer_{id}$ is the data consumer identifier
- $producer_{id}$ is the data producer identifier
- $P$ is a set of data consumption preferences where $\forall p \in P$ $p$ is a 4-tuple $p = \langle req_a, cons_f, dedup_{type}, dudup_c \rangle$ where:
  - $req_a = d_j.a \forall j \in \mathbb{N}$ is a required data attribute (cf. Definition 1)
  - $cons_f$ is the data consumption frequency of $req_a$
  - $dedup_{type}$ is the required deduplication type
  - $dedup_c : \langle Left_{Operand} \oplus Right_{Operand} \rangle$ is the deduplication condition.

*Remark 3* The $dedup_{type}$ could be ratio-based (a specified deduplication ratio or percentage), memory-based (expressed as required output data size in Bytes), processor-based (expressed as the required percentage of CPU to be consumed for deduplication), or energy-based (expressed in the amount of energy consumed during deduplication). ■

The subscriptions are translated into redundancy removal rules that we consider in the redundancy removal process (cf. Subscription-based Data Deduplication module - Figure 2). Various markup languages can be used to describe the aforementioned rules (e.g., RuleML [3], SWRL [9], ECA [2], ECA RuleML [15]). To illustrate, consider the following data consumption subscription example between a user (data consumer) and a device (data producer) where the user requires a specific deduplication ratio (e.g., for resource-related reasons). Table 5 shows the cleaned data collection based on the user's needs.

**Data Consumption Subscription - Example**

```
<dcs_1>
    <consumer_id>user1</consumer_id>
    <producer_id>device1</producer_id>
    <P>
        <p>
            <req_a>temperature</req_a>
            <cons_f>everyDay</cons_f>
            <dedup_type>ratio-based</dedup_type>
            <dedup_c>ratio ≤ 34</dedup_c>
        </p>
    </P>
</dcs_1>
```

Table 5: $dcs_1$ | Redundancy-free $\boldsymbol{CO_2}$ Data Collection

| a | v | t | | l | | | | s |
|---|---|---|---|---|---|---|---|---|
| | | format | value | format | value | | | |
| | | | | | x | y | z | |
| $CO_2$ | 98 | dd/mm/yyyy hh:mm:ss | 10/02/2019 10:00:00 | cartesian | 8 | 12 | 8 | S1 |
| $CO_2$ | 109 | dd/mm/yyyy hh:mm:ss | 10/02/2019 10:02:00 | cartesian | 6 | 8 | 6 | S1 |
| $CO_2$ | 110 | dd/mm/yyyy hh:mm:ss | 10/02/2019 10:04:00 | cartesian | 4 | 6 | 6 | S1 |

*Device Capabilities Discussion.* In a connected environment, devices might have different capabilities (i.e., some are more advanced in terms of storage, autonomy, processing, and communication than others). In our proposal, the bulk of the processing concerns the **Device Level Redundancy Management** modules (cf. Figure 2) where we identify and clean data duplicates. Therefore, in order to cover a wide plethora of devices, we consider two main scenarios: (i) the device has an acceptable level of capabilities/resources and can run the aforementioned modules locally; and (ii) the device has no or low/limited resources and delegates the processing to another capable edge device, or to the nearest sink.

## 5 Experiments & Results

We have implemented DRMF and have conducted a large battery of experiments to evaluate its redundancy identification and removal functionalities. Our experiments can be grouped in two main category: (i) performance evaluation (i.e., run-time, CPU consumption, and RAM size); and (ii) accuracy evaluation (i.e., deduplication quality, deduplication ratio, and threshold es-

timation). To do so, we used the publicly available Intel Lab Data set[1], and tested the algorithms (developed in Python 3.8) on 38,656 records. The data set provides various observations (e.g., temperature, humidity, light) from 54 sensors deployed in various locations in the Intel Berkeley Research lab between February 28th and April 5th, 2004.

### 5.1 Performance Evaluation

We ran the following experiments on a Dell machine with Windows 10, having a Core i5 $8^{th}$ Generation 1.8 GHZ processor, and 16 GB of RAM. We evaluate the performance of both redundancy detection algorithms and the auto-clean mode for redundancy removal (we left the evaluation of the consumer-centric deduplication for a separate work) by measuring the run-time, CPU consumption, and RAM size.

- Experiment 1: *Input Data Size Impact.* In this test, we gradually increase the input data size in order to assess its impact on performance.
- Experiment 2: *Deviation Threshold Impact.* In this test, we gradually increase the deviation threshold to generate clusters with various sizes and spreads to assess its impact on performance.

*Discussion.* Figures 3a, 3b, and 3c show the results of experiment 1. Increasing the number of input data items from 0 to 38656 had a visible impact on performance. The required time, CPU, and RAM for identifying and removing redundancies increase in a quasi-linear way. However, in the worst case scenario (i.e., 38656 values) the required time does not exceed 8 seconds and the required RAM/CPU does not surpass 34 MB and 9% respectively. Figures 3d, 3e, and 3f show the results of experiment 2. Increasing the deviation threshold generates fewer but bigger clusters of redundant data. This is reflected in the results: the bigger the threshold, the less time and CPU are required since less clusters are generated. RAM consumption slightly fluctuates between 31.9 and 32.9 MB, yet the difference is not significant since every data item eventually belongs to one cluster regardless of the number of generated clusters. Finally, for both tests, Algorithm 2 requires more resources since it considers both temporal and spatial dimensions in contrast with Algorithm 1 which only consider the temporal dimension.

### 5.2 Accuracy Evaluation

We also evaluate the accuracy of our proposal by measuring data reduction accuracy (using the Jaccard Similarity Index), and the data reduction ratio. Each experiment is done twice (with temperature and humidity values). For

