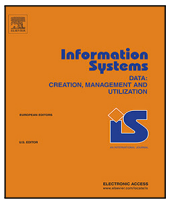
FEDAMON: A fully automated framework for communication-efficient continuous distributed monitoring with error guarantees

Information Systems 141 (2026) 102751

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is



FEDAMON: A fully automated framework for communication-efficient continuous distributed monitoring with error guarantees☆

Yixing Zhang *, Romaric Duvignau

Department of Computer Science and Engineering, Chalmers University of Technology and University of Gothenburg, Sweden

ARTICLE INFO

Keywords:
Continuous monitoring
Distributed data streams
Network monitoring
Distributed tracking
Data-aware approaches

ABSTRACT

Efficient monitoring of large distributed systems is critical for applications such as data center load balancing, fleet management, and smart grid energy optimization. This paper addresses the continuous distributed monitoring problem, where a central coordinator tracks statistics from numerous distributed nodes in real time. We present FEDAMON, a novel **F**orecast-based, **E**rror-bounded, and **D**ata-**A**ware approach to continuous distributed **Mon**itoring that significantly reduces communication costs while maintaining accuracy. Instead of transmitting all values to the coordinator, our event-based monitoring leverages lightweight forecasting models at the coordinator and distributed nodes to predict the evolution of observations, communicating when deviations exceed an error threshold. To adapt to dynamically changing data streams, we introduce a data-aware model selection strategy that optimizes the trade-off between communication and accuracy. Our solution is communication-efficient, fully automated, and equipped with dynamic error control, reducing system parametrization to a single error tolerance of three preset levels. FEDAMON reduces communication to 10% of the baseline with less than 2% error across all streams on diverse datasets on average. Moreover, the standard parameter solution surpasses even the best calibrated single models across all error bounds, achieving up to 33% improvement in communication efficiency with identical error guarantees. Further gains of 25% in accuracy is obtained by tuning the data-aware control factor without extra cost. In addition, our framework generalizes effectively to previously unseen datasets. Finally, our dynamic error control achieves comparable performance to fixed bounds. Results highlight the scalability and robustness of FEDAMON, enabling fully automatic, real-time monitoring with large communication savings and marginal error.

1. Introduction

The proliferation of connected devices has led to an unprecedented surge in the volume and velocity of data being generated across modern networks. Today's network infrastructure faces the challenge of managing high-velocity and high-volume data generated by billions of internet-connected devices, reflecting the pervasive reality of the Internet of Things (IoT) era [1]. Thus, reducing the amount of continuously transmitted data has become imperative for saving energy [2] and enhancing performance in modern packet-processing architectures [3,4]. Efficiently and continuously monitoring large-scale distributed systems is a crucial challenge with numerous applications, ranging from extending the lifespan of battery-powered devices [5] in sensor networks to improving load balancing in data centers [6,7].

This paper focuses on the Continuous Distributed Monitoring (CDM) problem, where a central coordinator C monitors a distributed system in a synchronous manner, retrieving observed values from all sensors during each round. The primary goal is to minimize the amount of data transmitted between distributed nodes and C while maintaining accurate monitoring. The field of CDM is a vibrant research area, with various approaches aiming to reduce network costs. In recent years, significant research has been devoted to developing communication-efficient algorithms across various models for problems like (approximate) event counting [10], computing item frequencies [11], and identifying the most popular items [12]. We focus here on the ***All-Values-Tracking* (AVT) problem**, i.e., keeping track of all observed distributed values at each round, an important task with numerous practical applications [3,13,14]. AVT focuses on tracking every individual value from each node and therefore provides a generic solution to the CDM problem. Since C receives all observed values from the nodes, it can compute any aggregation function based on these estimates. Fig. 1(a) illustrates the baseline monitoring approach, where all observed values are transmitted to the coordinator.

☆ This article is part of a Special issue entitled: 'Distributed Data Processing and Event-Based Systems' published in Information Systems.

* Corresponding author.
E-mail address: yixing@chalmers.se (Y. Zhang).

https://doi.org/10.1016/j.is.2026.102751
Received 11 October 2025; Received in revised form 22 February 2026; Accepted 15 May 2026
Available online 21 May 2026

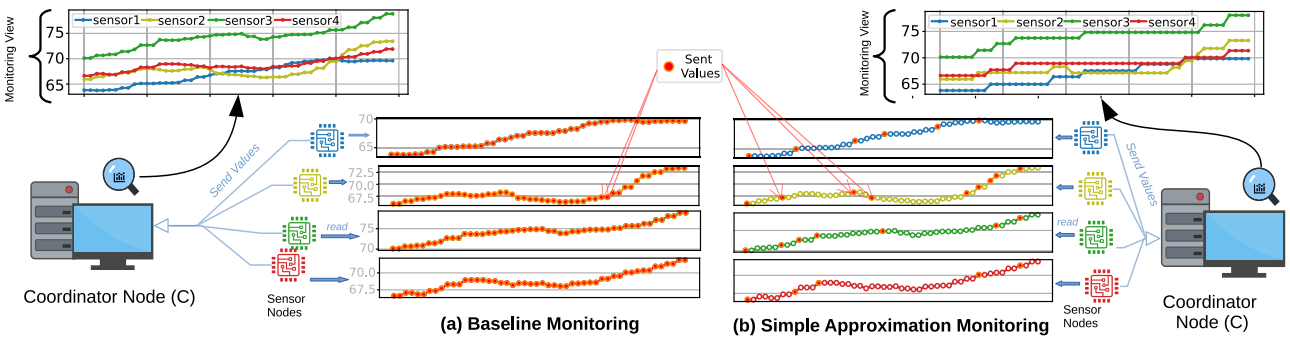


Fig. 1. Illustration of continuous monitoring comparing *(a) baseline monitoring* [8] with a *(b) simple approximation* [9] strategies, both on 4 distributed sensor nodes with monitoring views at the coordinator.

The existing literature highlights several approaches to communication-efficient distributed monitoring, such as prediction-based models and data compression techniques. Prediction-based frameworks, like the one in [15], employ forecasting models to minimize data transmission, but often rely on static modeling choices and do not account for the evolving nature of data streams. Similarly, prediction-based data reduction in Wireless Sensor Networks (WSNs) [5] assumes shared communication mediums and leverages inter-node correlations, which limits the applicability of per-node prediction schemes in more general distributed settings. Data compression techniques [16], while effective at reducing communication, fail to provide real-time monitoring capabilities, which are critical for applications requiring up-to-date resource tracking. Thus, there remains a gap in designing a scalable, adaptive monitoring approach that minimizes communication overhead while maintaining real-time accuracy, particularly in distributed systems with private communication channels and loosely correlated data streams, applicable to many Cyber-Physical Systems (CPS) and modern distributed environments.

1.1. Aim, motivations and challenges

To avoid continuously flooding the network with monitoring messages, we aim to reduce communication costs by allowing a small, bounded error in the tracked statistics at the coordinator. This is a challenging problem, as worst-case scenarios may require all data to be transmitted at all times. By embedding the monitoring logic close to data sources and leveraging the predictable nature of data evolution, we seek to drastically reduce network transmissions while incurring only a small reduction in tracking accuracy. Additionally, the dynamic nature of data streams necessitates data-aware monitoring model selection to account for changing data characteristics, thereby enhancing monitoring performances.

Despite advancements in distributed monitoring, current state-of-the-art solutions [10,13,14,17] often prioritize worst-case scenarios, focusing on tight theoretical bounds for communication complexity or near-optimal online algorithms. However, these approaches fall short in practical settings, where real-time data stream values typically exhibit strong temporal correlations. For instance, continuously tracking all node values remains a significant challenge due to the dynamic nature of the data and the requirement for lightweight approaches—excessive computational demands for monitoring would negate the benefits of reduced communication. Furthermore, CPS and IoT devices operate under stringent energy and computational constraints. In this context, monitoring algorithms must be efficient not only in communication but also in computation to ensure low energy consumption and sustain system-wide energy efficiency. This creates a pressing need for algorithms that are both lightweight and capable of leveraging temporal correlations in data streams. A shift towards data-driven, data-aware monitoring solutions is essential to address these challenges and meet the demands of real-world applications.

Table 1
Summary of notations and symbols used throughout the paper.

Notation	Description
C	Coordinator node
$\mathbb{S}$, $n = \lvert\mathbb{S}\rvert$	Set of distributed nodes and its size
$s^i \in \mathbb{S}$	ith distributed node
$v_t^i, \hat{v}_t^i$	Value observed, estimated on s^i at time t
T	Total no. of monitoring time steps
Q	Total no. of update messages sent
ρ	Communication ratio
$\ell, k, \kappa, \alpha, \beta, \gamma$	Model-specific parameters
R_D, R'_D	Static and dynamic range for dataset D
$R_{\mathcal{E}}$, $R_{\mathcal{G}}$, R_I, $R_{\mathcal{A}}$	Range of Ericsson, Geolife, IntelLab, ACSF1
L, D	Local buffer and its maximum size
$\mathcal{V}$, α	Values, control factor for score calculation
$M_{selected}$	Selected model running at distributed nodes
L_{range}	Buffer for dynamic range calculation

1.2. Contributions

The contributions of our **F**orecast-based, **E**rror-bounded, and **D**ata-**A**ware Continuous Distributed **Mon**itoring (FEDAMON) framework are summarized as follows:

- We present a novel **forecast-based, error-bounded framework** for continuous distributed monitoring. Both the coordinator and distributed nodes locally predict the evolution of observed values and communicate only when deviations exceed a predefined error threshold. This event-triggered strategy substantially reduces communication while preserving accuracy. The framework relies on lightweight yet effective forecasting models, including `Auto-Regressive (AR)` [21], `Least Mean Square (LMS)` [22,23], `Piecewise-Linear Approximation (PLA)` [20], `Holt Linear` [24], and `Holt Winters` [25], ensuring minimal computational overhead at distributed nodes.
- We introduce a **data-aware model selection mechanism** that dynamically identifies the most suitable forecasting model based on evolving data characteristics. By continuously evaluating candidate models, the coordinator optimizes the trade-off between communication cost and monitoring accuracy, updating distributed nodes with the most appropriate model when necessary.
- We propose a **standardized parameter configuration** together with a **dynamic data-range-based error control mechanism**, significantly simplifying deployment. This design reduces system parametrization to a minimal set of intuitive error tolerance levels while maintaining robust performance across diverse data streams.

To validate our approach, we conduct experiments on four large datasets spanning diverse environments, including CPU usage in mobile architectures, vehicular sensor data, temperature readings in sensor networks, and energy consumption of home appliances. We further apply our approach to three additional validation datasets, covering

Table 2
Parameters of FEDAMON and their suggested default value.

Param.	Description	Default value
ϵ	Absolute error upper-bound	$x\% \cdot R_D$
δ	Relative error tolerance for adaptive error control using dynamic range	5%, 10%, 15%
W_{range}	Buffer size for dynamic range	1000
τ_ϵ	Starting round for dynamic range	150
$\mathbb{M}$	Candidate model list	–
ξ	Model selection criterion	0.01
W_{score}	Size of the set for score calculations	100
τ	Starting round for data-aware selection	150

ϵ is used as a parameter when R_D is fixed, $x \in [1, 2, \ldots, 15]$;
$\delta, W_{range}, \tau_\epsilon$ are used when *dynamic range* is introduced;
$\mathbb{M}, \xi, W_{score}, \tau$ are used when *standard solution* is introduced.

Table 3
Summary of related work in distributed and forecast-based monitoring, highlighting our proposed FEDAMON framework.

Category	Model/Approach	Key features	Rounds	Channel	Node-wise	Limitations
CDM-related models	Cormode's Model [10]	Monitor aggregates over union of streams; events interchangeable; efficient for classical data mining functions	Async.	Bidirectional	✗	Typically studied monitoring functions and algorithms ignore node identities
	Distributed Data Streams [18]	Requires sublinear space with data-sketches; suitable for approximate summaries	Sync.	Unidirectional	✗	No feedback from C; limited for tasks requiring historical data; impractical in real systems
	Distributed Online Tracking [17]	Uses relays for local aggregation; competitive analysis vs. offline optimal	Sync.	Unidirectional	✗	Overhead of relays; limited applicability
	Online Monitoring [13,14]	Tracks top-k values; allows broadcast counted as one message	Sync.	Unidirectional	✗	Strong synchronization assumption; specific to top-k tracking; no experimental validation
Forecast-based approaches	Baseline Monitoring (TinyDB) [8]	Each node forwards all values; simple and accurate	Sync.	Bidirectional	✓	Not scalable due to high communication cost
	Simple Approximation [9]	Nodes send updates only when deviation exceeds threshold; simple and efficient	Sync.	Bidirectional	✓	Does not capture linear correlations; loss of accuracy; does not adapt to data evolution
	Adaptive AR Prediction [15]	Each node runs multiple AR models; Hoeffding bound discards weak models; single optimized AR per node	Sync.	Unidirectional	✓	Restricted to AR models; does not account for data evolution; requires parameter tuning (less flexible than a plugin approach)
	Prediction-based Reduction in WSNs [5,19]	Exploits spatial correlations; cluster-head aggregation; offloads prediction tasks	Sync.	Bidirectional	✗	Assumes shared medium; not feasible for per-node models; limited in low-correlation datasets
	Data Stream Compression [16,20]	Compress data and send in batches; significantly reduces communication overhead	Sync.	Unidirectional	✗	Breaks real-time monitoring; coordinator cannot track per-node values continuously
FEDAMON		Forecast-based, error-bounded, data-aware model selection; tracks all nodes individually; adapts dynamically; minimizes communication	Sync.	Bidirectional	✓	Extra computational overhead on C when using data-aware solution

highway speed records and electricity consumption from different regions. The results demonstrate significant reductions in communication with minimal impact on real-time monitoring accuracy, highlighting the practical benefits of our framework in real-world applications.

This work is an extension of our previous work [26]. Compared to the short conference version, this revised and extended version adds two additional prediction models to our evaluation and standard solution (`Holt Linear` and `Holt Winters`), a communication reduction analysis, a more detailed description of the monitoring algorithms (on the distributed nodes and at the coordinator) and scoring calculations, a novel approach to dynamically calculate per-node error bounds, and a fully revised evaluation that incorporates all refinements and additional components as well as additional validation datasets.

1.3. Paper structure

The rest of this paper is structured as follows. Section 2 provides an overview of related work in continuous distributed monitoring and forecasting approaches. Section 3 presents our proposed forecast-based monitoring framework, including the design of error-bounded communication protocols, prediction models, and communication analysis. Mechanisms for data-aware model selection and dynamic error control are introduced in Section 4. Section 5 discusses the results of our extensive performance evaluation, highlighting the effectiveness of our approach across various scenarios and comparing it to baseline solutions. Finally, Section 6 concludes the paper by summarizing our contributions.

