UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Samuel Amico Fidelis

A Distributed Architecture for Edge AI Computing

Florianópolis
2023

Samuel Amico Fidelis

A Distributed Architecture for Edge AI Computing

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação para a obtenção do título de Mestre em Ciência da Computação.
Supervisor: Prof. Márcio Bastos Castro, Dr.
Co-supervisor: Prof. Frank Augusto Siqueira, Dr.

Florianópolis
2023

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Fidelis, Samuel Amico
A Distributed Architecture for Edge AI Computing / Samuel Amico Fidelis ; orientador, Márcio Bastos Castro, coorientador, Frank Augusto Siqueira, 2023.
106 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Ciência da Computação, Florianópolis, 2023.

Inclui referências.

1. Ciência da Computação. 2. Computação em borda. 3. Aprendizagem Distribuida. 4. Edge AI. I. Castro, Márcio Bastos . II. Siqueira, Frank Augusto. III. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. IV. Título.

Samuel Amico Fidelis

A Distributed Architecture for Edge AI Computing

O presente trabalho em nível de mestrado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Prof. Ivanovitch Medeiros Dantas da Silva, Dr.
Universidade Federal do Rio Grande do Norte

Prof. Mario Antônio Ribeiro Dantas, Dr.
Universidade Federal de Juiz de Fora

Prof. Douglas Dyllon Jerônimo de Macedo, Dr.
Universidade Federal de Santa Catarina

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de Mestre em Ciência da Computação.

Documento assinado digitalmente
PATRICIA DELLA MEA PLENTZ
Data: 04/09/2023 15:53:55-0300
CPF: ***.243.700-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Patricia Della Méa Plentz, Dr.
Coordenadora do Programa

Documento assinado digitalmente
Marcio Bastos Castro
Data: 04/09/2023 09:51:24-0300
CPF: ***.707.720-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Márcio Bastos Castro, Dr.
Orientador

Florianópolis, 2023.

This work is dedicated to my mother, my father and my wife.

ACKNOWLEDGEMENTS

I first thank God for giving me the opportunity to study, my parents for all the help and support they gave me to live abroad, my wife who is always by my side, my advisors Márcio Castro and Frank Siqueira who always helped and encouraged me in this master program.

People are often ask me what's going to happen next in science that is important. Of course, the whole point is that if it's important, it's something we didn't expect. In order for science to go on, it has to have some mysteries. (DYSON, A., 2012)

RESUMO

Internet das Coisas ou *Internet of Things* (IoT) é uma rede massiva de dispositivos físicos incorporados com sensores, *software*, peças eletrônicas e rede que permite que os dispositivos IoT troquem ou coletem dados e executem certas ações sem intervenção humana. A maioria dos dispositivos *IoT* produzem quantidades massivas de dados em um curto intervalo de tempo e tais dados podem ser transformados em informações úteis para aplicações de negocio através do uso de modelos de Aprendizagem de Máquina, análise estatística e técnicas de mineração de dados. Transferir grandes quantidades de dados em um curto intervalo de tempo para a nuvem pode ser inaceitável para algumas aplicações que exigem processamento quase em tempo real. Uma solução para atender a esses requisitos é aproximar a maior parte do processamento de dados aos dispositivos IoT (ou seja, na borda). Nesse contexto, o presente trabalho propõe mover todo processamento e armazenamento de dados e a aplicação de algoritmos de Aprendizagem de Máquina para a camada em borda mantendo a camada em nuvem apenas para as tarefas relacionadas ao armazenamento de longo prazo de dados resumidos e hospedagem de relatórios e *dashboards*. A solução desenvolvida neste trabalho foi o desenvolvimento de uma nova arquitetura distribuída para computação *Edge AI* onde os nós são distribuidos pelas subcamadas da borda (*mist* e *fog*) além de um novo algoritmo para Aprendizagem de Máquina que utiliza o algoritmo de consenso para distribuir um conjunto de diferentes modelos leves (*lightweight models*) não supervisionados com o intuito de melhorar a precisão geral do sistema e manter o processamento em baixa latência. Resultados obtidos com três conjuntos de dados diferentes mostram que a abordagem proposta pode melhorar a precisão dos modelos de aprendizagem de máquinas sem comprometer significativamente o tempo de latência de resposta.

Palavras-chave: Computação em borda. Computação em névoa. Computação em neblina. *Edge AI*. Aprendizagem Distribuida

RESUMO EXPANDIDO

Introdução

Os dispositivos de IoT estão cada vez mais presentes em diversos cenários, seja *smart home*, indústria, carros autônomos, entre outros. Os dispositivos IoT são capazes de produzir grandes quantidades de dados, contribuindo para os fenômenos chamados de *Big Data*, mas possuem recursos limitados de armazenamento e processamento. A estratégia de enviar dados de IoT para serem processados e armazenados na nuvem apresenta vários novos desafios que não podem ser totalmente resolvidos apenas pela infraestrutura de nuvem. Neste contexto, surge um novo paradigma computacional, a computação de borda (*edge computing*) para mitigar os requisitos supracitados, uma vez que os dados podem ser processados nas proximidades dos dispositivos IoT ao invés de enviados para serviços remotos na nuvem. Tal solução pode fornecer serviços com resposta mais rápida e maior proteção de privacidade do que a computação em nuvem. Nesse contexto, os dados produzidos pelos dispositivos IoT são vistos como um ativo valioso para o sucesso dos negócios. Técnicas de mineração de dados são usadas para extrair informações significativas dos dados brutos e gerar insights. Algumas técnicas como *Machine Learning* (ML) e *Deep Learning* (DL) são utilizadas em diversas aplicações, como: recomendação de produtos, detecção de objetos, detecção de fraudes, previsão de eventos catastróficos, etc. No entanto, para as aplicações IoT, o modelo Cloud não será uma boa solução pelas razões já mencionadas. Uma das novas tendências da computação de borda é levar a tecnologia de Inteligência Artificial (IA) para as bordas da rede, a chamada *Edge AI*, e, graças às melhorias no hardware dos equipamentos de borda, o processamento de grandes quantidades de dados torna-se mais viável.

Objetivos

O paradigma de *Edge AI* carece de uma arquitetura de base comum e de uma definição de quais serviços de IA podem ser trazidos para a borda e quais podem estar na nuvem. Neste trabalho propomos uma arquitetura de base comum de *Edge AI* para ser usada em aplicações críticas de IoT, como IoT industrial, que precisam de serviços de IA. Através do uso de camadas específicas de computação distribuída, um novo algoritmo de aprendizado distribuído e o uso de hardware heterogêneo – ou seja: máquinas com desempenho e estrutura diferenciados para atender diferentes necessidades computacionais – são fornecidos os recursos computacionais para processamento e armazenamento de dados, treinamento de modelos de IA, deslocando os algoritmos de inferência para os dispositivos finais e evitando o envio de grandes quantidades de dados para a nuvem.

Metodologia

Para validar a arquitetura proposta com o algoritmo de consenso distribuído, projetamos um conjunto de experimentos para avaliar seu desempenho, levando em conta os principais requisitos impostos por aplicações IoT de baixa latência. Os experimentos projetados visam identificar: (i) Quanto tempo a arquitetura economizou executando na camada de borda; (ii) Qual é a melhoria do algoritmo de consenso em comparação com a execução de modelos leves de aprendizado de máquina de uma única maneira; (iii) Quão flexível é a arquitetura para diferentes cenários com múltiplas variáveis sendo monitoradas; (iv) Qual é o tempo de processamento executando a arquitetura proposta em um cenário real distribuído usando hardware embarcado. Para realizar os experimentos nós considera-

mos utilizar um cenário industrial e uma tarefa muito comum e importante para muitas aplicações IoT industriais que exigem o uso de modelos de aprendizado de máquina com resposta de baixa latência e alta precisão, a manutenção preditiva.

Resultados e Discussão

Em todos os experimentos, nós focamos em responder perguntas metodologicas propostas que visam validar a arquitetura proposta para aplicações IoT. Através de um experimento que utilizou a camada em borda e nuvem nós mostramos que a camada de nuvem tem um tempo de resposta maior do que a camada de borda, seja em Fog ou Mist, executando o mesmo modelo de aprendizado de máquina. O tempo de envio e recebimento de dados pela internet dos dispositivos IoT para o servidor em nuvem é alto o suficiente para aplicações de baixa latência. O segundo experimento usando um cenário de sensor único com a arquitetura proposta atuando na camada em borda mostrou que os modelos leves de ML usados de forma independente e separada acabaram apresentando uma grande ineficiência em termos de precisão de resposta, ou seja, não são adequados para aplicações IoT devido à sua alta taxa de erro. Por outro lado, quando aplicados na arquitetura juntamente com um algoritmo de consenso, a precisão final do sistema melhorou o suficiente para ser adequado ao cenário testado. Além da alta precisão do nosso algoritmo de distribuição, o tempo de processamento da camada de borda com hardware embarcado provou ser suficientemente baixo e adequado para muitas aplicações IoT de baixa latência. O último experimento teve como objetivo demonstrar a viabilidade da arquitetura usando o algoritmo de consenso em um cenário onde o número de sensores/recursos é alto. O uso do algoritmo de consenso com o algoritmo de seleção de recursos RFE provou ser altamente eficiente, pois um pequeno subconjunto de combinações de diferentes recursos e modelos foi encontrado, mostrando um aumento na precisão e recuperação do sistema em comparação com os resultados do modelo único. O experimento concluiu que mesmo diante de um cenário com múltiplos sensores, o algoritmo de consenso aplicado com técnicas de redução dimensional pode aumentar a precisão final do sistema. A última parte deste experimento mostrou que o tempo de processamento do algoritmo de consenso para o pequeno subconjunto de combinações de recursos e modelos foi adequado para as aplicações IoT de baixa latência.