[1] `http://db.csail.mit.edu/labdata/labdata.html`

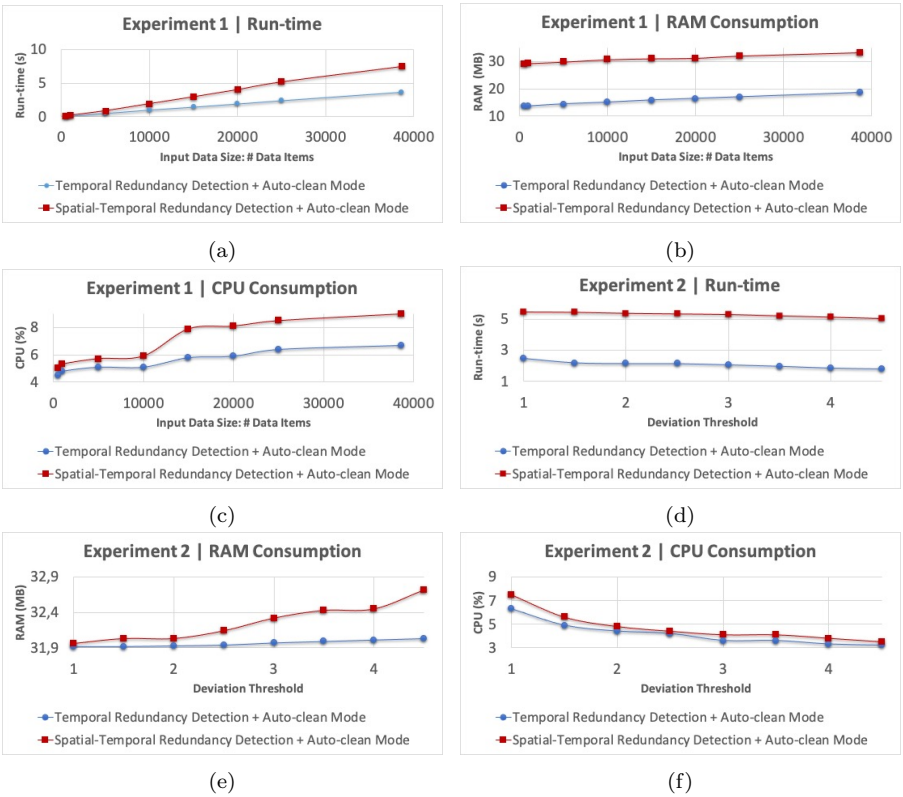



Fig. 3: Performance Results: Experiments 1 and 2

the two cases, we identify redundancies using both algorithms and clean duplicates using the auto-clean mode. We present here the temporal redundancy identification and cleaning results.

- Experiment 3: *Input Data Size Impact.* In this test, we gradually increase the input data size to assess its impact on accuracy.
- Experiment 4: *Deviation Threshold Impact.* In this test, we gradually increase the deviation threshold to assess its impact on accuracy.
- Experiment 5: *Threshold Estimation Impact.* In this test, we change the threshold estimation technique (i.e., Z-score, MAD, IQR) and measure the data reduction accuracy and ratio using the estimated thresholds.

*Discussion.* Figure 4a shows the results of experiment 3. The data reduction accuracy (with a fixed deviation threshold of 2.5) varies between 97.63% and 98.78% for humidity data. Moreover, accuracy varies between 95.03% and 96.54% for temperature. Figure 4b shows the results of experiment 4. Increasing the deviation threshold affects the clustering of redundancies. The optimal deviation threshold in the 0.5 to 5 range is 1 for humidity (accuracy of

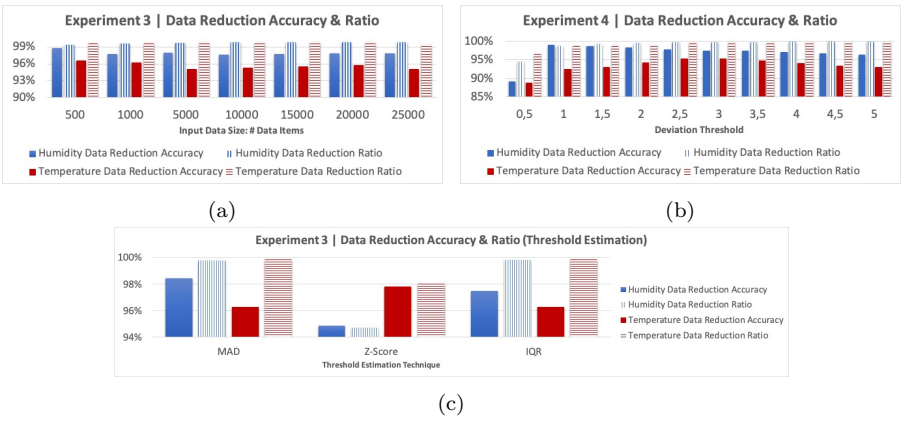



Fig. 4: Accuracy Results: Experiments 3, 4, and 5

99.10%) and 3 for temperature (accuracy of 95.43%). In addition, both experiments achieve high deduplication ratios (94.48% to 99.90%). Finally, Figure 4c shows that results of experiment 5 which highlights the importance of choosing the adequate technique based on the data distribution and historical patterns (i.e., the technique should not be randomly assigned/configured). The discovered distribution for both temperature and humidity collections is Weibull. This explains why Z-Score provided the worst result since it is based on the mean and standard deviation (i.e., more suited for normal distributions).

## 6 Conclusion & Future Works

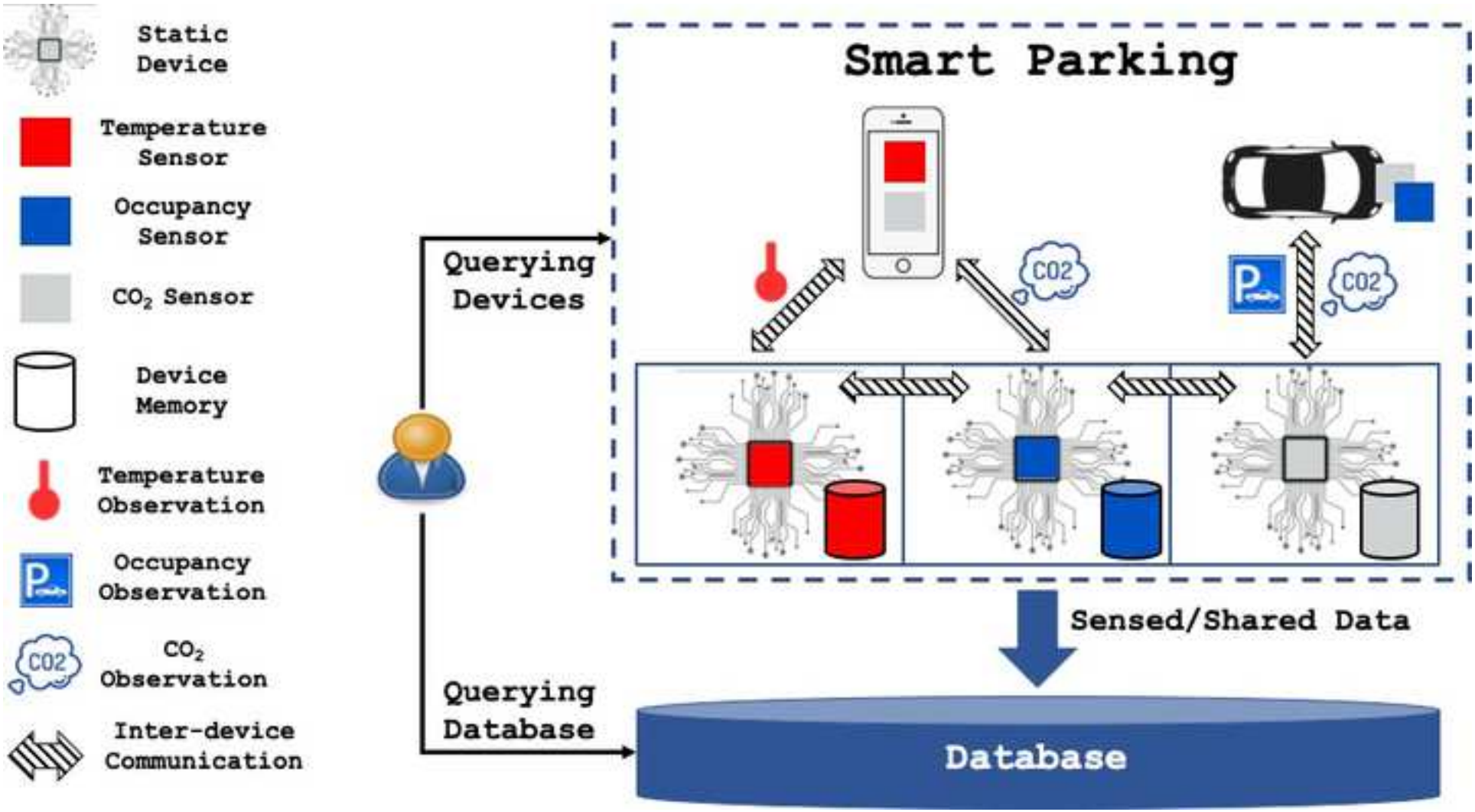
In this paper, we address the problem of handling data redundancy in connected environments. We introduce DRMF, a data redundancy management framework which handles sensor data redundancy at the edge device level, considering both static and mobile devices, in order to eliminate redundancies from the source before reaching the core of the network. DRMF includes two clustering algorithms that detect temporal and spatial-temporal data redundancies, and a module for redundancy removal/summarization that considers data consumer needs and device resources when deduplicating. We are currently extending DRMF to develop the consumer-centric redundancy removal process based on the data consumption subscriptions. Consequently, we plan to improve the deviation threshold estimation by considering pattern recognition and providing multiple thresholds for a data collection. In addition, we plan to investigate an approach based on fuzzy reasoning, to improve the accuracy of deduplication detection, especially when dealing with crisp cluster boundaries. In the near future, we aim to detect composite redundancies that are generated by data fusion from multiple sensors, where deduplication would be handled both at the edge level and at the sink level of the network. This