2. Background and related work

In this section, we introduce our system model and compare it to other related monitoring models, highlighting the differences in key assumptions and characteristics. We then outline the primary problem in focus and present the existing solutions solving it. Table 3 summarizes related CDM models and previous forecast-based approaches in comparison to FEDAMON.

2.1. Continuous distributed monitoring

2.1.1. CDM system model

The Continuous Distributed Monitoring (CDM) problem concerns the task of continuously gathering information at a central location, or *Coordinator* (referred thereafter as C), i.e., a sink node that collects values observed by a set of distributed *nodes*. Unlike traditional monitoring approaches, CDM emphasizes optimizing communication efficiency rather than computational performance. The notations used throughout the paper are summarized in Tables 1, 2.

We consider a set of n distributed nodes, or sensors, $\mathbb{S} = \{s^1, s^2, \dots, s^n\}$, where each node s^i observes a discrete stream of values $v_t^i \in \mathbb{R}^+$ at discrete timesteps $t \in \mathbb{N}$. The coordinator C is connected to each node via a bidirectional channel, while direct communication between nodes is not assumed. The distributed nodes can represent routers or worker nodes in a data center, IoT devices in a sensor network, or vehicles in a vehicular fleet. Fig. 1 provides an example of communication between C and the sensor nodes. For each node s^i, monitoring rounds have a fixed duration d_i, so that time t (or round t) corresponds to the interval starting at real-time $t \cdot d_i$ and ending at $(t+1) \cdot d_i$. The objective of CDM is to compute a monitoring function f at C using distributed inputs, while minimizing communication overhead. The function f may track all values individually or compute an aggregate (e.g., maximum or average) over the nodes' readings, such as CPU usage of worker nodes in a data center, temperature or humidity from IoT devices, or vehicle positions in a fleet management system.

A *monitoring strategy* is defined as an algorithm to compute the action (referred as the *monitoring decision*) taken by a node after recording a measurement: to either transmit an update to C or remain silent. The central challenge in CDM is to reduce communication overhead while ensuring that the monitoring function is accurately maintained at the coordinator. The *monitoring view* refers to C's (possibly approximate) view of all tracked values, which is updated at every round. Values observed at nodes are treated as observation events. Based on the monitoring decision, each node determines whether or not to trigger an update event.

For the subsequent analysis, we assume a reliable and stable communication channel between the coordinator C and each distributed node, as well as synchronized monitoring periods (i.e., consistent start and end times across nodes and the coordinator). In practice, these assumptions can be partially relaxed, as discussed in Section 4.6. Consistent with prior work [10,13,14,17], we assume a single instance of the central coordinator C and do not consider replication. Consequently, failures of C are beyond the scope of this study.

2.1.2. Other related CDM models

Cormode's Model [10] is one of the most well-known frameworks for distributed monitoring. It focuses on enabling C to compute an aggregate function over the union of streams from n observers, each directly connected to C via a bidirectional channel. In this context, an *item* refers to an application-specific *type* of observation detected at a remote site, which is then considered an event. The model operates asynchronously, where each observation triggers an event, followed by a monitoring decision and a message transmission to update the function f at C. The key strength of this model lies in leveraging the interchangeability of events to efficiently compute classical data mining functions. However, there is a major limitation that prevents its application to our setting, namely that f is treated as a function over the union of streams, thereby ignoring the identity of the nodes (i.e., the same event from different nodes is treated identically), whereas our problem requires tracking each node individually.

The Distributed Data Streams Model [18] introduces a variation of the CDM model with two notable differences: observers communicate with C via unidirectional channels, and they are constrained to use sublinear space, meaning they can only maintain approximate summaries or *data sketches* [27] of their input streams. Due to the unidirectional communication, observers must rely solely on their own observations to make monitoring decisions, without any feedback from C. While the space limitations are effective for counting problems (such as top-k items or frequency tracking), they are less applicable in scenarios where historical data is critical, which is often the case in our context. Additionally, the restriction to unidirectional channels significantly reduces the efficiency of monitoring algorithms, and no practical system justifies this limitation.

The Distributed Online Tracking Model [17] is another unidirectional, synchronous framework that introduces intermediary nodes (or relays) between observers and C. These relays, acting as cluster heads in WSNs [5], are responsible for local aggregation before forwarding data to C. This model emphasizes competitive analysis, comparing its online algorithms against an optimal offline solution. However, it also suffers from the assumption of unidirectional communication channels, which is required to achieve online performance bounds.

Lastly, the Online Monitoring Model [13,14] addresses the problem of tracking (approximately) the top-k greatest values from distributed nodes in a synchronous environment. Within each monitoring round, nodes are allowed to send a polylogarithmic number of messages to C using synchronized *sub-rounds*. This model also incorporates a broadcast channel, counted as a single message, to enhance communication. However, this synchronization assumption limits its applicability to specific types of distributed systems, and no experimental validation was provided in prior studies.

2.2. The All-Values-Tracking (AVT) problem

The AVT problem consists in maintaining at C an approximation of the current value for every node and every monitoring round, where observed values are assumed to arrive at the distributed nodes in discrete rounds.[1] See Fig. 1 for an illustration of AVT monitoring with 4 distributed nodes, one coordinator C, and 50 rounds; two monitoring strategies, "baseline monitoring [8]" and "simple approximation [9]", are presented (cf. Sections 2.3 and 3.3) along with the monitoring views at C.

We highlight that when a highly communication-efficient solution for AVT is presented, it can be extended to efficiently monitor any function f at C. This can be achieved without fine-tuning the monitoring logic to f, by simply computing f continuously over the (approximated) tracked values at C. As this work focuses on a general CDM setting rather than providing solutions for a specific monitoring function f, more efficient monitoring algorithms may exist for some f. Our design thus offers *query flexibility* by allowing new aggregation queries to be implemented without redesigning the monitoring system. For instance, abnormal values can be detected when all values are being tracked. Beyond supporting aggregation functions, note that tracking all values from all nodes in real time is also essential in many practical scenarios. For example, in a smart city scenario, vehicle tracking [28] requires real-time access to the position of each node to enable traffic flow optimization, safety monitoring, and incident detection. Similarly, in sensor networks, IoT devices often need to report raw, full-resolution readings (e.g., environmental or health monitoring [29]) rather than data summaries, since the original values may feature anomalies or correlations required for later analysis. In industrial, performance-critical systems — such as core mobile broadband networks processing millions of packets per second — efficient monitoring of distributed nodes is essential, as it must introduce only minimal overhead when transmitting statistics over the network, which is shared with dense operational

[1] Rounds may have different lengths that vary across nodes, as long as the coordinator is aware of the start of each round. Hereafter, we refer to rounds using a discrete timestamp $t \geq 1$, regardless of which monitored node is involved. Cf. Section 4.6 for how round synchrony is handled in our system model.

traffic. Consequently, reducing the communication cost of monitoring directly contributes to improved overall system performance as shown in [3]. In large-scale distributed systems such as cloud computing and data centers, comprehensive and high-precision AVT across all nodes is critical for service reliability and troubleshooting. However, continuously collecting massive amounts of probe data severely consumes network communication and computational resources, potentially even undermining core operations. Therefore, balancing monitoring precision against communication overhead without downgrading system efficiency has become a core problem. To tackle this, architectures like on-demand partial deep analysis and load-adaptive regulation are proposed in [30–33].

While there are efficient solutions in the CDM space [10–12,34,35] that significantly reduce data transfer over the network, these methods are not applicable to the AVT problem. This limitation arises because these approaches rely on the interchangeability of data sources, meaning that a value observed at one node can be treated as equivalent to that observed at another. In contrast, the AVT setting requires each individual data stream to be tracked separately.

One of the most practical approaches to AVT is *adaptive filtering* [36], which transmits only values that fall outside a predefined per-node interval. It has been extensively studied in the context of IoT devices [37]. While it yields significant reductions in communication costs, it often results in a loss of accuracy, whether in value accuracy by filtering minor fluctuations or in time accuracy. However, as demonstrated in our evaluation, it can be further refined by considering both data variation (not all nodes benefit equally from the same model) and data evolution (prediction models may need to adapt over time to better align with changing data trends).

Another well-studied technique is Geometric Monitoring [34,35], which enables efficient threshold-based monitoring of functions computed over network-wide aggregates. However, these global approaches are not directly applicable to AVT, which focuses on per-node data granularity. In addition to analytical approaches, heuristic solutions for distributed queries have also been explored [36], as well as algorithms for tracking popular items [38]. These heuristic methods prioritize reducing communication costs while maintaining acceptable accuracy, but they are generally limited to specific query types and do not address the per-node tracking challenges inherent in AVT.

To summarize, the inherent complexity of AVT lies in the need to monitor all individual streams independently, without leveraging data redundancy across nodes. Thus, there is no one-size-fits-all solution for arbitrary input data, which previous CDM approaches have generally targeted. The distributed nature of AVT compounds this challenge, requiring solutions that prioritize node-to-coordinator communication while remaining agnostic to the presence or absence of other nodes. This necessitates more sophisticated, communication-efficient strategies that can handle the non-transferable nature of events across distributed nodes.

2.3. Forecast-based monitoring

Simple Approaches. The most straightforward approach to CDM is for each node to directly forward all its observations to C, which corresponds to the **TinyDB** [8] strategy and serves as our baseline monitoring method. However, this approach quickly becomes impractical as the number of nodes or the volume of observations increases, leading to scalability issues [3,10]. Two common techniques to address this challenge are reducing the polling frequency and sampling values from only a subset of nodes. Reducing the polling frequency mitigates scalability concerns but sacrifices responsiveness. The main drawback of polling is its dependency on the chosen frequency: a high rate can easily overload the network, while a low rate may result in significant losses in monitoring accuracy [10]. Sampling, on the other hand, lowers monitoring overhead by collecting data from only a subset of nodes. While this approach reduces communication, it is unsuitable for the AVT problem, where every individual stream must be monitored.

State of the art. In [15], the authors outline a framework for communication-efficient distributed monitoring using predictive modeling. Similar to our proposal, the approach leverages predictions at C and sends real values only when the deviation from predictions exceeds a predefined threshold. It employs an AR model, where parameters are transmitted instead of raw data. Initially, nodes run multiple AR models with varying lags and use a racing mechanism based on the Hoeffding bound to evaluate and discard underperforming models. Ultimately, each node maintains a single optimized model, ensuring efficient and adaptive monitoring. The main differences compared to our approach are: (1) their reliance on a single AR model, whereas we consider the prediction model as a *plugin* that may have several possible implementations, and (2) their disregard for the evolution of data streams, in contrast to our data-aware forecasting approach that adapts to different data and network conditions.

Prediction-based data reduction has also been extensively studied in the context of WSNs [5]. A key difference from our setting is that WSN models typically assume a shared communication medium among sensor nodes, allowing values transmitted by nearby sensors to aid in data reduction. This assumption limits the feasibility of per-node prediction schemes [5], as they are often considered too energy-intensive to operate on individual sensor nodes. Instead, prediction tasks are usually offloaded to a cluster head or to C, which reduces the potential communication savings. For instance, in [19], the authors propose the approximate framework *Ken*, which employs probabilistic models to estimate a probability density function over node values and exploits spatial correlations among distributed nodes through inter-node communication. However, in the special case of the Disjoint-Cliques model with the maximum clique size restricted to one (DjC1), spatial correlations are no longer leveraged, as nodes are treated independently.

In contrast, our work considers a more general distributed setting where communication channels between nodes and the coordinator are private. As their work is compared with TinyDB [8] and Simple Approximation [9], we also utilize these methods for comparison with our approach. Furthermore, the datasets we use often exhibit lower correlations between the observations of distributed nodes, making our approach better suited for such scenarios.

The related problem of data stream compression [20,39,40], particularly studied in sensor networks [16], is also worth noting. The key distinction from our problem lies in the *real-time* dimension of monitoring. If the sole focus were on the data itself, nodes could compress their observations and transmit the compressed data at large intervals, significantly reducing communication with the coordinator. However, this would prevent the coordinator from tracking node values in real time, since it would need to wait for the next batch of compressed data before updating its view. In contrast, our approach ensures that C maintains a continuously updated, though possibly slightly inaccurate, global view of the system at every round. For many applications that require continuous monitoring of resources, such delays are unacceptable.

3. Error-bounded forecast-based monitoring

In this section, we introduce our monitoring framework, which is based on maintaining aligned time series prediction models at both the nodes and C. By treating the observations at each node as a time series, we can capture the underlying characteristics of evolving patterns with limited required prior knowledge, making the approach generalizable to any data stream. In each round, every node applies a monitoring strategy to decide whether to transmit an update to C or remain silent. We describe below how the monitoring strategy in FEDAMON operates.

3.1. Error-bounded value tracking

It is evident that striving for an error-free solution to the AVT problem is impractical: due to the unpredictable nature of stream changes over time, the only viable approach to *perfect* AVT (i.e., error-free) would be to transmit every value that differs from the last shared value. Instead, we adopt a more efficient strategy by permitting the coordinator to tolerate a small, bounded error for each stream independently, with this error constrained in absolute terms by a predefined absolute error upper bound. That is, for any given times $t \geq 1$, the value $\hat{v}_t^i$ that C has as estimate for the value on s^i is within ϵ of its true observation v_t^i, i.e.,

$$|\hat{v}_t^i - v_t^i| \leq \epsilon, \tag{1}$$

where $\epsilon \in \mathbb{R}^+$ is defined as a fraction of the range R_D, which captures a sufficient proportion of percentiles across all relevant data streams. For instance, a modest error threshold may be set to $15\% \cdot R_D$, while a more restrictive one would correspond to $1\% \cdot R_D$. Hereafter, we refer to Eq. (1) as the *error constraint* applied to each s^i. If not otherwise stated, ϵ is assumed to be identical across all nodes in the system and constant over time, referred to as a *static error threshold*.

3.2. Monitoring strategy

The monitoring decision is guided by the current monitoring strategy, which balances communication efficiency with the need for accuracy. The primary objective is to reduce unnecessary transmissions while ensuring that C maintains an approximate but sufficiently accurate view of the observed data streams. Our system is built on the principle of forecasting the evolution of the monitored values to reduce communication. As long as the constraint in Eq. (1) holds at time t, no update message (event) is needed for that round. At every round, observations are stored locally in a circular (ring) buffer L, referred to as the *local buffer*, with a maximum size[2] of $D \geq 1$. The exact value of D may depend on the prediction model but remains constant during execution. Additionally, when dynamic error control is enabled (cf. Section 4.4), a secondary, larger buffer is maintained to support range adaptation.