Considerações Finais

A maioria das aplicações de IoT gera uma vasta quantidade de dados em pouco tempo, e muitas utilizam de algoritmos de IA para transformar dados brutos em informações relevantes para a aplicação em si, sendo esse processamento aplicado na computação em nuvem. Entrentanto, o uso de servidores na nuvem não é adequado para aplicações que precisam de uma resposta do modelo de ML em baixa latência. Diante disso, a computação em borda pode ser utilizado pois os dados são enviados para hardware próximos aos dispositivos IoT para que sejam processados e analisados. No entanto, esses nós não podem processar modelos pesados e treiná-los devido ao tamanho e complexidade dos mesmos. Para isso, é proposta neste trabalho uma arquitetura capaz de processar os dados em baixa latência, usando modelos de ML leves e um algoritmo de aprendizagem distríbuida que consegue aumentar a acurácia do sistema processar os dados com baixa latência. Implementamos nossa arquitetura com o algoritmo de aprendizado distribuído em diferentes cenários e avaliamos seu desempenho, demonstrando que ele pode funcionar na camada de borda em conformidade com os requisitos impostos pela maioria das aplicações IoT, além de demonstrar que através do uso do algoritmo de consenso é possível utilizar modelos leves de aprendizado de máquina, obtendo excelentes resultados

comparáveis a modelos mais robustos que rodam na nuvem. Contudo, trabalhos futuros devem investigar aspectos mais detalhados, tais como: (i) adaptar a arquitetura para lidar com dispositivos IoT maliciosos e bizantinos; (ii) usar o algoritmo de consenso em GPU para acelerar o processamento; (iii) aplicar a arquitetura em cenários novos, como controladores inteligentes, carros autônomos e ambientes inteligentes.

Palavras-chave: Computação em borda. Computação em névoa. Computação em neblina. *Edge AI*. Aprendizagem Distribuida

ABSTRACT

Internet of Things (IoT) is a massive network of physical devices embedded with sensors, software, electronics, and network which allows IoT devices to exchange or collect data and perform certain actions without human intervention. Most IoT devices produce massive amounts of data in a very short time and these data can be transformed into useful information for the application business through the use of Machine Learning (ML) models, statistical analysis and data mining techniques. Transferring large amounts of data in a very short time to the cloud may be unacceptable for some applications that require near-real time processing. One solution to meet such requirements is to bring most data processing closer to IoT devices (i.e., to the edge). In this context, the present work proposes to move all processing and data storage and the application of Machine Learning algorithms to the edge layer, keeping the cloud layer only for tasks related to long-term storage of summarized data and hosting reports and dashboards. The solution developed in this work are: the formulation of a new distributed architecture for Edge AI where the nodes are distributed into two sub-layers of the edge (mist and fog) and a new distributed algorithm for Machine Learning models that uses the consensus algorithm to distribute a set of different unsupervised lightweight models in order to improve the overall accuracy of the system and keep the inference process at low latency. Results obtained with three different datasets show that the proposed approach can improve the accuracy of machine learning models without significantly compromising the response latency time.

Keywords: Edge Computing. Fog Computing. Mist Computing. Edge AI. Distributed Machine Learning.

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

CONTENTS

1 INTRODUCTION

Since the beginning of the 21st century, many companies have changed the way they provide services to their customers, adopting the concept of distributed resources as a commodity or a public service (COULOURIS; DOLLIMORE; KINDBERG,). The concept of computing as a public service was defined much earlier, in 1997, as *Cloud Computing* by professor Ramnath Chellappa (CHELLAPPA, 1997). The essential characteristics of Cloud Computing are (MELL; GRANCE et al., 2011): on-demand self-service; broad network access; resource pooling; rapid elasticity; measured service. Cloud Computing has been the predominant paradigm used by several small, medium, and large businesses in the world, where the cloud is responsible for centralizing the processing, control, and storage of applications (CHIANG; ZHANG, 2016).

Companies such as Google and Amazon provide a multitude of services to their customers, adopting service models such as *Platform as a Service (PaaS)*, *Infrastructure as a Service (IaaS)* and *Software as a Service (SaaS)*. This strategy allows customers to pay only for the required services without having to maintain or build all the necessary infrastructure. However, a new paradigm called *Internet of Things (IoT)* emerged in the last years and has pointed out some limitations of a centralized infrastructure such as the one provided by cloud vendors (DONNO; TANGE; DRAGONI, 2019).

The IoT can be defined as the connection of physical objects/devices with network connectivity that enables them to collect and exchange data (GOKHALE; BHAT; BHAT, 2018). The *thing* refers to an object of the physical world, and the *device* refers to a piece of equipment capable of communicating through the Internet and optionally processing and storing data (SUNYAEV, 2020). According to Cisco, 29.3 billion devices are expected to be connected to the Internet by 2023 (CISCO, 2020). Furthermore, IoT applications have been present in many fields, such as healthcare, logistics, and industry.

The IoT devices are capable of producing large amounts of data, contributing to the phenomena called Big Data, but they have limited storage and processing resources. The strategy of sending IoT data to be processed and stored in the cloud introduces several new challenges that cannot be fully addressed by the cloud infrastructure alone. These challenges are (DONNO; TANGE; DRAGONI, 2019): low latency response, which is needed by several classes of IoT applications; high network bandwidth, which is required due to the large amount of data generated by the *things*; intermittent connectivity on some IoT devices; data security and privacy issues. These requirements cannot be safely addressed through a centralized cloud computing infrastructure (WANG et al., 2020).

In this context, a new computing paradigm called Edge Computing emerges (DONNO; TANGE; DRAGONI, 2019). According to Shi et al. (2016), "Edge Computing refers to the enabling technologies allowing computation to be performed at the edge of the network". Edge Computing comes to mitigate the aforementioned challenges since the data can be processed nearby the IoT devices rather than shipped to remote

cloud services. It can provide services with faster response and more security and privacy protection than cloud computing (JAIN; MOHAPATRA, 2019). For some authors, the term "edge" refers to any computing resource between the end devices and the cloud, and there are some techniques that share the same principles of Edge Computing but with subtle differences, such as Fog Computing (KLAS, 2015; BYERS, 2017), Mobile Edge Computing (MEC) (KLAS, 2015), Cloudlets (SATYANARAYANAN et al., 2009) and Mist Computing (HENSH; GUPTA; NENE, 2021).

According to Vaquero & Rodero-Merino (2014), "Fog Computing is a scenario where a huge number of decentralized devices communicate and cooperate among them to perform storage and processing tasks". For most IoT applications, cooperation with a Fog Computing platform will be the most suitable approach to mitigate the limitations of IoT devices since it provides distributed computation, storage, control, and networking capabilities closer to the devices and users (GOKHALE; BHAT; BHAT, 2018).

Nowadays, data has become a valuable asset for business success. Data Mining techniques are used to extract meaningful information from the raw data and to generate insights. Some techniques such as Machine Learning (ML) and Deep Learning (DL) (SCHMIDHUBER, 2015) are used in many applications, such as product recommendation, object detection, fraud detection, prediction of catastrophic events, etc. However, these techniques come with very high computing requirements. Creating a good ML or DL algorithm requires a lot of data and powerful computers equipped with Graphics Processing Units (GPUs) or Tensor Processing Units (TPUs) to process them, which are only available in the cloud. Fortunately, however, one of the new trends of Edge Computing is to push Artificial Intelligence (AI) technology to the edge, a paradigm called Edge AI. Thanks to the improvements in embedded hardware, processing such large amounts of data becomes more feasible. There are many application scenarios where AI can be applied on the edge, such as in the context of Smart Cites and the Internet of Vehicles (IoV).

Overall, Edge AI aims to solve three major problems related to the use of AI in the cloud for IoT applications:

- **Cost**: AI models in the cloud require devices to transmit a lot of data through the network, demanding a large network bandwidth (WANG et al., 2020).
- **Latency**: Many time-critical IoT applications require a low latency response coming from the AI models, which cannot be guaranteed by the Cloud Computing model due to the distance between the servers and the devices (WANG et al., 2020).
- **Reliability**: Most IoT devices rely on wireless communication, but for many industrial applications, IoT services must be highly reliable even when the network connection is lost (WANG et al., 2020).

In summary, Edge AI tries to push AI computation closer to the devices/things as much as possible, where the classical Cloud Computing model is used when additional

processing is required (KANG et al., 2017).

1.1 TARGET PROBLEM AND PROPOSED APPROACH

We observed that Edge AI lacks a common base architecture and a definition of which AI services could be brought to the edge and which could be on the cloud (FIROUZI; FARAHANI; MARINŠEK, 2021). Thus, in this work, we propose a common base architecture of Edge AI to be used in critical IoT applications by using the advantages of specific distributed computing layers, a new distributed learning algorithm, and the use of heterogeneous hardware in order to meet different computing needs. Our goal is to allow the processing of ML tasks, such as training and inference, by leveraging the hardware used on the edge layer to store raw data closer to the end devices without having to send large amounts of data to the cloud.

1.2 GOALS AND CONTRIBUTIONS

The goal of this work is to design and present an architecture to be used in Edge AI applications that enable the usage of ML services on critical IoT applications that need real-time responses. The proposed architecture is divided into three layers:

(a) **Mist Computing Layer**: this layer is responsible for preprocessing the data and processing the inference of ML models since the main property of this type of computing is to operate directly with the IoT devices.
(b) **Fog Computing Layer**: the purpose of this computing layer is to process and store data in addition to training ML models. Due to the large volume and velocity with which IoT data are produced, it is necessary to avoid sending data over the Internet to a centralized cloud architecture due to network bandwidth limitations. The Fog Computing layer is located between the mist and the cloud layers.
(c) **Cloud Computing Layer**: this layer is responsible for centralizing the summarized data received from IoT devices, which will be presented in reports and dashboards.

To ensure that nodes in the edge layer are able to process ML models with high accuracy and low response latency, we propose a new distributed learning approach using consensus on Edge AI. To accomplish the proposed architecture, clusters are built in the fog and mist layers using embedded hardware (Raspberry PIs (PI, 2015) and NVIDIA Jetson Nano[1]). The proposed architecture was tested in various scenarios using real-world data from IoT industrial applications. In summary, this work delivers the following main contributions:

[1] https://developer.nvidia.com/embedded/jetson-nano-developer-kit

1. An Edge AI base architecture that allows critical IoT applications to run on fog and mist layers with low latency response, network bandwidth savings, and increased data security;
2. A new distributed learning algorithm that ensures a suitable accuracy using lightweight ML models with low latency processing time;
3. The implementation of the proposed learning algorithm on dedicated hardware in the Fog Computing Layer;
4. Implementation of a streaming database to fulfill the requirements imposed by IoT Big Data; and
5. Physical implementation of the architecture and demonstration of its impact using data from industrial IoT applications.

1.3 ORGANIZATION OF THE DISSERTATION

The remainder of this work is organized as follows. In Chapter 2, we cover the background of Edge Computing and ML models, including distributed learning algorithms. We discuss related work and the methodology used for the literature review in Chapter 3. We detail our proposed architecture and the consensus algorithm in Chapter 4. Then, we present our evaluation methodology in Chapter 5, which is then applied in Chapter 6 to evaluate the experimental results. Finally, we draw our conclusions in Chapter 7.

2 BACKGROUND

In this chapter, a brief introduction is given about IoT, emphasizing the data-related issues in IoT applications. Then, an overview of cloud, fog, and mist concepts is presented. Finally, a brief introduction to Machine Learning and Distributed Learning is presented.