entails special challenges depending on the structure, connectivity, dynamics, and overall properties of the connected environment.

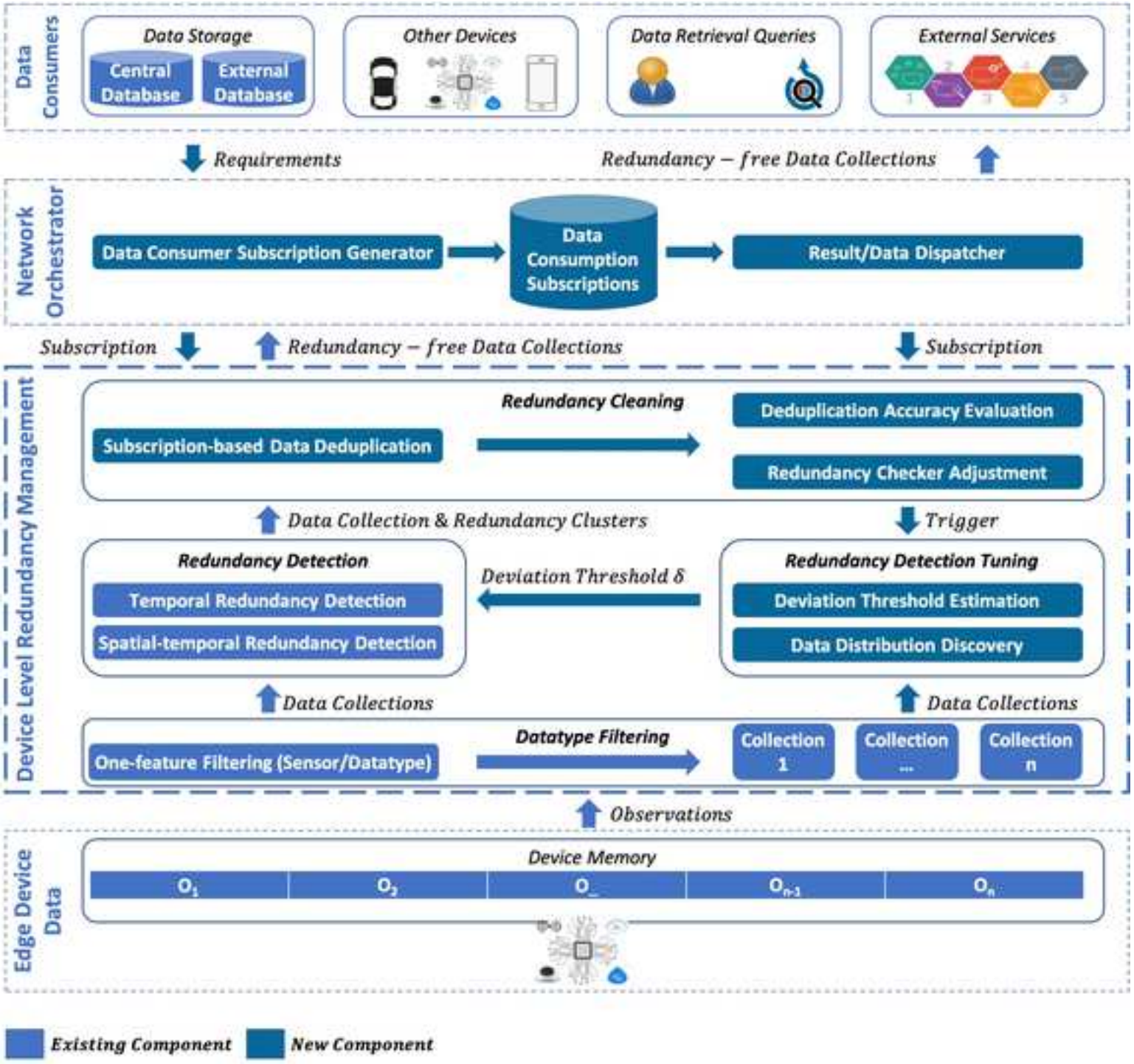
## References


1. Baba, A.I., Lu, H., Xie, X., Pedersen, T.B.: Spatiotemporal data cleansing for indoor rfid tracking data. In: 2013 IEEE 14th International Conference on Mobile Data Management, vol. 1, pp. 187–196. IEEE (2013)
2. Bailey, J., Papamarkos, G., Poulovassilis, A., Wood, P.T.: An event-condition-action language for xml. In: Web Dynamics, pp. 223–248. Springer (2004)
3. Boley, H., Tabet, S., Wagner, G.: Design rationale for ruleml: A markup language for semantic web rules. In: SWWS, vol. 1, pp. 381–401 (2001)
4. Chowdhury, S., Benslimane, A.: Relocating redundant sensors in randomly deployed wireless sensor networks. In: 2018 IEEE Global Communications Conference (GLOBECOM), pp. 1–6. IEEE (2018)
5. Fang, X., Misra, S., Xue, G., Yang, D.: Smart grid—the new and improved power grid: A survey. IEEE communications surveys & tutorials **14**(4), 944–980 (2011)
6. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of things (iot): A vision, architectural elements, and future directions. Future generation computer systems **29**(7), 1645–1660 (2013)
7. Harb, H., Makhoul, A., Abou Jaoude, C.: En-route data filtering technique for maximizing wireless sensor network lifetime. In: 2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC), pp. 298–303. IEEE (2018)
8. He, Q., Li, Z., Zhang, X.: Data deduplication techniques. In: 2010 International Conference on Future Information Technology and Management Engineering, vol. 1, pp. 430–433. IEEE (2010)
9. Horrocks, I., et al.: Swrl: A semantic web rule language combining owl and ruleml. W3C Member submission **21**(79), 1–31 (2004)
10. Idrees, A.K., Al-Yaseen, W.L., Abou Taam, M., Zahwe, O.: Distributed data aggregation based modified k-means technique for energy conservation in periodic wireless sensor networks. In: 2018 IEEE Middle East and North Africa Communications Conference (MENACOMM), pp. 1–6. IEEE (2018)