If the observed value deviates significantly from what C currently knows (i.e., the constraint is violated), the node sends an *update message*. When a violation occurs on node s^i at time t, both the observed value v_t^i and the previously buffered values from L are sent to C. Sending L is important as C and node s^i utilize those previous observations to estimate new model parameters for predicting future node s^i's values $\hat{v}_{t'}^i$ for $t' > t$. An alternative design described and analyzed in Section 3.4 forwards only the necessary model parameters to C instead of L. After sending an update message to C, the buffer L is cleared to save memory for new observations.

Our proposed framework is a 2-layer hierarchical model in which all nodes are directly connected to the central coordinator C. In real-world systems, it can be naturally extended to additional hierarchical layers, for example by running our monitoring framework both at a cluster level of aggregation between the end nodes and C, and again at the central coordinator C for a 3-layer hierarchy.

The communication benefit is measured by the *communication ratio* ρ, defined as the proportion of update messages sent to C, regardless of the information contained in those messages (i.e., one message may carry multiple values). For T denoting the number of timesteps in the studied period (assuming all nodes share the same monitoring period), it is calculated as:

$$\rho = \frac{Q}{T \cdot n}, \tag{2}$$

where Q is the number of updates sent by the monitoring strategy during T. If the system achieves a ρ as in Eq. (2), then when accounting for the cost of protocol headers (i.e., not only the data payload but also the packet header), the effective communication reduction is bounded by ρ together with the header size. In other words, the practical communication reduction is lower than ρ, and we further quantify this difference in Section 3.4.

3.3. Prediction models

Since observations vary across nodes, we provide multiple options for models to capture local observation behaviors at each node and at the coordinator C. By design, our framework is compatible with generic prediction models, allowing additional models to be incorporated to adapt to different data streams. This means that while the specific models presented later are effective, alternative models can also be employed within the framework. We present six well-documented prediction models (`SA`, `AR`, `LMS`, `PLA`, `Holtl`, and `Holts`) included in our standard solution. All models follow the same monitoring strategy: when s^i violates the constraint at time t, all buffered values L are sent to C. Both C and s^i then update their model parameters using the new data, including L, for predictions starting from time $t+1$. Since time-series prediction often relies on previously predicted values, some models (precisely, all introduced here except `SA` and `PLA`) gradually replace true observations with their own past predictions when no updates occur. The following sections describe in detail how model parameter calculation is performed within this framework. In the following, $v_{t'}$ represents what the model considers to be the "true" observation at some time $t' < t$, and $\hat{v}_t$ denotes the prediction made by the model for time t. In other words, we intentionally omit the node's identity i in the model's view for two reasons: (1) the model is agnostic to the node's identity, as each node runs its own prediction model independently, and (2) v_t may not correspond to the actual value for a specific node i (i.e., v_t^i) e.g., v_t itself may be an earlier prediction generated by the model.

3.3.1. Prediction model template

Let M be a model used for forecasting future values based on past observations. In our design, M must implement the following interface:

- M.`minValues` specifies the minimum number of past observations (i.e., exact values recently read) required for M to produce its first forecast. In other words, M must receive at least M.`minValues` observations before it can perform its initial prediction.[3]
- M.`predict`(t) forecasts the value $\hat{v}_t$ for the observation v_t that will occur at timestamp t. Following the definition of M.`minValues`, we assume M.`predict`(t) is called after receiving at least M.`minValues` values.
- M.`update`(t, L) updates the model M using the current timestamp t and an ordered list L, which M treats as the consecutive set of true observed values $L = \{v_{t-|L|+1}, \ldots, v_t\}$ up to and including timestamp t. If the addition of these $|L|$ samples brings the total number of observations to at least M.`minValues`, this operation updates M's internal state to improve the accuracy of forecasts for future timestamps $t' > t$ by recalculating model parameters. Otherwise, only M's internal buffer is updated.

[2] The buffer size is limited to reduce memory usage and data transmission for improved performance, although it would be straightforward to store all values that do not trigger an update if desired.

[3] M.`minValues` is mainly a constraint for the first prediction. Afterward, all previously seen values can be memorized by M, which is responsible for always maintaining a buffer of M.`minValues` in its local state.

3.3.2. Simple Approximation (SA)

The most basic forecasting approach produces a static prediction based on the last updated observation, i.e., it assumes that local observations remain constant over time. In other words, the prediction $\hat{v}_t$ for $t \geq t_{prev}$ does not change during the interval $[t_{prev}, t]$, where t_{prev} is the last time the node violated the constraint. The prediction model is thus simply given by:

$$\hat{v}_t = v_{t_{prev}}, \tag{3}$$

This model is trivial to implement, only requires maintaining a single parameter $v_{t_{prev}}$ and we have M.minValues = 1 for SA. This model, referred to as Approximate Caching [9], is one of the baseline methods introduced in [19]. In our study, we also implement it as the simplest forecasting model within our framework.

3.3.3. Auto-Regressive Model (AR)

Auto-Regressive model [21] is a widely used time-series prediction model. AR can be employed in our system, with parameters that can be updated online as new observations become available while remaining computationally efficient. The predicted value $\hat{v}_t$ at each node is based on a linear combination of past observations v_{t-j}, expressed as an AR model of *lag* ℓ:

$$\hat{v}_t = \delta_t + \sum_{j=1}^{\ell} w_{j,t} v_{t-j}, \tag{4}$$

where $\mathbf{w} = (\delta_t, w_{1,t}, w_{2,t}, \dots, w_{\ell,t})$ is the set of $\ell + 1$ model parameters with δ_t being a constant additive term. For M being an AR-ℓ-k prediction model, the model parameters $\mathbf{w}_M$ are estimated using Ordinary Least Squares (OLS) by minimizing the sum of squared errors (SSE) over the last $k - \ell$ known samples, i.e., $\mathbf{w}_M = \arg\min_{\mathbf{w}} \text{SSE}(\mathbf{w}, t, k, \ell)$ with

$$\text{SSE}(\mathbf{w}, t, k, \ell) = \sum_{r=t-(k-\ell)}^{t-1} \left[v_r - \left(\delta_t + \sum_{j=1}^{\ell} w_{j,t} v_{r-j} \right) \right]^2.$$

An AR-ℓ-k model, as defined above, requires k previous samples $\{v_{t'}\}_{t-k \leq t' < t}$ to calculate its parameters, setting M.minValues = k for AR-ℓ-k (if $k = 2\ell + 1$). This is because each predicted value $\hat{v}_j$ depends on samples that mostly overlap with $\hat{v}_{j-1}$ except for the most recent value (i.e., $\hat{v}_{j-1}$ requires $v_{j-\ell}$ in addition to more recent samples). Thus, the oldest sample in the SSE, $v_{t-(k-\ell)}$, implies that v_{t-k} is needed to calculate the weights for $\hat{v}_t$. A closed-form solution can be obtained by solving OLS, which requires at least $k \geq 2\ell + 1$ observations (i.e., $k - \ell \geq \ell + 1$), thus it can start prediction after $2\ell + 1$.

3.3.4. Least Mean Square Filter (LMS)

LMS is one of the most widely applied adaptive filters [41], known for its low computational overhead and memory usage, making it well-suited for predictive tasks. Essentially, LMS takes each observation at time t as input and computes the prediction as a linear combination of the last ℓ observations:

$$\hat{v}_t = \sum_{j=1}^{\ell} \theta_{j,t} v_{t-j}, \tag{5}$$

where $\boldsymbol{\theta} = (\theta_{1,t}, \theta_{2,t}, \dots, \theta_{\ell,t})$ is the set of model weights at time t. Initially, $\boldsymbol{\theta}$ is set to zero and is updated iteratively to approach the best suited weights. After collecting ℓ observations, the model can start making predictions. When a violation occurs at s^i, the prediction error is computed as:

$$e_t = v_t - \hat{v}_t.$$

The goal is to minimize $|e_t|$ as the cost function between the prediction and the true value. The weights are then updated in one iteration (which occurs when an update message is sent to C) as follows:

$$\theta_{j,t+1} = \theta_{j,t} + \eta\, e_t\, v_{t-j}, \tag{6}$$

where η is the step size, which must satisfy $0 \leq \eta \leq \frac{1}{E_t}$ to ensure convergence [22,42]. Here, we set $\eta = (\kappa E_t)^{-1}$, where κ is a fixed model parameter and E_t is computed as:

$$E_t = \frac{1}{\ell} \sum_{j=1}^{\ell} \left(v_{t-j} \right)^2.$$

This bound determines the upper limit for the step size η. Since our method enforces an error constraint, whenever update messages are sent, the model's ℓ parameters $\boldsymbol{\theta}$ are updated according to Eq. (6), where $\theta_{j,t}$ refers to the weight set that remained unchanged since the previous model update. Predictions in LMS require ℓ observations (i.e., M.minValues = ℓ for LMS). Prediction starts as soon as ℓ values are available, in contrast to the "normal" mode in [22], which requires additional iterations for $\boldsymbol{\theta}$ to first converge, that is, a fixed number of rounds below an error threshold, before predictions are used and observations cease to be transmitted.

3.3.5. Piecewise Linear Approximation (PLA)

Instead of using previous values themselves as input for prediction (i.e., using the output as the input for the next time step), we can compute the PLA over the data by representing the observations in the data stream as a function of time, modeled by line segments with a size of ℓ points. The time duration of one line segment spans the range $[t - \ell, t)$, where the points on the segment are represented as $\langle t - \ell, v_{t-\ell} \rangle, \langle t - \ell + 1, v_{t-\ell+1} \rangle, \dots, \langle t - 1, v_{t-1} \rangle$. Let the set of timestamps $\{t - \ell, t - \ell + 1, \dots, t - 1\}$ be denoted as X, and the corresponding set of values $\{v_{t-\ell}, v_{t-\ell+1}, \dots, v_{t-1}\}$ as Y. By extending the line segment of the best-fit line, we can estimate future values. The prediction at time t is expressed by the linear equation:

$$\hat{v}_t = at + b, \tag{7}$$

where a is the slope of the best-fit line given by the covariance between X and Y divided by the variance of X, i.e.,

$$a = \frac{\text{cov}(X, Y)}{\text{var}(X)}.$$

The intercept b is calculated as:

$$b = \mathbb{E}(Y) - a\mathbb{E}(X),$$

where $\mathbb{E}(X)$ and $\mathbb{E}(Y)$ represent the expected values of X and Y, respectively. When C receives an update triggered by an error constraint violation, it recalculates the parameters a and b. Because estimating these parameters requires ℓ data points, we set M.minValues = ℓ for PLA.

3.3.6. Holt Linear (Holtl)

Exponential smoothing [43] is widely used for time series forecasting, where it calculates the weighted average of past observations, with weights decaying exponentially as the observations get older. In other words, the most recent values are associated with higher weights. Holtl [24] is a type of exponential smoothing method that accounts for a trend (slope) in the data. The equations for predicting the value h timestamps away from the current time t are:

$$\begin{aligned} \hat{v}_{t+h} &= s_t + hb_t \\ s_t &= \alpha v_t + (1 - \alpha)(s_{t-1} + b_{t-1}) \\ b_t &= \beta(s_t - s_{t-1}) + (1 - \beta) b_{t-1}, \end{aligned} \tag{8}$$

Here, s_t represents the smoothed value at time t, and b_t is an estimate of the trend at time t. α $(0 \leq \alpha \leq 1)$ is the data smoothing factor, and β $(0 \leq \beta \leq 1)$ is the trend smoothing factor. As shown in Eq. (8), s_t is a weighted average of v_t and the one-step-ahead forecast $\hat{v}_t$ at time t. Similarly, b_t is a weighted average of the estimated trend at time t, calculated from $s_t - s_{t-1}$ and b_{t-1}. The smoothing factors α and β determine how much weight is placed on recent values, and their impact is evaluated in Section 5. The initial values s_0 and b_0 are

estimated by minimizing SSE over ℓ training samples. Consequently, the required number of initial input values is ℓ for `Holtl`. In our framework, h corresponds to the time difference between the current time and the last time an update message was sent.

3.3.7. Holt Winters (Holts)

`Holts` [25] is another type of exponential smoothing method that considers seasonality by introducing one more seasonal component c_t. When c_t is calculated in the same units as the original observation, it is called Holt-Winters' additive method and for all time t in one whole seasonal period t, the corresponding c_t are added up to approximately zero. The equations for forecasting a value h timestamps away from time t is:

$$\begin{aligned}\hat{v}_{t+h} &= s_t + hb_t + c_{t+h-\ell(k+1)}\\ s_t &= \alpha(v_t - c_{t-\ell}) + (1-\alpha)(s_{t-1} + b_{t-1})\\ b_t &= \beta(s_t - s_{t-1}) + (1-\beta)b_{t-1}\\ c_t &= \gamma(v_t - s_{t-1} - b_{t-1}) + (1-\gamma)c_{t-\ell},\end{aligned} \tag{9}$$

where γ $(0 \leq \gamma \leq 1)$ is the seasonality smoothing factor, ℓ the period of seasonality, $k = \lfloor \frac{h-1}{\ell} \rfloor$ to make sure the predicted value is always calculated from the last seasonal period of the observations, i.e., from $[t-\ell+1, t]$. s_t is a weighted average between the seasonally adjusted part $(v_t - c_{t-\ell})$ and one-step-ahead forecast $\hat{v}_t$ (same as `Holtl`) at time t. b_t is identical to `Holtl` as well. The seasonal component c_t is a weighted average between $(v_t - s_{t-1} - b_{t-1})$ and the seasonal component of the ℓ time ago. The initial values s_0, b_0, c_0 are estimated by using Maximum Likelihood Estimation [44] for the past two seasonal samples. Hence, for $M =$ `Holts`, M.`minValues` $= 2\ell$. The parameters α, β, γ are evaluated in Section 5.

3.4. Communication analysis

There are two variants of our framework with different impacts on communication overhead. The default, *Configuration 1*, requires nodes to send the buffer L to the coordinator, which then calculates the corresponding model parameters (coefficients) to make predictions. The alternative, *Configuration 2*, has nodes send all the values needed for forecasting directly, without requiring the coordinator to calculate model parameters. In this case, the transmitted values include both the model parameters and past observations (the output variable depends on its own previous values in `SA`, `AR`, `LMS`, `Holtl`, and `Holts`). For a fair comparison, Table 4 lists the number of values that must be sent in each configuration. The results indicate that *Configuration 2* reduces communication overhead only for `PLA` when $\ell > 2$. For other models, the communication cost of *Configuration 2* is not lower than that of *Configuration 1*, because only `PLA` does not use past observations as independent variables. Additionally, when updates are frequent, *Configuration 1* sends only the buffered values (usually fewer than the upper bound in Table 4), making *Configuration 2* less efficient since it redundantly sends values that may have been transmitted previously. Nevertheless, *Configuration 2* has the advantage of bypassing the parameter optimization phase entirely, which can be beneficial in rare situations involving message loss. By directly transmitting the model parameters together with the required past observations for forecasting, the system can recover immediately and resume consistent operation without first waiting for the buffers to be cleared. Therefore, *Configuration 2* should be viewed as a robustness-oriented alternative rather than the primary communication-optimized strategy. For these reasons, in this paper, we focus on analyzing communication reduction using our prediction-based method under *Configuration 1*.