2.1 INTERNET OF THINGS (IOT) AND BIG DATA

In a nutshell, IoT is the connection of multiple physical objects to the Internet. Sensors attached to these connected objects generate vast amounts of data, resulting in what is called a Big Data ecosystem. IoT data comes from many applications such as healthcare, industry, manufacturing, and others (KHARE; TOTARO, 2019).

Big Data can be defined according to Sawalha & Al-Naymat (2021) as "an application that has high volume, velocity and variety (the 3 V's of Big Data) of information assets that need processing to serve decision making". IoT applications are inserted in this context of the 3 V's of Big Data (KAUR; SOOD, 2017). However, IoT Big Data has different processing requirements than other common Big Data applications, such as social networks. IoT devices generate different data types with noise and redundant information. Moreover, most IoT applications impact everyday things and people rather than business decision-making. Therefore, they need actuation commands, notifications, and other approaches to arrive at clients in real-time with extremely low latency.

Due to these requirements, the 3 V's of IoT Big Data can be defined as follows:

- **Volume**: IoT devices generate large amounts of streaming data that need to be efficiently stored and managed. Relational databases are usually not suitable for this task (KHARE; TOTARO, 2019).
- **Velocity**: Data from sensors come at an extremely high rate, and the challenge is to collect, handle and generate notifications/commands with low latency. In some cases, sending this data to the cloud is not feasible (KHARE; TOTARO, 2019).
- **Variety**: IoT applications involve data from different kinds of sensors, such as audio, video, time series, etc. In many applications, it is necessary to keep the metadata and perform processing that uses different types of data together to generate a meaningful result.

Bansal, Chana & Clarke (2020) went even further in the concept of 3 V's and defined 13 V's for Big Data in IoT, which are illustrated in Figure 1. Besides that, there is a new (and unique) challenge in IoT applications: the data transmission problem. Multiple devices produce a lot of data and communicate with each other using network protocols. Because of this huge amount of data generated with different formats and with

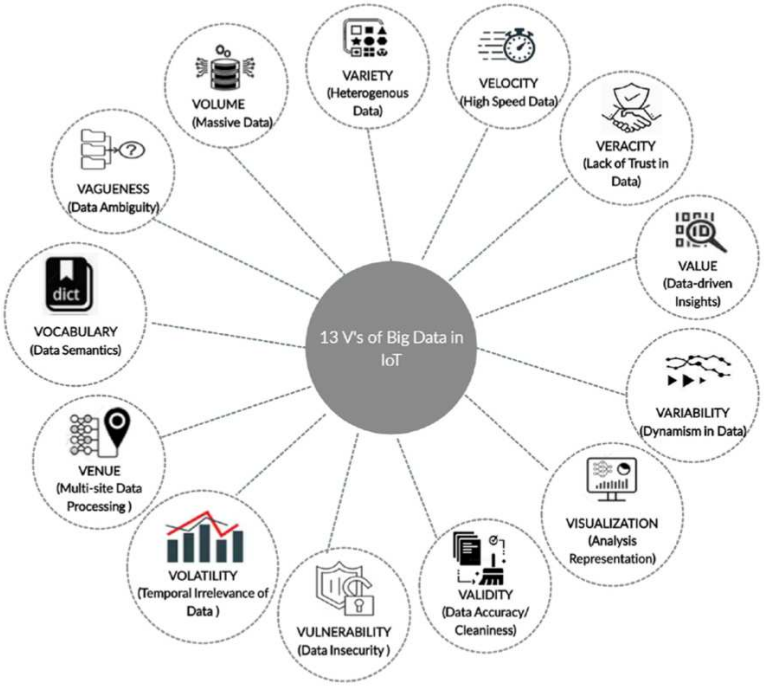

Figure 1 – 13 V's Big Data in IoT (BANSAL; CHANA; CLARKE, 2020).

high velocity, network bandwidth is saturated, and many devices compete for bandwidth to send data over the network. In addition, many hops are needed to send data to the cloud, which ends up exposing the data at multiple spots that represent the potential attack surface for private data (BANSAL; CHANA; CLARKE, 2020; TAHERKORDI; ELIASSEN; HORN, 2017).

2.2 MIST COMPUTING

Mist Computing is another subset of Edge Computing focused on running middleware and services together with the end devices. The term "mist" refers to the cloud on the ground, i.e., a layer connected directly to the end devices (SHAHID et al., 2021). Mist Computing allows different types of processing to be carried out, such as anomaly detection, data filtering, data aggregation, and ML inference (SHAHID et al., 2021).

The main goal of Mist Computing is to reduce the amount of data transmitted to the network by carrying out some processing locally. According to Satoh (2013), its main purpose is "to deal with time-sensitive issues of real-time systems using the mist nodes with a direct communication link with the IoT devices to avoid network bandwidth issues". In addition, it can enable more autonomous and fault-tolerant applications.

The main features of Mist Computing are:

- Distributed layer with lightweight nodes that require less computational power, storage, and memory (SATOH, 2013);
- Mist nodes provide better support for decision making in real-time (SATOH, 2013);

- Mist nodes could apply ML inference algorithms (the so-called TinyML) (SATOH, 2013);
- The nodes reduce the data volumetry before sending it through the network (SATOH, 2013).

Some authors addressed the combination of multiple edge layers. For example, Asif-Ur-Rahman et al. (2018) demonstrate a use case of Mist Computing with Fog and Cloud Computing together in a framework for an application of Internet of Healthcare Things (IoHT). Other works focused only on adopting one layer. Shahid et al. (2021) proposed the use of Mist Computing on an IoT military application where the mist nodes are responsible for preprocessing and applying ML inference on the data generated by the mobile devices. They concluded that the mist-based IoT system experiences less processing time, service time, latency, and task failures compared with other computing models (fog and cloud).

2.3 FOG COMPUTING

Fog Computing can be considered as a subset of Edge Computing (QIU et al., 2020), focused on the infrastructure. It orchestrates, manages, and secures resources and services across networks between edge devices and cloud data centers (JAIN; MOHAPATRA, 2019). The term "fog" is used to describe that the cloud is close to the ground (BONOMI et al., 2012), indicating the closeness to the end devices. According to Bonomi et al. (2012), Fog Computing is a "virtualized platform that provides computing power, storage, and network services". Fog nodes are located between the end devices and the cloud but are closer to the cloud data centers and are used to conduct analytics, recognize deviations, failure prediction, perform ML inference, and/or, in some cases, training models (MATT, 2018). Fog Computing is suitable for IoT applications where the data requires fast analysis and response time and the application is geographically distributed. It bridges the gap between the cloud and elements such as end-user devices in a way that consolidates important operational functions that are currently often processed in an isolated system (MATT, 2018).

Some of the main challenges of Fog Computing defined by Matt (2018) are the following: 1) technological heterogeneity makes the integration of fog nodes challenging; 2) Fog nodes are implemented in geographically widespread areas, and it is often expensive to provide support to all individual nodes; and 3) a high number of fog nodes and associated service providers can imply new forms of security and privacy risks.

Figure 2 describes the Fog Computing conceptual model according to the National Institute of Standards and Technology (NIST).

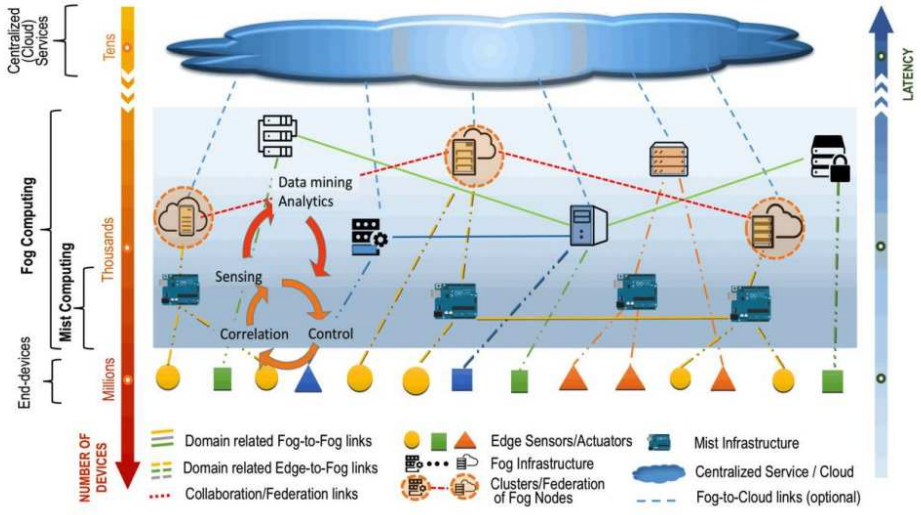

Figure 2 – Fog and Mist conceptual model - NIST SP 500-325.

2.4 EDGE COMPUTING

Cloud Computing is very useful for Big Data applications. Its advantages are very suitable for processing large amounts of data and performing analytical processing. Such advantages are: elasticity, low cost compared to on-premise infrastructures, fault tolerance, and easy-to-use infrastructure (SHI et al., 2016). Cloud applications are often user-driven, which is very suitable for large-scale data analytic functions (YU et al., 2017). However, using the cloud as a centralized server increases the frequency of communications between the client and the application (YU et al., 2017; KHAN et al., 2019).

Along with the emergence of Cloud Computing, a new Big Data ecosystem is showing a rapidly growing importance. IoT Big Data applications arrived in the post-cloud era (SHI et al., 2016) and have shown different requirements. In these applications, the data is generated in a high volume by the end devices (like sensors), and they often need a real-time response, which brings a concern for continuing usage of cloud services since the network bandwidth and speed will not be enough to support the transmission of the data from devices to a cloud server.

Therefore, Cloud Computing is not efficient enough to support current IoT Big Data applications because: 1) large network latency will be unacceptable and cannot satisfy the Quality of Service (QoS) requirements; 2) IoT devices produce massive data that need to be stored and, due to the high velocity and volume of data generated by them, sending all data to centralized storage will be unfeasible; and 3) IoT devices have limited computation power, so they act as simple producers and need to transmit data to a more powerful computing node/server to perform more complex analytical tasks. For these reasons, researchers have proposed a new solution, which moves some of the processing power of the cloud to the edge of the network. This approach is called Edge

Computing (KHAN et al., 2019; QIU et al., 2020; CAO; ZHANG; SHI, 2018).

Edge Computing is a distributed system that contains multiple edge nodes, which will provide reliability, security, and scalability to the system (KHAN et al., 2019). It brings services and processing power closer to the end devices to avoid sending data to a centralized cloud server and is characterized by fast processing and response time (KHAN et al., 2019; LI et al., 2018). However, the cloud is still needed, as shown in Figure **??** proposed by Khan et al. (2019).