11. Ismael, W.M., Gao, M., Al-Shargabi, A.A., Zahary, A.: An in-networking double-layered data reduction for internet of things (iot). Sensors **19**(4), 795 (2019)
12. Li, S., et al.: Ef-dedup: Enabling collaborative data deduplication at the network edge. In: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), pp. 986–996. IEEE (2019)
13. Mansour, E., et al.: Data redundancy management in connected environments. In: Proceedings of the 16th ACM Symposium on QoS and Security for Wireless and Mobile Networks, Q2SWinet '20, p. 75–80. Association for Computing Machinery, New York, NY, USA (2020). DOI 10.1145/3416013.3426451. URL `https://doi.org/10.1145/3416013.3426451`
14. Marinakis, V., Doukas, H.: An advanced iot-based system for intelligent energy management in buildings. Sensors **18**(2), 610 (2018)
15. Paschke, A.: Eca-ruleml: An approach combining eca rules with temporal interval-based kr event/action logics and transactional update logics. arXiv preprint cs/0610167 (2006)
16. Patil, P., Kulkarni, U.: Svm based data redundancy elimination for data aggregation in wireless sensor networks. In: 2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 1309–1316. IEEE (2013)
17. Suma, S., Mehmood, R., Albeshri, A.: Automatic event detection in smart cities using big data analytics. In: International Conference on Smart Cities, Infrastructure, Technologies and Applications, pp. 111–122. Springer (2017)
18. Ullah, A., Sehr, I., Akbar, M., Ning, H.: Fog assisted secure de-duplicated data dissemination in smart healthcare iot. In: 2018 IEEE International Conference on Smart Internet of Things (SmartIoT), pp. 166–171. IEEE (2018)
19. Ullah, A., et al.: Secure healthcare data aggregation and deduplication scheme for fog-orineted iot. In: 2019 IEEE International Conference on Smart Internet of Things (SmartIoT), pp. 314–319. IEEE (2019)

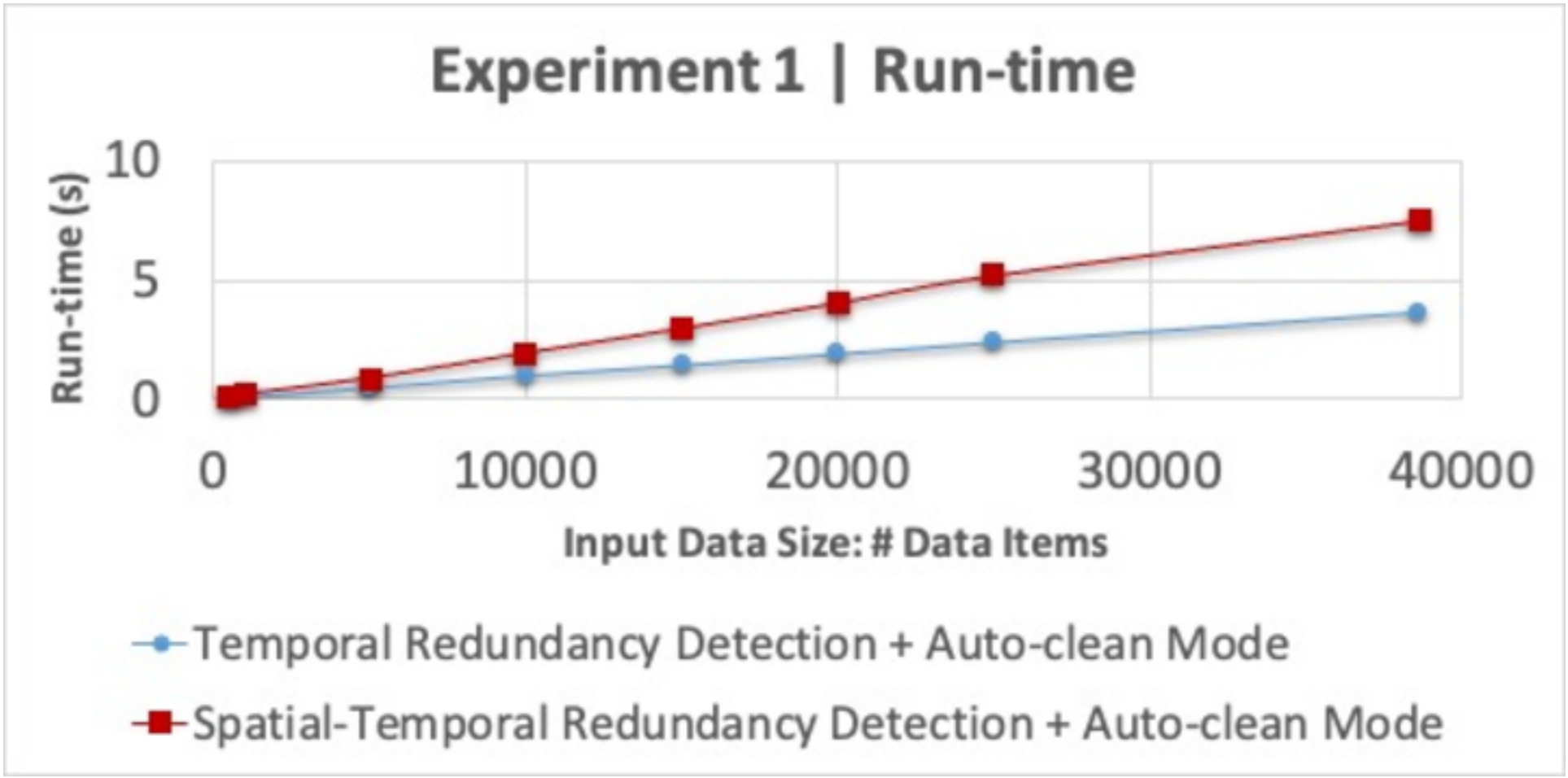
Click here to access/download;Figure;Fig1.jpg