To ease the analysis, consider as a baseline a protocol that would transmit every observed value to the coordinator in a separate message. Hence, the baseline cost for the network usage (communication cost accounting for the protocol overhead) is

$$U_{\text{baseline}} = T \cdot n \cdot (H + V),$$

Table 4
Number of values required to transmit to update the model for the two configurations of each model type. (upper bound for *Configuration 1*).

Model type	Configuration 1	Configuration 2
`SA`	1	1
`AR`-ℓ-k	k	$k + \ell + 1$
`LMS`-ℓ-κ	ℓ	2ℓ
`PLA`-ℓ	ℓ	2
`Holtl`-ℓ-α-β	ℓ	ℓ
`Holts`-ℓ-α-β-γ	2ℓ	2ℓ

where H is the protocol header overhead (in bytes) and V the size of one observed value (in bytes) and both are assumed constant. Let us call $U_{\text{baseline}}/(T \cdot n) = H + V$ the per-value overhead. Let us consider now that our system achieves a communication ratio of ρ as per Eq. (2) while denoting r_i for the number of rounds where s^i sends an update message, i.e., let $q_{i,j}$ be the jth update message from s^i and $|q_{i,j}| \leq D$ be the number of values contained in message $q_{i,j}$ following *Configuration 1*. For the sake of simplicity, let us assume the system ends with a signal sent to the node to send a final update and that a sequence of buffered values always fits in a single transmitted packet (which can fit 160–320 values even considering large protocol overhead). For the models utilized in this work, D is constrained to a fixed value, which depends on the selection of model parameters. Since we have limited the size of the local buffer on each node, we can bound the amount of sent values:

$$\sum_{i=1}^{n} \sum_{j=1}^{r_i} |q_{i,j}| \leq Q \cdot D. \tag{10}$$

Moreover, since obviously there cannot be more buffered values than observed values, we also have:

$$\sum_{i=1}^{n} \sum_{j=1}^{r_i} |q_{i,j}| \leq T \cdot n. \tag{11}$$

By using Eqs. (10) and (11), and assuming $c = \mathcal{O}(1)$ denotes the small number of bytes required for additional information in an update message (3 variables and a 2-bit flag in our system, explained in later sections), we can calculate the network usage as:

$$\begin{aligned}U &= \sum_{i=1}^{n} \sum_{j=1}^{r_i} \left(H + V \cdot |q_{i,j}| + c\right)\\ U &= \sum_{i=1}^{n} \sum_{j=1}^{r_i} (H + c) + V \sum_{i=1}^{n} \sum_{j=1}^{r_i} |q_{i,j}|\\ U &\leq Q \cdot (H + c) + V \min\{Q \cdot D, T \cdot n\}.\end{aligned}$$

Hence the network usage in comparison to the baseline can be upper-bounded by the communication ratio up to constants depending on the header and value sizes:

$$\begin{aligned}\frac{U}{U_{\text{baseline}}} &\leq \frac{Q \cdot (H + c) + V \cdot \min\{Q \cdot D, T \cdot n\}}{T \cdot n \cdot (H + V)}\\ &\leq \frac{\rho \cdot (H + c) + \min\{\rho \cdot D, 1\} \cdot V}{H + V}\\ &\leq \rho \left(\frac{H + c}{H + V}\right) + \frac{V}{H + V}.\end{aligned}$$

In other word, as long as the protocol overhead H is relatively large in comparison to the size of a single value V and the extra few bytes c, the network usage ratio can be well approximated by ρ. Note that in the above calculations, we have abstracted connection and retransmission overhead which in any case will influence in a similar manner both the baseline and FEDAMON's network usage.

4. Data-aware model selection

In this section, we propose a data-aware model selection framework to adapt the prediction model to the diversity of data stream

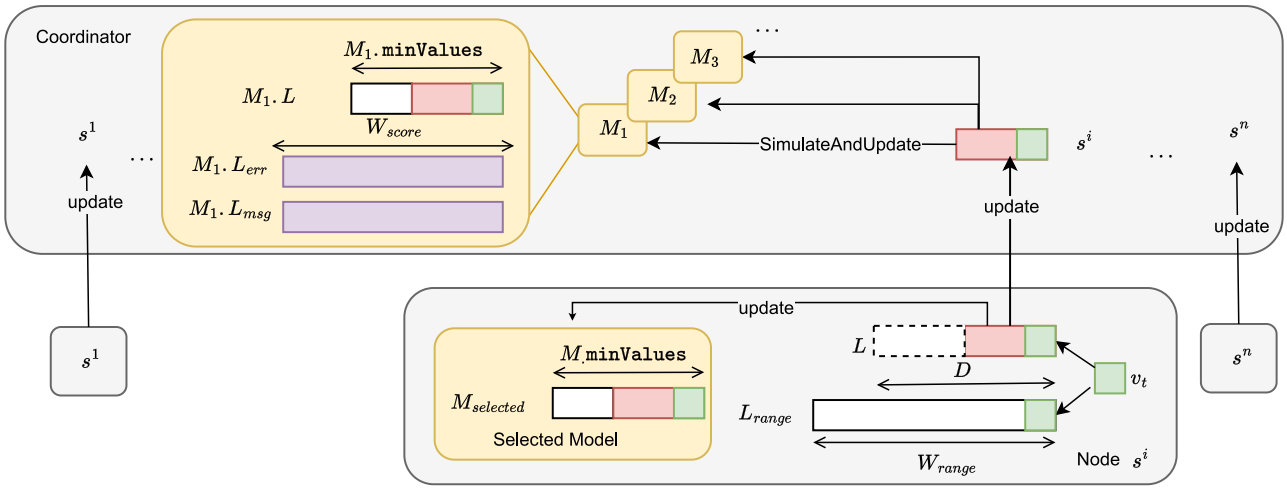


Fig. 2. Overview of our model simulation and update process, illustrating the various circular buffers (L, L_{range}, as well as $M.L$, $M.L_{err}$, and $M.L_{msg}$ associated to a prediction model M), together with their respective maximum sizes: D, W_{range}, M.minValues and W_{score}.

characteristics. Without prior knowledge of the observations, selecting an optimal model for the nodes can be challenging. Additionally, due to the skewed distribution of data across distributed nodes, applying a single model type is not suitable for all nodes. Thus, we advocate for a data-aware model selection approach based on model score, enabling automatic control in determining the appropriate model type and following the data streams continuously generated on the nodes. The selection is done by C using *node-wise* model selection over a list of candidate models denoted as $\mathbb{M} = \{M_1, M_2, M_3 \dots\}$ simultaneously. Hence, s^i runs a personalized model type independently, without requiring alignment with other nodes, while being guided by messages from C regarding model type switching.

4.1. Circular buffers

Our implementation uses a set of circular buffers that cache recent values at different levels; below, the details of each:

- L denotes a local buffer (first introduced at Section 3.2) maintained at every node that gets reset upon every update. Every value observed at the node is added to L, and the buffer can grow up to size D, a parameter set by the coordinator. Once an error violation occurs, L is transmitted to C.
- L_{range} is a larger buffer maintained at each node that contains the most recent W_{range} values. That buffer is never flushed and is used to calculate a dynamic error threshold per node as described in Section 4.4. When the error is static and directly provided as input, this buffer can be omitted.
- $\{L^i[M], L^i_{err}[M], L^i_{msg}[M]\}_{1\leq i\leq n, M\in\mathbb{M}}$ is a collection of $3n|\mathbb{M}|$ buffers maintained at the coordinator to use for calculating a score for each candidate model $M \in \mathbb{M}^i$ for each node s^i, with $\mathbb{M}^i$ being a pool of models associated to node s^i. To ease the code description (see Algorithm 1), for a given instantiated model M, we denote those buffer as $M.L$, $M.L_{err}$, $M.L_{msg}$ but those fields are not accessible by the model M for calculating its predictions.

Fig. 2 shows the buffers of different levels. In the pseudo-code, buffer(k) initializes a circular buffer of size up to k, and if L is a buffer, $L \leftarrow L+\{v\}$ appends the value v to the buffer L. Once L's size reaches k, each new value flushes out the oldest value in the buffer. The notation $L \leftarrow \varnothing$ means L is reset (or emptied) and $L \leftarrow L \cup L'$ appends L' into L while keeping L under its maximum size (equivalent to adding each element of L' to L one by one).

Algorithm 1: Model simulation and calculation of $score^{\epsilon}_{\alpha}(M, t, \mathcal{V})$

Function: `SimulateAndUpdateModel`$(M, t, L, \epsilon, \alpha)$
  // Assumption: $L = \{v_{t_0}, \cdots, v_t\}$ with $t_0 = t - |L| + 1$
  for $t - |L| < j \leq t$ **do**
    $M.L \leftarrow M.L + \{v_j\}$;
    if $j \geq M$.minValues **then**
      $\hat{v}_j \leftarrow M$.predict(j);
    if $j < M$.minValues *or* $\left|\hat{v}_j - v_j\right| > \epsilon$ **then**
      M.update$(j, M.L)$;
      $M.L \leftarrow \varnothing$;
      $n_{msg} \leftarrow 1, error \leftarrow 0$;
    else
      $n_{msg} \leftarrow 0, error \leftarrow \left|\hat{v}_j - v_j\right|$;
    $M.L_{err} \leftarrow M.L_{err} + \{error\}$;
    $M.L_{msg} \leftarrow M.L_{msg} + \{n_{msg}\}$;
  $acc \leftarrow 1 - \left(\frac{1}{W_{score}} \Sigma_{e\in M.L_{err}} e\right)/\epsilon$;
  $omiss \leftarrow 1 - \Sigma_{m\in M.L_{msg}} m/W_{score}$;
  M.score $\leftarrow \alpha * acc + (1-\alpha) * omiss$;

4.2. Model score

As the score calculation procedure is performed independently for each node, the node index i is omitted from the notation throughout this section for better clarity.

Score calculation with an updated model. Let us denote $\mathcal{V} = \{v_j \mid t - W_{score} + 1 \leq j \leq t\}$ the set of the last W_{score} recently observed true observations, with $|\mathcal{V}| = W_{score}$, and let us assume here that M is a model that has been updated exactly at timestamp $t - W_{score}$. The selection process is done by considering the performance of each candidate model upon receiving an update message. Specifically, $\forall M \in \mathbb{M}$ at time t, the performance of candidate models is calculated using the *score* function (cf. Algorithm 1), which calculates a trade-off between the monitoring accuracy acc and that of messages omission rate *omiss* over the last W_{score} rounds. Here follows how a model's score is computed. At time t, a model $score^{\epsilon}_{\alpha}(M, t, \mathcal{V})$ for model M, a set $\mathcal{V}$ of recent exact values and error threshold ϵ is computed as:

$$score^{\epsilon}_{\alpha}(M, t, \mathcal{V}) = \alpha \cdot acc^{\epsilon}(M, t, \mathcal{V}) +$$

Algorithm 2: Forecast-based Error-bounded Data-Aware MONitoring: **FEDAMON-Node**

Input: absolute error upper-bound ϵ
(or an error tolerance δ, W_{range} and τ_ϵ when using dynamic range R'_D; ϵ is initially set to 0 then)

Node $s^i \in \mathbb{S}$ **executes:**
// Initialize buffer(s) and parameters
$M, D \leftarrow$ receive($\langle$"init", $M_{selected}, D\rangle$);
$L \leftarrow$ buffer(D), $L_{range} \leftarrow$ buffer(W_{range}), $t' \leftarrow D$;
for $t \geq 1$ **do**
  $v_t \leftarrow$ read(t) ; // monitored stream
  $L \leftarrow L + \{v_t\}$; // $|L| \leq D$
  $L_{range} \leftarrow L_{range} + \{v_t\}$; // $|L_{range}| \leq W_{range}$
  if $t <$ M.minValues **then**
    send(C, $\langle$"update", $\{v_t\}, t-1, t, \epsilon\rangle$);
    M.update($t, \{v_t\}$);
  else
    if receive($\langle$"switch", $M_{top}\rangle$) **then**
      $M \leftarrow M_{top}$; // up-to-date model
    $\hat{v}_t \leftarrow M$.predict(t) ;
    if $|\hat{v}_t - v_t| > \epsilon$ **then**
      if $t \geq \tau_\epsilon$ **then**
        $\epsilon \leftarrow$ calculateRange(L_{range});
      send(C, $\langle$"update", $L, t', t, \epsilon\rangle$);
      M.update(t, L);
      $L = \emptyset$, $t' \leftarrow t$; // clear buffer

$$(1-\alpha)\cdot omiss^\epsilon(M,t,\mathcal{V}),$$

where $0 \leq \alpha \leq 1$ is the *control factor* that weights the contribution of monitoring accuracy. In the score's calculation, setting $\alpha = 0$, the emphasis is put on reducing communication overhead, while setting $\alpha = 1$ places all the importance on accuracy. Thus, tuning α enables the ability to take both communication and accuracy into account.

In detail, acc^ϵ is calculated over the last W_{score} rounds as follows:

$$acc^\epsilon(M,t,\mathcal{V}) = 1 - \frac{\sum_{v_t \in \mathcal{V}} (|\hat{v}_t - v_t|)}{\epsilon W_{score}},$$

noting that if there is an update message, $\hat{v}_t$ is replaced with v_t on the coordinator, thus the error is zero at that round; as for $omiss^\epsilon$, the fraction of omitted messages among all observations over the last W_{score} rounds, it is calculated as:

$$omiss^\epsilon(M,t,\mathcal{V}) = 1 - \frac{Q^\epsilon_{\mathcal{V},M}}{W_{score}}$$

where $Q^\epsilon_{\mathcal{V},M}$ is the potential number of updates that would have been transmitted during the last W_{score} rounds if model M were used for prediction, calculated by the coordinator using the error constraint that is introduced at Eq. (1).

