ARTICLE IN PRESS

Internet of Things xxx (xxxx) xxx

Contents lists available at ScienceDirect

Internet of Things

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

AnoML-IoT: An end to end re-configurable multi-protocol anomaly detection pipeline for Internet of Things

Hakan Kayan [a], Yasar Majib [a], Wael Alsafery [a], Mahmoud Barhamgi [b], Charith Perera [a,*]

[a] *Cardiff University, UK*
[b] *Claude Bernard Lyon 1 University, France*

ARTICLE INFO

Keywords:
Internet of Things
Data science
Pipeline
Data analytics
Multi-protocol

ABSTRACT

The rapid development in ubiquitous computing has enabled the use of microcontrollers as edge devices. These devices are used to develop truly distributed IoT-based mechanisms where machine learning (ML) models are utilized. However, integrating ML models to edge devices requires an understanding of various software tools such as programming languages and domain-specific knowledge. Anomaly detection is one of the domains where a high level of expertise is required to achieve promising results. In this work, we present AnoML which is an end-to-end data science pipeline that allows the integration of multiple wireless communication protocols, anomaly detection algorithms, deployment to the edge, fog, and cloud platforms with minimal user interaction. We facilitate the development of IoT anomaly detection mechanisms by reducing the barriers that are formed due to the heterogeneity of an IoT environment. The proposed pipeline supports four main phases: (i) data ingestion, (ii) model training, (iii) model deployment, (iv) inference and maintaining. We evaluate the pipeline with two anomaly detection datasets while comparing the efficiency of several machine learning algorithms within different nodes. We also provide the source code of the developed tools which are the main components of the pipeline.

1. Introduction

Edge AI which is critical for resource-constrained environments that operates in the Internet of Things (IoT) domain where intelligent tasks are performed has started to become a hot topic with the arrival of Industry 4.0 [1]. It manages the interaction with the physical world that is provided by sensors and actuators. Management of such an environment requires series of tasks (e.g., data collection, anomaly detection) that are operated by microcontrollers running ML models. Data-related professions (e.g., data scientists, ML engineers) define rules/ranges and search for the best practices to increase the operability of edge mechanisms in their relevant scientific disciplines. Finding hidden information from big data can enhance the quality of living but it is not a straightforward task [2].

While for data scientists, being an expert in edge-related infrastructures (e.g., programming languages, microcontrollers, sensors) is not expected, they should be able to utilize data science pipelines which are executable workflows of data-related tasks that automate the desired process. Thus, we developed a reconfigurable data science pipeline based on an IoT sensing infrastructure that utilizes open-source software to facilitate developing an interconnected anomaly detection system that runs on edge, fog, and cloud

* Corresponding author.
E-mail address: pererac@cardiff.ac.uk (C. Perera).

https://doi.org/10.1016/j.iot.2021.100437
Received 15 February 2021; Received in revised form 12 July 2021; Accepted 13 July 2021
Available online 30 July 2021
2542-6605/© 2021 Elsevier B.V. All rights reserved.

platforms. We define the edge as the platform where the first interaction between the cyber and physical world happens. Hence, microcontrollers (e.g., Raspberry Pi Pico) that gather physical data are edge devices. We define fog as the platform where several edge devices can be supervised. Hence, single-board computers (e.g., Raspberry Pi 4B) are fog devices that might act as edge devices as well. Cloud is the platform where real-world data gathered by the edge and fog devices are progressed. We implemented our system based on an example use case scenario to describe how to proposed system works while providing some results.

The contributions of this paper are as follows:

- We provide reconfigurable IoT sensing infrastructure that consists of two main open-source components: (i) The Edge to Cloud Code Generator (EECG) that generates ready-to-deploy codes to enable data circulation from edge to fog. (ii) The Node-RED package is hosted on Node-RED servers that enables accessing and processing to the edge data from anywhere that has access to Node-RED servers while offering visualization via the graphical user interface (GUI). We also provide one Python library and executable shell script that facilitate data training and inference phases.
- We propose a data science pipeline that interconnects edge, fog and cloud devices/services to provide end-to-end anomaly detection system development. The pipeline contains four main stages: (i) the data collection which is provided by the components mentioned at the first contribution point, (ii) the anomaly detection model training, (iii) model deployment to the edge, fog, and cloud, (iv) inference, and maintaining the model based on the new data. We demonstrate how the proposed tools are utilized during these stages.
- We provide a dataset that is generated via the utilization of proposed tool. We analyze the performance of Convolutional Neural Network (CNN) [3], Recurrent Neural Network (RNN) [4], Isolation Forest [5] and One-class Support Vector Machines (OC-SVM) [6] on the proposed dataset [7] and the WADI dataset [8]. We also evaluate them according to the platform (edge, for or cloud) where the anomaly detection model is deployed via the utilization of proposed pipeline. In the edge, we only evaluated CNN due to lack of application programming interface (API).

Structure of the Paper: This section provides a high-level understanding of what we proposed. We outlined the previous commercial and academic works in Section 2. Section 3 contains the architecture of an IoT anomaly detection pipeline infrastructure. Section 4 presents details about how the data circulated and progressed within the AnoML-IoT pipeline. We demonstrate our evaluations and results in Section 5. Then, we discuss about the results in Section 6 and finally provide our conclusions in Section 7.

2. Related work

In this section, we introduce the data science pipelines that are offered either by academia or commercial entities, and anomaly detection techniques in time-series sensor data. We also analyze the capabilities of open-source platforms that facilitates data circulation.

2.1. Unsupervised anomaly detection in time series sensor data

Anomaly detection is one of the fundamental fields that utilize the machine learning (ML) model as the main component. There is extensive research being done in this field [9–11]. There are three types of anomalies: (i) point anomalies, (ii) contextual anomalies, (iii) collective anomalies. If the anomalies are contextual where the context is time, the time series anomaly detection models are applied. For example, in an environment where the weather temperature decreases at night if the temperature value generated by the sensor acts otherwise, there is a contextual anomaly. While point anomalies are easier to detect, contextual and collective anomaly detection requires additional tasks to identify the normal behavior of the system.

The nature of the input data is the core element that determines the efficiency of the ML model. The features of the data may depend on several complementary terms such as labels, context, and domain. For example, if the input data do not contain any labels that define normality, unsupervised algorithms [12] are applied, if the data is related to a certain context, context-aware [13] methods are selected, if the environment is industrial, because of the importance of detection time, faster models with reduced complexity [14] are preferred.

In an interconnected domain such as IoT, cyber–physical systems [15] are utilized to supervise the environment. These systems observe the behavioral changes (e.g., change in the temperature or movement) in surroundings through modules that manage sensors [16]. They can also act as controllers if they contain actuators. In such environments, the anomaly might occur either by independent or dependent events. If the events are independent, univariate analysis [17] is applied. For example, the behavioral changes in temperature, loudness, light density, and humidity can be detected via the related data only, hence require univariate analysis. However, changes in the angular momentum or acceleration are measured by sensors (e.g., accelerometer, gyroscope) that generate data per dimension. Hence, the relation between the data points should also be analyzed to detect anomalies. Then, the multivariate analysis [18] is applied.

While academia keeps offering new anomaly detection algorithms [19,20], most of the time these are based on the fundamental ones [21]. Hence, for this work, we selected the following algorithms as they are the most common ones that are utilized for the unsupervised anomaly detection in time series data and accessible via common ML programming libraries/frameworks (e.g., scikit-learn [22], TensorFlow [23]) : (i) convolutional neural networks (CNN) [24,25], (ii) recurrent neural networks (RNN) [26], (iii) isolation forest (IF) [5], and (iv) one-class support vector machines (OC-SVM) [6]. Fig. 1 demonstrates the used algorithms in this study.

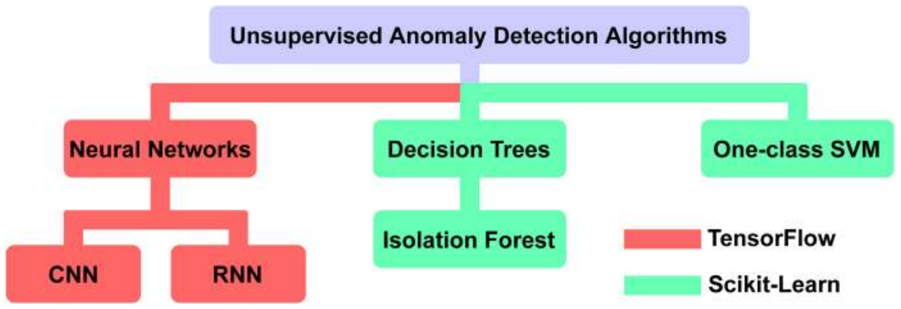


Fig. 1. Illustrates the utilized algorithms. TensorFlow also has an API [27] for decision trees, but due to having better documentation we prefer using scikit-learn for implementing Isolation Forest.

2.2. Machine learning platforms and data science pipelines

Machine learning platforms. After showing promising results in a variety of tasks including speech recognition, image processing, anomaly detection, and medical diagnosis, ML has taken an interest in both academic and commercial entities, hence resulting in the creation of many open-source and proprietary ML platforms and pipelines. ML models can be generated via hand-coding, code generators, or interpreters. *Hand coding.* There are many machine learning libraries available [22,23,28] that allows user to create ML models or to deploy and evaluate ML algorithms. A person with the profession might prefer hand-coding as it offers high customization, allows the development and employment of novel algorithms, and is easy to maintain. However, hand-coding might be very resource-consuming, thus most of the time it is done by a group of programmers. *Code generators*. ML consists of many steps (e.g., data acquisition, data pre-processing, and fitting). Rather than hand-coding all these steps, code generators [29,30] might be utilized to facilitate the process. Due to the variety of complicated tasks, most code generators provide a specific code for a specific task. *Interpreters*. One of the main challenges of ML is the portability of the generated model. Interpreters provide portability by generating a model file that can be run on other platforms with minimal coding. TensorFlow [23] is the most common one that offers model generation for resource-constrained platforms.

Data science pipelines. Raw data are needed to be interpreted to be utilized within data science-related tasks. If the data science pipeline contains all the steps that are required to interpret the data from data gathering to deployment of a machine learning model, it is called end-to-end. These end-to-end pipelines can be either manual where the user provides many inputs and sets parameters each time before a new model is generated or automated where little to no input is taken. Due to a variety of data types, automated pipelines put a certain set of rules (e.g., time format) for their system to accept the input data [31]. These pipelines can also be named according to the performed tasks (e.g., anomaly detection pipeline). Now we introduce pipelines that are presented by either industry or academia.

Azure Machine Learning Pipeline [32]. Microsoft provides an ML pipeline based on running Python scripts on the cloud while automatically handling resource usage. Each step of the pipeline can be independently customized hence offering scalability to the end-user. One of the practical features that Azure Machine Learning Pipeline offers is the automated dependency handling that allows the usage of a variety of hardware and software environments. Microsoft also provides Azure Cognitive Services [33] where you can utilize their ML pipeline and Anomaly Detector [34] service. They apply Graph Attention Network (GAN) [35] for multivariate analysis, apply SR-CNN [31] for the univariate analysis.

Amazon Web Services (AWS) Machine Learning Pipeline [36]. Amazon provides an end-to-end ML pipeline as a service for detecting anomalies in real-time. Inside the pipeline, there are many different services (e.g., database, data formatting) that can be utilized for pipeline tasks. Amazon SageMaker [37] is the main service that provides anomaly detection for both univariate and multivariate data. It allows users to either use a built-in unsupervised anomaly detection algorithm based on Random Cut Forest (RCF) [38] or use a custom algorithm that can be deployed via a Docker image. Now we introduce the pipelines proposed by the academia.