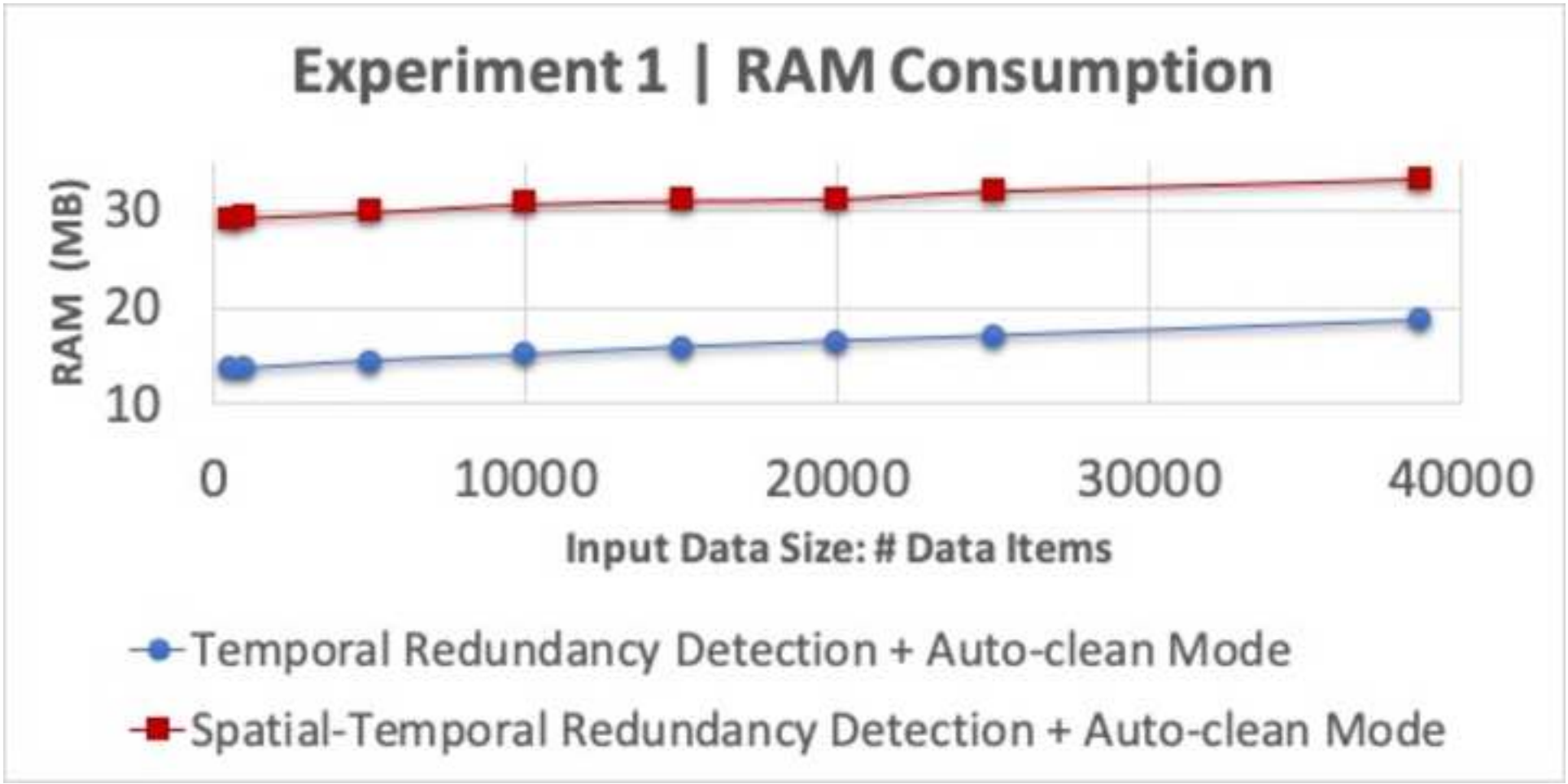
Click here to access/download;Figure;Fig2.jpg



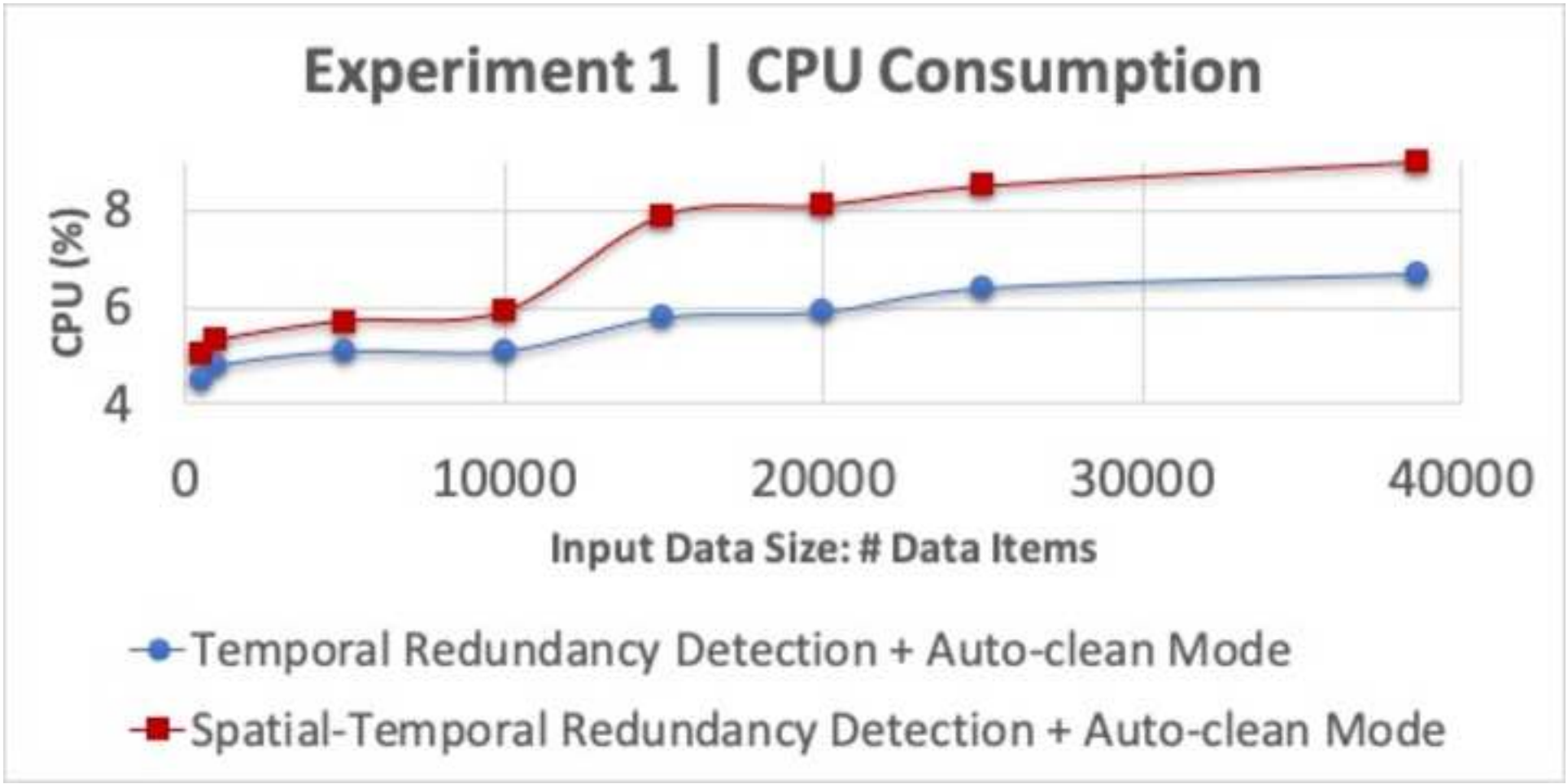
Click here to access/download;Figure;Fig3.jpg



