electronics

MDPI

Article

A Dataflow-Oriented Approach for Machine-Learning-Powered Internet of Things Applications

Gabriele Baldoni [1,2,*,†], Rafael Teixeira [3,4,*,†], Carlos Guimarães [1], Mário Antunes [3,4], Diogo Gomes [3,4] and Angelo Corsaro [1]

[1] ZettaScale Technology, 91190 Saint-Aubin, France; carlos.guimaraes@zettascale.tech (C.G.); angelo.corsaro@zettascale.tech (A.C.)
[2] U3CM Telematic Engineering Department, Universidad Carlos III de Madrid, 28911 Leganés, Madrid, Spain
[3] DETI, Universidade de Aveiro, 3810-193 Aveiro, Portugal; mario.antunes@av.it.pt (M.A.); dgomes@av.it.pt (D.G.)
[4] Instituto de Telecomunicações, Universidade de Aveiro, 3810-193 Aveiro, Portugal
* Correspondence: gabriele.baldoni@zettascale.tech (G.B.); rafaelgteixeira@av.it.pt (R.T.)
† These authors contributed equally to this work.

Abstract: The rise of the Internet of Things (IoT) has led to an exponential increase in data generated by connected devices. Machine Learning (ML) has emerged as a powerful tool to analyze these data and enable intelligent IoT applications. However, developing and managing ML applications in the decentralized Cloud-to-Things continuum is extremely complex. This paper proposes Zenoh-Flow, a dataflow programming framework that supports the implementation of End-to-End (E2E) ML pipelines in a fully decentralized manner and abstracted from communication aspects. Thus, it simplifies the development and upgrade process of the next-generation ML-powered applications in the IoT domain. The proposed framework was demonstrated using a real-world use case, and the results showcased a significant improvement in overall performance and network usage compared to the original implementation. Additionally, other of its inherent benefits are a significant step towards developing efficient and scalable ML applications in the decentralized IoT ecosystem.

Keywords: IoT; dataflow programming; machine learning; MLOps

Citation: Baldoni, G.; Teixeira, R.; Guimarães, C.; Antunes M.; Gomes D.; Corsaro A. A Dataflow-Oriented Approach for Machine-Learning-Powered Internet of Things Applications. *Electronics* **2023**, *12*, 3940. https://doi.org/10.3390/electronics12183940

Academic Editor: Bahman Javadi

Received: 12 July 2023
Revised: 12 September 2023
Accepted: 14 September 2023
Published: 18 September 2023

1. Introduction

The rapid development of the Internet of Things (IoT) market [1] has led to the integration of smart devices and sensors. These devices gather vast amounts of data, to which Machine Learning (ML) models have proved to be effective in analyzing and deriving useful insights [2]. ML-powered IoT applications are widely adopting a *pipes-and-filters* [3,4] pattern to implement the entire End-to-End (E2E) ML pipeline. In doing so, large applications are decomposed into a series of steps, each applying different transformations to data. In the context of ML pipeline, these include complex tasks, such as data pre-processing and model training, validation, deployment, and inference. These are then split into a well-defined series of simple and independent processing steps called *filters*. While this pattern has proven to work well when the application runs on a single and isolated machine, forthcoming ML [5,6] is clearly showing that *(i)* computations will need to span across multiple locations, from devices and gateways up to the edge or cloud (i.e., Cloud-to-Things continuum); and *(ii)* complex and dynamic data interactions need to be taken into account. By continuing to adopt the *pipes-and-filters* pattern (as proposed in recent MLOps best practices), developers will face new challenges to interconnect and deploy these components transparently, increasing the complexity of applications and their time-to-market and development costs.

Thus, the next-generation of ML-powered IoT applications are envisioned as a set of individual components, spanning from traditional sensors and IoT gateways to computational resources (e.g., storage and computation) deployed anywhere in the continuum.

When stitched together, these components achieve the overall application purpose. However, deploying ML models for IoT scenarios is challenging due to several factors, such as resource constraints, network limitations, and privacy/security concerns. Based on our previous research, a *Data Flow Programming (DFP)* [7] pattern is the most suitable candidate to tackle the challenge mentioned above. By allowing applications to be represented as a directed graph of components (called *operators*), as opposed to a linear pipeline [8], DFP generalizes *pipes-and-filters* and provides a more suitable programming pattern for the applications development. Based on the result of each processing step, DFP determines at runtime the *path* through which data flows, allowing it to express more complex and distributed pipelines.

This paper presents Zenoh-Flow, a novel framework designed to simplify the implementation of E2E ML pipelines using Data Flow Programming patterns. The distinctive contribution of Zenoh-Flow lies in its ability to streamline the entire process of designing, defining, implementing, and deploying IoT applications. Noteworthy advancements offered by this work include the definition of a unified abstraction and computing model capable of accommodating the diverse Cloud-to-Things continuum. Additionally, Zenoh-Flow empowers developers with the ease of declarative application definition, facilitating efficient development and deployment across the entire Cloud-to-Things spectrum. The framework also supports real-time IoT applications, incorporating essential features such as deadlines, time-stamping, and progress tracking. Moreover, this work validates the feasibility of Zenoh-Flow to implement the next-generation ML-powered IoT applications by porting a real-world scenario from the Smart Green Homes project [9], facilitating the definition, deployment, and lifecycle management of its applications without any performance degradation regarding the original deployment. Furthermore, it paves the way for the implementation of more complex applications, which otherwise would be extremely hard to achieve only with MQTT.

The remainder of the article is structured as follows. In Section 2, we briefly present a background reference; Section 3 describes the proposed Zenoh-Flow framework, which is later validated and evaluated over a proposed scenario on Section 4; advantages and future challenges are discussed in Section 5; and Section 6 concludes the article.

2. Related Work

As previously mentioned, ML has become integral to advanced IoT scenarios. However, building an ML model is just one part of the process. To effectively use ML models, it is vital to consider the entire data flow, from data collection to model deployment. Data flow programming is a well-suited programming paradigm for building data pipelines. In this paradigm, data are treated as a stream that flows through various processing nodes. Each node performs a specific transformation on the data and passes it on to the next node. This approach enables developers to build complex data pipelines by chaining simple processing nodes. In addition to DFP, it is also essential to consider the transport layer used to move data between processing nodes. The data transport layer plays a critical role in the performance and scalability of a DFP. A data-centric transport layer is a transport layer designed specifically for moving data between processing nodes. It is optimized for high throughput and low latency, making it well-suited for building ML pipelines.