Prado et al. [39] propose an end-to-end modular AI pipeline that allows users with less expertise to implement their AI applications such as keyword spotting, image classification, and object detection to systems that contain embedded devices. Their framework relies on Low Power Deep Neural Network (LPDNN) that contains an Inference Engine (LNE) that is compatible with Caffe [40]. LNE is a code generator that facilitates the deployment to the embedded devices. The authors use FIWARE [41] for IoT hub integration and Kurento Media server [42] for media streaming which are required to run live inference. The authors define Raspberry Pi devices as edge and evaluate the efficiency of LPDNN compared TF Lite [43] on these devices by running benchmarks that are included in the TF Lite repository.

Drori et al. [44] propose an automatic ML (AutoML) system that optimizes the ML pipeline according to the given dataset. Their pipeline utilizes LSTM-RNN as a base ML algorithm. Monte Carlo Tree Search (MCTS) [45] is applied to the predictions generated by the LSTM-RNN to evaluate the performance of the pipeline and decide on the better pipeline. They evaluate the proposed pipeline compared to baseline stochastic gradient descent (SGD) [46] estimators from scikit-learn [22]. They claim their pipeline provides faster run time according to its peers.

Sutton et al. [47] propose an open-source ML pipeline that receives physiological data that is used to identify anomalous behaviors as an input in real-time. The authors try to detect Paroxysmal atrial fibrillation (PAF) by applying Probabilistic Symbolic

Table 1
Comparison of AnoML-IoT with the previous works.

Related Work	Topic	Environment	Commercial	Open-source	End-to-End	Time-series Data	Adaptability
Azure Machine Learning Pipeline [32]	General	General	✓		✓	✓	
AWS Machine Learning Pipeline [36]	General	General	✓		✓	✓	✓
Prado et al. [39]	Classification	IoT	✓		✓	✓	
Sutton et al. [47]	Anomaly Detection	Medical		✓		✓	
Nitsche and Halbritter [49]	Classification	Linguistics					
Shaikh et al. [50]	Policy	General			✓	✓	
Boovaraghavan et al. [51]	Classification	IoT			✓	✓	
Molinara et al. [52]	Classification	IoT			✓		
Vinzamuri et al. [53]	Anomaly Detection	IoT			✓	✓	
Li et al. [54]	Anomaly Detection	General			✓	✓	✓
AnoML-IoT	Anomaly Detection	IoT		✓	✓	✓	✓

Pattern Recognition (PSPR) to the Electrocardiogram (ECG) signals. PSPR is used for online feature extraction while they apply random forest (RF) to classify ECG data. The proposed pipeline is based on Spark's ML library (MLlib) [48], hence allows other anomaly detection techniques included within MLlib.

Nitsche and Halbritter [49] propose a data science pipeline that is optimized for text classification. The authors benchmark different GPUs to evaluate the performance of their hardware setup which consists of 10 NVIDIA Quadro P6000 and the effect of the number of GPUs on the image processing time. They apply the Naive Bayes classifier that is included in scikit-learn API and achieve above 90% accuracy on Deutsche Presse-Agentur (dpa) dataset.

Shaikh et al. [50] focus the challenges of ensuring policy fairness within end-to-end ML pipelines. They claim the ML-based tasks are done by engineers that have a variety of professions including data creators and future engineers. Hence, each step of the ML pipeline might be subjected to a policy violation. The authors provide an end-to-end ML pipeline that is based on log management to prevent these violations as manually ensuring policy fairness is highly resource-consuming.

Boovaraghavan et al. [51] propose an adaptive end-to-end ML system for IoT applications. Their pipeline is optimized for activity recognition-based tasks including object recognition. Authors claim that the main challenge regarding end-to-end pipeline is due to the heterogeneity of IoT applications. Authors evaluate their pipeline with various hardware platforms and datasets while comparing prediction time and accuracy per each machine learning technique they applied.

Molinara et al. [52] propose an end-to-end ML-based indoor air monitoring system for contaminant classification. Authors compare the performances of Multi Layer Perceptron (MLP) to CNN and LSTM based deep learning techniques while testing the performance of MLP and CNN on ESP32 MCU. They investigate the power consumption of the MCU regarding the utilized ML technique. They claim the proposed system is only lacked to classify alcohol and acetone due to their chemical similarities.

Vinzamuri et al. [53] propose an end-to-end context-aware anomaly detection system that requires time-series data. The proposed system utilizes a semi-supervised algorithm with Sparse Gaussian Graphical Models. They benchmark the pipeline on several public datasets. The authors claim semantics can improve the Gaussian Graphical Models further beyond other anomaly detection techniques. Their ML comparison is based on F-Score as the authors mention that the proposed pipeline is promising for industrial IoT environments.

Li et al. [54] develop an end-to-end automated anomaly detection system. They utilize Apache Spark backend server to run the query-based operations. After the user provides a dataset, the proposed system automatically selects the most appropriate algorithm then applies anomaly detection. Finally, the results are shown in figures within the pipeline. They benchmark the proposed system based on several datasets while applying quantification analysis.

Our pipeline utilizes scikit-learn and TensorFlow for the anomaly detection while relying on TensorFlow Lite for the deployment on the fog, TensorFlow Lite Micro for the deployment on the edge devices. Hence, it allows the lightweight implementation of anomaly detection techniques while offering a variety of communication protocols (e.g., Bluetooth low energy (BLE)) and sensor types for the inference. The Table 1 compares our pipeline with the previous works based on significant features that determine the efficiency of the pipeline.

3. The architecture of the AnoML-IoT pipeline

In this part, we present the overall architecture of our pipeline by defining main components, describing the workflows that differ according to application scenario, and explaining how to automate these workflows to maintain the pipeline.

3.1. AnoML-IoT layers and application scenarios

IoT infrastructures should provide semantic data exchange to be considered as completely ubiquitous. Current technologies that are utilized in IoT architectures are rapidly evolving to achieve semantic interoperability, hence causing the debate of what kind of infrastructure is needed for a certain task. Due to each technology has a variety of pros and cons per IoT environment, extensive testing is required to decide on IoT elements (e.g., wireless technologies, edge/fog node types, sensors, and actuators). Building an IoT application from scratch to perform these tests requires intensive labor. Thus, reconfigurable IoT sensing architecture that

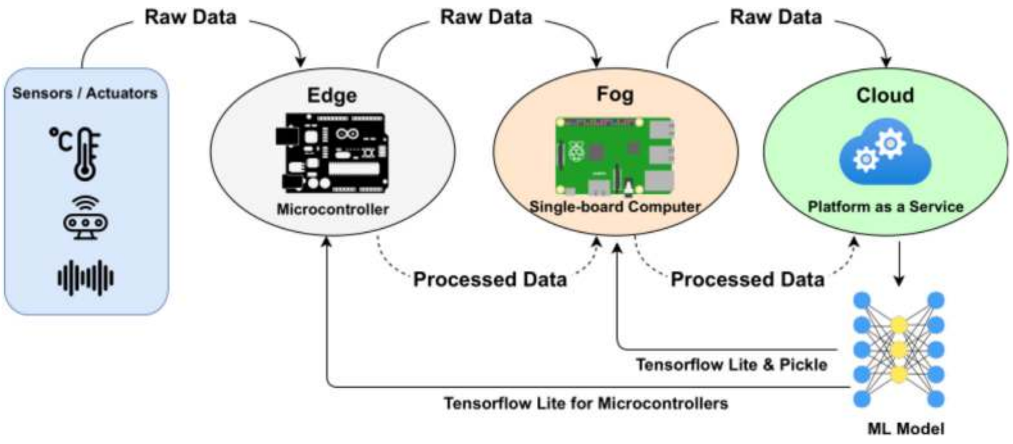


Fig. 2. Overview of the data circulation and the pipeline application scenarios based on the location of anomaly detection: (i) If the anomaly detection is done on the cloud, the edge and fog devices only forward data, (ii) if the anomaly detection is done on the fog, the edge sends raw data to fog, then the fog might send processed data to the cloud, (iii) if the anomaly detection is done on the edge, the processed data might be sent to fog and then to cloud. Each scenario has its pros and cons that we introduce in Section 5. Here processed data contains information to be used to decide if the data is anomalous or not. Thus, it might be either binary (e.g., 0 for normal data, 1 for anomalous data) or a decimal (float) as an anomaly score that varies in the range of -1 to 1. The model in the cloud will be updated with the new normal data according to the predetermined intervals.

allows the implementation of various ML-based application scenarios including anomaly detection with minimum user (e.g., data scientist) interaction is required. AnoML-IoT allows the implementation of a variety of application scenarios where the scenarios are evolved around the platform that the anomaly detection is implemented. In this context, AnoML-IoT consists of three layers:

- *Edge Layer*: The edge layer contains edge nodes that consist of microcontrollers, sensors, and actuators. These nodes are physically observing the IoT environment by gathering data via sensors while conducting physical operations via actuators (e.g., the fan stops working when a certain degree is reached).
- *Fog Layer*: The fog layer contains fog nodes that get the data from edge nodes and process it according to the end user's preferences. Usually, fog nodes offer more computing power than edge nodes while allowing flexible deployment options. Even though fog computing may be considered as an alternative to cloud computing, fog nodes can also act as an IoT gateway between the edge nodes and the cloud. In this work, we utilize Raspberry Pi 4 as a fog device and present our results. We believe, similar Linux-based devices can be used as fog devices instead of the Raspberry Pi for the proposed pipeline.
- *Cloud Layer*: The cloud layer provides services ranging from data management, storing, applying anomaly detection to developing multi-purpose frameworks. Even though, cloud computing offers many benefits (e.g., automatic service integration, and high accessibility), edge/fog computing is preferred where time-critical or confidential applications (e.g., industrial) are present to reduce reliance on cloud services. While the edge and fog layer might contain standalone nodes, the cloud layer requires interaction with other layers.

Fig. 2 summarizes the working principle of the machine learning pipeline. The anomaly detection can be done within the pipeline on edge, fog, or cloud. The edge and fog devices also can be utilized just to forward raw data to the platform at one upper level. The initial training requires prior data. Hence, if there is no dataset to be used for initial training, the edge and fog will send only raw data until the cloud can generate an efficient ML model. Then the model will be deployed to edge, fog, or cloud. According to the given intervals, the new model will replace the old model to keep the system up-to-date. Having an up-to-date ML model is significant to prevent a decrease in efficiency that depends on the dynamic context.

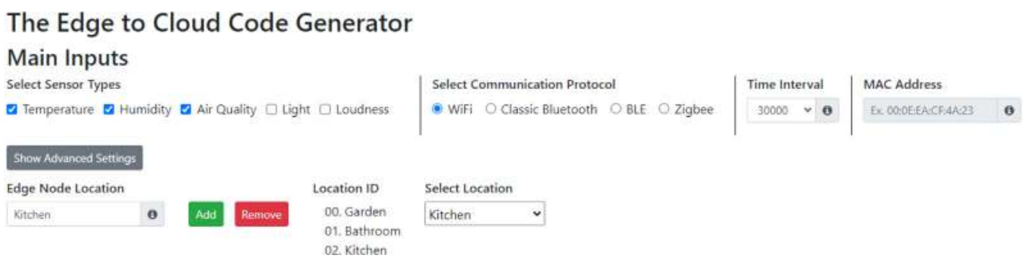
3.2. The AnoML-IoT open-source tools

AnoML-IoT consists of three main tools: (i) Edge to Cloud Code Generator (ECCG), (ii) Node-RED package, (iii) python library. Now we introduce what these tools provide, what are their functionalities, use cases, and roles within the AnoML-IoT pipeline. The source code of the tools is published in GitLab.[1]

3.2.1. The Edge to Cloud Code Generator

The ECCG is a web-based code generator that generates edge code to be either used for inference or to transmit data to the fog node. Currently, it is only compatible with Arduino IDE [55]. It has a user-friendly interface, where even the data scientist with minimal IoT knowledge can design a basic IoT application that contains several sensors where the sensor data can be transferred between layers. The characteristics of ECCG including the justification for design choices are given below:

[1] https://gitlab.com/IOTGarage/anoml-iot-analytics.