According to Yu et al. (2017), the main advantages of Edge Computing are the following:

- **Latency**: it reduces the response time, improving computing latency and transmission latency.
- **Bandwidth**: it controls the data traffic flow by optimally migrating data processing and aggregating tasks to reduce the bandwidth usage of the network while maintaining the quality of data.
- **Energy**: it incorporates a flexible task offloading scheme, which considers the resources available in each device to save as much energy as possible.

Although Edge Computing has many benefits, some challenges are still present:

1. **Partitioning and offloading tasks**: the challenge is not only related to partitioning the computation efficiently but rather doing this in an automated manner without requiring the explicit definition of the edge nodes (SHI et al., 2016; CAO; ZHANG; SHI, 2018). In this context, load balancing mechanisms are needed to balance the load across nodes (QIU et al., 2020).
2. **QoS guarantees**: the challenge is to ensure that nodes achieve high throughput and are reliable when delivering for their intended workloads (SHI et al., 2016).
3. **Privacy and security**: classical security mechanisms demand considerable computing power and memory, so they are not suitable for IoT devices (SHA et al., 2020).

2.5 CLOUD COMPUTING

Cloud Computing is the on-demand delivery of IT resources over the Internet with pay-as-you-go pricing (TAURION, 2009; CHOPRA; THAKUR; SHARMA, 2019). Instead of buying, owning, and maintaining physical data centers and servers, users can access technology services, such as computing power, storage, and databases, on an as-needed basis from a cloud provider. Cloud service providers such as Amazon, Google, and Azure offer services such as:

- **Infrastructure as a Service (IaaS)**: provides IT infrastructure such as storage, network, and computing capacity (MELL; GRANCE et al., 2011).

- **Platform as a Service (PaaS)**: provides high-level infrastructure services to the consumer. The consumer does not manage or control the underlying cloud infrastructure but has control over the deployed applications and possibly configuration settings for the application-hosting environment (e.g., AWS Lambda) (MELL; GRANCE et al., 2011).
- **Software as a Service (SaaS)**: Software applications are provided through the network using the cloud infrastructure for execution. Applications such as web-based email, database servers and content management systems are accessible from various client devices through either a thin client interface (e.g., a web browser) or a software interface (MELL; GRANCE et al., 2011).

In summary, the cloud can be seen as the most evolved stage of the virtualization concept (TAURION, 2009), giving to users high availability of services and virtually "infinite" resources on demand. However, Cloud Computing also has some disadvantages, such as:

- **Privacy and security**: storing data in a cloud storage service (usually buckets) might not be secure due to the vulnerabilities of the Internet (CHOPRA; THAKUR; SHARMA, 2019; MIRASHE; KALYANKAR, 2010).
- **Network stability**: Cloud Computing services rely on a constant and good internet connection with the client (MIRASHE; KALYANKAR, 2010).
- **Limited control**: clients are limited to using the services made available by the provider, and they do not have entire control over the services (CHOPRA; THAKUR; SHARMA, 2019).

Cloud infrastructures can be public, private, or hybrid. A brief description of each type of cloud is given below:

- **Public Cloud** is available to any user through a public interface. This type of cloud has a more affordable cost. The resources are virtually "infinite" (on-demand, pay-per-use model), but the infrastructure is shared with other users (MELL; GRANCE et al., 2011).

  Private Cloud is available for companies/customers who want to allocate a large number of resources without sharing with other users, to avoid security problems with their data and to have more control over the services offered by the cloud provider. In this type of cloud, the availability of resources is no longer "infinite" since there is a ceiling on the amount of infrastructure allocated under a business contract (MELL; GRANCE et al., 2011).
- **Hybrid Cloud** is a blend of public and private cloud concepts where the providers argue to protect their data as in private clouds in addition to making resources available on demand in a simpler way as in public clouds (MELL; GRANCE et al., 2011).

The Table 1 summarizes the differences between the Mist, Fog and Cloud Computing platforms.

Features	**Fog**	**Mist**	**Cloud**
Latency	Low	Negligible	High
Network	LAN/WLAN	PAN	WAN
Maintenance	Expert	Expert	Negligible
Storage Volatility	Semi-permanent	Temporary	Permanent
Location of Node	Near Devices	On devices	Far devices
Mobility Support	Medium	High	Limited
computing Power	Medium	Low	High

Table 1 – Main differences between Fog, Mist, and Cloud Computing layers.

2.6 MACHINE LEARNING (ML)

Conventional algorithms can be described in Computer Science as a finite sequence of executable actions that aim to obtain a solution to a given type of problem. ML algorithms are more sophisticated ones, where a dataset is given previously to the algorithm to enable the creation of a model or policy that "learns" from that dataset and is able to give a solution for new input data (unobserved instances). In other words, the model is capable of recognizing patterns from data.

ML is an interdisciplinary area that makes strong use of statistical theories, information theory, and optimization to build mathematical models for the automated detection of significant patterns in data (SHALEV-SHWARTZ; BEN-DAVID, 2014). Most of the ML algorithms follow the same workflow: preprocessing the data, feature engineering, splitting the data, training the model, testing/validating results, and making inferences for new data. This workflow is presented in Figure 3.

Moving the machine learning workflow pipeline to production needs an extra step that are not present on the figure above and it's related to the calculation of the model degratation. The most commong algorithms for that are: concept drift and data drift. They are used to retrain the model to adpat to a new statistic distribution not seen during the model training and it's the key point to continue the sucess of the model in the production.

Machine learning algorithms are present in many areas from financial business to industrial sectors. Advances in AI techniques are increasingly changing, especially in the automation sector in industries, where the use of machine learning has ceased to be

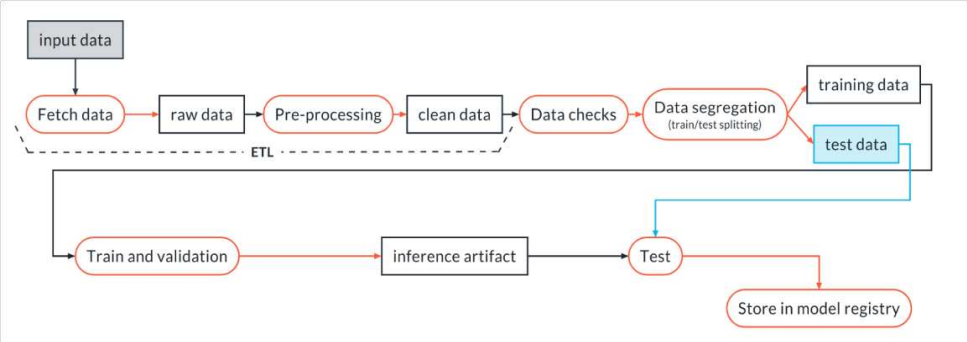

Figure 3 – Typical Machine Learning workflow (BURKOV, 2020).

an innovation for the success of companies and has begun to dictate the new industrial revolution, the so-called Industry 4.0. Common examples of the use of these AI techniques in process automation are: Inspection of parts in an industrial chain using image detection algorithms, predictive maintenance algorithms to optimize the useful life of machines, among others.

The algorithms used to create ML models are classified into three main categories:

- **Supervised Learning**: The labels/targets of the input dataset are available. During the training process, the algorithm tries to learn the parameters of the model function to map the input features to the corresponding label (SHANTHAMALLU et al., 2017).
- **Unsupervised Learning**: There are no explicit labels associated with the dataset (SHANTHAMALLU et al., 2017). The goal is to make inferences by segmenting and clustering the data into proper groups of similar features and distribution (JINDAL; GUPTA; BHUSHAN, 2019).
- **Reinforcement Learning**: What distinguishes Reinforcement Learning from Supervised Learning is that only partial feedback is given to learners about their predictions, and learners try to maximize a numerical performance measure that expresses a long-term objective (SZEPESVÁRI, 2010).

In the ML field, the specific type of ML model derived from the neural network algorithms (thus, inspired by the structure of the human brain) is called Deep Learning (DL) (SHANTHAMALLU et al., 2017; JINDAL; GUPTA; BHUSHAN, 2019; LI et al., 2019). This kind of model is characterized by having more hidden layers and is suitable for processing huge amounts of data. Its algorithms gained more popularity due to the great advancement in GPU research, where the GPUs are responsible for accelerating the training phase using a massive amount of data and achieving great performance (LI et al., 2019).

2.7 DISTRIBUTED MACHINE LEARNING (DML)

With billions of connected IoT devices, commercial websites (e-commerce), social networks, and other applications, a great transformation in the way the data is processed and stored took place. With the increasing volume of data and the way it is generated by geographically dispersed sources, data centralization techniques that were adopted until then became obsolete and unable to support the new growing wave of distributed data.

Data generated at great speed and volume by distributed sources led to a change in the way ML models are processed. To meet the large demand for data, methods such as vertical scaling are proposed to increase the processing power and storage of machines that centralize the data and the ML model to be trained. However, vertical scaling leads to problems such as high investment capital, low resilience to failures (i.e., the machine is a single point of failure), and inability to serve distributed sources if the data lies in several places.

Another technique to solve this problem is to horizontally distribute the processing and storage of data across several machines, making it possible to distribute the processing of ML models in this architecture. In DML, there are two fundamentally different ways of partitioning the training problem across all machines: parallelizing the data or the model or applying them simultaneously. Figure 4 shows the parallelism in DML (VERBRAEKEN et al., 2020).

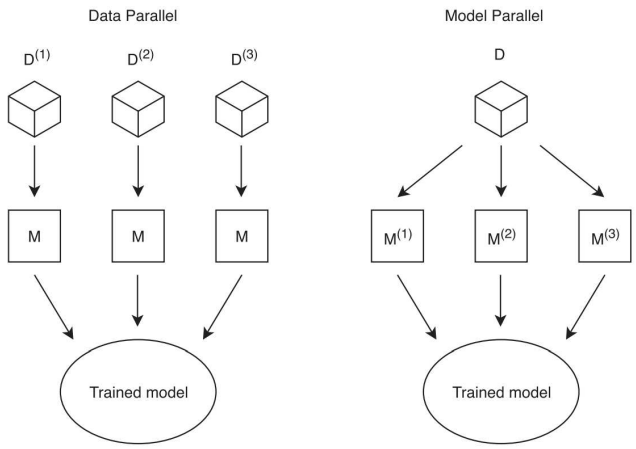

Figure 4 – Data and model parallelism.