The following subsections will dive deeper into these concepts and explore their importance in building effective ML pipelines.

2.1. Machine Learning

With the emergence of Cloud-to-Things and Edge Computing architectures, ML models can be deployed closer to where the data are generated, enabling faster and more efficient processing. This makes ML essential for unlocking the full potential of IoT and driving innovation in various industries.

MLOps refers to the set of practices and tools used to streamline and automate the process of building, deploying, and managing ML models [10,11]. With the increasing demand for ML, MLOps has become essential for ensuring the reliability and scalability of machine learning projects. MLOps provides a standardized set of best practices and tools that enable organizations to overcome unique challenges associated with ML, including data preparation, model development, testing, deployment, and monitoring.

Following that methodology, most ML deployments can be described as a single pipeline. They all need data; using said data, they need to be trained; once trained, they need to be deployed. This pipeline can be roughly structured into four connected components, as depicted in Figure 1: *(i)* Data Collection; *(ii)* Data Transformation; *(iii)* (Continuous) ML (re-)training; and *(iv)* (Continuous) ML (re-)deployment [10,12].

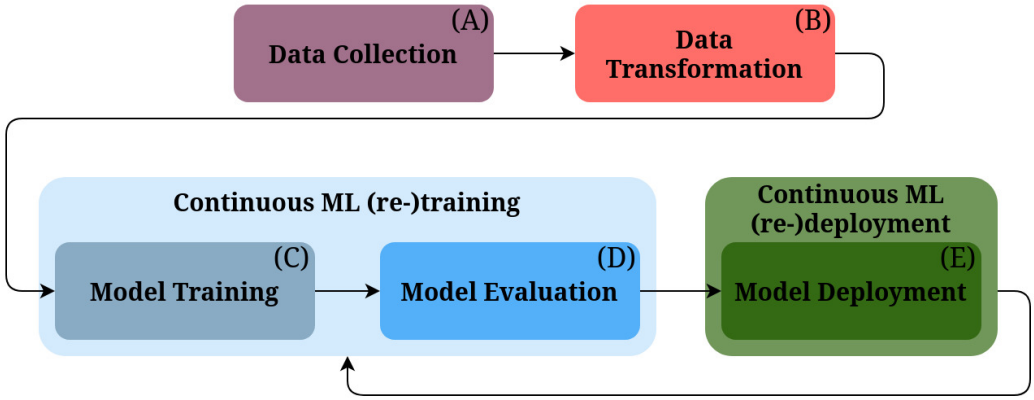

Figure 1. Standard ML pipeline.

In data collection (A), the system collects data that will later be used as the input for the model. The collected data can come from various sources (sensors, network sniffers, or even user input). Given the possible heterogeneity of the data, it is crucial that, after it is collected and aggregated, it is also transformed (B) into a standardized format that the model can use. It is essential to mention that the data collection and data transformation steps are continuous operations used for the training and deployment phases.

After it, the model (re-)training (C) takes place, where the model goes through a loop that only stops once the model achieves the desired performance. In this loop, the model is trained and evaluated (D) with different hyperparameters to optimize them according to a set of metrics. This step is the hardest to perform as the results can be non-deterministic, and human interaction is often required. In addition, the model may need to be periodically retrained. Since the model, in its essence, learns the patterns in the training dataset, any pattern change in the environment not reflected in the training set can decrease its performance, and once it drops below a certain threshold, the model must be retrained.

After the model's training, it needs to be deployed (E) into production, where the trained model is used to predict outputs using the live data still being collected in step (A) and transformed in step (B). It is also in this step that the model is monitored for performance drops that trigger the retraining discussed above. This step usually requires its own (smaller) pipeline, and the process should be automated. Every time the model is retrained, it will be redeployed if the performance meets the desired value.

Despite offering a comprehensive approach to building efficient pipelines for continuous training and deployment of machine learning ML models, MLOps fails to address the specific challenges of data transfer and parallelization across various pipeline stages. This work addresses this gap by providing a concrete IoT scenario with a complete ML deployment.

2.2. Data Flow Programming

DFP [7] is a computational model that represents applications as directed graphs, where nodes are computational units called operators and edges are unbounded First-In-First-Out (FIFO) streams known as links. These links, categorized as inputs or outputs, connect operators via typed ports, enabling concurrent execution and communication. Notably, DFP facilitates application decomposition into more straightforward operators arranged in graphs for adequate communication abstraction.

Two notable DFP models are Kahn Processing Networks (KPNs) [13] and Dataflow Process Networks (DPNs) [14]. KPN enforces a firing strategy requiring all inputs to be present before any processing, ensuring functional behavior and determinism. This is vital for safety-critical scenarios and debugging. However, it limits the processing of incomplete data. In contrast, DPNs allows operators to fire with subsets of inputs, enabling computations on partial data. However, to guarantee determinism and functional behavior, DPNs operators are not allowed to perform side effects, including having an internal state. Thus, it does not apply to complex applications requiring historical data. Nevertheless, DFP lacks guidance on handling real-time processing, deadlines, computation periodicity, progress tracking, and geo-distributed deployment of operators. This places additional responsibilities on developers, leading to a more complex development process, increased maintenance efforts, slower time to market, and elevated costs.

2.3. Data-Centric Transport Layer

IoT has relied on protocols above the transport layer to reflect a more web-based services approach to device reachability and data consumption. However, a more complex networking stack must be supported by IoT devices, while also requiring logically centralized communication points to be deployed in the network infrastructure (e.g., communication brokers). The explosion of inter-connected IoT devices across the Cloud-to-Things continuum and the need for a scalable, decentralized, and distributed content delivery with intrinsic security aspects is motivating next-generation IoT applications to shift towards data-centric approaches for the transport layer.

Unlike host-centric approaches, data-centric approaches place the data itself as the central networking element of its architecture, moving away from the host-oriented addressing schemes and E2E principles. Data are decoupled from their location and their specific hosts, allowing more efficient distribution and consumption. In doing so, applications (or their components) can be deployed anywhere in the Cloud-to-Things continuum without requiring changes in the application or its configurations.

Zenoh and Message Queue Telemetry Transport (MQTT) are innovative or widely used data-centric network protocols designed to optimize data transfer, enable secure communication, and enhance interoperability across distributed systems.

Other protocol solutions, such as Kafka and CoAP, were initially taken into consideration but, given their limitations, have not been implemented for evaluation. Table 1 provides a qualitative analysis of the different solutions. In particular, Kafka only supports Cloud-only deployments, while CoAP supports Things-only deployment. Therefore, we focused our evaluation on solutions capable of communicating E2E from Things to Cloud without requiring any protocol conversion.