The Edge to Cloud Code Generator

Main Inputs

Select Sensor Types: ☑ Temperature ☑ Humidity ☑ Air Quality ☐ Light ☐ Loudness

Select Communication Protocol: ● WiFi ○ Classic Bluetooth ○ BLE ○ Zigbee

Time Interval: 30000

MAC Address: Ex. 00:0E:EA:CF:4A:23

Show Advanced Settings

Edge Node Location: Kitchen

Add Remove

Location ID: 00. Garden, 01. Bathroom, 02. Kitchen

Select Location: Kitchen

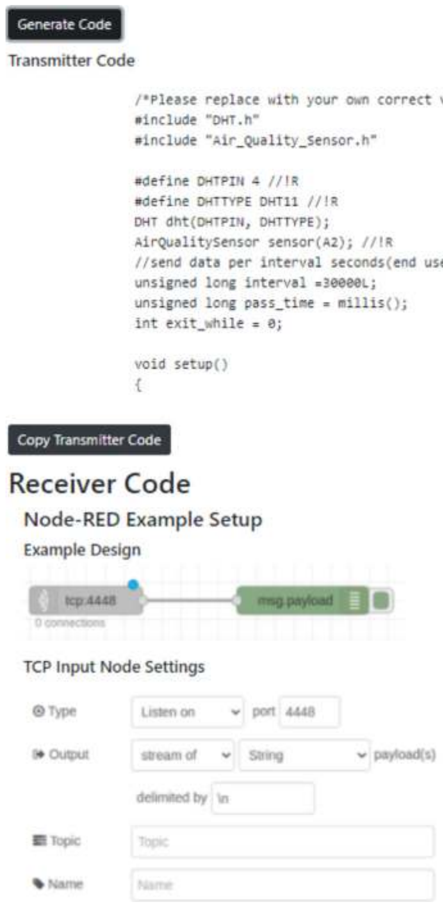
Fig. 3. Illustrates the main input section of the ECCG. Each input is strictly controlled for the following reasons: (i) to prevent cross-site scripting attacks, (ii) to prevent bugs that may occur in the code due to mistyping. Each text input has a tooltip that clarifies what kind of information should be given. In addition, placeholders demonstrate an example input.

- Currently, five sensor types are available: *temperature*, *humidity*, *air quality*, *light*, *loudness*. The end-user can simultaneously select all sensors. Selected sensor data will be included within the pipeline. *Justification*. These five sensor types are among the most common sensors that are utilized in IoT applications. The generated code clearly describes how the sensor data is received, and processed, thus allowing the easy adaptation of the code for a similar type of sensor.
- Four communication protocols are available: *Wi-Fi* [56], *Bluetooth Classic* [57], *BLE* [58], and *Zigbee* [59]. The end-user can only select one. The application also includes additional settings regarding communication protocols for advanced users. *Justification*. The selected four communication protocols occupy the vast majority of the IoT market and offer a variety of topologies (e.g., mesh, star, tree). Understanding the basics of how to establish these wireless technologies enables the implementation of high-range IoT applications. The generated code by ECCG defines how to handle the required network elements (e.g., MAC address, personal area network (PAN) ID). An advanced user may conveniently adapt the generated code to be used with other IoT communication protocols that are out-of-scope of this project such as WirelessHART [60].
- The data transfer rate determines the time between two data blocks. It is in milliseconds that ranges from 30 000 to 300 000. After copying to code the user can set the data transfer rate as desired. However, we do not recommend setting it below 30 s as it is the time that is required for the module to initialize. *Justification*. Data generating time differs per sensor module. Controlling the data transfer rate is necessary to ensure the integrity of transmitted data.
- The sensor locations are determined by the end-user. Identification (ID) number starting from 00 is given per location. The application supports up to 99 locations. *Justification*. Data scientists work with comma-separated value (CSV) files, to handle the further progressing of the data. The data in CSV files mostly in pairs as *text-value* where *text* is the identifier, and *value* is digital presentation of a physical quantity. To generate such files, data is transmitted in data-serialization formats (e.g., JavaScript Object Notation (JSON), Extensible Markup Language (XML)) supported by IoT application standards. Thus, the ECCG allows an end-user to define location. Then, it generates a unique location ID number to be used in data transmission.
- Three microcontroller types are available: Arduino Nano 33 BLE Sense, Arduino Nano RP2040 Connect, and Raspberry Pi Pico. The end-user can select one of the microcontrollers to obtain the edge node ID number to be used to identify the microcontroller types when needed. *Justification.* The ECCG supports the top three microcontrollers that are officially supported by TensorFlow Lite for Microcontrollers (TFLM) [61]. Supporting a variety of microcontrollers allows users to design a heterogeneous IoT application, where the environment benefits from different features of these devices (e.g., RAM, flash memory).
- The ECCG allows sending lowest, mean, or highest sensor data which are generated during the time interval determined by the user. *Justification*. In some cases, the normal range might just be determined by lower or upper limits. Hence, we allow the user to decide on the data to be sent. If the user selects mean, the mean of the number of data points generated by the sensor during the predetermined time interval will be sent.

The user interface of the ECCG facilitates usability by navigating users via input controls. Fig. 3 demonstrates the main input section of the code generator while the example inputs are given. Minimalist design is preferred to assist a data scientist with no prior IoT knowledge. Thus inputs are controlled, tooltips are included, and example inputs are given as placeholders.

After the inputs are given, the final step is clicking on the "Generate Code" button. The ECCG will output the followings: (i) the edge code that is ready to be deployed to the microcontroller via Arduino IDE, (ii) the Python3 script that should be run on the fog device, (iii) the Linux commands to be run via terminal, (iv) example Node-RED setup. The outputs differ according to the given inputs. For example, the Linux commands are only required if the Bluetooth Classic will be used within the pipeline. The Fig. 4 illustrates how the generated code blocks are presented. Two main buttons are included for each code block: the first button generates the code shown in the pre-scrollable division to let a data scientist examine the codes before further progression, the second button copies the code without breaking the format to prevent possible errors. The user manually uploads the code into the microcontroller.

We assume the following scenario: the edge node (micro-controller) gathers physically observed data via sensors and sends it to a fog node(a single-board computer (SBC)) where the data is either processed or forwarded to the cloud via Node-RED. Thus, the ECCG consists of three main sections: (i) the input field where the end-user determines the basic characteristics of the desired IoT application, (ii) the transmitter field where the edge node code is generated for a microcontroller, and (iii) the receiver field where the fog node code is generated for a single-board computer. Table 2 illustrates the specifications of the ECCG in detail.



Transmitter Code

```
/*Please replace with your own correct variables if you see "!R" near the given variable.*/
#include "DHT.h"
#include "Air_Quality_Sensor.h"

#define DHTPIN 4 //!R
#define DHTTYPE DHT11 //!R
DHT dht(DHTPIN, DHTTYPE);
AirQualitySensor sensor(A2); //!R
//send data per interval seconds(end user decides)
unsigned long interval =30000L;
unsigned long pass_time = millis();
int exit_while = 0;

void setup()
{
```

Receiver Code

Node-RED Example Setup

Example Design

TCP Input Node Settings

Type	Listen on	port	4448
Output	stream of	String	payload(s)
	delimited by	\n	
Topic	Topic		
Name	Name		

Fig. 4. Illustrates the generated code block and the example Node-RED setup. In the example scenario, Raspberry Pi Pico is selected as an edge node/transmitter. The generated transmitter code is written in C++ and should be deployed via Arduino IDE. The Raspberry Pi 4B is acting as a fog node/receiver.

3.2.2. The Node-RED package

Node-RED [62] is an open-source browser-based workflow development tool that runs on Node.js [63] based runtime. We can develop workflows via utilizing drag-and-drop nodes on resource-constraint environments such as Raspberry Pi thanks to the non-blocking nature of Node.js. Hence, we use Node-RED on the fog device to develop and control the workflows within the AnoML-IoT pipeline. Even though Node-RED contains many open-source packages developed for the IoT networks, we could not find any up-to-date package that provides the development of Bluetooth Low Energy (BLE) modules while offering customization choices. Hence, we developed and published our package under the name of "node-red-contrib-ble-sense". The package contains the following nodes:

- **BLE Scanner**. BLE devices can operate in four different roles: broadcaster, observer, central, and peripheral. The task of the BLE Scanner node is to scan for the advertising BLE peripherals. It can specifically search for a local name and output one of the followings: whole peripheral as an object, MAC address, and advertisement data. So, the user can use this node either to discover the MAC address of the target BLE peripheral or to get the advertisement data.
- **BLE Connect**. This node establishes a BLE connection with the target peripheral device. While one peripheral device can only be connected to one central, central devices can manage multiple peripherals. Hence, our pipeline allows the deployment of multiple edge devices via BLE while the maximum number of peripherals that can be connected depends on the system-on-a-chip (SoC) of the microcontroller. For example, the Arduino Nano 33 BLE Sense [64] SoC nRF52840 [65] supports up to 20 parallel connections.

3.2.3. The AnoML.py and SetupAnoML.sh

AnoML.py. Library of functions in python which can support most of the tasks in developing ML models with various types of normalization and data points. Generating different models for the cloud, fog and edge platforms using various normalization methods and data points is a labor-intensive task. Evaluating different models for unsupervised ML when there is a small amount of anomalous data is also another challenge. Our Python library supports data preprocessing to generating ready to deploy ML models for the cloud, fog, and edge for unsupervised ML. It also provides performance visualization for each data point while

Table 2
The specifications of the Edge to Cloud Code Generator.

Main inputs	Sensor types	Temperature Humidity Loudness Light Air Quality
	Communication protocols	Wi-Fi Bluetooth Classic BLE Zigbee
	Edge node types	Arduino Nano 33 BLE Sense Arduino Nano RP2040 Connect Raspberry Pi Pico
	Edge node identifiers	Edge Node Location Edge Node ID Number
Advanced inputs	Wi-Fi	Service Set Identifier (SSID) Password Host IP Address Host Port
	Bluetooth & BLE	*MAC Address Module Name Module PIN
	Zigbee	PAN ID Destination Address High Destination Address Low
Generated outputs	Node-RED example setup	Node Settings Workspace Design Example Functions
	Transmitter code	Arduino Scripts Python Scripts
	Receiver code	JavaScript Function Nodes Python Scripts

*Among advanced inputs, only the MAC address is obligatory. The default options that are included in the code are explained via comments in detail.

including normalization techniques which helps to select the most efficient model. The proposed library provides data cleaning, normalization, reduction, scaler, and visualization functions before feeding into machine learning models. The library also provides various functions that allow training and testing. The inference part of the library provides functions to evaluate different machine learning models generated using our library at both fog and cloud platforms so the performance of the platforms can be compared.

SetupAnoML.sh. It acts as an installer to prepare both fog and cloud platforms for action. It installs correct versions of necessary packages and libraries required to run inference. Once the prerequisites are handled, it can then download, install and configure packages such as Node-RED and TensorFlow runtime. Also, it sets our custom-developed services to listen on web ports to order to allow microcontrollers to interact with fog devices and fog devices to interact with the cloud platform. The script can be configured to download files and configurations either from Google Drive or a local/remote FTP Server. Users also can manually download, install and configure all prerequisite requirements and model files if it is desired.