In the data-parallel approach, data is partitioned as many times as there are worker nodes in the system, and all worker nodes subsequently apply the same algorithm to different datasets. The same model is available to all worker nodes (VERBRAEKEN et al., 2020). In the model-parallel approach, exact copies of the entire dataset are processed by the worker nodes that operate on different parts of the model. The model is, therefore, the aggregation of all model parts. However, this approach cannot automatically be applied to every ML algorithm (VERBRAEKEN et al., 2020).

The main advantages of DML are:

- Using different learning processes to train several classifiers from distributed data sets increases the possibility of achieving higher accuracy, especially on a large-size domain (PETEIRO-BARRAL; GUIJARRO-BERDIÑAS, 2013);
- Learning in a distributed manner provides a natural solution for large-scale learning, where algorithm complexity and memory limitation are always the main obstacles (PETEIRO-BARRAL; GUIJARRO-BERDIÑAS, 2013);
- Distributed learning is inherently scalable (PETEIRO-BARRAL; GUIJARRO-BERDIÑAS, 2013);
- Distributed learning overcomes the already mentioned problems of centralized storage.

The distribution of ML is not limited only to training techniques as previously described. It is possible to distribute the prediction task where a single model is not accurate enough to solve the problem. To alleviate this issue, multiple models can be combined, the so-called Ensemble Learning (VERBRAEKEN et al., 2020). It is more common to find ensemble algorithms for supervised models for classification problems. Accordingly to Peteiro-Barral & Guijarro-Berdiñas (2013), the most common algorithms are:

- **Bagging**: the process of building multiple classifiers and combining them into one (VERBRAEKEN et al., 2020);
- **Stacked generalization**: involves learning a global classifier that combines the outputs of a number of classifiers instead of using fixed rules (PETEIRO-BARRAL; GUIJARRO-BERDIÑAS, 2013);
- **Distributed pasting votes**: builds ensembles of classifiers from small pieces or "bites" of data (PETEIRO-BARRAL; GUIJARRO-BERDIÑAS, 2013);
- **Effective voting**: an effective extension to classifier evaluation and selection. Effective voting attempts to select the most significant classifiers based on statistical tests and then combine them by voting (PETEIRO-BARRAL; GUIJARRO-BERDIÑAS, 2013);
- **Distributed boosting**: this algorithm proceeds in a series of T rounds. In every round t, a classifier is trained using a different distribution D_t for its training data that is altered by emphasizing particular training examples. The entire weighted training set is given to the classifier to compute the hypothesis h_t. In the end, all hypotheses are combined into a final hypothesis h_{fn} (PETEIRO-BARRAL; GUIJARRO-BERDIÑAS, 2013).

2.8 DISCUSSION

Most IoT Big Data applications need to mine data generated by devices to extract valuable information and enhance the quality of the produced results. In order to extract insights from the high volume of data in real-time, computational processing needs to happen near the location where data is generated. To bring valuable information for IoT applications, AI or ML techniques are usually required. Applications such as image/video recognition in smart cities, automated vehicles, and industrial/manufacturing services need AI/ML algorithms to aggregate valuable information to improve the quality of their services.

However, a wide range of IoT applications cannot integrate ML services by adopting only the Cloud Computing model due to some factors: 1) **cost**: training and prediction of AI/ML models in the cloud requires the IoT devices to transmit massive amounts of data through the internet (WANG et al., 2020); **latency**: the delay to access cloud services is generally not guaranteed and might not be short enough to satisfy the requirements of many applications (WANG et al., 2020); **reliability**: Cloud Computing relies on local and wide area (backbone) networks, but for many IoT scenarios services must be highly reliable, even when the network connection is lost (WANG et al., 2020); **privacy**: the data required for ML services might carry private information (WANG et al., 2020).

Meanwhile, hardware enterprises and developers created new "miniaturized" AI accelerators (RAUSCH; DUSTDAR, 2019). Those accelerators tend to be small and efficient, thereby making it feasible to fit them easily on the edge layer (RAUSCH; DUSTDAR, 2019). Consequently, by moving those AI accelerators to the edge, it is possible to run AI/ML services closer to IoT devices. Therefore, the AI/ML services will gradually be pushed from the cloud closer to the Edge. The usage of intelligent services on the edge results in a new paradigm, the Edge AI. In this context, Greengard (2020) emphasize that "to truly and pervasively engage AI in the process within our lives (IoT), there is a need to push AI computation away from the data centers and toward the edge". For Adi et al. (2020), the convergence of ML and IoT will increase the efficiency, accuracy, productivity, and overall cost-saving for resource-constraint in IoT devices.

Edge AI will add a new responsibility to the Edge Computing paradigm since the ML training and/or inference models run locally on-or-near the devices rather than in distant cloud datacenters. Therefore, the combination of IoT and ML has enormous potential to improve the quality of human life and applications for industrial growth (ADI et al., 2020). However, Edge AI has its own design challenges that must be addressed: 1) **Distributed ML**: one particularly important requirement to allow ML to operate in the edge is to successfully distribute the training and inference algorithms on the heterogeneous edge environment; 2) **Power efficiency**: for applications that need to operate with long processing duration on battery-powered devices, energy efficiency becomes very important and necessary (LEE; TSUNG; WU, 2018); 3) **Security and privacy**: for some

Edge AI applications, it is necessary to protect private information, such as fingerprint, voice, and face recognition data (LEE; TSUNG; WU, 2018).

3 RELATED WORK

Surveys and systematic reviews in the area of Edge AI focus on a variety of domains such as: 1) DML (PONCINELLI et al., 2022); 2) Operations aspects of ML (MURSHED et al., 2021) in the edge layer; 3) Edge AI applications (KUBIAK; DEC; STADNICKA, 2022); 4) Introducing the state of the art, challenges and issues (LIU et al., 2019; ROSENDO et al., 2022; SU et al., 2022; NAIN; PATTANAIK; SHARMA, 2022). However, they do not go deeper into studying and discussing architectures for Edge AI. In this chapter, we will survey the research literature in the area of Edge AI.

Poncinelli et al. (2022) focused on DML. The authors conducted their systematic review in the context of challenges, open problems, techniques, strategies, and frameworks used in distributed learning. They discussed general issues and challenges faced in Edge Computing and pointed out some studies that tackle some of them. In addition, the authors addressed distributed learning techniques and the main frameworks that make use of them. However, they did not cover Edge AI architectures that support distributed learning techniques.

The systematic review executed by Rosendo et al. (2022) focused on problems arising from the application of ML and Data Analytics (DA) in the edge and edge-cloud layers, and they address the frameworks used to solve such problems. The authors defined their research questions within state-of-the-art methods and frameworks for ML and DA on the edge and edge-cloud, in addition to addressing the open challenges and research opportunities in this area. The authors classified the studies in the area of ML and DA based on 4 aspects: framework, hardware, application, and metrics used in the experiments. Aspects of ML and DA will be covered in this systematic review, but unlike what was addressed by the authors, this work focuses on architectures that allow such functionalities for edge computing.

The survey proposed by Liu et al. (2019) describes some Edge Computing tools and systems, such as Cloudlet, Cloudpath, SpanEdge, and others. The authors addressed, for each system, some possible application scenarios (general use, smart home, video streaming, etc.) and unique features/targets of each scenario. In addition to these systems, the authors briefly investigated DL techniques for the edge layer, where toolkits, packages, hardware, and frameworks that enable the use of DL on edge nodes are presented. However, the authors did not address architectures or systems for the edge that use such DL packages and frameworks on specialized hardware.

Su et al. (2022) address extensively the state-of-the-art of AI services on Edge Computing platforms. The authors address two main concerns related to AI services: the first is the training process in the edge layer. They listed the main architectures, emphasizing the Federated Learning (FL) model and the main open-source frameworks that corroborate with this type of model. The second concern is related to techniques and architectures for inferring AI models at the edge and how to optimize them. The authors

covered very well the state-of-the-art of AI at the edge, but they did not discuss models, techniques, and architectures that bring ML/AI services and data analytics to the edge layer.

In summary, all these related systematic reviews focused on a general domain of Edge AI and some frameworks that bring DML and ML inference to the edge layer. In this scenario, the contribution of our systematic review to existing research and review articles is mainly in the coverage of studies that address architectural models that bring AI and ML services with data analytics to the edge layer.

3.1 REVIEW METHODOLOGY

The systematic review methodology proposed in this work is based on (KEELE et al., 2007) with the three main processes, which are: 1) planning the review; 2) conducting the review; and 3) reporting the review. Figure 5 illustrates the planning process followed by this systematic review.

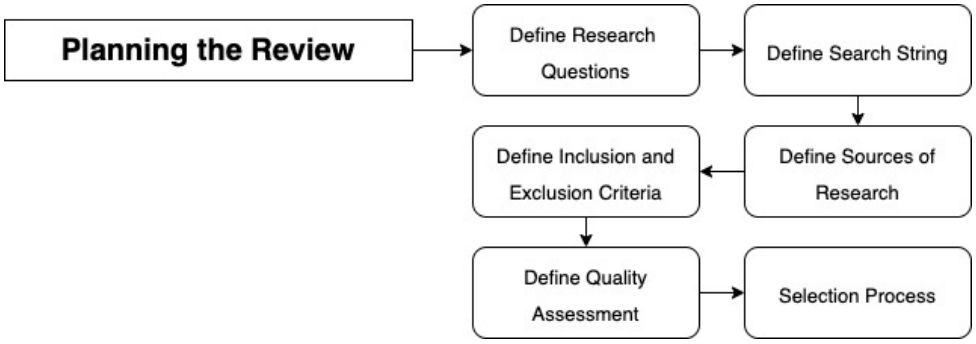

Figure 5 – Systematic review methodology.

3.1.1 Motivation

The objective of this Systematic Literature Review (SLR) and, therefore, the goal of this chapter is "to identify architectural models used in the edge layer that bring AI services and data analysis closer to the devices (Edge AI) in the context of IoT applications".

As opposed to the studies mentioned above in the Edge AI field, this SLR stands out in three main perspectives: 1) discusses the use of different layers of systems, such as fog only, mist-fog, mist-fog-cloud or edge-cloud for building an architectural model; 2) provides a detailed analysis of the integration of different frameworks in the construction of an architecture model, including the distributed learning strategies; 3) presents relevant discussions on open challenges and research opportunities identified after reviewing the articles.

3.1.2 Research questions

As a result of this SLR, we intend to answer the following research questions:

- RQ1. How do **AI/ML** services fit into an architecture that uses only edge nodes and in architectures composed of edge and cloud layers?
- RQ2. How are the architectures structured, only **mist**, **fog/edge** and **cloud** or a combination of these?
- RQ3. What are the open challenges and research opportunities in this area?
- RQ4. Which existing Edge AI architectures target **IoT applications**?