2.3.1. Zenoh

Zenoh (/zeno/) is a pub/sub/query protocol unifying data in motion, data at rest, and computations. Additionally, it is entirely decentralized and offers improved functionalities such as dynamic node discovery, batching at the wire level, multiple levels of reliability and priority, and minimal network overhead. Nevertheless, Zenoh connects to other communication middleware, such as MQTT and Data Distribution Service (DDS). In doing so, Zenoh can interoperate with legacy systems in the IoT domain, allowing information to be exchanged across protocols and avoiding the creation of information silos. Moreover, its plugin interface enables extensibility by allowing new connectors to

be quickly developed. Such capabilities have already proven their suitability as transport for ML workflows in the context of DAEMON, where it has been used to implement an N-MAPE-K framework [15].

Table 1. Qualitative analysis of IoT communication solutions.

	Zenoh-Flow (Proposed)	MQTT	Kafka	CoAP
Paradigm	Data-centric	Data-centric	Data-centric	Host-centric
Topology	Peer-to-Peer, Brokered, Routed	Brokered	Brokered	Peer-to-Peer
Communication Model	Query (Request/Reply) Publish/Subscribe (Push) Publish/Subscribe (Pull)	Publish/Subscribe (Push)	Publish/Subscribe (Pull)	Request Reply
Multiparty Communication	Yes	Limited	Limited	No
Types Aware	Yes	Blob	Blob	Blob
Composability	Declarative Definition	No	No	No
Deployment Model	Cloud-to-Things	Cloud-to-Things	Cloud	Thing
Considered for evaluation	Yes	Yes	No	No

2.3.2. MQTT

MQTT is a messaging protocol between clients and brokers designed for resource-constrained devices and networks. Thus, it is commonly used in IoT applications. It follows a publish–subscribe model where clients subscribe to specific topics and receive messages published by other clients on those same topics. MQTT relies on a communication broker, a logically centralized intermediary for distributing all messages exchanged between publishers and subscribers. MQTT operates on top of TCP/IP, making it less suitable for a wide range of IoT applications that consider resource-constrained IoT devices with limited processing power, memory, and battery life.

3. A Data Flow Framework for the Cloud-to-Things Continuum

This section describes Zenoh-Flow: a *decentralized data flow programming framework for the Cloud-to-Things continuum* that leverages data-centric networking concepts [16]. Departing from novel IoT applications' requirements and existing solutions' shortcomings, Zenoh-Flow is designed to meet specific needs, including being *Cloud-to-Things native* for seamless deployment, employing a *declarative approach* for explicit application definitions and analysis, enabling *composition* of applications by reusing or grouping operators, offering core features like *time-stamping*, *deadlines*, *logging*, and *data replay* for next-gen IoT applications, and ensuring *high performance* with low-latency, low-overhead, and high throughput to support advanced IoT applications.

Note that the generic approach taken by Zenoh-Flow makes it agnostic to the application domains. In fact, its integration into automotive and robotics applications, which mostly rely on implicit dataflow programming approaches, are two industrial verticals where Zenoh-Flow is being successfully deployed and integrated [17].

3.1. Solution Workflow

In Zenoh-Flow, the user starts by defining and implementing the different operators, which are chained together in a *descriptor* (step 1 in Figure 2). The descriptor acts as a contract specifying the application operators, their interconnections, and their requirements, e.g., deadlines and access to specific hardware resources. Operators can be composed to create more complex applications, effectively fostering code reuse. The descriptor also contains additional information to provide automatic deployment of applications across the

Cloud-to-Things continuum with automatic allocation. The application descriptor, along with any operators, is on-boarded into Zenoh-Flow (step 2 in Figure 2), which, in turn, stores them in a distributed registry (step 3 in Figure 2).

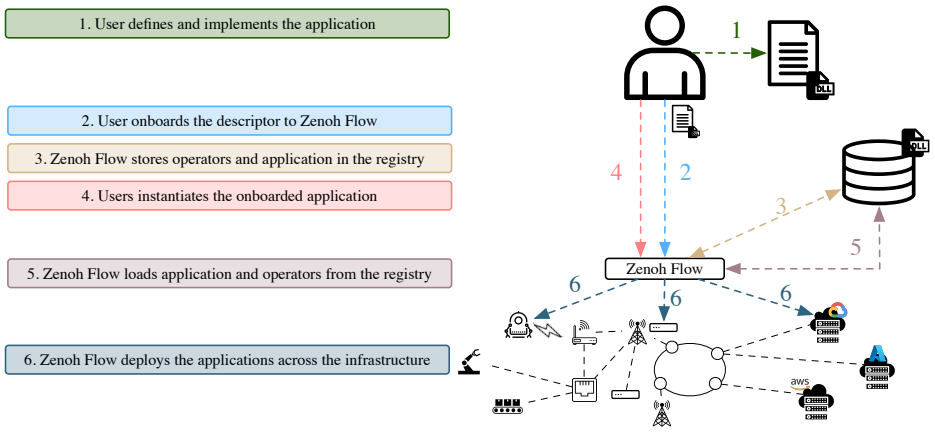

Figure 2. Zenoh-Flow overview and E2E application deployment workflow.

Upon an application instantiation request (step 4 in Figure 2), Zenoh-Flow creates a runtime graph by loading operators from the distributed registry (step 5 in Figure 2), and establishing the underlying communication between them in a location transparent manner (step 6 in Figure 2).

3.2. Solution Overview

Following the DFP model, in Zenoh-Flow, each application is composed of a graph of nodes. Each node is associated with different inputs and outputs, called *ports*. In the same way, ports are typed and interconnected via unbounded FIFOs, called *links*.

Note that Zenoh-Flow presents some differences concerning state-of-the-art DFP. In particular, it differentiates three kinds of graph nodes: *source*, *sink*, and *operators*. Sources and sinks are dedicated to performing I/O operations that have side-effects, making them not *purely functional*, while operators follow the KPN model. *Sources* can also be triggered within a specified period, allowing sensor readings to produce new data periodically. The differentiation of *source* and *sink* enables the application graph to communicate with the external world, thus performing I/O only before entering or when leaving the application graph.