4. The data circulation

The nature of the input data shapes the characteristics of the data science pipeline. Hence, gathering raw data is the first step in this kind of pipeline. In the IoT environment, the sensors generate a variety of data in various formats. While most of the temperature sensors generate data in floating-point numbers (floats) to provide more accuracy, the light sensors that measure light density usually provide data in integer. This is significant for two main reasons: (i) each data type occupies memory in different sizes, (ii) there might be different regulations [66,67] according to the application domain for certain data types such as floats. Hence, the memory usage should be optimized for microcontrollers that have very limited memory. The memory usage might also differ according to the microcontroller architecture. For instance, while Arduino Uno [68] stores int values in two bytes, Arduino Due [69] stores int values in four bytes. The data circulation within the AnoML-IoT pipeline consist of four main steps: (i) data ingestion, (ii) data training, (iii) model deployment, (iv) inference and maintaining.

4.1. Data ingestion

Data ingestion is the first step of all kinds of data science pipelines unless the user already has a ready-to-deploy ML model. In our pipeline, the ECCG is the main tool that utilizes the data ingestion process. Fig. 3 illustrates the choices offered by the ECCG.

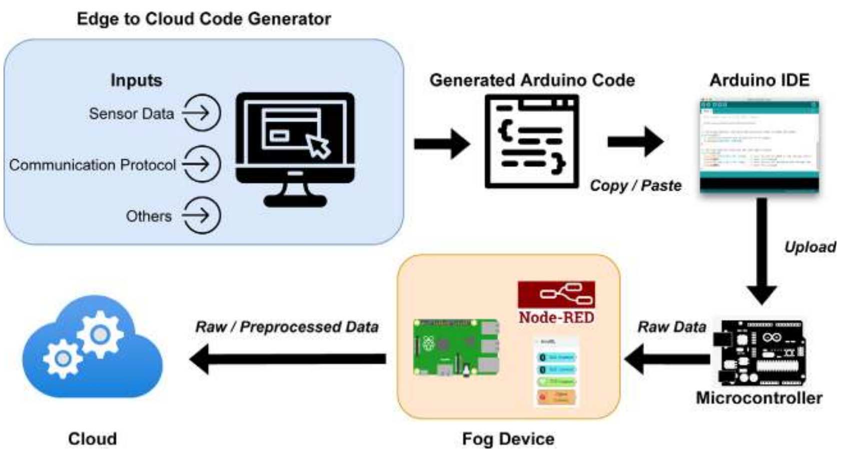


Fig. 5. Illustrates data ingestion phase. The data preprocessing can be done on all platforms. However, due to the limited computing power of edge and fog devices, we prefer to utilize the cloud. In addition, if more than one edge device is connected to the fog device, data preprocessing can be done on the fog device to identify the edge devices.

The user can select among the offered choices to obtain a generated code. The generated code can be copied without disrupting the format. Multiple communication protocols and Node-RED example that demonstrates how to receive data via fog device are also provided.

What the ECCG generates is an example edge code. Besides, in our setup, we use Grove sensors and shields [70] while utilizing Raspberry Pi 4B as a fog device. The IoT environment is very heterogeneous hence, the user might have or want to use different sensors, microcontrollers, or fog devices. The generated code by the ECCG should require minimal editing even when this is the case as we also provide instructions via comments for the exact lines within the code that might require editing due to individual preferences. If needed, the user can edit the generated code after copying it from the ECCG to Arduino IDE. Then, the code can be uploaded to a microcontroller. To complete these tasks the user needs a computer that can access the internet and run Arduino IDE. So, the tasks of copying the code and uploading it to a microcontroller are handled manually. Fig. 5 demonstrates the data ingestion process.

4.2. Model training

Model training can be done before ingestion if the user already has a training dataset. Otherwise, the user should follow the steps mentioned in the data ingestion phase. The AnoML.py library contains all the required functions that are needed to generate a ML model. Uploading the dataset to a cloud platform is handled manually. After uploading the dataset, the user should preprocess the data to convert to an appropriate format for the ML algorithms. Then, the ML models are generated based on the user preferences. User can generate multiple ML models at the same time. Here, we facilitate the model training by providing user a python library that is capable of preprocessing the time series data and generating multiple ML models based on given parameters with only a few line of codes. The required time depends on the capabilities of a cloud platform that is used during the training. The user should decide on the followings: (i) the size of the training dataset, (ii) the algorithm specific parameters (e.g., contamination factor for Isolation Forest). Fig. 6 illustrates the model training phase.

4.3. Model deployment

The next step after training the model is model deployment. Here we provide an executable shell script SetupAnoML.sh that facilitates the model deployment phase by automating several key tasks. The user should provide the following inputs to the script: (i) the platform (edge or cloud) where the anomaly detection technique will be deployed, (ii) the details of the place where the model is stored (e.g., server, username, password, and port names for an FTP server, Google Drive token or Google Drive). After these inputs are given, the script will automatically download, install, and configure all the required software packages (e.g., libraries, models, configurations). Then the script automatically will deploy the models to cloud, fog, or both cloud and fog platforms. The user can deploy multiple models at the same time to different platforms. Fig. 7 demonstrates the model deployment process.

4.4. Inference and maintaining

Successful deployment results in inferring all models on each type of platform. We developed an end-to-end pipeline which allows a user to generate ML models for anomaly detection from sensor data at all platforms. We developed a performance monitor which can present visualization of performance and accuracy of all type platforms, ML models, data-points and normalization/reduction techniques. Maintenance of data is a key aspect for evolution, we recommend that user should make it a practice to visualize performance comparison as a process of decision making in order to enhance performance and accuracy on all platforms.

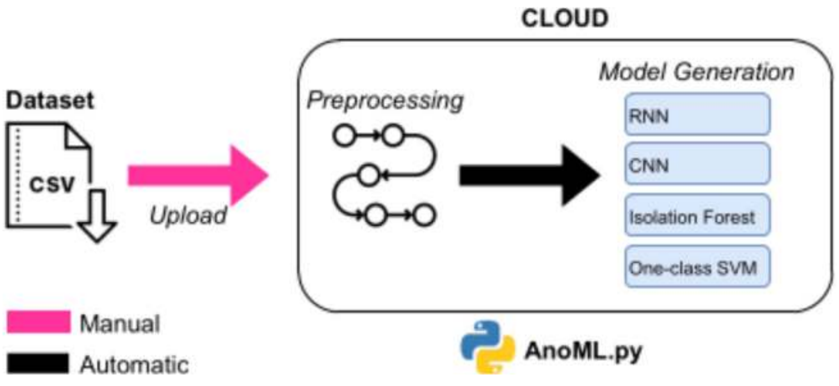


Fig. 6. Demonstrates the model training phase. Currently, the built-in algorithms within the AnoML.py are RNN, CNN, isolation forest, and One-class SVM. Multiple datasets can be utilized at the same. The storage place of these models depends on the preferences of the user. During our evaluation we used Google Drive [71] to store our models as it can be easily integrated into Google Colab [72]. Another option would be running an FTP server. The user also can utilize both at the same time.

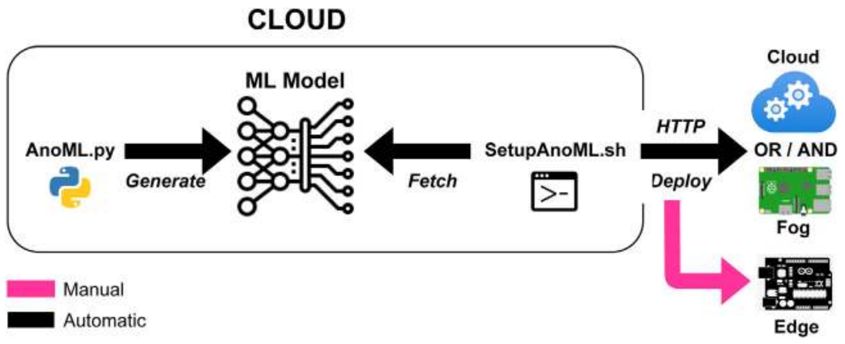


Fig. 7. Illustrates the model deployment phase. SetupAnoML is an executable Linux shell script that does the followings in order: (i) fetches generated ML models from Google Drive or FTP server, (ii) installs required libraries and packages, (iii) deploys models over HTTP. The deployment to edge should be handled manually. We are considering to provide automated edge ML deployment function via over-the-air (OTA) transmission in the future versions of AnoML-IoT.

5. Evaluation

In this section, we present the evaluations that are done to measure the efficiency of the pipeline with various configurations enabled. A data science pipeline evaluation is not a trivial task if the environment is heterogeneous such as IoT. Fig. 8 demonstrates possible scenarios available within our pipeline where different sensors and wireless communication protocols are used.

5.1. Wireless protocol comparison

The decision on the wireless protocol selection depends on the application domain as each protocol has its benefits and disadvantages. There are several important features to be considered when designing an anomaly detection pipeline for the IoT environments:

- **Latency**. The time that takes for one network package to be transmitted from the transmitting endpoint to the receiving endpoint. Latency optimization is very significant to achieve near real-time processing and inference. It is one of the significant features that determine the anomaly detection time. Low latency is aimed for the applications (e.g., industrial) where the anomaly detection time is critical [73].
- **Power consumption**. IoT is a resource-constraint environment, hence power consumption is one of the significant features to be considered.
- **Network Topology**. The data circulation and the system design depend on the network topology. While there are many topologies (e.g., star, mesh, ring) offered by the current communication protocols, the mesh, and star networks are the most common ones that are established in IoT environments.

Table 3 compares the wireless communication protocols utilized in the proposed pipeline based on the aforementioned features. We measured the latency by taking the mean of the passed time for the 1000 transmitted packages. Edge and fog devices were placed next to each other during the tests hence there was no physical barrier between the devices. No packet drop is observed.

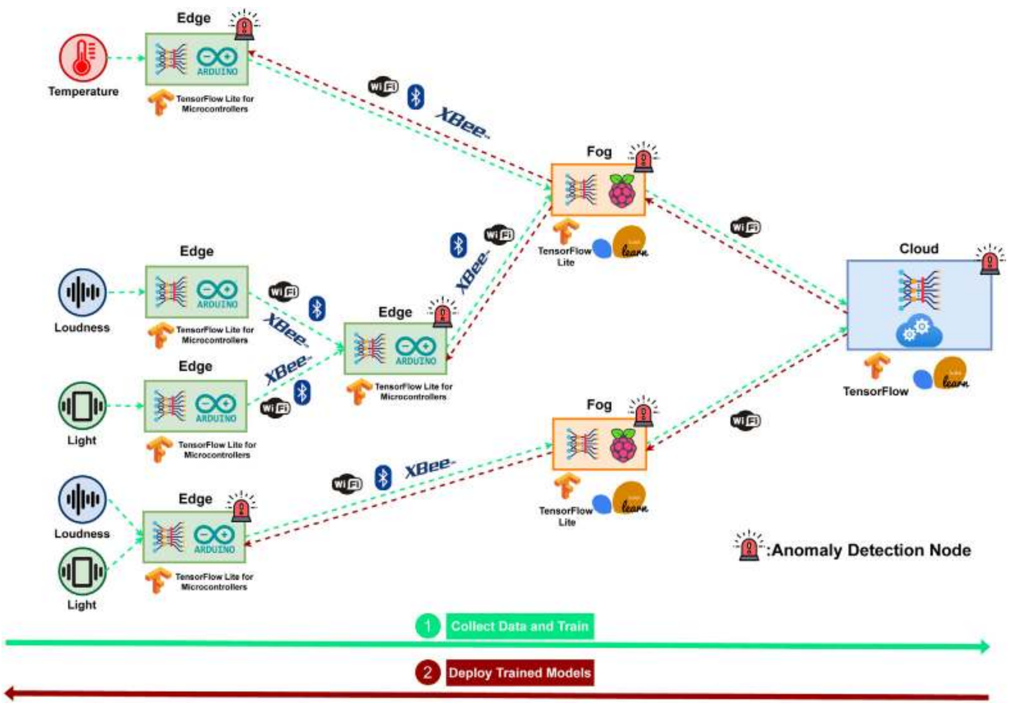


Fig. 8. Illustrates the possible scenarios that can be generated within the AnoML-IoT pipeline. We allow the use of various topologies, wireless communication protocols, and execution of use case scenarios shaped around the decision of anomaly detection platform.