At last, let us observe that both acc and $omiss$ lie in the interval $[0,1]$, hence $score$ is bounded to the same interval. The extremal cases are occurring when $acc = 0$ and every prediction reaches the error threshold ϵ while $acc = 1$ indicates not a single prediction contain any error (note that we compute the accuracy over the predicted values and not the values eventually transmitted to C). Similarly, $omiss = 0$ indicates every single prediction broke the error constraint whereas $omiss = 1$ means not a single prediction occurred beyond the set error bound and no update message was transmitted to C during the last W_{score} rounds.

Score calculation for out-of-sync models. Assume an update is triggered at time t, and the previous "global" update (i.e., the last execution of Algorithm 1) occurred at time t'. Models may be out of sync with one another, meaning that the coordinator's simulation of each model must take into account the last state the model was in at time $t_{last} \leq t'$ when calculating its score for time t. The previously defined score can be computed at the coordinator for a given model M if M was last updated at time t', since in that case, the coordinator can easily run the "fresh" model over the values contained in $\mathcal{V}$ (for example, by storing those values at C).

To handle this, we use the previously introduced buffers $M.L$, $M.L_{err}$, and $M.L_{msg}$, which respectively store the values since the last internal update of M, the corresponding errors, and the update message indicators over the last (at most) W_{score} rounds. The buffer L transmitted to Algorithm 1 corresponds to the unprocessed values received at the coordinator when an update message arrives. Since L is the local buffer of a node s^i, it always contains consecutive true observations from s^i; L can grow up to size D, which is set larger than W_{score}.

This means that when updates are sent relatively frequently (i.e., when $|L| < D$), all simulated models are updated under the exact same conditions as they would be if executed on s^i, since C has access to all true observations between times t_{last} and t. However, when the selected model performs very well and $|L| = D$, some values recorded between times $[t'+1, t-|L|]$ are never transmitted to C, creating a *gap* in the true observations available at the coordinator. In this case, the simulation

Algorithm 3: Forecast-based Error-bounded Data-Aware MONitoring: **FEDAMON-Coordinator**

Input: absolute error upper-bound ϵ (initially set to 0 when using dynamic range), pool of candidate models $\mathbb{M}$, criterion ξ, score buffer size W_{score} and starting round for data-aware selection τ
Output: C's monitoring view $\{\hat{v}^i_t\}_{1\leq i\leq n, t\geq 1}$

$D_{\max} \leftarrow \max_{M\in\mathbb{M}} M$.minValues;
$D \leftarrow \max\{D_{\max}, W_{score}\}$; // Assumption: $\tau \geq D$
Coordinator C **executes in parallel for each** $s^i \in \mathbb{S}$
  $\mathbb{M}^i$ is a set of candidate models for node i;
  // Initial Setup
  $M^i_{selected} \leftarrow \mathbb{M}^i[0]$; // default model
  send(s^i, $\langle$"init", $M^i_{selected}, D\rangle$);
  for $M \in \mathbb{M}^i$ **do**
    $M.L_{err}, M.L_{msg} \leftarrow$ buffer(W_{score}), buffer(W_{score});
    $M.L \leftarrow$ buffer(M.minValues);
  $t^i_{update} \leftarrow 0$; // last update at node s^i
  // Forecast-based Monitoring Phase
  for $t \geq 1$ **do**
    // Estimation of v^i_t at C
    if $t \geq M^i_{selected}$.minValues **then**
      $\hat{v}^i_t \leftarrow M^i_{selected}$.predict($t$);
    // Model updating and switching
    if receive(s^i, $\langle$"update", $L, t', t, \epsilon\rangle$) **then**
      // Assumption: $t' = t^i_{update}$ to proceed
      $\hat{v}^i_t \leftarrow L[-1]$; // last value in L's buffer
      for $M \in \mathbb{M}^i$ **do**
        SimulateAndUpdateModel($M, t, L, \epsilon, \alpha$)
      if $t \geq \tau$ **then**
        $M^i_{top} \leftarrow \arg\max_{M\in\mathbb{M}^i} M$.score;
        if M^i_{top}.score $> M^i_{selected}$.score $+ \xi$ **then**
          send(s^i, $\langle$"switch", $M^i_{top}\rangle$) ;
          $M^i_{selected} \leftarrow M^i_{top}$;
      $t^i_{update} \leftarrow t$;

proceeds on a best-effort basis: each model continues running from its state at time t_{last}, accumulating predicted values into $M.L$ despite the gap in time. Once M violates the error constraint at some time $j \geq t-|L|+1$, $M.L$ is flushed, and M considers those buffered values as the "true observations" corresponding to timestamps $[j-|M.L|+1, j]$.

4.3. Node-wise prediction and model selection

Node's side. Algorithm 2 defines the behavior of a node in FEDAMON. The process is straightforward and follows the monitoring strategy described earlier. In short, each node s^i runs a single model to make predictions and reports to C whenever its error exceeds the defined threshold. Each node is first initialized by the coordinator with a buffer size D and an initial model M_0. Except during the initial rounds, when the current model may require more values than have been collected, a prediction is made in every round. When the error constraint is violated, an update message is sent to C, and the local buffer is cleared. Dynamic error control is discussed later in Section 4.4.

Coordinator's side. The model selection process is based on the coordinator C maintaining the list $\mathbb{M}$ and concurrently simulating each model upon receiving an update message.

Initialization: The coordinator initializes the maximum buffer size D to $\max\{W_{score}, D_{\max}\}$, allowing the local buffers to grow up to D. This setup ensures that (1) gaps (i.e., missed true observations at C) never overlap with timestamps used for model score calculation, and (2) C can update all candidate models to their most up-to-date state. The first model M_0 in the candidate pool is used as the initial selected model. If the pool contains only a single model, the data-aware component is not triggered and can be omitted. Initially, all buffers of the candidate models are created, and an "init" message is sent to the monitored node s^i with D and the initial model M_0. Before the default model M_0 has accumulated its minimum required number of inputs, each round results in the coordinator receiving an update from s^i, ensuring that the monitoring view remains error-free during these initial rounds.

Model Update and Simulation: Upon receiving an update message from the selected model $M_{selected}$ (Algo. 3, line 14), C updates the state and score of all candidate models in $\mathbb{M}$. For each $M \in \mathbb{M}$, `Simulate-AndUpdateModel` (Algo. 1) is invoked. Recall that for each model M, the coordinator maintains two buffers, $M.L_{err}$ and $M.L_{msg}$, each of size W_{score}, to store values required for computing acc and $omiss$. Each model M is simulated and updated using the true observations contained in L, which were transmitted in the latest update message. Predictions are used once t exceeds M's required number of inputs. To replicate the behavior the model would exhibit if executed on a node, M updates its internal parameters only when its prediction violates the error constraint. When this occurs, the values stored in $M.L$ since the last internal update are flushed, and M's parameters are refreshed. Once all values from L have been processed, M's score for the current timestamp t is computed based on the contents of $M.L_{err}$ and $M.L_{msg}$.

Model Selection: After τ rounds (a configurable parameter assumed to be larger than D) have elapsed, the data-aware selection process begins. Let $M^i_{top} \in \mathbb{M}^i$ denote the model at round t with the highest score, i.e., $M^i_{top} = \arg\max_{M \in \mathbb{M}^i} score^{\epsilon}_{\alpha}(M, t, \mathcal{V})$. Ties are resolved by selecting the model with the smallest index. The highest score is then compared with the score of the currently selected model running on s^i. If

$$score^{\epsilon}_{\alpha}(M^i_{top}, t, \mathcal{V}) > score^{\epsilon}_{\alpha}(M^i_{selected}, t, \mathcal{V}) + \xi$$

holds, where ξ is the model selection *criterion*, a new model is selected. A model-switching message $\langle$"switch", $M^i_{top}\rangle$ is then sent from C to s^i (Algo. 3, line 21). Since s^i retains its previous observations and has been configured with $D \geq D_{max}$, it can estimate the parameters of the newly selected model. Otherwise, as the studied models have relatively few parameters (cf. Table 4), we can also alternatively assume that the required parameters, along with the model type, are transmitted in the switching message received by s^i. Thus, in Algo. 2, after line 13, M corresponds to $M_{selected}$ on the coordinator's side. This model selection process is performed independently for each node in the system.

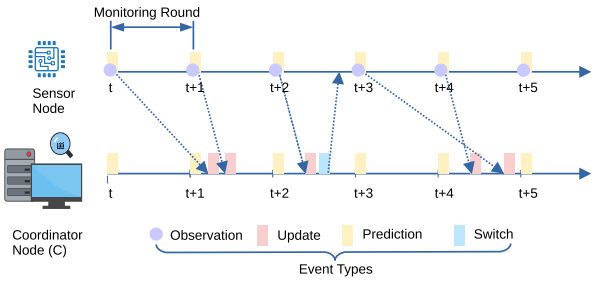


Fig. 3. Example of different types of events arriving in different orders with one single node and coordinator C.

4.4. Dynamic range

We introduce a method that enables fully automatic control of our data-aware system without requiring prior knowledge of R_D. Each node maintains a circular buffer L_{range} of maximum length W_{range} to store the most recent W_{range} observations. At each time t, the observation v_t is added to L_{range} (Algo. 2, line 7). Using the values in this buffer, the node calculates a dynamic absolute error upper bound ϵ after τ_ϵ rounds have elapsed (ϵ is initially set to 0 before that). The dynamic range is defined as

$$R'_D = \max_{v_t \in L_{range}} v_t - \min_{v_t \in L_{range}} v_t.$$

An error tolerance δ (as a fraction) controls the allowed deviation between the predicted value $\hat{v}_t$ and the actual observation v_t. The updated absolute error upper bound is then

$$\epsilon = \delta \cdot R'_D.$$

This ϵ is recalculated whenever a constraint violation occurs and sent from node s^i to the coordinator as part of the update message (Algo. 2, line 17). The dynamic error calculation starts after τ_ϵ rounds and thus without waiting for W_{range} observations.

If a static error ϵ is provided as input to the monitoring algorithms, lines 7 and 17 can be omitted.

4.5. Standard solution

To handle entirely new data streams at distributed nodes, we provide a *standard model list* for our framework designed to overcome the challenges posed by unknown data distributions and characteristics. Normally, selecting and tuning candidate models requires manual effort, which conflicts with our goal of fully automatic monitoring. In our standard solution, a fixed set of candidate models are already predefined with fixed types and parameter combinations, removing the need for manual adjustments. This standard solution performs reliably across heterogeneous datasets with varying data dynamics, making it especially suitable for completely new data streams. In the experimental section, we present a comprehensive evaluation to demonstrate the effectiveness of the proposed approach and to systematically analyze the impact of parameter configurations.

4.6. Robustness and event consistency

As stated in Section 2.1.1, our theoretical guarantees assume reliable communication and synchrony between the coordinator C and the distributed nodes. In this subsection, we briefly discuss how the framework behaves in practical scenarios where temporary communication irregularities (e.g., delayed, out-of-order, or loss events) may occur.

Table 5
Datasets with type of statistics, no. of nodes, total duration T, standard data range R_D, $\epsilon = 5\% \cdot R_D$, and model *criteria* ξ.

Dataset D	Statistic	# Nodes	T	Standard data range R_D	ϵ at $5\% \cdot R_D$	ξ
Ericsson	CPU usage	720	2022	$R_{\mathcal{E}} = 97$	4.85	0.01
Geolife	Vehicle speed	100	54,743	$R_{\mathcal{G}} = 120$	6.0	0.029
IntelLab	Sensor temperature	47	300,000	$R_{\mathcal{I}} = 44.6$	2.23	0.015
ACFS1	Power consumption	10	2920	$R_{\mathcal{A}} = 13$	0.65	0.01

Event handling. At the beginning of each monitoring round, a new observation event is generated at each node. If the error constraint is violated at time t on node s^i, the node sends an *update event* to the coordinator. This event corresponds to transmitting the message $\langle$"update", $L, t', t, \epsilon\rangle$ (Algorithm 2, line 9), where L denotes the local buffer of s^i, t' is the timestamp of the previous update event at node s^i, t is the current timestamp, and ϵ represents the current dynamic error threshold. Upon receiving an update event under normal (failure-free) conditions, the coordinator replaces the predicted value for node s^i at time t with the exact observed value (i.e., the last element of L), provided that t matches the current monitoring round.

Out-of-order events. With unreliable communication, update events may arrive out of order due to network delays. For instance, an update generated at time t may arrive after the update for time $t + 1$, or an update for time $t + 3$ may be received after the one for $t + 4$ (cf. Fig. 3). The coordinator can detect and properly process such situations by inspecting the timestamps t' and t contained in the message. Based on this information, it decides whether to process the event immediately or defer it until preceding updates have been handled. For clarity of presentation, we abstract from these low-level ordering details in the pseudo-code. Given typical monitoring periods and network latencies, such events are rare in practice. Moreover, mechanisms for handling out-of-order message delivery are well studied in distributed systems [45].

Loss events. In the presence of occasional message loss, temporary inconsistencies may arise. However, the coordinator maintains a collection of buffers $\{L^i[M], L^i_{err}[M], L^i_{msg}[M]\}_{1\leq i\leq n,\, M\in\mathbb{M}}$, which are continuously updated by subsequent correct events. As new consistent updates are received, any outdated or corrupted entries are eventually overwritten. Consequently, the system automatically recovers after a bounded delay and converges back to a consistent state. While reliable communication is assumed for the formal guarantees, the buffer-based design of FEDAMON provides practical robustness against transient communication irregularities.

Synchronization. Perfect clock synchronization between the distributed nodes and the coordinator C is not required. Since events are labeled with their corresponding monitoring period, the framework does not rely on exact alignment of local clocks across nodes and the coordinator. To mitigate potential clock drift over extended periods, a lightweight synchronization protocol may be executed infrequently (e.g., once per hour), as commonly adopted in practical distributed monitoring deployments [23]. Furthermore, due to the lightweight design of FEDAMON, the time required to compute a monitoring decision at a node is often negligible compared to the duration of a monitoring round. In typical deployments, network latencies also allow multiple communication round-trips between C and the nodes within a single monitoring interval, ensuring stable operation even under minor timing deviations.

5. Evaluation

In this section, we evaluate the experimental results on different datasets using our proposed methods under the assumption of reliable communication with no failures. First, we evaluate how different single prediction models perform on four real-world datasets, and how they differ in terms of messages sent, thereby motivating the necessity of our data-aware method. We then apply data-aware model selection to choose candidate models of the same type in one dataset, followed by testing the framework with standard model parameter settings across four datasets. This allows us to explore the trade-off between communication overhead and monitoring accuracy, as well as observe how different models are selected throughout the process. Next, we apply our data-aware framework with standard solutions to three validation datasets to test its effectiveness. Finally, we present results demonstrating how our approach performs with fully automatic control using dynamic range.

5.1. Experimental setup

In this section, we describe the datasets, parameters, evaluation metrics, and parameter selection process. All experiments are conducted in a simulator under the assumption of no failures or out-of-order events.