Note that data produced by any operator of the application graph is always timestamped by a Hybrid Logical Clock (HLC) [18] combined with unique IDs. Thus, HLC, together with the unique ID, provides total ordering guarantees to Zenoh-Flow. As data are ordered, Zenoh-Flow additionally guarantees that if some data with timestamp T are received, all the data with timestamp $t' < T$ have already been received. Hence, the timestamp of the data enables progress tracking. Furthermore, ordered and timestamped data are the building block for deadlines. As data timestamp is checked upon reception, Zenoh-Flow provides notification of operators in the case of deadline miss.

Similarly to DPN, Zenoh-Flow operators can have an internal state. An internal state allows for more complex applications that require *historic* information to compute an output. The combination of the internal state of operators and the total ordering of data enables Zenoh-Flow to support determinism. In the context of an application composed of *purely functional* operators, Zenoh-Flow supports the determinism for the whole application. In such cases, the same sequence of inputs produces the same sequence of outputs.

3.3. Architecture

Figure 3 depicts the functional architecture of Zenoh-Flow:

- The *Runners* are in charge of executing the *user code*, abstracting data delivery, and ensuring timely operations like deadlines. Each runner is in charge of a single operator;
- A *DF instance* represents a DFP application running across the continuum. Each instance keeps track of its runtime information and has access to the runners executing their operators;
- The *Framework runtime* is in charge of mapping applications to the infrastructure, managing the life-cycle of operators, and configuring the application's *data plane*. A runtime runs on a single machine. All runtimes collaborate to provide a decentralized allocation;
- The *Control/Management plane* is in charge of the control and management communications across different runtimes in the continuum. It also stores information about the state of the infrastructure and any running application;
- The *Data plane* is in charge of moving the data between the operators, and it is configured by the runtime.

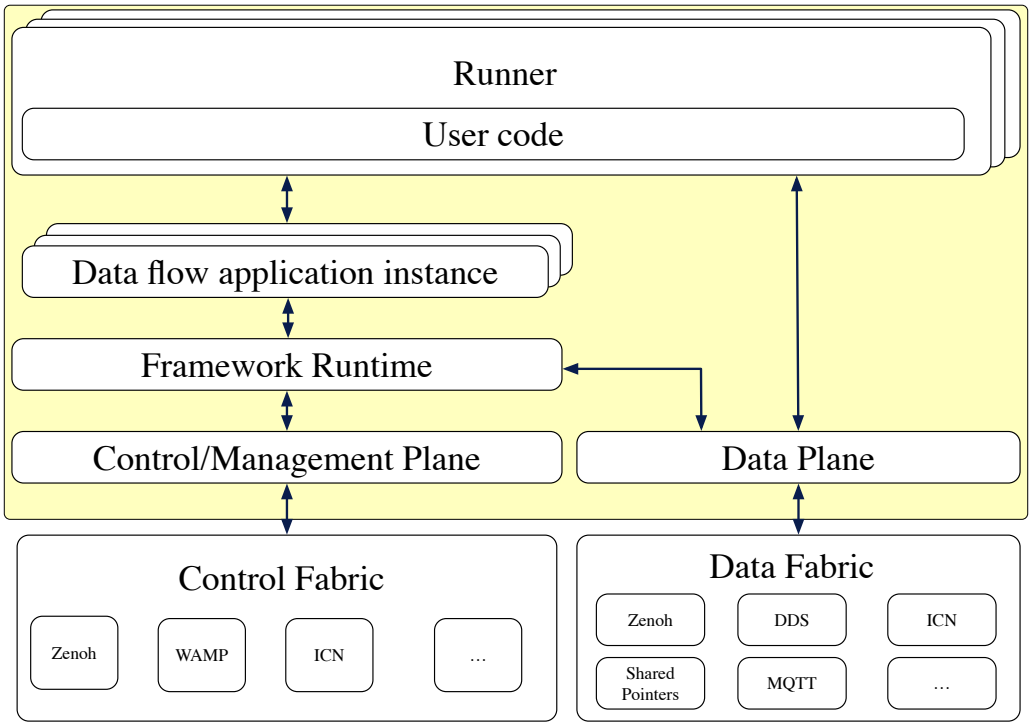

Figure 3. Functional architecture of Zenoh-Flow.

3.4. Highlights

Summarizing, Zenoh-Flow introduces a wide range of functionalities that suits the next-generation IoT applications, going beyond state-of-the-art solutions:

1. Application graphs are defined with a *declarative* approach, in contrast to most common *programmatic* approaches used nowadays;
2. Input data are timestamped (via HLC), and total ordering is provided by augmenting the timestamps with unique IDs;
3. Deployment of applications across the Cloud-to-Things continuum is supported natively, instead of relying on external tools;
4. Native support for functionalities relevant for next-generation IoT applications, like loops, deadlines, and progress tracking;
5. Agnostic to the transport protocol, allowing transparently stitching dataflow graphs across legacy and greenfield systems.

4. Implementing an ML Pipeline over DataFlow Concepts

This section explores the potential of Zenoh-Flow in a real-world scenario by implementing a ML pipeline in an IoT environment. The IoT presents several challenges for data processing, including the sheer volume and heterogeneity of data sources and the need for real-time processing. By leveraging the capabilities of Zenoh-Flow, next-generation ML-powered IoT applications are empowered with a robust and scalable solution that can efficiently process data from various IoT devices and seamlessly integrate with machine learning algorithms across the entire Cloud-to-Things continuum.

The code used for this evaluation is publicly available on GitHub [19]).

4.1. Proposed Scenario

The proposed scenario mimics a real-world deployment from the Smart Green Homes (SGH) project [9] with 13 houses, each with five different sensors. Although the original project intent was to learn the comfort temperature of the user, in this scenario, another task is solved. The objective is to deploy a distributed ML pipeline capable of predicting if a user is present within the house. Since the output from the motion sensors is used as the truth value, the models are solving a classification task. The readings used by the model as features are: *(i)* indoor and outdoor temperature; *(ii)* indoor and outdoor humidity; *(iii)* indoor and outdoor pressure; *(iv)* windspeed; *(v)* precipitation; and *(vi)* door status.

4.1.1. ML Pipeline

The distributed ML pipeline implemented to support the proposed scenario is presented in Figure 4. As one can see from the figure, there are more components than pipeline steps. This results from the data preprocessing being distributed through the filler, preprocessor, and batcher. It is also visible that there is more than one trainer. Although the trainers could be combined into a single entity, model training consumes the most time in the system, so distributing the trained models among various trainers helped reduce the overall time spent training.

Although the sensors provide valuable information, their publishing mechanism is not ideal for ML, as each sensor only publishes data when it changes beyond a threshold, instead of periodically. This is mitigated by the implemented pipeline starting in the gateway, where the sensor data are synced and transformed into a periodic stream, filling in missing values whenever necessary.