Table 3
Wireless protocol comparison.

	Latency (ms)				Power consumption	Topology*
	Edge to Edge	Edge to Fog	Fog to Fog	Fog to Cloud		
Wi-Fi	18.24	14.566	17.25	21.23	High	Star
Bluetooth Classic	195.13	171.15	187.15	NA	Medium	Point-to-Point
BLE	11.23	13.45	13.21	NA	Low	Mesh
Zigbee	18.56	16.66	14.56	NA	Low	Mesh

NA: Not Applicable. ⋆: The most common topology is given.

While we observed similar latency for the BLE, Wi-Fi, and Zigbee, the Bluetooth Classic had the highest latency. However, it is challenging to have a conclusion as many factors that might affect the latency. Hence, to get the most realistic results, the tests should run in a real environment where anomaly detection takes place.

5.2. Dataset and testbed

Realistic datasets and testbeds are required to achieve the most promising results. To evaluate the efficiency of the pipeline, we built a testbed that contains components that are low-cost and accessible to most of the IoT community. When designing the IoT testbed, the sensor selection is the initial task that shapes the main futures of the testbed. In cyber–physical environments, there are several behaviors (e.g., temperature, noise, humidity) that can be considered as common. Hence, we observe these common behaviors via our testbed and generate a dataset. Table 4 demonstrates the components utilized during the evaluation.

We recorded the temperature, humidity, light, loudness, air quality data for around two days. We observed the data at the beginning to form an initial opinion about physical behavioral changes in the test environment. We realized the only temperature and humidity sensors are working as expected and let us creating anomalous behaviors. Hence, we only utilize these two data during the evaluation. Fig. 9 provides visualization of the generated dataset. Arduino Nano RP2040 Connect, Grove sensors, and Putty is used to generate the dataset which is published on Kaggle [7] where more details are also provided regarding the testing environment. In addition, we utilize the WADI [8] dataset during the evaluation to analyze how the key ML parameters such as accuracy, F1-score, and prediction time differ according to the ML platform. The dataset contains data from 123 sensors and actuators from a water distribution testbed that had non-stop run for 16 days while being attacked during the last two days. Table 5 demonstrates the differences between two utilized datasets. By using these datasets, we also evaluate the relationship between the power consumption and data volume.

Table 4
Testbed components.

Sensors	Grove — Temperature & Humidity Sensor (High-Accuracy & Mini) v1.0 Grove — Light Sensor Grove — Loudness Sensor Grove — Air Quality Sensor v1.3 Grove — UART Wifi V2 Digi XBee 3 Zigbee 3 RF Module
Microcontrollers	Arduino Nano 33 BLE Sense Raspberry Pi Pico Arduino Nano RP2040 Connect
Shields	Grove — Bee Socket Grove Shield for Pi Pico V1.0 Arduino Tiny Machine Learning Shield
Single-board Computer	Raspberry Pi 4 Model B
Software Tools	Putty Raspberry Pi Imager Node-RED Arduino IDE 1.8.15 Visual Studio Code Google Colab

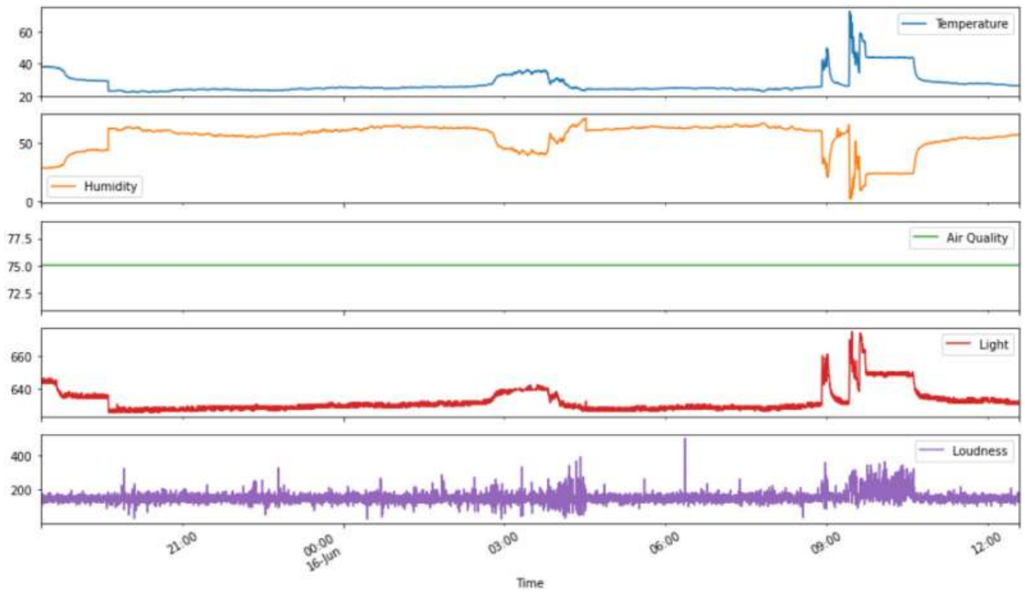


Fig. 9. Demonstrates the behavior per data type. The controlled anomalies for temperature and humidity data are identifiable. The anomalies in temperature and humidity are created via hair dryer. The air quality, light, and loudness can be considered as faulty due to not responding to our anomaly creation attempts. There might be two reasons for the occurrence of faulty data: (i) the sensors are cheap quality, (ii) these sensors are designed for the AVR [74] architecture. However, during the evaluation we used ARM-based microcontrollers [75].

Table 5
Comparison of datasets.

Features	WADI [8]	AnoML [7]
Amount of Data	122,543,744	45,906
Number of Rows	957,374	6,559
Number of Columns	129	7
Number of Sensors & Actuators	123	5
Time-series	✓	✓
Labeled	✓	✓
File size	588,906 KB	265 KB

Table 6
Evaluation results of WADI dataset.

Model details			Inference time (ms)		AUC		Accuracy		Recall		Precision		F1-Score		Scaling/Reduction time (s)		Model size (KB)	
Algorithm	SR	API	Fog	Cloud	Fog	Cloud	Fog	Cloud	Fog	Cloud	Fog	Cloud	Fog	Cloud	Fog	Cloud	Fog	Cloud
CNN-AE	Average	TF	1.01413	23.3021	0.418136	0.418136	0.323962	0.323962	0.311666	0.311666	0.91451	0.91451	0.464895	0.464895	6.88913	1.416631	34.64844	291.4189
	Kurtosis	TF	0.995147	24.0796	0.519019	0.519019	0.728089	0.728089	0.755389	0.755389	0.945001	0.945001	0.839623	0.839623	67.18984	23.71106	34.60938	291.0166
	MAD	TF	1.025978	24.03	0.448848	0.448848	0.7072	0.7072	0.740935	0.740935	0.934799	0.934799	0.826653	0.826653	72.07326	21.63906	34.64844	291.4189
	MM	TF	440.2933	96.6556	0.498538	0.498538	0.18035	0.18035	0.138802	0.138802	0.941108	0.941108	0.241923	0.241923	1.433161	0.481541	397203.7	1191798
	NS	TF	440	103.3094	0.5	0.50022	0.059669	0.059669	0.002144	0.002144	0.942253	0.953552	0.004278	0.004278	NA	NA	397194.2	1191799
	Skew	TF	1.001088	24.6464	0.361076	0.361073	0.261203	0.261197	0.248162	0.248156	0.885031	0.885028	0.387632	0.387624	69.34095	34.05004	34.60938	291.0166
	SS	TF	440	106.2	0.699918	0.699918	0.758349	0.758349	0.765979	0.765979	0.971539	0.971539	0.856599	0.856599	0.814415	0.790764	397203.6	1191796
	StDev	TF	1.051563	25.3799	0.497397	0.497397	0.929901	0.929901	0.986375	0.986375	0.941966	0.941966	0.963659	0.963659	20.04371	3.039728	34.64844	291.4189
RNN	Average	TF	8.164231	29.9278	0.431887	0.431887	0.174428	0.174428	0.140811	0.140811	0.892397	0.892397	0.243241	0.243241	6.911079	1.273132	797.8203	4045.537
	Kurtosis	TF	8.020618	30.07291	0.515931	0.515931	0.433345	0.433345	0.422561	0.422561	0.946374	0.946374	0.584251	0.584251	67.16076	22.99305	797.8047	4042.888
	MAD	TF	8.014902	29.02	0.421892	0.421892	0.69089	0.69089	0.726014	0.726014	0.930689	0.930689	0.815708	0.815708	71.88432	19.5469	797.8164	4044.854
	Skew	TF	7.621182	29.82979	0.448924	0.448965	0.544221	0.544209	0.556664	0.556646	0.932372	0.93238	0.697119	0.697107	69.21164	31.53431	797.8125	4044.202
	StDev	TF	8.09226	30.83805	0.420087	0.42009	0.436152	0.436158	0.43825	0.438256	0.922818	0.922819	0.594276	0.594282	20.04073	3.640271	797.8203	4045.537
RNN-AE	MM	TF	18.32667	40.9852	0.470241	0.470231	0.166829	0.166811	0.127211	0.127192	0.917464	0.917453	0.22344	0.223411	1.330966	0.328342	1414.535	10386.43
	NS	TF	17.90922	41.1471	0.5	0.500252	0.942253	0.05964	1	0.002107	0.942253	0.955432	0.970268	0.004205	NA	NA	1410.465	10386.67
	SS	TF	18.08283	49.55683	0.699284	0.699284	0.671158	0.671158	0.667486	0.667486	0.975904	0.975904	0.792754	0.792754	3.304777	1.005414	1414.512	10378.48
OC-SVM	Average	SK	0.894195	0.342938	0.491203	0.491203	0.7086	0.7086	0.801859	0.801859	0.84711	0.84711	0.823864	0.823864	6.88913	1.416631	56.62305	56.62305
	Kurtosis	SK	0.855004	0.341614	0.481406	0.481406	0.8127	0.8127	0.954818	0.954818	0.84496	0.84496	0.896536	0.896536	67.18984	23.71106	50.1582	50.1582
	MAD	SK	0.512389	0.207312	0.500747	0.500747	0.7141	0.7141	0.805624	0.805624	0.850137	0.850137	0.827282	0.827282	72.07326	21.63906	8.673828	8.673828
	MM	SK	0.687799	0.367881	0.5	0.5	0.1501	0.1501	0	0	1	1	0	0	1.433161	0.481541	407.5889	407.5889
	NS	SK	0.680684	0.335778	0.5	0.5	0.1501	0.1501	0	0	1	1	0	0	NA	NA	395.542	395.542
	SS	SK	1.314422	0.576975	0.496702	0.496702	0.8028	0.8028	0.93411	0.93411	0.849	0.849	0.889524	0.889524	0.814415	0.790764	1442.616	1442.616
	Skew	SK	0.545252	0.189985	0.493176	0.493176	0.8383	0.8383	0.986351	0.986351	0.848138	0.848138	0.912038	0.912038	69.34095	34.05004	10.2168	10.2168
	StDev	SK	1.13791	0.469978	0.501338	0.501338	0.7897	0.7897	0.913402	0.913402	0.850274	0.850274	0.880708	0.880708	20.04371	3.039728	92.62793	92.62793
IF	Average	SK	0.1884	0.2868	0.5891	0.5891	0.7152	0.7152	0.7179	0.7179	0.9921	0.9921	0.8330	0.8330	6.88913	1.416631	962.10	962.10
	Kurtosis	SK	0.1884	0.2868	0.6433	0.6433	0.7205	0.7205	0.7221	0.7221	0.9936	0.9936	0.8364	0.8364	67.18984	23.71106	775.20	775.20
	MAD	SK	0.1884	0.2868	0.4325	0.4325	0.7123	0.7123	0.7183	0.7183	0.9876	0.9876	0.8317	0.8317	72.07326	21.63906	634.68	634.68
	MM	SK	0.1978	0.3081	0.7155	0.7155	0.8399	0.8399	0.8426	0.8426	0.9948	0.9948	0.9124	0.9124	1.433161	0.481541	952.58	952.58
	NS	SK	0.1884	0.2868	0.6951	0.6951	0.8444	0.8220	0.8247	0.8247	0.9944	0.9944	0.9016	0.9016	133.64	133.64	NA	NA
	SS	SK	0.2119	0.3103	0.6951	0.6951	0.8444	0.8220	0.8247	0.8247	0.9944	0.9944	0.9016	0.9016	0.814415	0.790764	944.20	944.20
	Skew	SK	0.1884	0.2868	0.6415	0.6415	0.7036	0.7036	0.7049	0.7049	0.9937	0.9937	0.8247	0.8247	69.34095	34.05004	761.20	761.20
	StDev	SK	0.1884	0.2868	0.5519	0.5519	0.7034	0.7034	0.7066	0.7066	0.9910	0.9910	0.8250	0.8250	20.04371	3.039728	765.32	765.32