5.1.1. Testing datasets

We evaluate our approach on a variety of datasets with different characteristics. All experiments use timestamp sequences starting from 0 and increasing by 1 at every data point. Range R_D is an empirical knowledge we collect from each dataset D which represents the possible fluctuation of the data streams. We have set the value of R_D in relation to D so that it covers the 97.2% percentiles of all values within D, hence discarding some outliers with extremal values. Then ϵ is calculated as a proportion of R_D. The characteristics of the datasets we used are listed in Table 5.

- **Ericsson** [3] is 4 h of hardware CPU usage retrieved during 8 runs of a load testing procedure in an Evolved Packet Core testing infrastructure.
- **Geolife** [46] is GPS trajectories generated from vehicles within the scope of the (Microsoft Research Asia) Geolife project. The vehicular speed based was calculated on the longitude, latitude, and timestamps from the raw data following the preprocessing done in [47].
- **IntelLab** [48] is collected from 54 IoT sensors in IntelLab, from which we select 47 sensors for temperature readings.
- **ACSF1** is from UCR Time Series Classification Archive [49], which contains power consumption from home appliances across 10 classes. In this study, we select class 3 and aggregate all the time series data (training and test datasets). Each node represents a subset of data from class 3.

For clarity, we denote by $R_{\mathcal{E}}$, $R_{\mathcal{G}}$, $R_{\mathcal{I}}$ and $R_{\mathcal{A}}$ (corresponding respectively to Ericsson, Geolife, IntelLab and ACSF1) when referencing a specific R_D of one of the datasets. The values of each are listed in Table 5.

5.1.2. Evaluation metrics

FEDAMON is evaluated for two metrics among all scenarios:

- **Communication ratio**: ρ is the number of update messages Q sent to C over the total duration of time T by the number of nodes n as described in Section 3.2.
- **Mean Absolute Error** (MAE) over R_D: MAE is the difference between the reported value on C and v^i_t, note that the reported value is different from $\hat{v}^i_t$ since when the constraint is broken, $\hat{v}^i_t$ is replaced by v^i_t.

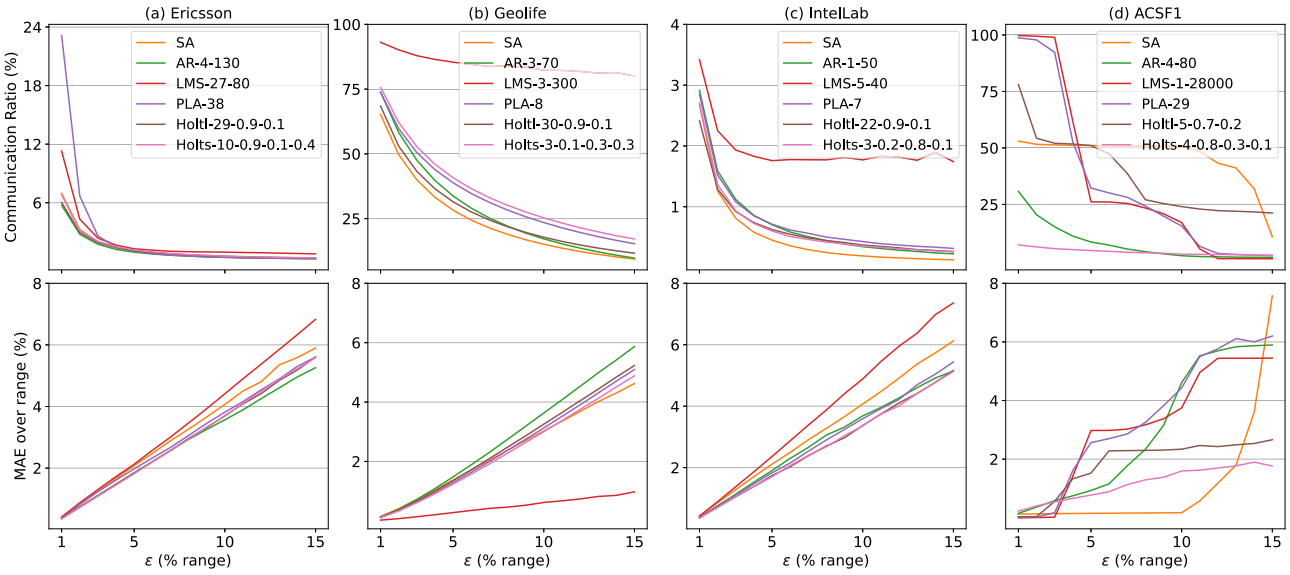


Fig. 4. Forecast-based model performance in four datasets. Here, AR-4-130 indicates that $\ell = 4$ and $k = 130$, LMS-3-300 indicates that $\ell = 3$ and $\kappa = 300$, PLA-8 indicates that $\ell = 8$, Holtl-29-0.9-0.1 indicates that $\ell = 29, \alpha = 0.9, \beta = 0.1$, while Holts-10-0.9-0.1-0.4 indicates that $\ell = 10, \alpha = 0.9, \beta = 0.1, \gamma = 0.4$.

5.1.3. Model and system parameters

A wide range of model parameters was tested for all selected model types, see Fig. 6. The values of the parameters $\ell, k, \kappa, \alpha, \beta, \gamma$ that provide the best performance for each of the testing datasets are listed in the legend of Fig. 4. Holtl and Holts have more model parameters to tune compared to the other models. To avoid the tedious task of testing the full parameter space, we used a systematic procedure as follows: (1) all possible values of α, β, γ in the range $[0, 1]$ with increments of 0.1 were tested on a shorter monitoring period of 100 timestamps for an arbitrarily chosen node, with $\epsilon = x\% \cdot R_D$ where $x \in \{1, 2, \ldots, 15\}$ for each dataset, (2) the top 3 parameter settings (in terms of communication efficiency) are further tested on 5 nodes over the full monitoring period, (3) the average communication ratios are computed across all thresholds for each parameter combination, and the best 3 combinations are then selected. For comparability and clearer interpretation, ϵ is set to a static value based on the range of each dataset (as shown in Table 5), except in Section 5.5, where dynamic error control is evaluated. The buffer size W_{range} of L_{range} is set to 1000 for the dynamic range.

Standard solution. We evaluated multiple system configurations and selected those that consistently achieved strong performance across all datasets. Recurrent top-performing models were identified based on empirical results obtained from four datasets (cf. Section 5.3.1) and subsequently validated on three additional datasets (cf. Section 5.4). We further conducted a systematic exploration of the parameter space for W_{score}, α, and ξ. Specifically, we varied $W_{score} \in [50, 150]$ in increments of 10, $\alpha \in [0, 1.0]$ in increments of 0.1, and $\xi \in [0.001, 0.05]$ in increments of 0.001. Experimental results across all datasets indicate that ξ has a relatively limited overall impact; therefore, we selected the best-performing value. In contrast, α plays a more significant role, particularly with respect to communication overhead. Based on these observations, the standard parameter configuration is set to $W_{score} = 100$, $\alpha = 0$, and $\xi = 0.01$. The number of candidate models $|\mathbb{M}|$ is set to eleven, comprising one SA, two ARs, two LMSs, two PLAs, two HoltLs, and two HoltWs, with SA serving as the initial model. The maximum local buffer size at each node, denoted by D, is computed according to Algo. 3, line 2, as the maximum between the largest minValues among all models and W_{score}. Under the standard parameter configuration, this results in $D = 100$ in our experiments. Note that D may vary if alternative (non-standard) parameter settings are used. In the standard configuration, both the dynamic error calculation and the data-aware mechanism are activated after $\tau_\epsilon = \tau = 150$ monitoring rounds. A summary of the parameter settings for the data-aware standard solution is provided in Table 6.

Table 6
Standard model parameters for our data-aware system, with $\alpha = 0$, $\xi = 0.01$, $W_{score} = 100$ and $\tau_\epsilon = \tau = 150$.

Parameter	SA	AR	LMS	PLA	Holtl	Holts
ℓ	–	6,8	6,28	29,39	5,30	3,4
k	–	100	–	–	–	–
κ	–	–	100	–	–	–
α	–	–	–	–	0.9,0.8	0.3,0.8
β	–	–	–	–	0.1,0.2	0.1,0.3
γ	–	–	–	–	–	0.1,0.3

5.2. Single model prediction performance

We start our evaluation by measuring the communication saved in each of the 4 datasets. Fig. 4 shows ρ achieved by SA, AR, LMS, PLA, Holtl and Holts together with the measurements in terms of MAE over R_D with different parameter settings. We report ρ for varying ϵ values. More specifically, we show results for $1\% \cdot R_D \leq \epsilon \leq 15\% \cdot R_D$ for all datasets.

Regarding ρ, the performance of our forecast-based method is evident in Ericsson, IntelLab, and ACFS1 across all models. When ϵ exceeds $7\% \cdot R_D$, the best model for each dataset achieves a ρ of less than 20%. However, other models still achieve a communication efficiency below 20% when ϵ is $15\% \cdot R_D$. In Fig. 4, AR stands out as the most communication-efficient prediction model, in Fig. 4d, Holts achieves the best performance, in Fig. 4b and c, SA outperforms all other models. Considering the significantly smaller resource overhead of the SA algorithm, its performance on those datasets is quite remarkable. We observe that ϵ not only bounds the maximum error in relation to the range R_D but on average entails a MAE of half its allowed range over all tested datasets. Furthermore, increasing ϵ results in greater communication savings; however, the MAE over R_D also increases. This highlights well the inherent trade-off between communication efficiency and error, which we will examine in the following discussion.

We analyze the performance of each model over time by examining the number of messages sent across all nodes (excluding the initialization phase before τ rounds have elapsed). We use the same parameter

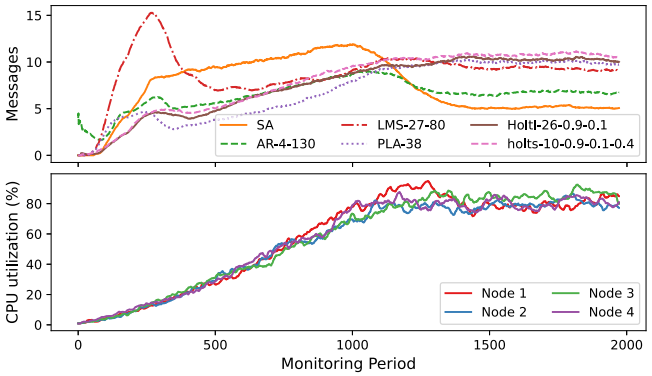


Fig. 5. Messages sent (upper plot) over time across all nodes of Ericsson dataset ($\epsilon = 5\% \cdot R_{\mathcal{E}}$) using a subset of models from Fig. 4(a) and original data streams from four selected nodes (lower plot).

settings as in Fig. 4a, under the Ericsson dataset with $\epsilon = 5\% \cdot R_{\mathcal{E}}$. As shown in Fig. 5, `Holtl` and `Holts` initially perform best for $t \in [0, 200]$, but their performance declines toward the end of the period. Similarly, `SA` sends the most messages for $t \in [400, 1000]$, yet becomes the most communication-efficient model by the end. `PLA` performs best for $t \in [400, 1000]$ but does not consistently maintain top performance. These differences largely arise from the temporal evolution of data streams, as illustrated in Fig. 5 using selected original data streams from various nodes in the Ericsson dataset. Consequently, selecting a fixed model does not guarantee consistent superiority over time, emphasizing the need for our data-aware model selection approach.

5.3. Data-aware system evaluation

In this section, we evaluate the performance of the data-aware system. Specifically, we focus on communication, accuracy respectively and move on to the trade-off evaluation. The introduced communication overhead of switching events and potential scalability are also considered in this section. Here, `SA` is always set as the default selected model $M^i_{selected}$ (Algo. 3, line 5). If not otherwise stated, $\alpha = 0, \xi = 0.01$.

5.3.1. Communication focus

Performance of data-aware with one model type. We continue to explore the performance of the selected model in ρ when having candidate models with the same model type but different parameter settings. Specifically, we evaluate the performance under ACSF1 with `AR-only`, `LMS-only`, `PLA-only`, `Holtl-only` and `Holts-only` with $\alpha = 0$. From Fig. 6, the results show that the data-aware method can always guarantee ρ close to the best parameter setting in each experiment. Moreover, from `PLA` in Fig. 6, the selected model outperforms all the others when ϵ is smaller than $10\% \cdot R_{\mathcal{A}}$ and running different `Holtl` as candidate models shows our data-aware method gives the best results in terms of ρ when $\epsilon > 9\% \cdot R_{\mathcal{A}}$.

Performance of the standard solution. As shown in Fig. 7, the average performance of our data-aware method surpasses that of all individual models in terms of ρ when averaged over different datasets and this across all ϵ, achieving a +12.7% relative improvement on average compared to the best model type, `AR`. In the best-case scenario, when ϵ is 1%, it achieves a +7.1% absolute increase in communication reduction (+33% relative improvement) compared to `AR`. While our selection involves empirical judgment based on quantitative results, it is guided by systematic testing and clearly demonstrates the effectiveness of the data-aware approach in adapting and switching models. Moreover, in [19], the Disjoint-Cliques algorithm with a maximum clique size of one (DjC1) and `SA` achieve comparable communication ratios of approximately 65% on the two datasets evaluated by the authors. In our experiments, the proposed data-aware mechanism consistently improves over `SA`, yielding up to a 10 percentage point absolute reduction in communication ratio. While a direct experimental comparison is not available, this observation indicates that FEDAMON achieves competitive, and potentially improved, communication efficiency relative to the DjC1 algorithm.

Table 7
Average number of rounds per node between two switching events and average introduced communication overhead for all ϵ across all datasets, comparing with the communication ratio without taking switching messages into account.

Dataset D	Ericsson	Geolife	IntelLab	ACSF1	Avg.
Rounds	633	127	2265	539	891
Overhead	0.16%	0.79%	0.04%	0.19%	0.29%
Comm. ratio	1.47%	25.71%	0.58%	5.40%	8.29%
Total	1.63%	26.50%	0.62%	5.59%	8.58%

5.3.2. Accuracy focus

When nodes send more messages to C, monitoring accuracy increases accordingly. We explore the trade-off between communication and accuracy by setting $\epsilon = 15\% \cdot R_D$. Smaller values of ϵ constrain the system's error to a narrower range, while larger values allow more flexibility to adjust communication and accuracy based on model performance. We run the data-aware selection algorithm with the standard solution over each dataset. Fig. 8 illustrates the effectiveness of α in controlling this trade-off. Specifically, when α is 0, the system prioritizes communication, resulting in minimal ρ. As α increases, the communication ratio rises linearly, indicating that the system places more emphasis on accuracy over communication cost.