As the name indicates, the aggregator component combines the data from the various houses into a single stream that will be used in the remainder of the pipeline. The filler mitigates any failed communication between the gateway and aggregator, or even aggregator and filler, by checking if there are no missing values between consecutive messages from a given house and filling them when necessary. The preprocessor prepares the raw data received from the sensors into the input expected by the trained models, removing unnecessary information and dividing the input and output of the presence-checking task.

Since training a model each time a new example is published is unreasonable, the batcher is a buffer storing examples from the preprocessor until a batch of 10,000 samples is achieved. Once the batch is published, six different ML algorithms are trained in parallel in a five-fold cross-validation scheme on the collected data, and their results compared by the comparator. Whenever one of the models' performance surpasses the best model so far, the comparator publishes it to the evaluator, where the model is used to infer the presence in the houses. The constant retraining and redeployment are essential as the various seasonal changes related to the indoor presence (e.g., vacations) can cause prediction drift.

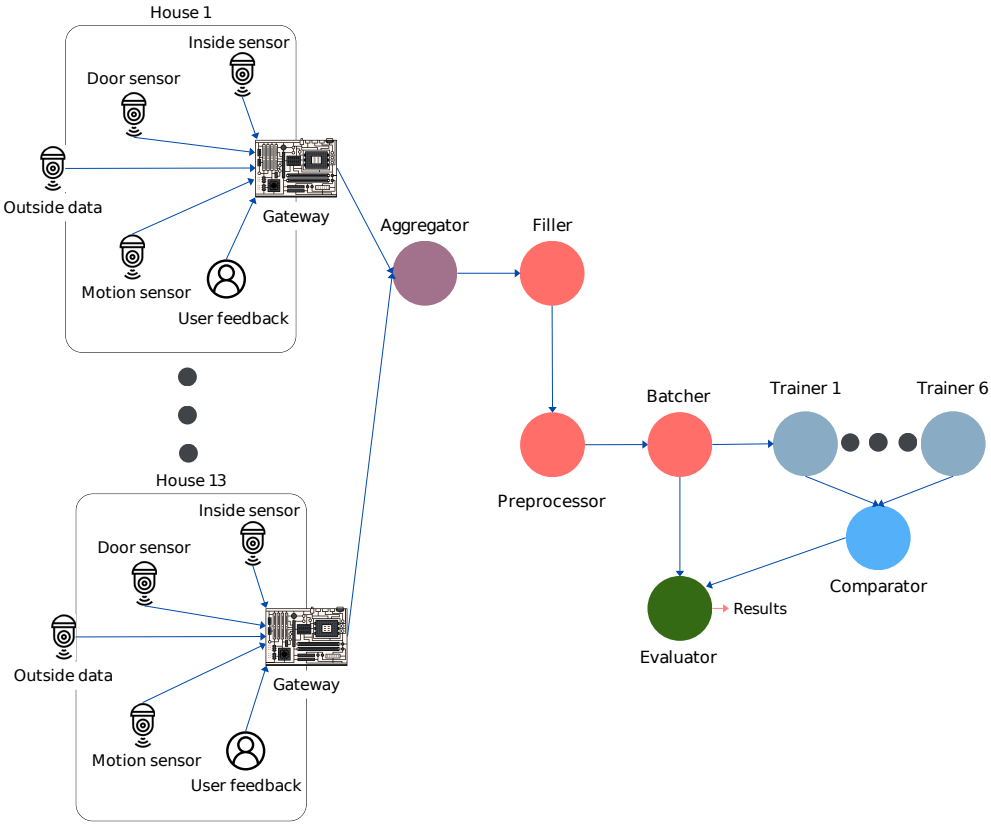

Figure 4. Overview of the proposed scenario.

4.1.2. ML Models

The models used in the trainers are six Scikit-learn models: Logistic Regression (LR), Support Vector Machine (SVM), K-Nearest Neighbors (k-NN), Decision Tree (DT), Random Forest (RF), and Multi-layer Perceptron (MLP), with a set seed of 42. Based on the dataset analysis [20], a shallow model should be sufficient to capture the underlying pattern:

- The simplest model is the LR [21]. Borrowed from statistical analysis, LR estimates the probability of an event happening given a set of features;
- SVM, is very similar to LR. The difference between them is how the fitted curve is created. While LR only cares about separating the different classes, the SVM maximizes the margins between the decision boundary and the examples [22];
- The k-NN algorithm uses an entirely different approach to classifying examples. Instead of fitting a curve that separates the two classes, when a new data point needs to be classified, the algorithm searches the closest k terms and uses them as a voting system [22];
- Like k-NN, the DT does not try to fit a curve on a hyperplane. As its name states, the class prediction is formulated as a tree structure where the nodes verify features [21];
- The RF is a bagging classifier that creates several DTs and trains them on different subsets of the training dataset [22];
- Finally, the MLP considered is a classical Artificial Neural Network (ANN) with one hidden layer comprised of simple dense neurons that performed non-linear calculations [23].

4.1.3. Dataset

Data were gathered from 13 houses (within Portugal), each with five data sources. There are three indoor sensors (a motion sensor, a door sensor, and a temperature and humidity sensor), one outdoor sensor that registers the outside conditions, and one feedback

source where the user can state whether he is comfortable. The data were collected, compiled, and published as an anonymized dataset [20].

4.2. Scenario Implementation and Deployment

To validate the feasibility of Zenoh-Flow as a solution for implementing ML pipelines for next-generation IoT applications, the proposed scenario was implemented using MQTT as the widely used legacy protocol for IoT applications, and Zenoh-Flow as the novel solution proposed in this work. The following subsections present an overview of the implementation process for both approaches.

4.2.1. MQTT Implementation

Although the proposed scenario envisioned sequential communication between the pipeline nodes, implementing the system with MQTT was impossible, as communications must be sent to a central broker. Instead, the communications between nodes would use different topics to define the message flow. For example, the communication between the gateway and aggregator would flow through the topic "house_data" while the communication between the aggregator and filler would flow through the topic "filler". Figure 5 displays how the different components interact with the MQTT broker. Sensors deployed within the house are not represented, since the communications inside the house do not rely on MQTT. Since most components receive and send messages, they all send and receive messages from the MQTT broker. The only exceptions are the gateways, which only send data, and the evaluator, which only receives.