NA: Not Applied. MM: MinMax scaler. NS: No scaler applied. SS: Standard Scaler. MAD: Median Absolute Deviation. StDev: Standard Deviation. IF: Isolation Forest. AE: Autoencoder. TF: TensorFlow. SK: scikit-learn.

Table 7
Evaluation results of AnoML dataset.

Model details			Inference time (ms)			Accuracy			F1 Score			AUC			Recall			Precision			Scale time (s)		Model size (KB)		
Model	SR	API	Edge	Fog	Cloud	Edge	Fog	Cloud	Edge	Fog	Cloud	Edge	Fog	Cloud	Edge	Fog	Cloud	Edge	Fog	Cloud	Fog	Cloud	Edge	Fog	Cloud
	StDev	TF	174.24	1.95	32.22	0.258	0.251	0.251	0.000	0	0	0.4991	0.485	0.485	0.000	0	0	0.000	0	0	0.716	0.12	19.62	3.172	80
	Average	TF	172.05	1.13	32.63	0.769	0.871	0.871	0.862	0.91	0.91	0.5738	0.867	0.867	0.979	0.876	0.876	0.770	0.946	0.946	0.241	0.04	19.50	3.152	80
CNN	Skew	TF	NA	8E−15	32.20	NA	0.475	0.475	NA	0.57	0.57	NA	0.474	0.474	NA	0.476	0.476	NA	0.72	0.72	2.487	0.8	19.52	3.156	80
	Kurtosis	TF	NA	2E−07	32.04	NA	0.259	0.259	NA	0	0	NA	0.5	0.5	NA	0	0	NA	1	1	2.425	0.57	19.50	3.152	80
	MAD	TF	NA	1.81	32.16	NA	0.576	0.576	NA	0.60	0.60	NA	0.702	0.702	NA	0.441	0.441	NA	0.972	0.972	2.558	0.88	19.52	3.156	80

NA: Not Applied. MAD: Median Absolute Deviation. StDev: Standard Deviation. IF: Isolation Forest. TF: TensorFlow.

5.3. Anomaly detection methods

We tested the anomaly detection algorithms mentioned in Section 2 on WADI and AnoML datasets, as well as edge, fog, and cloud platforms by applying several scaling/reduction techniques. During the evaluations, we only used the baseline version of anomaly detection algorithms, hence we did not tune the algorithms to get better results to demonstrate the performance of baseline versions. We evaluate the following parameters as they are the core elements that determine the efficiency of an anomaly detection algorithm. *Accuracy* [76] determines the overall correct prediction ratio. *Precision* [77] defines how close the predictions are. *Recall* [77] demonstrates how the anomaly detection algorithm is successful at detecting normal data. *F1-Score* [76] is an evaluation metric that takes class distribution into considering. It might be preferred as the main metric when false detections matter (e.g., industrial environments). *True positive (TP).* The normal data is detected as normal. *True negative (TN).* The anomalous data is detected as anomalous. *False positive (FP).* The normal data is detected as anomalous. *False negative (FN).* The anomalous data is detected as normal. The following equations demonstrate how these parameters are calculated:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \qquad Precision = \frac{TP}{TP + FP}$$

$$F1 - Score = \frac{2 * TP}{2 * TP + FP + FN} \qquad Recall = \frac{TP}{TP + FN}$$

The TensorFlow has three different APIs. The main API (TensorFlow) contains all the available methods and utilizes high-level API Keras [78] to build neural networks. TensorFlow Lite [43] which is the lightweight version of TensorFlow that is specifically designed for mobile/IoT devices is generated via the main TensorFlow API. TensorFlow Lite for Microcontrollers [79] only allows a subset of TensorFlow operations to be run on 32-bit microcontrollers. Hence, currently only CNN is supported. TensorFlow Lite models are converted into micro models via TensorFlow Lite converter Python API [80]. Then, inference is possible at the edge.

Tables 6 and 7 demonstrates the results of anomaly detection tests. We see that baseline CNN at the edge achieves 77% accuracy and 86% F1-score. We did not observe any significant differences regarding accuracy and F1-Score. We also see those reduction methods help to reduce the inference time. If we compare scikit-learn methods and TensorFlow methods, we see that scikit-learn performs better when there is more computing power, but TensorFlow might generate better results on fog rather than the cloud. If we use CNN-AE, we see that the accuracy is higher when the standard deviation is applied, but for one-class SVM, we see that skew performs better than the standard deviation. Also, we see that the TF Lite models are significantly less in size than TF models. During the evaluation of the Isolation Forest, the whole dataset must be fit at once, due to nature algorithm. The Raspberry Pi 4B with 4 GB RAM fails due to high RAM usage that occurs because of fitting the whole dataset at once. Hence, we utilized the 64-bit 8 GB version of Raspberry Pi to evaluate the performance of Isolation Forest.

6. Lessons learned

IoT infrastructures are expanding thanks to the benefits of ubiquitous computing. The rapid developments in lightweight edge mechanisms made them desirable for labor-intensive tasks such as anomaly detection. The increasing variety of 32-bit microcontrollers allows us to choose a specific device with desired features (e.g., relatively high RAM) that can run ML algorithms. In the edge environment where these microcontrollers are utilized, while ML-based tasks such as keyword spotting, natural language processing, image processing have proven to be promising, anomaly detection is still in a very early phase [81]. Machine learning pipelines are the key elements that let us evaluate these edge ML algorithms. Thus, we develop an end-to-end ML pipeline that facilities developing anomaly detection systems where the detection can be done on different layers (edge, fog, and cloud). We make the following observations based on our findings: (i) lack of complete multi-protocol edge anomaly detection frameworks, (ii) the lack of control over converted ML models, (iii) there is a lack of sensor data terminology, (iv) manufacturers/developers provide platform-specific support, (vi) there is lack of support for multi-language/protocol development frameworks, (vii) automated is actually not automated.

Lack of complete multi-protocol edge anomaly detection frameworks. The two tasks that are executed within anomaly detection pipelines are: (i) data gathering/ingestion, (ii) anomaly detection model development. While these tasks are different by nature, they are essential to implement a complete IoT anomaly detection mechanism. During the evaluations, we see that the required tools are very diverse and no ML framework offers a complete solution. For data ingestion, flexibility and scalability are the most desired features. Hence, microcontrollers are the primary elements that are deployed in IoT environments to gather data. Due to the resource constraints of these environments, we need low power-consuming communication protocols. This is one of the main reasons that why BLE [58] is invented. In the edge, during the evaluations we see that there are three conditions needed for data acquiring: (i) sensors should be compatible with the microcontroller, (ii) communication protocol libraries should be available to establish a network both for the edge and the fog, and (iii) configurable platform that provides visual assistance is needed. As no framework ensures all three conditions, we had to use tools that require different programming languages (e.g., Arduino, JavaScript, Python).

The lack of control over converted ML models. TensorFlow allows us to convert the neural network models that are generated via the main API to more lightweight models that can be deployed to edge and fog. However it offers zero control over the model conversion. During the evaluations, we realized that some lite models perform better than their main peers which were unexpected as there is a known trade-off between model accuracy and power consumption. Due to not having any control over this conversion, the only way to compare these models is by running inference. Also, scikit-learn models do not have a lite version. Pickle [82] library is used to pack and deploy the same model to another platform. Hence deploying these models to microcontrollers is currently not

Table 8
Data format.

Categories	Sensor type					Protocols				Others		
Features	Temperature	Humidity	Air Quality	Light	Sound	WiFi	Bluetooth Classic	BLE	Zigbee	Location ID	Sensor ID	Microcontroller ID
Identifiers	TH	HU	AQ	LI	SO	WF	BC	BL	ZB	Numbers	Numbers	Numbers
Example Data	24.45	44.31	75	255	644	NA	NA	NA	NA	01	001	1

NA: Not Applicable.

possible. This is why during the evaluations we did not observe any differences rather than inference time when running scikit-learn models.

Lack of unified IoT sensor data terminology. In an IoT environment, the sensors generate data in a similar format. Showing such similarities might be confusing if the data are not identified with certain terms/identifiers especially where big data are present. Hence, we apply further processing to identify the data context. However, there is no sensor terminology that explicitly states the sensor data format. Thus, we designed ECCG in a way that it generates a code that provides all required information such as the location of the sensor, sensor type, and the microcontroller model. Also, another important point to consider regarding IoT sensor data terminology is the data with floating points. While WiFi, Zigbee, and Bluetooth Classic have no issues when sending floating-point data, BLE by nature designed to send buffer (byte array) as it provides power optimization. Hence, when sending data over BLE we might require further indicators such as "F" for floats or "I" for integers to correctly identify the data. Table 8 presents the data format that is generated via ECCG. We set the number IDs in different lengths to prevent possible confusion.

We evaluated our AnoML pipeline with two datasets, WADI and AnoML. WADI dataset has data from 127 different sensors, which is not possible to be replayed on edge devices due to limited computing power. WADI dataset was only evaluated on fog and cloud. We used a private machine with NVidia Tesla GPU and TensorFlow-GPU library version 2.4.1 for ML model training. AnoML dataset has data from five environmental sensors but two of them (temperature and humidity) were used to build and evaluate models on edge, fog, and cloud. We used Google Colab with GPU support to build models for the AnoML dataset with TensorFlow-GPU library version 2.1.1. AnoML dataset was transformed using reduction techniques only, to convert it into univariate.