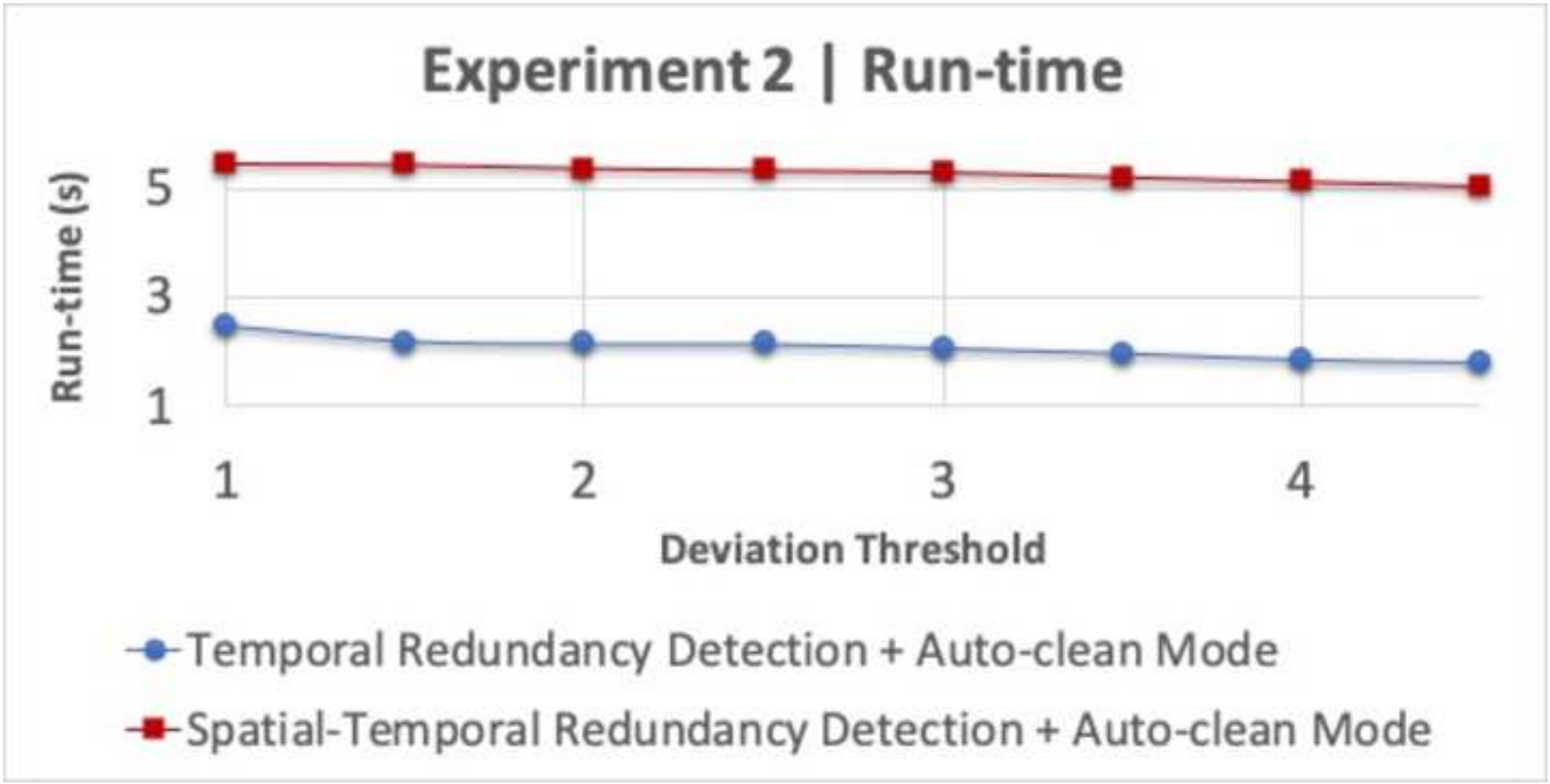
Click here to access/download;Figure;Fig4.jpg



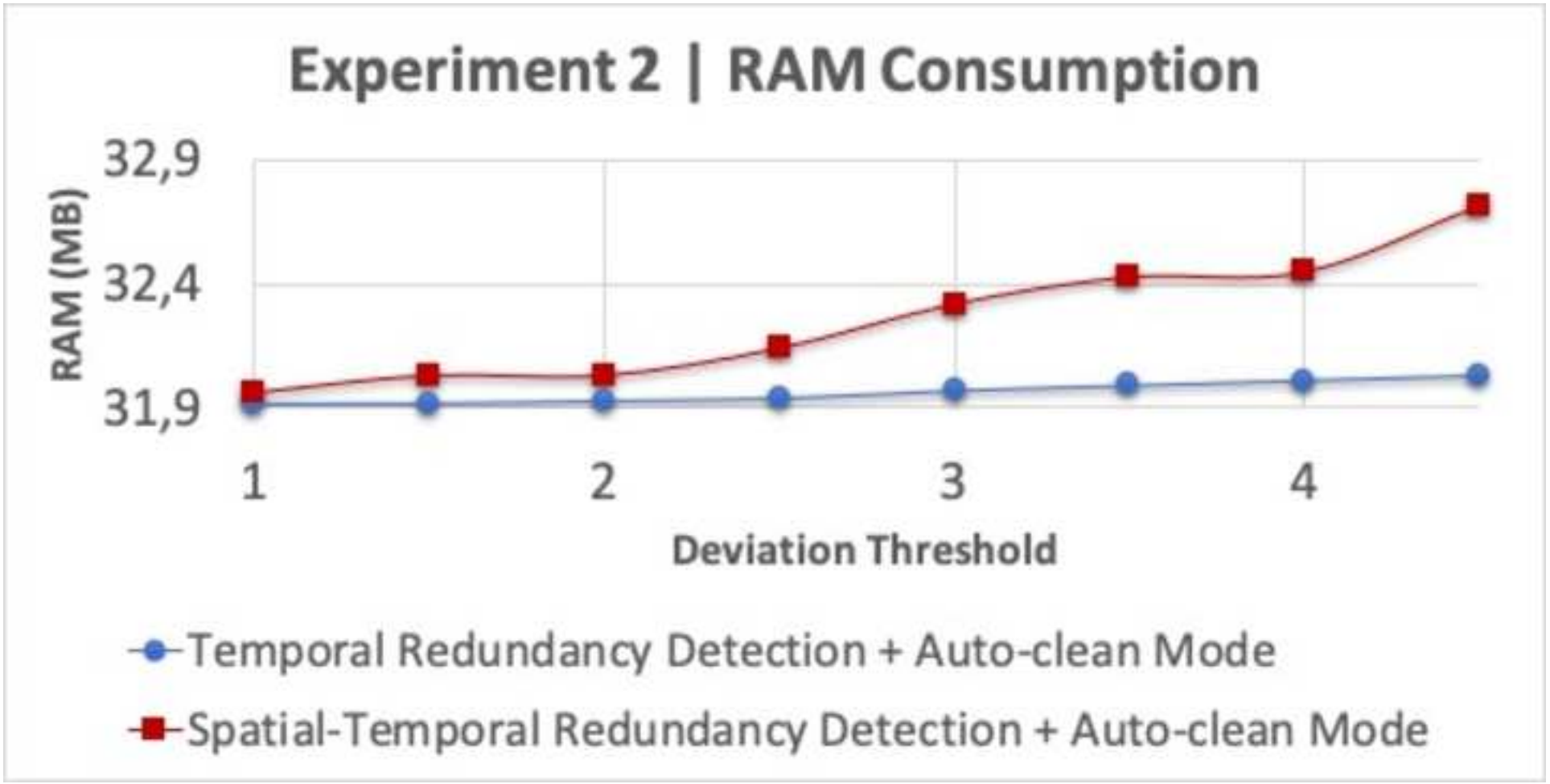
Click here to access/download;Figure;Fig5.jpg