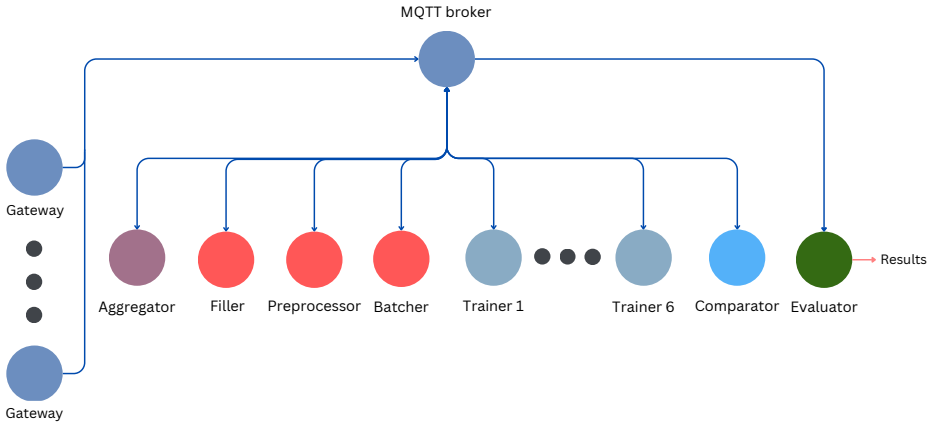

Figure 5. Overview of the scenario implemented in MQTT.

4.2.2. Zenoh-Flow Implementation

Zenoh-Flow takes a descriptive approach to the definition of the message flux. Thus, the scenario depicted in Figure 6 represents the exact data flow implemented in Zenoh-Flow. Zenoh-Flow defines unique topic names between nodes running on different hosts, and thus alleviates the developers in determining terms that could conflict, causing data mixing between applications. Two parts can be distinguished in the Zenoh-Flow implementation: the gateways and the ML pipeline. The nodes represented by a right-pointing triangle are the source of the pipeline (i.e., the ones injecting data from the external world), which are the gateways in this particular scenario. The left-pointing triangle represents the sink (i.e., the one sending back data to the external world). All other nodes, represented as circles, are operators which perform data computation from their inputs, sending them to their output. As Zenoh-Flow relies on Zenoh as its transport protocol, the actual data flow implementation leverage peer-to-peer communication, thus reducing the number of communication hops and avoiding a single point-of-failure.

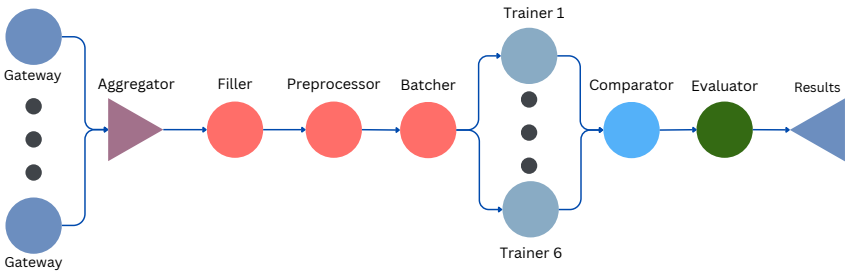

Figure 6. Overview of the scenario implemented with Zenoh-Flow.

Both implementations were written in Python, as it is a popular choice for ML pipelines due to its ease of use, flexibility, and the availability of numerous open-source libraries and frameworks. Additionally, Python's popularity means that a large community of developers and resources is available for support, making it a reliable choice for building and maintaining ML pipelines and leveraging the proposed solution. An implementation analysis of both solutions is summarized in Table 1, alongside qualitative analysis of Kafka and CoAP.

Each component of the pipeline has been run as an isolated docker container. The containers are managed via docker-compose for reproducbility and to enable interconnectivity between them. Specifications of the testbed are presented in Table 2.

Table 2. Testbed specifications.

Hardware		
CPU	**RAM**	**Operating System**
AMD EPYC 7502	512 GB	Ubuntu 20.04.5 LTS
Software		
Environment	**Communication Protocols**	**Language/Libraries**
Docker 23.0.1	MQTT 1.6.1 Zenoh 0.7.0rc0 Zenoh-Flow 0.4.0rc0	Python 3.8.10 Pandas 1.4.3 SciKit-Learn 1.1.2

4.3. Evaluation

Although this work focuses primarily on proposing a novel dataflow programming framework, a proper ML model evaluation is required to validate that its performance, data transferred, and model accuracy do not differ between implementations. To do so, Mathews Correlation Coefficient (MCC) was selected, as it correctly handles imbalanced datasets, penalizing misclassifications harder than the f1-score.

Given the distributed approach adopted in the ML pipeline design, it is vital to understand how different solutions impact the overall system performance in terms of time taken for message passing and network utilization. Since the code executed in each node is always the same, any considerable delay seen in the results will result from how the data were communicated. Figure 7 depicts the variation in the MCC across time. Since the models had set seeds, they would consistently achieve the same performance given the same batch of data, so performance values can be used to compare different experiment runs. At time 12,000 s, the pipeline implemented using Zenoh-Flow is slightly ahead of the one implemented using MQTT, which *delta* continues to accumulate until the end of the experiment (e.g., by time 17,000 s, this difference becomes even more apparent). This difference is around 1.20% in the evaluated scenario, with the tests running for a total of 5.5 h. This trend shows that, along with time passing, the Zenoh-Flow-based pipeline creates less overhead than the MQTT-based one allowing the data to flow faster.

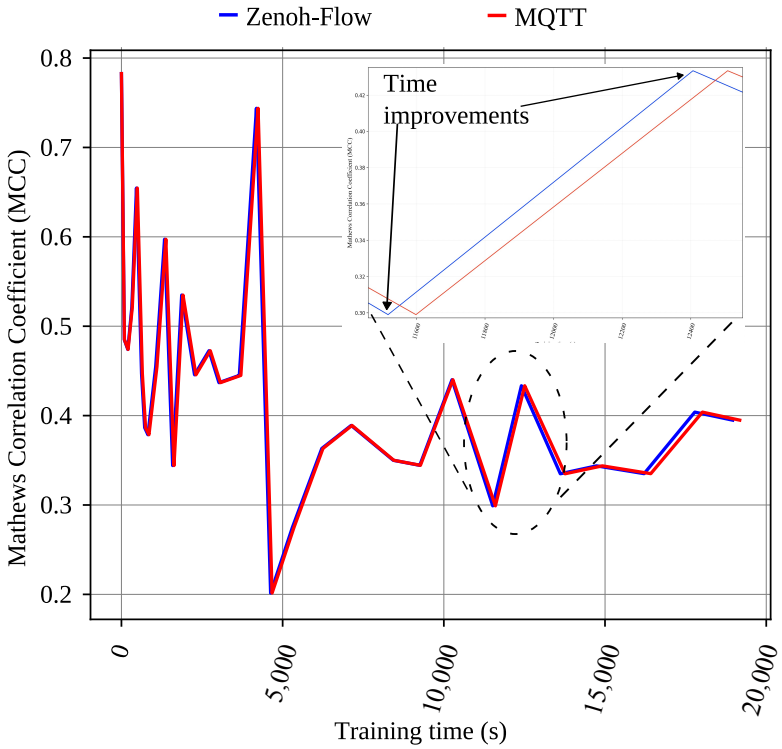

Figure 7. MCC over training time.