Moreover, a reduction of +1.5% in MAE over range (25% relative improvement) is achieved by setting $\alpha = 0.4$ almost without increasing communication overhead compared to $\alpha = 0$. The robustness of our data-aware system in absolute error is shown in Fig. 9, with $\epsilon = 15\% \cdot R_{\mathcal{E}}$ using the standard solution under the Ericsson dataset at $\alpha = 0.7$. The distribution of MAE over range for each node over time is comparable with `SA`, and the mean is around 5% of $R_{\mathcal{E}}$ even when $\epsilon = 15\%$, demonstrating the effectiveness of error control via α.

5.3.3. Occurrence of each model

We continue our investigation of how each model is selected over time, with model occurrences reported in Fig. 10. The evolution is studied on the Ericsson dataset with $\epsilon = 5\% \cdot R_{\mathcal{E}}$ and $\alpha = 0.7$, using the standard solution candidate models from Table 6. The entire monitoring period is divided into twelve time slots (slot 0 corresponds to $t \in [150, 306]$ and so on, as metrics begin counting after $\tau = 150$ leaving 1872 remaining timestamps). Time increases along the x-axis from left to right. At slot 0, `SA` is the most selected model, as all nodes initially use `SA` by default. Over time, other models are gradually chosen. In slots 1 and 2, `Holtl` models become predominant, while slots 3 to 7 are dominated by `AR` models. This illustrates how the data-aware standard system effectively adapts to time-dependent variations in data, selecting models based on their `score` without requiring prior knowledge of model parameters.

5.3.4. Introduced overhead

The data-aware system introduces one additional message from C to the nodes whenever a switch event occurs. We evaluate the average introduced communication overhead across all datasets in Table 7, using the standard solution candidate models from Table 6. The overhead is measured as the number of switching messages divided by the total dataset size (i.e., nT) for $\epsilon \in \{1\% \cdot R_D, \ldots, 15\% \cdot R_D\}$. The average introduced overhead across all datasets is 0.29%. Given that the average communication ratio is 8.29%, the overhead from switching messages is minor at about 3%–4% of all communication. The total communication cost, including switches, is 8.58% on average

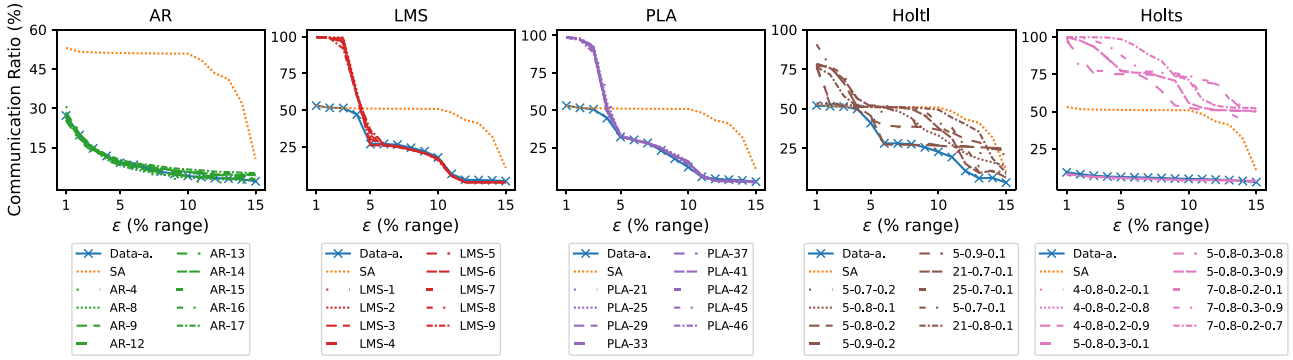


Fig. 6. Comparison between the selected model and candidate models (same type) on ACSF1 ($\alpha = 0$, $k = 80$, $\kappa = 280000$, model names for `Holtl` and `Holts` are not shown on legends to save space); `SA` is also plotted as reference.

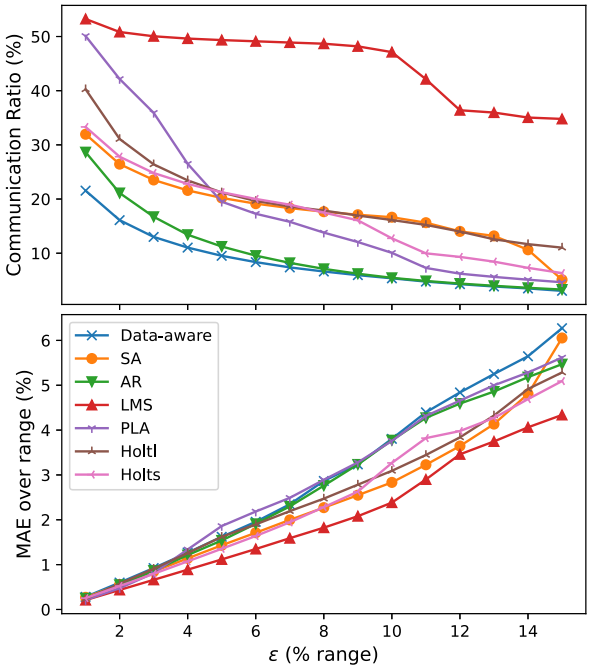


Fig. 7. Average performance on selected vs. candidate models with standard parameters over all testing datasets ($\alpha = 0, \xi = 0.01$).

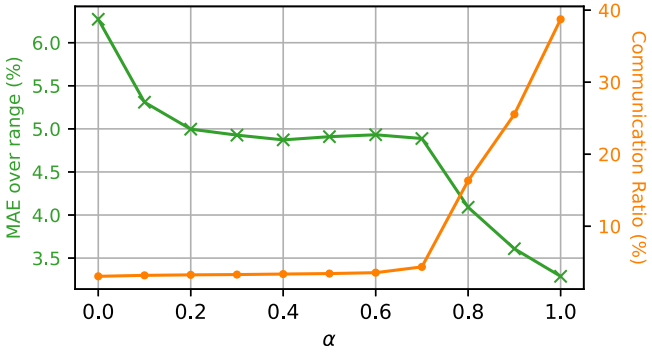


Fig. 8. The trade-off of the data-aware system between ρ and MAE over range on four datasets ($\epsilon = 15\% \cdot R_D$).

across all ϵ. Moreover, the average number of monitoring rounds between two switching messages is 891 for a single node, indicating that the overhead is small relative to the total monitoring period of each dataset.

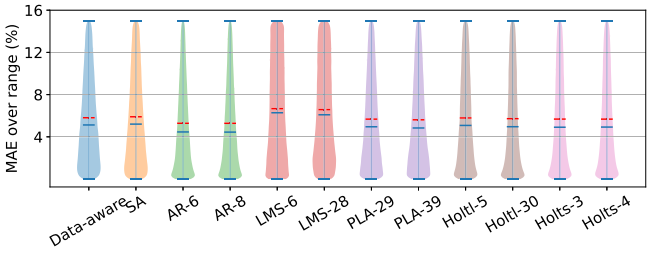


Fig. 9. MAE over range on Ericsson over time of each node; the red dashed lines represent the mean values ($\epsilon = 15\% \cdot R_\mathcal{E}$, $\alpha = 0.7$, model parameters are the same as Table 6).

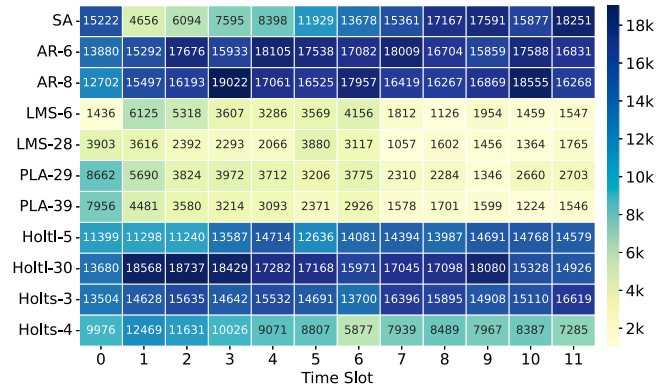


Fig. 10. Occurrence of each model over time on Ericsson ($\epsilon = 5\% \cdot R_\mathcal{E}$, $\alpha = 0.7$), model parameters are the same as Table 6, total duration is divided into 12 timeslots.

5.3.5. System stability

Thanks to the use of lightweight forecasting models, our framework is designed to scale efficiently, allowing a single coordinator to monitor a large number of distributed nodes. In the evaluated datasets, our framework consistently performs well with the number of nodes ranging from 10 to 720. Within our evaluation environment, the chosen metrics (communication savings and error) are largely independent of the number of nodes in the system. We examine how varying the number of nodes affects the stability of these metrics within the same dataset. The Geolife dataset, which consists of speed measurements from individual vehicles, exhibits very different data profiles across nodes. Fig. 11 shows the MAE and communication ratio as the number of nodes changes, with $\epsilon = 5\% \cdot R_\mathcal{E}$, $\alpha = 0$, using the standard solution. Despite the variation in data profiles, the MAE remains very stable at approximately 1.7%, while the communication overhead stays close to 29.65%. These results illustrate that the main trade-off within a dataset

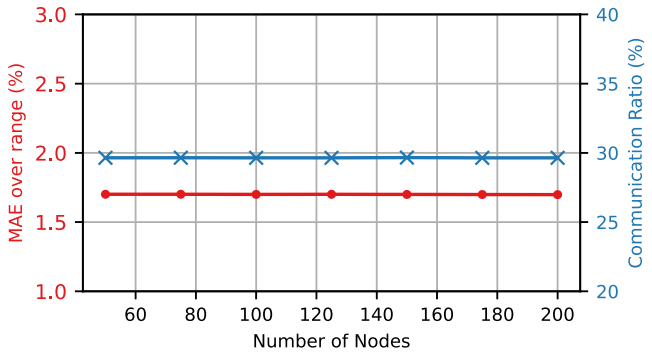


Fig. 11. System performance using standard parameters on Geolife ($\epsilon = 5\% \cdot R_\mathcal{E}$, $\alpha = 0$) with varying node set.

Table 8
Validation datasets with no. of nodes, total duration T, standard data range R_D.

Dataset D	# Nodes	T	Data range R_D
PeMSD7(M)	228	12,672	79.6
ELD	25	2975	1940.6
Energy consumption	50	8760	17.8

is controlled by the error threshold, largely independent of the nodes being involved.

5.4. Validation

We apply our standard solution of FEDAMON to three new datasets to demonstrate generality. Although the standard solution already outperforms all single prediction models, the chosen parameters were still selected empirically based on the four studied "testing datasets". The details of each *validation dataset* are listed in Table 8, with R_D covering at least 95% of all data points.

- **PeMSD7(M)** [50] is originally collected from the Caltrans Performance Measurement System (PeMS) [51] in real time by over 39,000 sensor stations deployed across major metropolitan areas of the California state highway system. PeMSD7(M) is preprocessed in [50], where 228 stations from District 7 of California are randomly selected for weekdays in May and June 2012 from historical speed records.
- **Electricity Load Diagrams (ELD)** [52] contains electricity consumption in kW from 370 clients, collected every 15 min. Here, 25 nodes are randomly selected from May 1st to May 30th, 2013, containing 2975 records per node.
- **Energy Consumption** corresponds to consumption profiles (in kW/h) of 2221 real households (as preprocessed following [53,54], originally from [55]) with each trace containing yearly electricity consumption for one household; we picked 50 traces from that dataset.

The results are summarized in Table 9. Even on unseen datasets, the standard data-aware method performs close to the best single prediction model for each dataset. Compared to the testing datasets, the communication ratio ρ for single model types is lower on the validation datasets with the same ϵ (except for SA at 5%), while the data-aware method achieves the best average ρ on the validation datasets, consistent with the results on existing datasets. Regarding error on validation datasets and in line with our previous results, MAE over range varies from 0.69% to 1.75% for each single model at 5%, from 1.59% to 3.77% at 10%, and from 1.82% to 5.82% at 15%.

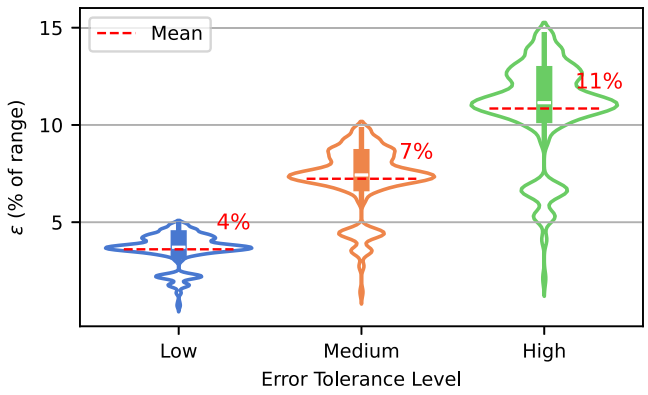


Fig. 12. Distribution of thresholds over time across three levels of error tolerance on ACSF1.

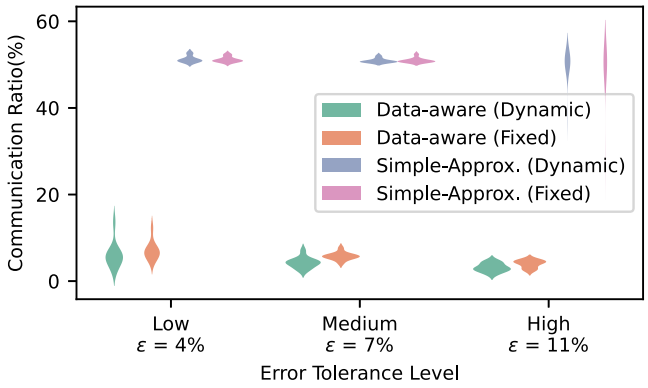


Fig. 13. Distribution of communication ratio on each node over three levels of error tolerance for data-aware and SA (dynamic range v.s. fixed range) on ACSF1.

5.5. Dynamic range

We evaluate the performance of dynamically adjusting the range using the standard solution on the ACSF1 dataset. We set three error tolerance levels δ at 5%, 10%, and 15%, corresponding to low, medium, and high tolerances. Fig. 12 shows the distribution of ϵ (converted to percentage, i.e., $100 \cdot \delta \cdot R'_D/R_D$) over time. The mean values for the three levels are 4%, 7%, and 11%, respectively. We then compare the communication ratio for equivalent error tolerances, for example, comparing the low tolerance level ($5\% \cdot R'_D$) to $4\% \cdot R_D$. Fig. 13 illustrates the results, contrasting the data-aware method with SA using both dynamic and fixed ranges at each node. Our dynamic range method achieves a comparable communication ratio without requiring prior knowledge. Furthermore, while SA reaches a communication ratio of about 50%, our data-aware method reduces it to below 10%. This demonstrates that SA may perform poorly on some datasets and emphasizes the importance of the data-aware method in ensuring system performance.