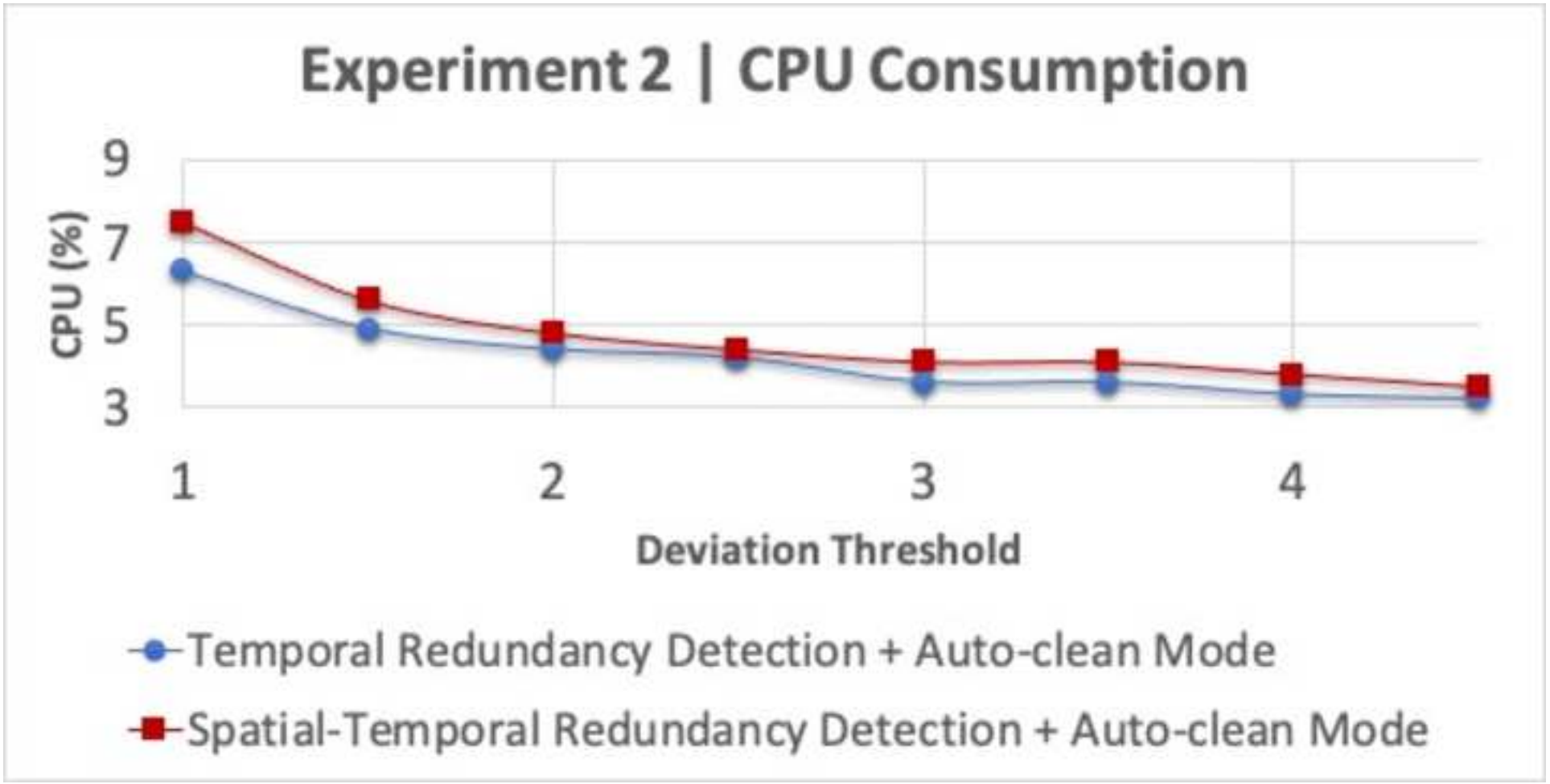
Click here to access/download;Figure;Fig6.jpg