Such behavior can be explained by the lower network overhead the underlying Zenoh protocol provides, as shown in Table 3. Thus, Zenoh-Flow accounts for 40% less network overhead in the evaluated scenario, compared to MQTT. Moreover, the Round-Trip-Time (RTT) of a message between two solutions varies significantly, as depicted in Figure 8. On the one hand, see that 99th-percentile Zenoh-Flow messages are delivered within 50 µs and 100 µs, with a consistent distribution. On the other hand, MQTT presents a step distribution pattern, where only a small amount of data are delivered within less than 500 µs (i.e., less than 20%), and it can take up to 1 ms to deliver a message.

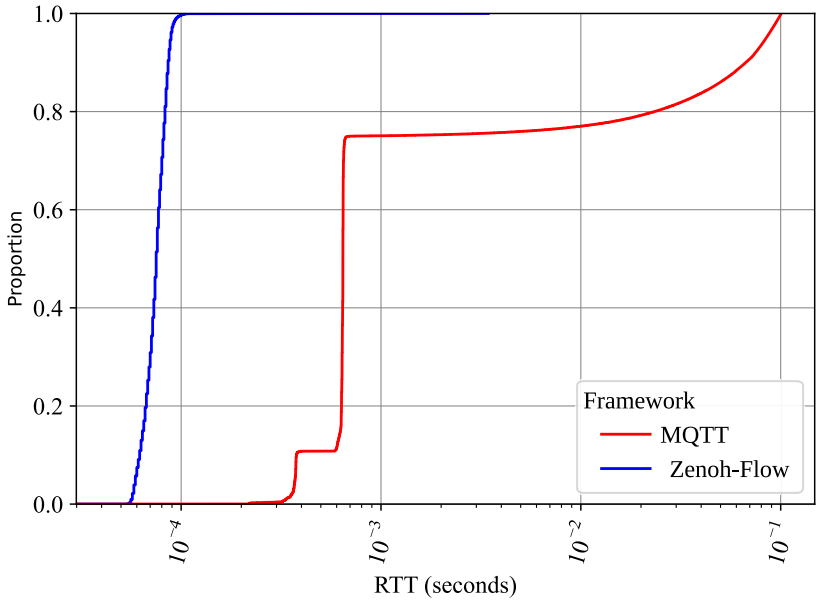

Figure 8. eCDFs of communication RTT.

Table 3. Analysis of CLOC and wire overhead for the different solutions.

	Zenoh-Flow	**MQTT**
Component	**CLOC**	
Aggregator	43	48
Filler	145	161
Preprocessor	83	105
Batcher	75	98
Trainers	102	116
Comparator	135	155
Inference	70	91
Total	522	774
	Overhead	
Payload Size (bytes)	20,969	20,969
Total Sent (bytes)	20,977	20,982
Overhead (bytes)	8	13

Finally, Zenoh-Flow abstracts the developer of defining unique topic names between nodes running on different hosts, thus requiring fewer lines of code to implement each application's component. As depicted in Table 3, the overall code base was shrunk by 32%.

5. Discussion

This section further elaborates on the evaluation results, contextualizing them in the overall solution and discussing some of its benefits.

The differences in message overhead and RTT come from the distinct communication approaches taken by the underlying network protocols. While Zenoh-Flow enables decentralized communication across components by leveraging Zenoh protocol, MQTT requires a message broker to mediate the communication. Thus, MQTT introduces an additional hop in the communication, resulting in higher latency and a single point of failure. The MQTT broker becomes a bottleneck for the pipeline and, in case of failure, no component can communicate anymore. In turn, Zenoh-Flow provides a dynamic discovery mechanism that allows components to discover and connect between themselves, thus creating a full mesh of connected applications with an optimal path for exchanging data.

By abstracting all the communication aspects, Zenoh-Flow allows the developers to focus solely on their application logic, thus reducing the overall amount of code to be written to support the communication between components of their application. The less the code to be implemented, the lower the likelihood of errors and bugs, as there are fewer opportunities for mistakes to be introduced into the codebase. In addition, such abstraction also facilitates the re-definition of the ML pipeline, together with better scalability, without requiring additional lines of code as opposed to MQTT. Altogether, such an approach enables reduced development times and costs, while providing faster time-to-market and while improving both the efficiency and quality of software development.

Regarding the ML models' performance, the fact that there is no difference between the two solutions indicates that the pipeline is the same in both approaches. Furthermore, the performance obtained is aligned with the previous analysis of the dataset [20].

This finding suggests that Zenoh-Flow is a more efficient and effective framework than MQTT, not only for applications that require low latency and minimal overhead. Its latency, overhead, and CLOC improvements are beneficial for different sets of applications, including pipeline-based data processing like the proposed ML IoT scenario, or even in robotics and automotive ones.

6. Conclusions

The next generation of IoT applications is expected to truly embody an ML approach while, at the same time, becoming fully decentralized across the entire Cloud-to-Things continuum. Not only will the development of such applications become more complex, but also their lifecycle management.

This work proposes Zenoh-Flow as a framework for implementing E2E ML pipelines employing dataflow programming patterns. This framework was demonstrated over a real-world application from the Smart Green Homes project, with results showcasing a better performance in terms of throughput and latency and lower network overheads without a negative impact on the model training compared to the MQTT implementation. Benefits in terms of development, including re-definition of the ML pipeline or its scalability, are also achieved by abstracting all the communication aspects and requiring lines of code to be implemented only for the application logic.

In future work, we intend to improve the pipeline by creating trainers and evaluators dedicated to each house, as it will help the model results improve significantly. In addition, we want to apply the proposed solution in different tasks, such as distributed learning, as the tool to move model weights for training using various machines.