From the research questions, some sub-questions were defined, aiming to raise important aspects of the chosen works. Table 2 shows the sub-questions.

Table 2 – Sub-Questions description.

RQ	Sub-Questions
RQ1	RQ1-1. Where the inference and training are performed?
	RQ1-2. Where do the processing and the storage of data take place ?
	RQ1-3. How the strategy of distributed learning has been applied?
RQ2	RQ2-1. Are AI/ML services situated on a single-layer or on multiple layers of the architecture?
	RQ2-2. How does each layer influence ML/AI services and data processing?
	RQ2-3. What ML techniques are used in architectures to achieve QoS?

3.2 RESULTS

This section provides answers to the research questions obtained during the data extraction phase based on the related work selected by this SLR. More information about the extraction phase and the other phases of this SLR are available in Appendix 8.1.

3.2.1 RQ1. How do AI/ML services fit into an architecture that uses only edge nodes and in architectures composed of edge and cloud layers?

The goal of this research question is to identify where the ML services - inference and training - as well as data processing and storage are distributed between layers of the proposed architecture. From this question, two sub-questions were created to better identify and separate the tasks in each layer. The sub-question RQ1-1 refers to the ML service of inference and training. In the vast majority of the included articles, the model inference was found in the edge, fog or mist layer, while the training was done in the cloud. Nonetheless, Bassetti & Panizzi (2022), Natesha & Guddeti (2021), Brik et al. (2019), and Ogore, Nkurikiyeyezu & Nsenga (2021) proposed the use of only one layer for ML tasks where the training and inference phases are performed at the edge. Some papers - e.g., Ogore, Nkurikiyeyezu & Nsenga (2021) and Brik et al. (2019) - did not simulate their

architectures on real hardware; they only focused on the field of computer simulation to demonstrate their proposals. Figure 6 shows the layers used by each primary study.

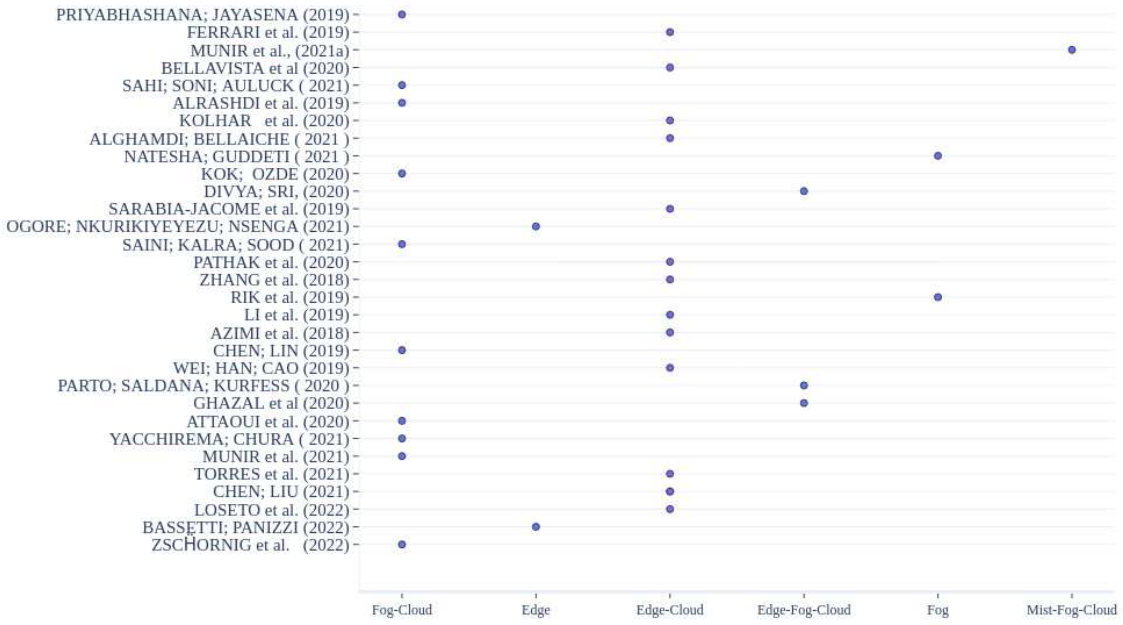

Figure 6 – Articles by ML service location layer.

The sub-question RQ1-2, on the other hand, addresses in which layer the data processing and storage is performed. Unfortunately, only a few articles selected in the SLR describe where the pre-processing and data storage took place. From the articles where this information was available, the data storage took place in the cloud, where the training was done with the data obtained from the IoT devices, and the generated model was transferred to the edge layer. In addition, only a few works explain how and where data pre-processing took place. Wei, Han & Cao (2019) proposed a pre-processing function within the embedded hardware itself, a device from which they performed several functions on the data, including the inference of the ML model. (SENGUPTA; SRIVASTAVA, 2022) used a ready-made device from Azure specifically for computer vision for ML, where both data processing, verification, and validation functions are already built into the device. However, the training service is still cloud-dependent.

3.2.1.1 How Distributed Learning algorithms have been used ?

The purpose of sub-question RQ1-3 is to identify how distributed learning is being used, whether for training or for prediction and which algorithms and models are usually employed. Some works seek to improve and adapt Federated Learning in the

nodes of the edge layer. *Nguyen et al.* (NGUYEN et al., 2021) proposed a decentralized blockchain approach that eliminates the need for a central server, where the learning updates are appended to immutable blocks for information exchange among nodes. In (NGUYEN et al., 2021), the authors focused on making efficient use of a certain amount of available resources in edge systems to minimize the model training loss function used in a federated learning context. The authors proposed a control algorithm to achieve the desired trade-off between local updates and global aggregation. However, the Federated Learning solution only focused on the training part of the model.

One popular strategy for model inference on distributed nodes is the *divide-and-conquer*, which only requires one-shot communication, but needs to constrain the number of machines. *Wang et al.* (WANG et al., 2019) proposed an algorithm for distributed inference for linear Support Vector Machine (SVM), where the new multi-round aggregations successfully eliminate this condition on the number of machines. However, this strategy is quite limited in terms of models that can be used in the divide-and-conquer approach, in addition to requiring third-party algorithms to avoid limiting the number of machines.

Divide-and-conquer and federated learning are not the only approaches in the literature to achieve distributed ML. *Georgopoulos and Hasler* (GEORGOPOULOS; HASLER, 2014) assumed an environment distributed across nodes, where each node receives different pieces of data due to the nature of the dataset. The authors proposed an algorithm that uses consensus to train the model, which is divided into two phases. First, each node with the same ML model is trained using different pieces of data. Then, the parameters of the model are communicated to initiate the consensus algorithm. The authors *did not* propose the use of consensus when *classifying or predicting new data with the same ML model*.

3.2.2 RQ2. How are the architectures structured, only mist, fog/edge and cloud or a combination of these ?

The way the architecture is geographically distributed is very important to guarantee the quality of service in IoT applications. Each layer before the cloud has different characteristics and limitations, and the combination of these can be a factor that differentiates the works. Disregarding the use of the cloud, where most of the architecture still used it, only 23 articles used edge, 12 used fog, and 4 used a multi-layered architecture. From Figure 6, it is possible to notice that works such as (GHAZAL et al., 2020), (PARTO; SALDANA; KURFESS, 2020), (DIVYA; SRI, 2020), and (MUNIR et al., 2021) were the only ones that proposed the use of multiple layers to guarantee QoS of their respective IoT applications.

In view of the vast majority of works that used only 1 layer to perform ML services, the sub-questions RQ2-2 and RQ2-3 were answered and analyzed by extracting

the data only in the works that used multiple layers. Parto, Saldana & Kurfess (2020) presented the best discussion and results of the use of multiple layers. The authors proposed a 3-layer architecture, composed by edge, fog and cloud, where most of the ML service is done at the edge and fog. The authors described the objectives of each layer. The edge is where the data is preprocessed (including the inference of models), and the fog is responsible for training the model incrementally with the help of the cloud. The authors proposed the use of the Federated Learning technique to carry out distributed and incremental training between fog and cloud.

Knowing which layers to use and how to orchestrate them is very important. However, Edge Computing requires its nodes to be made up of more dedicated, embedded hardware that can meet the demands of the necessary services. In the included articles, the Raspberry Pi hardware is the most used in the edge and fog layers (Figure 7).

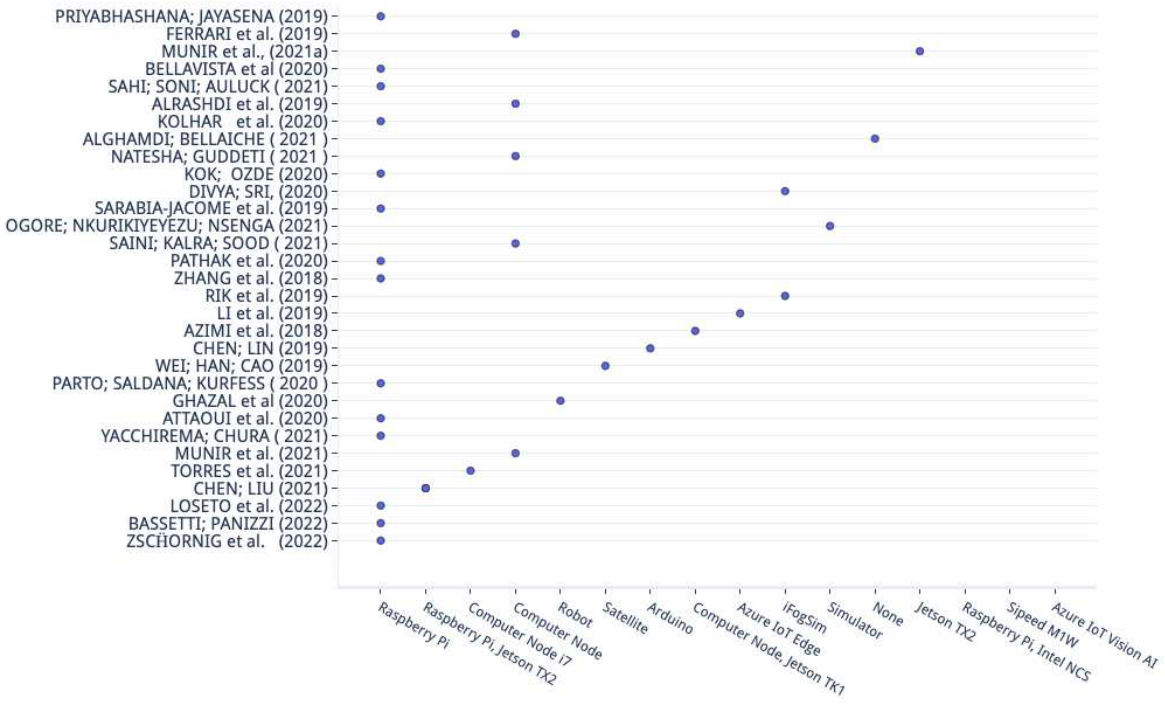

Figure 7 – Hardware used in edge layer.