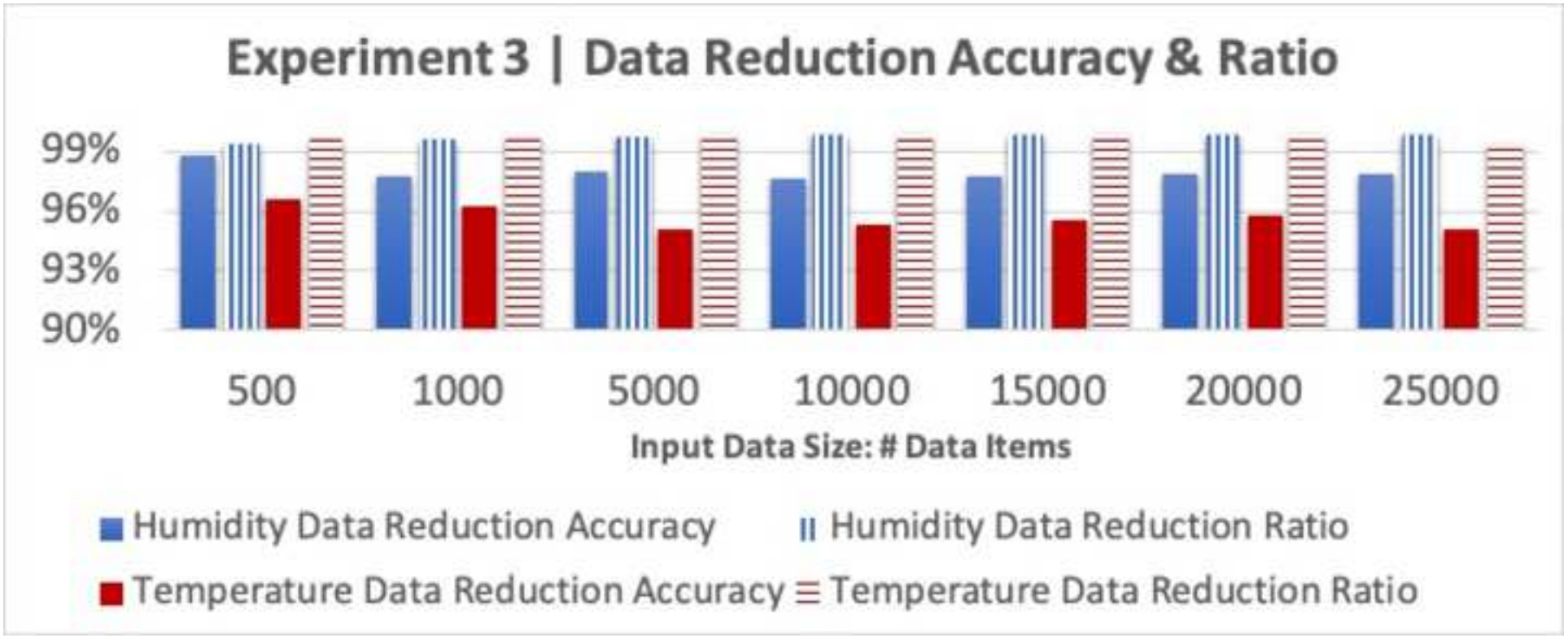
Click here to access/download;Figure;Fig7.jpg



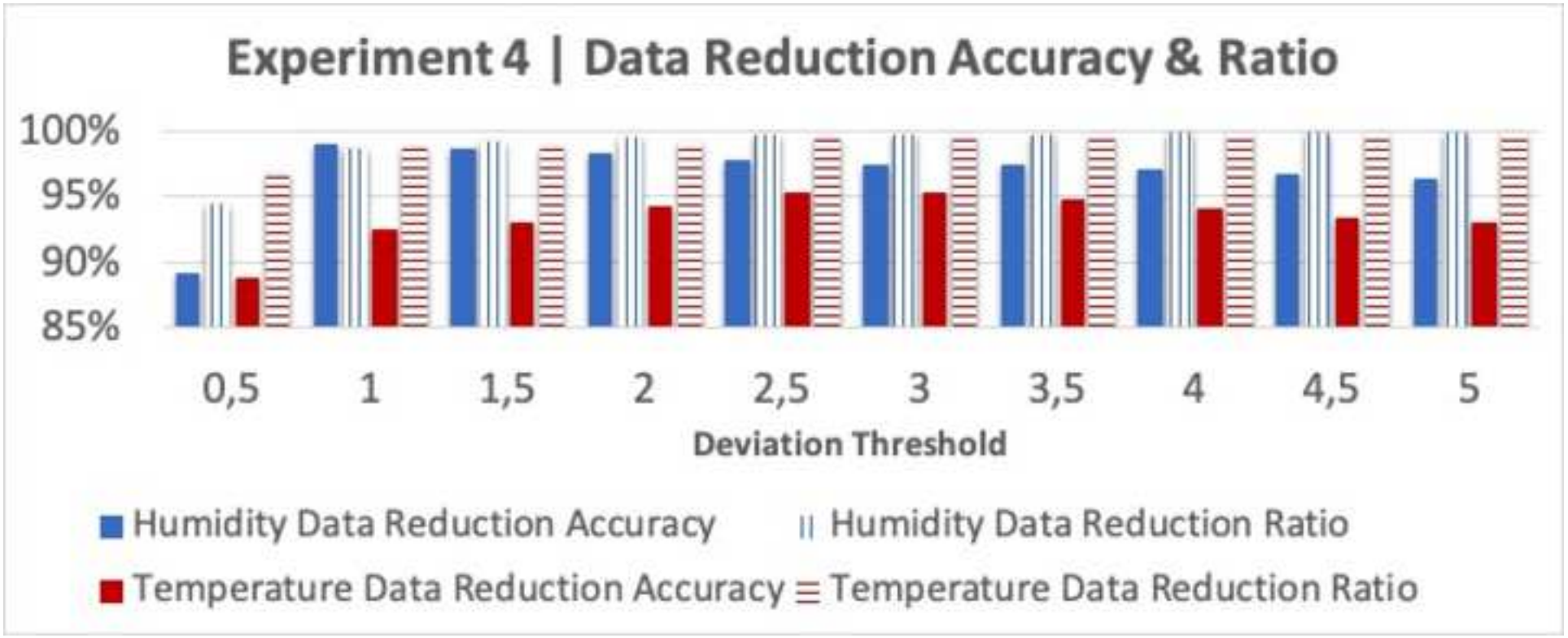
Click here to access/download;Figure;Fig8.jpg



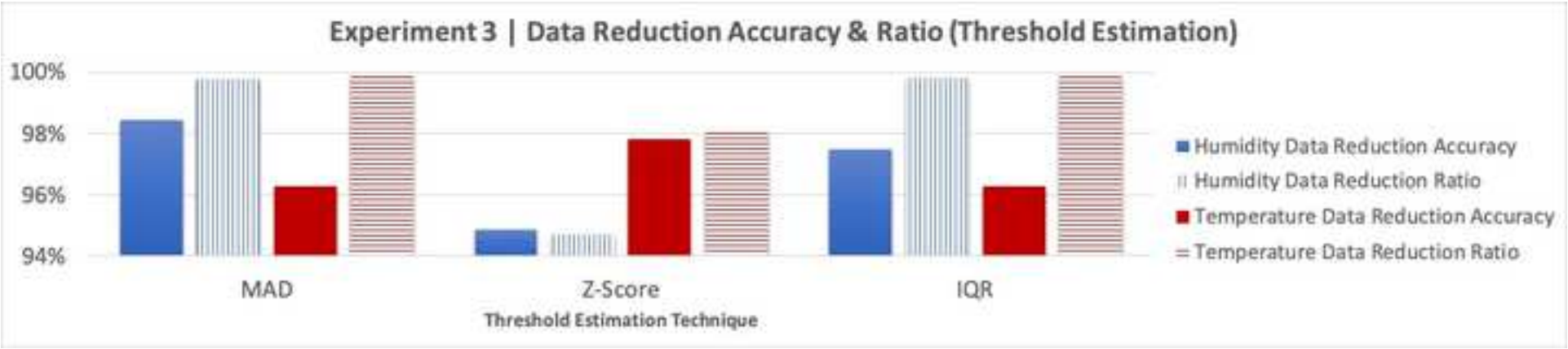
Click here to access/download;Figure;Fig9.jpg



Click here to access/download;Figure;Fig10.jpg



Click here to access/download;Figure;Fig11.jpg



Summary of Differences

Click here to access/download
**Supplementary Material**
Summary of Differences.pdf