In terms of accuracy, F1-score, precision, recall, and Area Under Curve (AUC), there was no significant difference recorded when comparing fog and cloud inferences in all types of models for the WADI Dataset. It was also observed that batch processing (all at once) in TensorFlow on the cloud does not affect the efficiency of the pipeline. Fig. 10 shows the comparison of each metric for all models for the WADI Dataset. Regarding fog and cloud, we observed a similar trend for the AnoML Dataset. However, we noticed that edge results were comparatively less efficient.

We noticed that inference on fog takes distinctly less time as compared to inference on the cloud. But, we observed the opposite results for CNN-AE when doing multivariate analysis where we recorded that cloud inference was faster than fog. Fig. 10(a) provides a visual comparison between inference times of fog and cloud. We also experienced that when using batch prediction in the cloud (all rows at once), the average prediction time was extremely faster. But, this is not applicable in the real-world scenario where data is being streamed. Both scaling and reduction techniques were faster on cloud (using batch-prediction) when compared to fog as seen in Fig. 10(g). Inference time trends were also identical in ML models for the AnoML dataset as fog inference time was less than cloud inference. The fog was more than 10 times faster than cloud inference. Inference time on edge was extremely slow when compared to fog and cloud, it was more than 100 times slower than the fog and almost 5 times slower than the cloud.

WADI Dataset is based on two sub-datasets recorded at different times, normal and attack, as we discussed that normal dataset was used for ML model training thus we used the attack dataset for testing. The attack dataset consists of 172 800 rows which were then converted into time-series data. As a result, the dimension of the data became (172 770, 30, 127) for scale-based and (172 770, 30, 1) for reduction-based models. Reduction-based models took exceptionally less time for training because of the univariate nature of the data. The size-on-disk of CNN-AE scaled-based models is too big as compared to CNN-AE reduction-based models. For RNN-AE models, the size-on-disk of reduction-based models are not significantly different from scale-based RNN models. The main reason for this was that we used LSTM layers for RNN. The size-on-disk variation trend was identical in fog models as seen in Fig. 10(h). We also learned that scikit-learn models maintain consistency over fog and cloud. The time-related results in the fog were always slower due to the difference in computing power, but there was no change observed in accuracy-related metrics. The obvious reason was that there was no platform/format conversion done for scikit-learn models.

Models for the AnoML dataset were lightweight but there was a notable difference for the fog models in cloud model size-on-disk. We also observed that the size of edge models was significantly greater than fog models, even though these models were converted from them. The reason was that edge models were in plain-text (hex-dump) as they were a C-array, but fog models were flat-buffered-binary format.

7. Conclusion

The proposed system offers support to a data scientist with minimal IoT knowledge to deploy a reconfigurable IoT anomaly detection infrastructure that utilizes a data science pipeline. The proposed framework contains: edge nodes consist of microcontrollers, fog nodes consist of single board computers and virtual cloud nodes. The communication between nodes is not limited to edge node types but limited with protocol specifications (e.g., Bluetooth only allows seven slaves/servers), hence allowing the

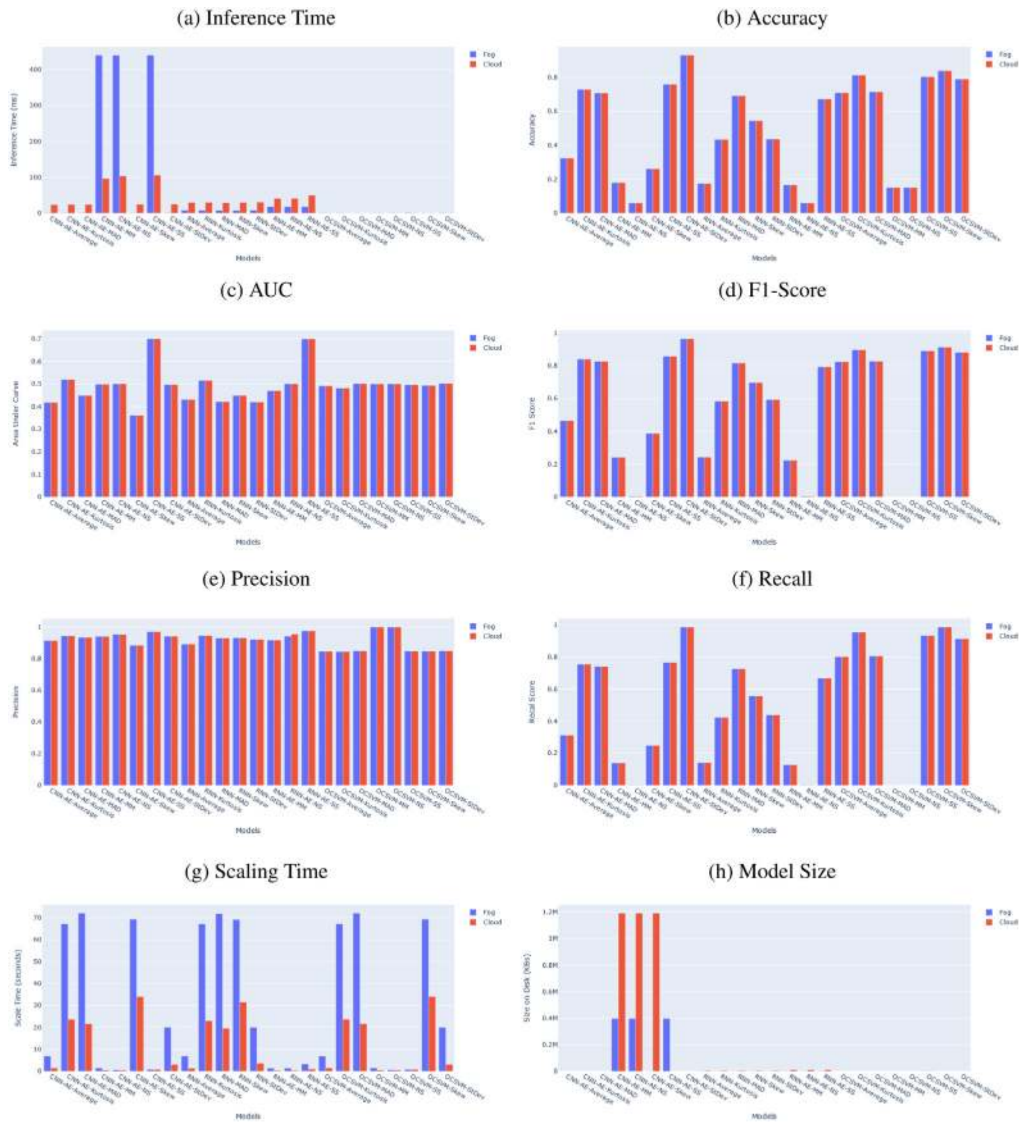


Fig. 10. WADI dataset evaluation.

implementation of different network topologies (e.g., bus, tree, and star). The system also explicitly supports the four major phases of data processing: gathering, training, deployment, and inference. We evaluated several combinations to measure the scalability that is offered by the proposed framework. We observed that the proposed anomaly detection pipeline reduces the required labor for building an IoT anomaly detection system. We identified the drawbacks of the proposed system including the reasons behind them. We believe our work encourages the future studies that aim to build open-source ML-based pipelines.

As future work, we are seeking to improve the ECCG web application by including more edge nodes, sensors, and protocol types. In addition, we envision converting required manual processes (e.g., deploying ML model to edge, uploading training data, deploying edge code) within the AnoML-IoT to automated.

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.

Acknowledgments

This work has been supported by GCHQ National Resilience Fellowship, Cardiff University EPSRC Capital Award support for Early Career Researchers, and the PETRAS National Centre of Excellence for IoT Systems Cybersecurity, which has been funded by the UK EPSRC under grant number EP/S035362/1.

References

[1] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, M. Hoffmann, Industry 4.0, Bus. Inf. Syst. Eng. 6 (4) (2014) 239–242.
[2] M. Mohammadi, A. Al-Fuqaha, S. Sorour, M. Guizani, Deep learning for IoT big data and streaming analytics: A survey, IEEE Commun. Surv. Tutor. 20 (4) (2018) 2923–2960, http://dx.doi.org/10.1109/COMST.2018.2844341, arXiv:1712.04301.
[3] Y. Jiang, C. Li, Convolutional neural networks for image-based high-throughput plant phenotyping: a review, Plant Phenomics 2020 (2020).
[4] J. Goh, S. Adepu, M. Tan, Z.S. Lee, Anomaly detection in cyber physical systems using recurrent neural networks, in: 2017 IEEE 18th International Symposium on High Assurance Systems Engineering, HASE, IEEE, 2017, pp. 140–145.
[5] F.T. Liu, K.M. Ting, Z.-H. Zhou, Isolation forest, in: 2008 Eighth IEEE International Conference on Data Mining, IEEE, 2008, pp. 413–422.
[6] J. Ma, S. Perkins, Time-series novelty detection using one-class support vector machines, in: Proceedings of the International Joint Conference on Neural Networks, 2003, vol. 3, IEEE, 2003, pp. 1741–1745.
[7] H. Kayan, AnoML-IoT, 2021, URL: https://kaggle.com/hkayan/anomliot.
[8] C.M. Ahmed, V.R. Palleti, A.P. Mathur, WADI: a water distribution testbed for research in the design of secure cyber physical systems, in; Proceedings of the 3rd International Workshop on Cyber-Physical Systems for Smart Water Networks, 2017, pp. 25–28.
[9] Y. Liu, Z. Pang, M. Karlsson, S. Gong, Anomaly detection based on machine learning in IoT-based vertical plant wall for indoor climate control, Build. Environ. 183 (2020) http://dx.doi.org/10.1016/j.buildenv.2020.107212.
[10] R. Chalapathy, S. Chawla, Deep learning for anomaly detection: A survey, (ISSN: 23318422) 2019, pp. 1–50, arXiv:1901.03407.
[11] A. Blázquez-García, A. Conde, U. Mori, J.A. Lozano, A review on outlier/anomaly detection in time series data, (ISSN: 23318422) 2020, arXiv:2002.04236.
[12] J. Song, H. Takakura, Y. Okabe, K. Nakao, Toward a more practical unsupervised anomaly detection system, Inform. Sci. 231 (2013) 4–14.
[13] Y. Zhu, N.M. Nayak, A.K. Roy-Chowdhury, Context-aware activity recognition and anomaly detection in video, IEEE J. Sel. Top. Sign. Proces. 7 (1) (2012) 91–101.
[14] L. Martí, N. Sanchez-Pi, J.M. Molina, A.C.B. Garcia, Anomaly detection based on sensor data in petroleum industry applications, Sensors 15 (2) (2015) 2774–2797.
[15] N. Jazdi, Cyber physical systems in the context of industry 4.0, in: 2014 IEEE International Conference on Automation, Quality and Testing, Robotics, IEEE, 2014, pp. 1–4.
[16] C.S. Raghavendra, K.M. Sivalingam, T. Znati, Wireless Sensor Networks, Springer, 2006.
[17] J. Pang, D. Liu, Y. Peng, X. Peng, Anomaly detection based on uncertainty fusion for univariate monitoring series, Measurement 95 (2017) 280–292.
[18] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, D. Pei, Robust anomaly detection for multivariate time series through stochastic recurrent neural network, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 2828–2837.
[19] M. Teng, Anomaly detection on time series, in: Proc. 2010 IEEE Int. Conf. Prog. Informatics Comput. PIC 2010, vol. 1, 2010, pp. 603–608, http://dx.doi.org/10.1109/PIC.2010.5687485.
[20] H.S. Wu, A survey of research on anomaly detection for time series, in: 2016 13th Int. Comput. Conf. Wavelet Act. Media Technol. Inf. Process, ICCWAMTIP 2017, no. 1, IEEE, 2017, pp. 426–431, http://dx.doi.org/10.1109/ICCWAMTIP.2016.8079887.
[21] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: A survey, ACM Comput. Surv. 41 (3) (2009) 1–58.
[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[23] M. Abadi, TensorFlow: learning functions at scale, in: Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, 2016, p. 1.
[24] M. Munir, S.A. Siddiqui, A. Dengel, S. Ahmed, DeepAnT: A deep learning approach for unsupervised anomaly detection in time series, IEEE Access 7 (2018) 1991–2005.
[25] T. Wen, R. Keyes, Time series anomaly detection using convolutional neural networks and transfer learning, 2019, arXiv preprint arXiv:1905.13628.
[26] D. Li, D. Chen, B. Jin, L. Shi, J. Goh, S.-K. Ng, MAD-GAN: Multivariate anomaly detection for time series data with generative adversarial networks, in: International Conference on Artificial Neural Networks, Springer, 2019, pp. 703–716.
[27] TensorFlow, Introducing tensorflow decision forests, 2021, URL: https://blog.tensorflow.org/.
[28] N. Ketkar, Introduction to pytorch, in: Deep Learning with Python, Springer, 2017, pp. 195–208.
[29] QuinnRadich, Automatic code generation with mlgen, 2021, URL: https://docs.microsoft.com/.
[30] MATLAB, Deep Learning Code Generation - MATLAB & Simulink - MathWorks United Kingdom, 2021, URL: https://uk.mathworks.com/help/deeplearning/deep-learning-code-generation.html.
[31] H. Ren, B. Xu, Y. Wang, C. Yi, C. Huang, X. Kou, T. Xing, M. Yang, J. Tong, Q. Zhang, Time-series anomaly detection service at microsoft, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 3009–3017.
[32] NilsPohlmann, Create and run ML pipelines - Azure Machine Learning, 2021, URL: https://docs.microsoft.com/en-us/azure/machine-learning/.
[33] Microsoft, Cognitive Services – APIs for AI Developers | Microsoft Azure, 2021, URL: https://azure.microsoft.com/en-gb/services/cognitive-services/.
[34] Microsoft, Anomaly Detector - Anomaly Detection System | Microsoft Azure, 2021, URL: https://azure.microsoft.com/en-us/services/cognitive-services/anomaly-detector/.
[35] H. Zhao, Y. Wang, J. Duan, C. Huang, D. Cao, Y. Tong, B. Xu, J. Bai, J. Tong, Q. Zhang, Multivariate time-series anomaly detection via graph attention network, 2020, arXiv preprint arXiv:2009.02040.
[36] AWS, Build your own Anomaly Detection ML Pipeline, 2021, p. 1.
[37] E. Liberty, Z. Karnin, B. Xiang, L. Rouesnel, B. Coskun, R. Nallapati, J. Delgado, A. Sadoughi, Y. Astashonok, P. Das, et al., Elastic machine learning algorithms in amazon sagemaker, in: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, 2020, pp. 731–737.
[38] S. Guha, N. Mishra, G. Roy, O. Schrijvers, Robust random cut forest based anomaly detection on streams, in: International Conference on Machine Learning, PMLR, 2016, pp. 2712–2721.
[39] M.D. Prado, J. Su, R. Saeed, L. Keller, N. Vallez, A. Anderson, D. Gregg, L. Benini, T. Llewellynn, N. Ouerhani, et al., Bonseyes AI pipeline—Bringing AI to you: End-to-end integration of data, algorithms, and deployment tools, ACM Trans. Internet Things 1 (4) (2020) 1–25.
[40] BAIR, Caffe | Deep Learning Framework, 2021, URL: http://caffe.berkeleyvision.org/.
[41] The FIWARE Foundation, The Open Source Platform for Our Smart Digital Future, FIWARE, URL: https://www.fiware.org/.