Geetha et al. (2021) and Priyabhashana & Jayasena (2019) used Raspberry Pi nodes together with an accelerator (Intel Neural Stick) to allow running ML models with greater computational power. Due to the easy access, price, and wide variety of models of the Raspberry Pi hardware, many authors prefer to use it to carry out their experiments, but such hardware does not have an accelerator, nor is it the best suited to run ML models. Some works, such as (AZIMI et al., 2018), (CHEN; LIU, 2021) and (WHITE; CLARKE, 2020) used hardware from the Jetson family from NVIDIA that has greater computational capacity due to the embedded GPU. It is possible to notice that embedded hardware with GPU or TPU that are specialized for ML tasks. However, they are not

the most used in Edge AI architectures due to the high cost.

Even in the face of a wide variety of ML models, Edge Computing cannot take advantage of many of them due to its limitation in terms of the hardware used. Figure 8 shows the most commonly used ML models in Edge AI architectures. The Deep Neural Network (37.2%) is the most used model in the architectures of the related works, followed by Random Forest (11.6%). A conclusion about the types of models used suggests that most applications where the architectures were proposed are fundamentally classification problems. Figure 9 shows the main model used by each work.

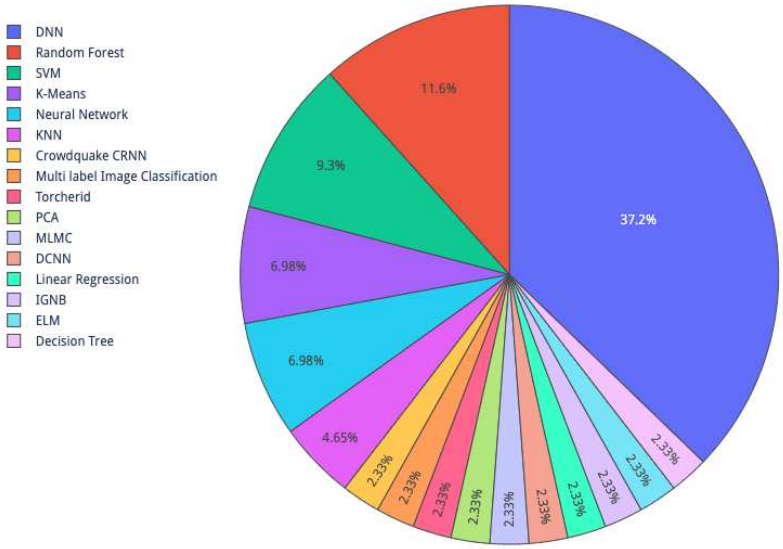

Figure 8 – Most common ML models.

3.2.3 RQ3. What are the open challenges and research opportunities in this area ?

An architecture or framework for Edge AI that can meet most IoT applications is still an open challenge. Works such as (ZSCHÖRNIG et al., 2022), (PARTO; SALDANA; KURFESS, 2020), (BELLAVISTA et al., 2020) have proposed architectures that try to be more generalist. However, the QoS requirements of each IoT application can vary a lot. Metrics such as response time latency and intermittent connectivity are parameters that greatly influence the choice of layers as well as the ML models and the data distribution techniques.

Figure 10 demonstrates a greater concern in reducing the response time (the latency between sending the data to a machine and the return of its response to the IoT device). Another very common result is the improvement of the accuracy of some ML models to be more suitable for the edge layer. Results like the modification of the architecture of a neural network by changing some parameters and layers are present in

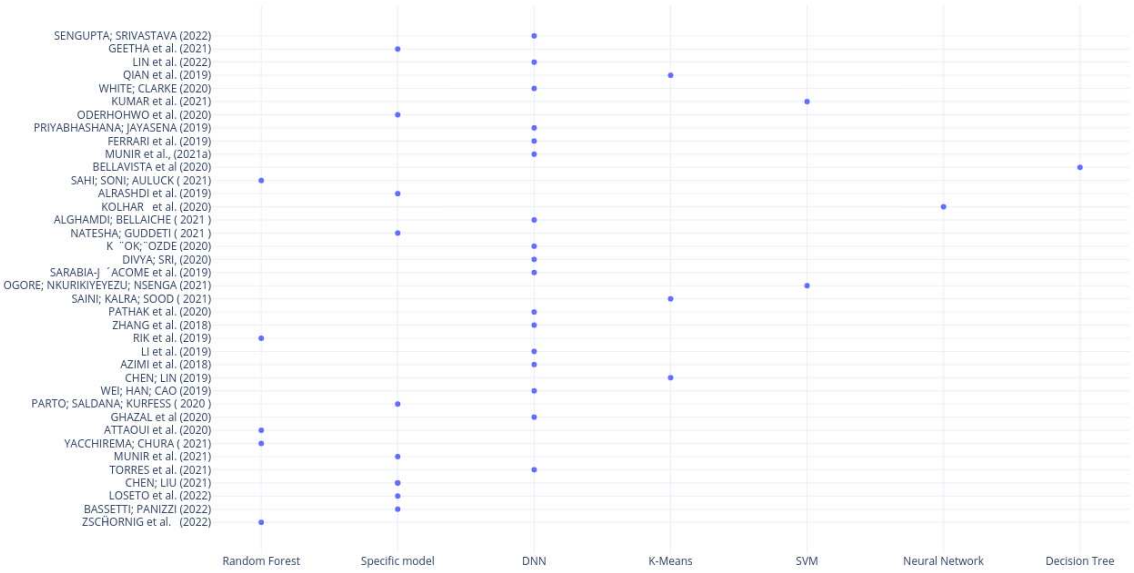

Figure 9 – Models used in the related works.

(ALGHAMDI; BELLAICHE, 2021) or the creation of a new ensemble of different models (SAHI; SONI; AULUCK, 2021).

In view of these results, the articles mostly reported two open challenges: (a) ML models that present a good trade-off between accuracy and response time suitable for the specific hardware of the edge layer and (b) energy consumption of the hardware that has embedded GPU or TPU.

3.2.4 RQ4. Which Edge AI architectures serve IoT applications, whether in a generic or specific context ?

Architectures for Edge AI generally meet the requirements imposed by some IoT applications. Developing a generic architecture is quite complex to meet different ML models that vary with the very nature of the data. Among the primary works included, only 10% described architectures able to meet any IoT application, while 90% developed an architecture to solve only one application context. Figure 11 shows each main article and its development context.

The most popular contexts for IoT are Industry 4.0 (18%), Smart Home, and Smart Cities (11%). Such contexts are growing in recent years, where end-to-end industrial process automation, smart devices in homes such as Alexa and Google Home in addition to autonomous vehicles are increasingly present in everyday life.

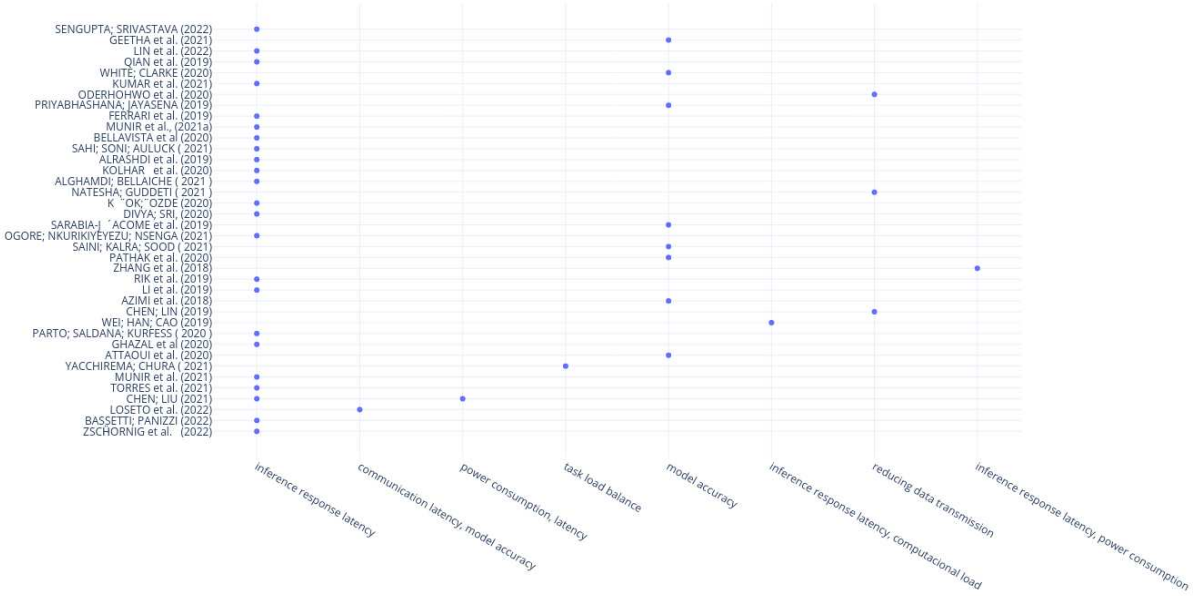

Figure 10 – Main result achieved by each work.