6. Conclusions

In this paper, we propose FEDAMON, a fully automated, forecast-based and error-bounded monitoring framework for continuously tracking values generated by distributed nodes at a central coordinator. The framework incorporates a data-aware model selection mechanism to minimize communication overhead in large-scale distributed systems. Our experimental evaluation shows that FEDAMON sends only 10% of the updates compared to baseline monitoring while maintaining less than 2% average error across all monitored streams. Moreover, it is particularly suited for distributed applications where one aims to

Table 9
Communication ratio on validation datasets, and the average of the three validation datasets and the testing datasets (Ericsson, Geolife, IntelLab, ACSF1).

Dataset	Epsilon	Method/Model type						
		Data-aware	SA	AR	LMS	PLA	Holtl	Holts
PeMSD7(M)	5%	13.44%	**10.94%**	13.82%	17.45%	27.06%	15.85%	19.38%
	10%	5.58%	**4.44%**	6.21%	8.97%	14.30%	7.62%	9.05%
	15%	3.02%	**2.46%**	3.75%	6.22%	8.41%	4.87%	5.51%
ELD	5%	5.62%	**5.42%**	6.08%	7.36%	10.19%	7.33%	7.89%
	10%	4.13%	**4.06%**	4.25%	4.86%	6.83%	5.16%	5.50%
	15%	3.52%	**3.42%**	3.58%	3.99%	5.50%	4.22%	4.54%
Energy	5%	14.8%	17.6%	**14.1%**	69.4%	18.0%	24.5%	30.1%
	10%	3.5%	5.2%	**3.2%**	58.4%	4.3%	10.5%	12.0%
	15%	1.0%	1.7%	**0.88%**	42.6%	1.4%	5.4%	5.7%
Averaged (Validation datasets)	5%	**11.28%**	11.31%	11.34%	31.42%	18.42%	15.89%	19.14%
	10%	**4.41%**	4.56%	4.56%	24.09%	8.47%	7.77%	8.86%
	15%	**2.53%**	2.53%	2.73%	17.59%	5.11%	4.84%	5.26%
Averaged (Testing datasets)	5%	**9.52%**	20.21%	11.24%	49.34%	19.51%	21.21%	21.26%
	10%	**5.37%**	16.64%	5.46%	47.12%	10.07%	16.13%	12.74%
	15%	**3.05%**	5.11%	3.29%	34.79%	4.64%	11.03%	6.33%

track all node values without extensive prior knowledge. In contrast to assigning fine-calibrated models for a specific dataset, FEDAMON performs well using standard candidate models across different datasets, achieving up to 33% improvement in communication overhead with identical guarantees on maximum error, validating the generalization of our data-aware model selection method. Furthermore, the trade-off between communication overhead and monitoring accuracy is effectively controlled by our data-aware model selection, achieving a 25% improvement in average monitoring error without incurring additional communication cost. By evaluating our framework with the proposed standard solution to validation datasets, results show it generalizes well. We further show that the dynamic range achieves comparable results to acquiring a full range of values of the datasets. Our evaluation results are promising, demonstrating the framework's effectiveness and highlighting its potential for large-scale, efficient monitoring of distributed systems.

CRediT authorship contribution statement

Yixing Zhang: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Conceptualization. **Romaric Duvignau:** Writing – review & editing, Writing – original draft, Supervision, Methodology, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] M. Agiwal, A. Roy, N. Saxena, Next generation 5G wireless networks: A comprehensive survey, IEEE Commun. Surv. & Tutorials 18 (3) (2016) 1617–1655.

[2] G. Anastasi, M. Conti, M. Di Francesco, A. Passarella, Energy conservation in wireless sensor networks: A survey, Ad Hoc Networks 7 (3) (2009) 537–568.

[3] R. Duvignau, M. Papatriantafilou, K. Peratinos, E. Nordström, P. Nyman, Continuous distributed monitoring in the evolved packet core, in: Proc. of the 13th ACM International Conference on Distributed and Event-Based Systems, DEBS, 2019, pp. 187–192.

[4] G. Kakkavas, A. Stamou, V. Karyotis, S. Papavassiliou, Network tomography for efficient monitoring in SDN-enabled 5G networks and beyond: Challenges and opportunities, IEEE Commun. Mag. 59 (3) (2021) 70–76.

[5] G.M. Dias, B. Bellalta, S. Oechsner, A survey about prediction-based data reduction in wireless sensor networks, ACM Comput. Surv. 49 (3) (2016) 1–35.

[6] A.M. Alakeel, et al., A guide to dynamic load balancing in distributed computer systems, Int. J. Comput. Sci. Inf. Secur. 10 (6) (2010) 153–160.

[7] Y. Jiang, A survey of task allocation and load balancing in distributed systems, IEEE Trans. Parallel Distrib. Syst. 27 (2) (2015) 585–599.

[8] S. Madden, M.J. Franklin, J.M. Hellerstein, W. Hong, The design of an acquisitional query processor for sensor networks, in: Proc. of the 2003 ACM SIGMOD International Conference on Management of Data, 2003, pp. 491–502.

[9] C. Olston, B.T. Loo, J. Widom, Adaptive precision setting for cached approximate values, ACM SIGMOD Rec. 30 (2) (2001) 355–366.

[10] G. Cormode, The continuous distributed monitoring model, ACM SIGMOD Rec. 42 (1) (2013) 5–14.

[11] G. Cormode, M. Garofalakis, S. Muthukrishnan, R. Rastogi, Holistic aggregates in a networked world: Distributed tracking of approximate quantiles, in: Proc. of the 2005 ACM SIGMOD International Conference on Management of Data, ACM, 2005, pp. 25–36.

[12] K. Yi, Q. Zhang, Optimal tracking of distributed heavy hitters and quantiles, Algorithmica 65 (1) (2013) 206–223.

[13] A. Mäcker, M. Malatyali, F.M. auf der Heide, Online Top-k-position monitoring of distributed data streams, in: 2015 IEEE International Parallel and Distributed Processing Symposium, IPDPS, IEEE, 2015, pp. 357–364.

[14] A. Mäcker, M. Malatyali, F.M. auf der Heide, On competitive algorithms for approximations of Top-k-position monitoring of distributed streams, in: 2016 IEEE International Parallel and Distributed Processing Symposium, IPDPS, IEEE, 2016, pp. 700–709.

[15] Y.-A. Le Borgne, S. Santini, G. Bontempi, Adaptive model selection for time series prediction in wireless sensor networks, Signal Process. 87 (12) (2007) 3010–3020.

[16] M.A. Razzaque, C. Bleakley, S. Dobson, Compression in wireless sensor networks: A survey and comparative evaluation, ACM Trans. Sens. Networks (TOSN) 10 (1) (2013) 1–44.

[17] M. Tang, F. Li, Y. Tao, Distributed online tracking, in: Proc. of the 2015 ACM SIGMOD International Conference on Management of Data, 2015, pp. 2047–2061.

[18] H.-L. Chan, T.-W. Lam, L.-K. Lee, H.-F. Ting, Continuous monitoring of distributed data streams over a time-based sliding window, Algorithmica 62 (3–4) (2012) 1088–1111.

[19] D. Chu, A. Deshpande, J.M. Hellerstein, W. Hong, Approximate data collection in sensor networks using probabilistic models, in: 22nd International Conference on Data Engineering, ICDE, IEEE, 2006, p. 48.

[20] R. Duvignau, V. Gulisano, M. Papatriantafilou, V. Savic, Streaming piecewise linear approximation for efficient data management in edge computing, in: Proc. of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC, 2019, pp. 593–596.

[21] J.D. Hamilton, Time Series Analysis, Princeton University Press, 2020.

[22] S. Santini, K. Romer, An adaptive strategy for quality-based data reduction in wireless sensor networks, in: Proc. of the 3rd International Conference on Networked Sensing Systems, INSS 2006, TRF Chicago, 2006, pp. 29–36.

[23] J. Körner, S. Bach, A. Karlsson, L. Sundkvist, R. Duvignau, Efficient monitoring of CPS and IoT systems: A deployment guide for empirical evaluations, in: 2024 13th Mediterranean Conference on Embedded Computing, MECO, IEEE, 2024, pp. 1–6.

[24] C.C. Holt, Forecasting seasonals and trends by exponentially weighted moving averages, Int. J. Forecast. 20 (1) (2004) 5–10.
[25] P.R. Winters, Forecasting sales by exponentially weighted moving averages, Manag. Sci. 6 (3) (1960) 324–342.
[26] Y. Zhang, R. Duvignau, FEDAMON: A forecast-based, error-bounded and data-aware approach to continuous distributed monitoring, in: Proceedings of the 19th ACM International Conference on Distributed and Event-based Systems (DEBS), 2025, pp. 39–50.
[27] L. Olsen, F.F. Samavati, M.C. Sousa, J.A. Jorge, Sketch-based modeling: A survey, Comput. Graph. (2009).
[28] J. Azimjonov, A. Özmen, A real-time vehicle detection and a novel vehicle tracking systems for estimating and monitoring traffic flow on highways, Adv. Eng. Informatics 50 (2021) 101393.
[29] D. Yacchirema, D. Sarabia-Jácome, C.E. Palau, M. Esteve, System for monitoring and supporting the treatment of sleep apnea using IoT and big data, Pervasive Mob. Comput. 50 (2018) 25–40.
[30] B. Costa, J. Bachiega Jr., L.R. Carvalho, M. Rosa, A. Araujo, Monitoring fog computing: A review, taxonomy and open challenges, Comput. Netw. 215 (2022) 109189.
[31] G. Aceto, A. Botta, W. De Donato, A. Pescapè, Cloud monitoring: A survey, Comput. Netw. 57 (9) (2013) 2093–2115.
[32] C. Wang, K. Schwan, V. Talwar, G. Eisenhauer, L. Hu, M. Wolf, A flexible architecture integrating monitoring and analytics for managing large-scale data centers, in: Proceedings of the 8th ACM International Conference on Autonomic Computing, 2011, pp. 141–150.
[33] A. Tundo, M. Mobilio, O. Riganelli, L. Mariani, Automated probe life-cycle management for monitoring-as-a-service, IEEE Trans. Serv. Comput. 16 (2) (2022) 969–982.
[34] I. Sharfman, A. Schuster, D. Keren, A geometric approach to monitoring threshold functions over distributed data streams, ACM Trans. Database Syst. 32 (4) (2007) 23.
[35] N. Giatrakos, A. Deligiannakis, M. Garofalakis, D. Keren, V. Samoladas, Scalable approximate query tracking over highly distributed data streams with tunable accuracy guarantees, Inf. Syst. 76 (2018) 59–87.
[36] C. Olston, J. Jiang, J. Widom, Adaptive filters for continuous queries over distributed data streams, in: Proc. of the 2003 ACM SIGMOD International Conference on Management of Data, 2003, pp. 563–574.
[37] D. Giouroukis, A. Dadiani, J. Traub, S. Zeuch, V. Markl, A survey of adaptive sampling and filtering algorithms for the internet of things, in: Proc. of the 14th ACM International Conference on Distributed and Event-Based Systems, DEBS, 2020, pp. 27–38.
[38] B. Babcock, C. Olston, Distributed top-k monitoring, in: Proc. of the 2003 ACM SIGMOD International Conference on Management of Data, ACM, 2003, pp. 28–39.
[39] R. Duvignau, V. Gulisano, M. Papatriantafilou, V. Savic, Piecewise linear approximation in data streaming: Algorithmic implementations and experimental analysis, 2018, arXiv preprint arXiv:1808.08877.
[40] B. Havers, R. Duvignau, H. Najdataei, V. Gulisano, M. Papatriantafilou, A.C. Koppisetty, DRIVEN: A framework for efficient data retrieval and clustering in vehicular networks, Future Gener. Comput. Syst. 107 (2020) 1–17.
[41] B. Farhang-Boroujeny, Adaptive Filters: Theory and Applications, John wiley & sons, 2013.
[42] S.S. Haykin, Adaptive Filter Theory, Pearson Education India, 2002.
[43] R.G. Brown, Statistical forecasting for inventory control, 1959, (No Title).
[44] R. Hyndman, A. Koehler, K. Ord, R. Snyder, Forecasting with Exponential Smoothing: The State Space Approach, Springer, 2008.
[45] J. Verwiebe, P.M. Grulich, J. Traub, V. Markl, Survey of window types for aggregation in stream processing systems, VLDB J. 32 (5) (2023) 985–1011.
[46] Y. Zheng, X. Xie, W.-Y. Ma, et al., GeoLife: A collaborative social networking service among user, location and trajectory, IEEE Data Eng. Bull. 33 (2) (2010) 32–39.
[47] R. Duvignau, B. Havers, V. Gulisano, M. Papatriantafilou, Time-and computation-efficient data localization at vehicular networks' edge, IEEE Access 9 (2021) 137714–137732.
[48] I.B.R. Lab, Intel lab data, 2004, (Accessed 06 November 2024).
[49] H.A. Dau, E. Keogh, K. Kamgar, C.-C.M. Yeh, Y. Zhu, S. Gharghabi, C.A. Ratanamahatana, Yanping, B. Hu, N. Begum, A. Bagnall, A. Mueen, G. Batista, Hexagon-ML, The UCR time series classification archive, 2018, https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.
[50] B. Yu, H. Yin, Z. Zhu, Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting, in: Proceedings of the 27th International Joint Conference on Artificial Intelligence, 2018, pp. 3634–3640.
[51] C. Chen, K. Petty, A. Skabardonis, P. Varaiya, Z. Jia, Freeway performance measurement system: mining loop detector data, Transp. Res. Rec. 1748 (1) (2001) 96–102.
[52] A. Trindade, ElectricityLoadDiagrams20112014, 2015, http://dx.doi.org/10.24432/C58C86, UCI Machine Learning Repository.
[53] R. Duvignau, V. Heinisch, L. Göransson, V. Gulisano, M. Papatriantafilou, Benefits of small-size communities for continuous cost-optimization in peer-to-peer energy sharing, Appl. Energy 301 (2021) 117402.
[54] R. Duvignau, V. Gulisano, M. Papatriantafilou, R. Klasing, Geographical peer matching for P2P energy sharing, IEEE Access (2024).
[55] E. Nyholm, J. Goop, M. Odenberger, F. Johnsson, Solar photovoltaic-battery systems in Swedish households–self-consumption and self-sufficiency, Appl. Energy 183 (2016) 148–159.