[42] L.L. Fernández, M.P. Díaz, R.B. Mejías, F.J. López, J.A. Santos, Kurento: a media server technology for convergent WWW/mobile real-time multimedia communications supporting WebRTC, in: 2013 IEEE 14th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" WoWMoM, IEEE, 2013, pp. 1–6.
[43] TensorFlow, TensorFlow Lite | ML for mobile and edge devices, 2021, URL: https://www.tensorflow.org/lite.
[44] I. Drori, Y. Krishnamurthy, R. Rampin, R. Lourenço, J. One, K. Cho, C. Silva, J. Freire, AlphaD3M: Machine learning pipeline synthesis, in: AutoML Workshop at ICML, 2018.
[45] C.B. Browne, E. Powley, D. Whitehouse, S.M. Lucas, P.I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton, A survey of Monte Carlo tree search methods, IEEE Trans. Comput. Intell. AI Games 4 (1) (2012) 1–43.
[46] L. Bottou, Large-scale machine learning with stochastic gradient descent, in: Proceedings of COMPSTAT'2010, Springer, 2010, pp. 177–186.
[47] J.R. Sutton, R. Mahajan, O. Akbilgic, R. Kamaleswaran, Physonline: an open source machine learning pipeline for real-time analysis of streaming physiological waveform, IEEE J. Biomed. Health Inf. 23 (1) (2018) 59–65.
[48] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, et al., Mllib: Machine learning in apache spark, J. Mach. Learn. Res. 17 (1) (2016) 1235–1241.
[49] M. Nitsche, S. Halbritter, Development of an end-to-end deep learning pipeline, Hochsch. Angew. Wiss. Hamburg (2019).
[50] S. Shaikh, H. Vishwakarma, S. Mehta, K.R. Varshney, K.N. Ramamurthy, D. Wei, An end-to-end machine learning pipeline that ensures fairness policies, 2017, arXiv preprint arXiv:1710.06876.
[51] S. Boovaraghavan, A. Maravi, P. Mallela, Y. Agarwal, MLIoT: An end-to-end machine learning system for the Internet-of-Things, in: Proceedings of the International Conference on Internet-of-Things Design and Implementation, 2021, pp. 169–181.
[52] M. Molinara, M. Ferdinandi, G. Cerro, L. Ferrigno, E. Massera, An end to end indoor air monitoring system based on machine learning and SENSIPLUS platform, IEEE Access 8 (2020) 72204–72215.
[53] B. Vinzamuri, E. Khabiri, A. Bhamidipaty, G. Mckim, B. Gandhi, An end-to-end context aware anomaly detection system, in: 2020 IEEE International Conference on Big Data (Big Data), IEEE, 2020, pp. 1689–1698.
[54] Y. Li, D. Zha, P. Venugopal, N. Zou, X. Hu, PyODDS: An end-to-end outlier detection system with automated machine learning, in: Companion Proceedings of the Web Conference 2020, 2000, pp. 153–157.
[55] M. Fezari, A. Al Dahoud, Integrated Development Environment "IDE" For Arduino, WSN Appl. (2018) 1–12.
[56] G.R. Hiertz, D. Denteneer, L. Stibor, Y. Zang, X.P. Costa, B. Walke, The IEEE 802.11 universe, IEEE Commun. Mag. 48 (1) (2010) 62–70.
[57] K.-H. Chang, Bluetooth: a viable solution for IoT?[Industry Perspectives], IEEE Wirel. Commun. 21 (6) (2014) 6–7.
[58] R. Heydon, N. Hunn, Bluetooth low energy, 2012, CSR Presentation, Bluetooth SIG https://www.bluetooth.org/docman/handlers/downloaddoc.ashx.
[59] S.C. Ergen, ZigBee/IEEE 802.15. 4 Summary, vol. 10, no. 17, UC Berkeley, September, 2004, p. 11.
[60] J. Song, S. Han, A. Mok, D. Chen, M. Lucas, M. Nixon, W. Pratt, WirelessHART: Applying wireless technology in real-time industrial process control, in: 2008 IEEE Real-Time and Embedded Technology and Applications Symposium, IEEE, 2008, pp. 377–386.
[61] P. Warden, D. Situnayake, Tinyml: Machine Learning with Tensorflow Lite on Arduino and Ultra-Low-Power Microcontrollers, O'Reilly Media, Inc., 2019.
[62] Node-RED, Node-RED, 2021, URL: https://nodered.org/.
[63] S. Tilkov, S. Vinoski, Node. js: Using JavaScript to build high-performance network programs, IEEE Internet Comput. 14 (6) (2010) 80–83.
[64] Arduino, Arduino Nano 33 BLE Sense | Arduino Official Store, 2021, URL: https://store.arduino.cc/arduino-nano-33-ble-sense.
[65] Nordic Semiconductor, nRF52840 - Nordic semiconductor, 2021, URL: http://nordicsemi.com/Products/nRF52840.
[66] IEEE, IEEE Standard for floating-point arithmetic, in: IEEE Std 754-2019 (Revision of IEEE 754-2008), 2019, pp. 1–84, http://dx.doi.org/10.1109/IEEESTD.2019.8766229.
[67] J. Yao, S. Warren, Applying the ISO/IEEE 11073 standards to wearable home health monitoring systems, J. Clin. Monitor. Comput. 19 (6) (2005) 427–436.
[68] Y.A. Badamasi, The working principle of an Arduino, in: 2014 11th International Conference on Electronics, Computer and Computation, ICECCO, IEEE, 2014, pp. 1–4.
[69] A. Due, A. Core, Arduino due, Retrieved 9 (16) (2017) 2019.
[70] Grove, Sensors - seeed studio electronics, 2021, URL: https://www.seeedstudio.com/category.
[71] D. Quick, K.-K.R. Choo, Google Drive: Forensic analysis of data remnants, J. Netw. Comput. Appl. 40 (2014) 179–193.
[72] T. Carneiro, R.V.M. Da Nóbrega, T. Nepomuceno, G.-B. Bian, V.H.C. De Albuquerque, P.P. Reboucas Filho, Performance analysis of google colaboratory as a tool for accelerating deep learning applications, IEEE Access 6 (2018) 61677–61685.
[73] J. Giraldo, D. Urbina, A. Cardenas, J. Valente, M. Faisal, J. Ruths, N.O. Tippenhauer, H. Sandberg, R. Candell, A survey of physics-based attack detection in cyber-physical systems, ACM Comput. Surv. 51 (4) (2018) 1–36.
[74] S.F. Barrett, D.J. Pack, Atmel avr microcontroller primer: Programming and interfacing, Synth. Lect. Digit. Circuits Syst. 7 (2) (2012) 1–244.
[75] Y. Bai, Practical Microcontroller Engineering with ARM Technology, John Wiley & Sons, 2015.
[76] M. Hasan, M.M. Islam, M.I.I. Zarif, M. Hashem, Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches, Internet of Things 7 (2019) 100059.
[77] J. Davis, M. Goadrich, The relationship between Precision-Recall and ROC curves, in: Proceedings of the 23rd International Conference on Machine Learning, 2006, pp. 233–240.
[78] A. Gulli, S. Pal, Deep Learning with Keras, Packt Publishing Ltd, 2017.
[79] Google, TensorFlow Lite Micro, 2021, URL: https://www.tensorflow.org/lite/microcontrollers.
[80] TensorFlow, Tensorflow lite converter, 2021, URL: https://www.tensorflow.org/lite/convert.
[81] C.R. Banbury, V.J. Reddi, M. Lam, W. Fu, A. Fazel, J. Holleman, X. Huang, R. Hurtado, D. Kanter, A. Lokhmotov, et al., Benchmarking TinyML systems: Challenges and direction, 2020, arXiv preprint arXiv:2003.04821.
[82] Python, pickle — Python object serialization — Python 3.9.6 documentation, pickle, URL: https://docs.python.org/3/library/pickle.html.