Author Contributions: Conceptualization, R.T. and C.G.; Methodology, G.B. and R.T.; Software, G.B. and R.T.; Validation, C.G., M.A., D.G. and A.C.; Data curation, M.A.; Writing—original draft, G.B. and R.T.; Writing—review & editing, C.G., M.A., D.G. and A.C. All authors have read and agreed to the published version of the manuscript.

Funding: This project has been partially founded by Horizon 2020 DAEMON (grant no. 101017109), Horizon Europe ICOS (grant no. 101070177), and partially supported by CT/MCTES through national funds and when applicable co-funded EU funds under the project UIDB/50008/2020-UIDP/50008/2020.

Data Availability Statement: The code used to obtain the results presented is publicly available at https://github.com/gabrik/dataflow-oriented-ml-powered-iot and the dataset considered is available at https://doi.org/10.48527/RXSATI.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Laghari, A.A.; Wu, K.; Laghari, R.A.; Ali, M.; Khan, A.A. A Review and State of Art of Internet of Things (IoT). *Arch. Comput. Methods Eng.* **2021**, *29*, 1395–1413. [CrossRef]
2. Adi, E.; Anwar, A.; Baig, Z.; Zeadally, S. Machine learning and data analytics for the IoT. *Neural Comput. Appl.* **2020**, *32*, 16205–16233. [CrossRef]
3. Mallozzi, P.; Pelliccione, P.; Knauss, A.; Berger, C.; Mohammadiha, N. Autonomous Vehicles: State of the Art, Future Trends, and Challenges. In *Automotive Systems and Software Engineering: State of the Art and Future Trends*; Springer International Publishing: Berlin/Heidelberg, Germany, 2019; pp. 347–367.
4. Buschmann, F.; Henney, K.; Schmidt, D.C. *Pattern-Oriented Software Architecture, Volume 5: On Patterns and Pattern Languages*; Wiley: Hoboken, NJ, USA, 2007.
5. Samie, F.; Bauer, L.; Henkel, J. From Cloud Down to Things: An Overview of Machine Learning in Internet of Things. *IEEE Internet Things J.* **2019**, *6*, 4921–4934. [CrossRef]
6. Paraskevoulakou, E.; Kyriazis, D. Leveraging the serverless paradigm for realizing machine learning pipelines across the edge-cloud continuum. In Proceedings of the 2021 24th Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN), Paris, France, 1–4 March 2021. [CrossRef]
7. Johnston, W.M.; Hanna, J.R.P.; Millar, R.J. Advances in dataflow programming languages. *ACM Comput. Surv.* **2004**, *36*, 1–34. [CrossRef]
8. Khan, F.; Kumar, R.L.; Kadry, S.; Nam, Y.; Meqdad, M.N. Autonomous vehicles: A study of implementation and security. *Int. J. Electr. Comput. Eng.* **2021**, *11*, 3013–3021. [CrossRef]
9. Bosch Termotecnologia S.A, Universidade de Aveiro. Smart Green Homes. 2023. Available online: https://www.ua.pt/pt/smartgreenhomes/ (accessed on 11 September 2023).
10. Tamburri, D.A. Sustainable MLOps: Trends and Challenges. In Proceedings of the 2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), Timisoara, Romania, 1–4 September 2020. [CrossRef]
11. Kreuzberger, D.; Kühl, N.; Hirschl, S. Machine Learning Operations (MLOps): Overview, Definition, and Architecture. *IEEE Access* **2023**, *11*, 31866–31879. [CrossRef]

12. Mäkinen, S.; Skogström, H.; Laaksonen, E.; Mikkonen, T. Who Needs MLOps: What Data Scientists Seek to Accomplish and How Can MLOps Help? In Proceedings of the 2021 IEEE/ACM 1st Workshop on AI Engineering—Software Engineering for AI (WAIN), Madrid, Spain, 30–31 May 2021; pp. 109–112. [CrossRef]
13. Kahn, G. The Semantics of a Simple Language for Parallel Programming. In Proceedings of the Information Processing, Proceedings of the 6th IFIP Congress 1974, Stockholm, Sweden, 5–10 August 1974; North-Holland: Amsterdam, The Netherlands, 1974; pp. 471–475.
14. Lee, E.; Parks, T. Dataflow Process Networks. *Proc. IEEE* **1995**, *83*, 773–801. [CrossRef]
15. Gramaglia, M.; Camelo, M.; Fuentes, L.; Ballesteros, J.; Baldoni, G.; Cominardi, L.; Garcia-Saavedra, A.; Fiore, M. Network Intelligence for Virtualized RAN Orchestration: The DAEMON Approach. In Proceedings of the 2022 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit), Grenoble, France, 7–10 June 2022; pp. 482–487. [CrossRef]
16. Ahlgren, B.; Dannewitz, C.; Imbrenda, C.; Kutscher, D.; Ohlman, B. A survey of information-centric networking. *IEEE Commun. Mag.* **2012**, *50*, 26–36. [CrossRef]
17. Baldoni, G.; Loudet, J.; Cominardi, L.; Corsaro, A.; He, Y. Zenoh-based Dataflow Framework for Autonomous Vehicles. In Proceedings of the 2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C), Hainan, China, 6–10 December 2021; pp. 555–560. [CrossRef]
18. Kulkarni, S.S.; Demirbas, M.; Madappa, D.; Avva, B.; Leone, M. Logical Physical Clocks. In *Principles of Distributed Systems*; Springer: Cham, Switzerland, 2014; pp. 17–32.
19. Baldoni, G.; Teixeira, R. Repository for the paper entitled A Dataflow-Oriented Approach for Machine-Learning-Powered Internet of Things Applications. Available online: https://github.com/gabrik/dataflow-oriented-ml-powered-iot (accessed on 11 September 2023).
20. Santos, M.; Antunes, M.; Gomes, D.; Aguiar, R.L. Home Comfort Dataset: Acquired from SGH. *Data* **2023**, *8*, 58. [CrossRef]
21. Ray, S. A Quick Review of Machine Learning Algorithms. In Proceedings of the 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon), Faridabad, India, 14–16 February 2019; pp. 35–39. [CrossRef]
22. Singh, A.; Thakur, N.; Sharma, A. A review of supervised machine learning algorithms. In Proceedings of the 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 16–18 March 2016; pp. 1310–1315.
23. Xu, Y.; Zhou, Y.; Sekula, P.; Ding, L. Machine learning in construction: From shallow to deep learning. *Dev. Built Environ.* **2021**, *6*, 100045. [CrossRef]