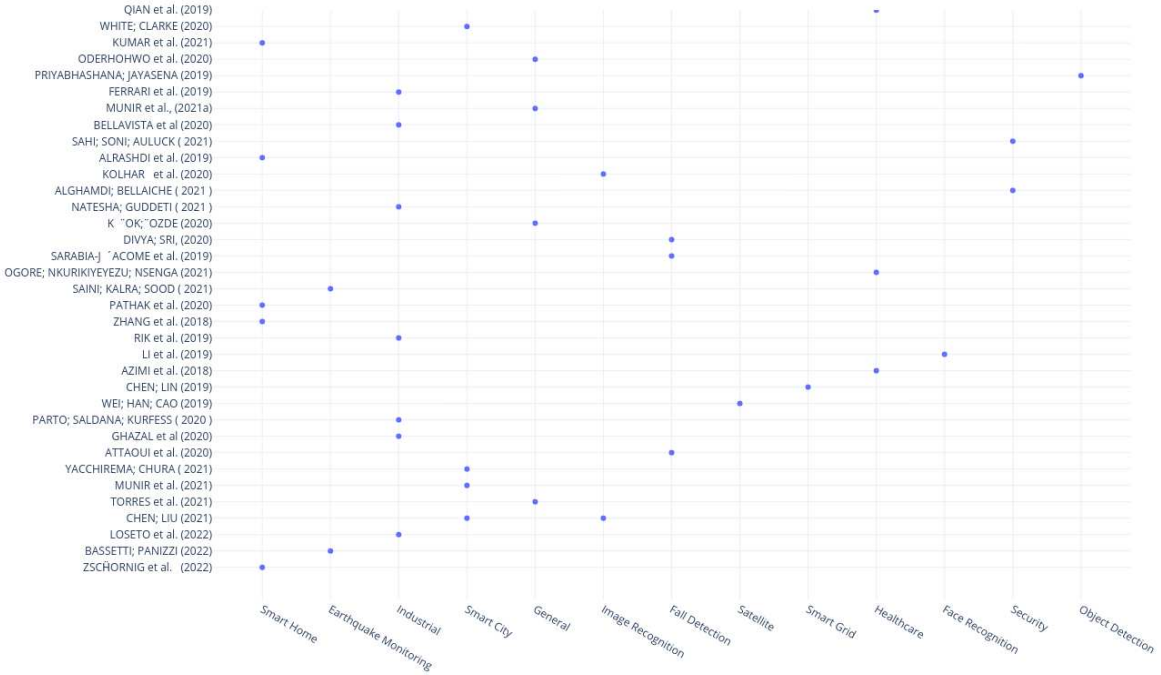

Figure 11 – Context of each work.

3.3 DISCUSSION

Given the results obtained in this systematic mapping, it is possible to notice the lack of articles in the area of Edge AI architecture, although the number of publications

has been increasing. The results showed four main points. The first point is related to layers. The architectures usually focused on using only one layer (fog or edge). The lack of experiments involving multiple layers simultaneously is an open challenge in the area of architectures for edge AI. This master's thesis, on the other hand, demonstrates the use of the architecture in different layers simultaneously and how these can be applied together with the distributed learning algorithm.

The second point concerns the generalization of the proposed architectures. Most of the works focused on edge-cloud frameworks or architectures designed for a specific scenario, and few works have proposed architectures capable of achieving QoS for multiple applications. Usually, the works are limited to the development of frameworks for distributed data processing that involve ML for multiple data formats.

The third point is the use of embedded hardware at the edge layer. The results show the engagement for viable solutions that are consistent with the requirements of Edge Computing, with the Raspberry PI being the most used hardware. Although specialized accelerators can run more complex models and perform inference and training tasks more quickly, they still have high cost and high energy consumption, and for this reason, the use of these in Edge AI architectures is still scarce.

The last point is related to ML models. According to the results, Deep Learning is the most used model due to its good accuracy compared to other models. However, it is a heavy model and requires more complex training, so it is possible to notice that 90% of the architectures use the cloud layer to help in the training phase. It is possible to highlight that distributed learning methods, such as Federated Learning, are still not used in most of the articles. Also, none of the works discussed distributed inference methods that allow the use of less complex models to achieve good aggregate accuracy. In this context, the work carried out during this master's thesis differs from the others, as it proposes a new distributed learning algorithm that uses light ML models capable of operating in the edge layer. The new algorithm aims to guarantee high accuracy and low response time while allowing the possibility of local training without the cloud.

4 EDGE AI ARCHITECTURE

Architectures commonly used in Big Data projects, such as Kappa and Lambda (LIN, 2017), are generally designed to run on servers located in the cloud. Despite the ease of instantiating machines, the advantage of processing powerful ML models, and the capability of storing a massive amount of data on these servers, some disadvantages of using those architectures in the cloud are still evident and critical for applications where the data is generated by IoT devices. Fortunately, the Edge Computing paradigm solves several issues, as the data is processed as close as possible to the IoT devices, reducing the latency and avoiding sending data to cloud servers. However, as discussed in Chapter 1, there is a lack of architectures capable of dealing with a broader range of IoT applications.

In this master's thesis, we propose a new architecture capable of meeting the requirements imposed by IoT applications that need to apply ML models with high response accuracy and low latency processing. The proposed architecture makes it possible to use lightweight ML models that can be processed in specialized hardware in the edge. These models are processed in a distributed fashion through the use of a new proposed consensus algorithm. Overall, our solution aims at the following goals:

- Allow data processing to happen in edge sub-layers, such as mist and fog;
- Improve the accuracy of lightweight ML models;
- Process data as quickly as possible;
- Enable models to be trained locally and the data to be retained at the edge layer.

This chapter covers the main aspects of the proposed solution. First, we present an overview of the architecture. Then, we present our distributed learning solution that runs within the proposed architecture, using a new ensemble algorithm that uses consensus to improve the system's accuracy by merging the results of the chosen ML models.

4.1 ARCHITECTURE OVERVIEW

The architecture is designed to sit on the edge layer, where it can be divided into mist and fog, the sub-layers of the edge, or it can be used on a single edge layer. The idea behind splitting into different layers on the edge is to incorporate the advantages of both computing infrastructures. In order to work with multiple edge layers, the architecture proposes a highly defined division of tasks. Figure 12 describes the responsibilities for each layer in our Edge AI architecture.

The base of the pyramid shown in Figure 12 represents all IoT devices that are directly connected to some machine, tool, or device. There is no restriction regarding the type of sensor to be used, the architecture is agnostic to the type of device. All the layers above the IoT devices will be composed of heterogeneous embedded hardware that will work in a distributed way. In the next sections, we give more details about each layer.

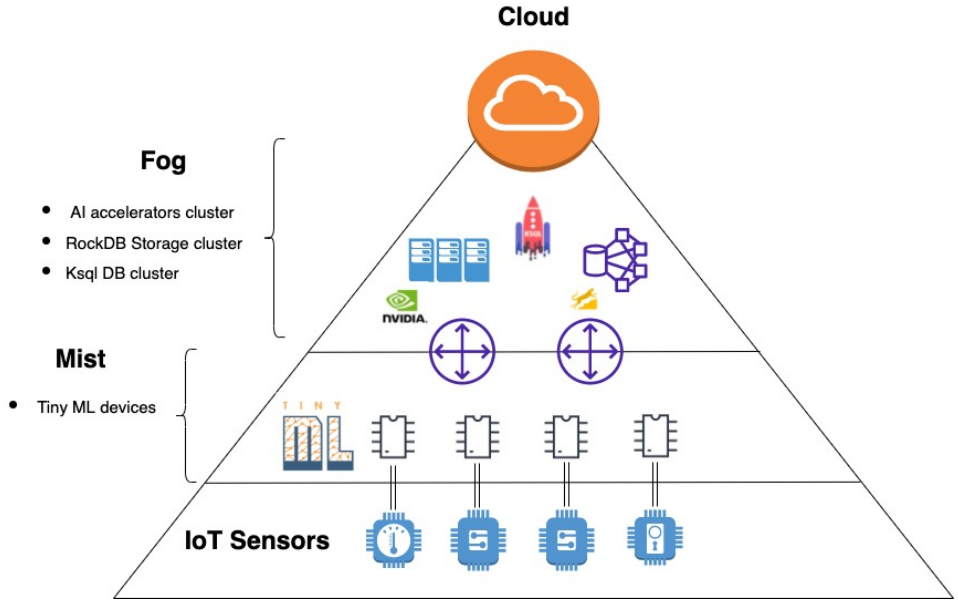

Figure 12 – Layers of the Edge AI.

4.1.1 Mist Layer

The mist layer is responsible for processing fast and simple transformations in the raw data coming from the sensor. The task of predicting patterns or states using extremely lightweight and compact ML models can be performed in this layer. The hardware used in this layer has some characteristics that can define them as tiny devices. The main characteristics of these are: (i) they are generally small in size to suit different situations; (ii) they have a low energy consumption, allowing them to operate in extreme conditions; (iii) they can operate without a persistent connection to a dedicated network or to the Internet; and (iv) they can process simple data transformations and carry out some ML prediction tasks.

The communication between the IoT sensors and the mist layer is done through direct wiring in most cases. In scenarios where the devices are not connected to the Internet, they will perform tasks locally. In scenarios where they have an Internet connection or communicate directly with fog layer hardware, communication takes place via LAN or WAN. Our architecture defines the use of a common and very robust framework to guarantee the processing of ML models in this hardware, the so-called *Tiny ML* framework.

4.1.2 Fog Layer

The fog layer is characterized by more robust hardware and with a higher computational power than the mist layer. Examples of devices used in this layer are the NVIDIA Jetson Nano, the TX2, and the Raspberry Pi with Intel Neural Compute Stick. For instance, this layer can be responsible for hosting specific databases for streaming, processing both prediction and training of ML models, and performing more complex

tasks such as image and audio processing. Usually, devices in this layer need an Internet connection to work most of the time and consume more energy than those devices used in the mist layer.

The fog layer is the most important one in the proposed architecture since it is responsible for running the distributed learning algorithm using consensus. Usually, communications between nodes in this layer take place either using WLAN (sensors) or LAN and communications with cloud servers are carried out through the Internet.

Our architecture employs the RocksDB NoSQL Streaming Database within a streaming processing framework, the KSQL, that is responsible for data processing, aggregation, and other kinds of data transformations. RocksDB uses a append-log-only structured database engine, written entirely in C++ for maximum performance, and is optimized for fast, low latency storage such as flash drives and high-speed disk drives. For that reason, RocksDB is suitable for handling IoT data. The KSQL is a more complete framework that uses RocksDB as its underlying processing engine, being responsible for processing data transformations in streaming, in addition to allowing queries to be made continuously, i.e.: continuous queries that updates the values in a dashboard automatically, and also as the common ad-hoc query, making easy to built static and dynamic dashboards. The combination of KSQL with RocksDB allows fast data storage, easy use of transformations and queries without affecting the system performance. Furthermore, KSQL is easily compatible with Kafka and with many ML processing libraries such as TensorFlow.

4.1.3 Execution Flow

The architecture employs a specific flow of executions to take advantage of layers and frameworks. Starting with the mist layer, where the nodes will be responsible for performing quick data transformations, such as filtering, summarizing, and in some cases being able to perform the prediction task using lighter ML models. Data flows from the mist layer to the fog layer using a messaging system, and the hot data (the most recent data that was generated) is stored in RocksDB. Fog nodes periodically flush the data in batch mode to persistent cloud storage if needed. The KSQL will perform more complex transformations that were not done in the mist layer, such as summarizing, aggregation, etc. In the fog layer, the distributed nodes will perform the inference together using the consensus algorithm. The cloud layer can be used for storing the cold data (data flushed by the fog storage layer on a persistent database). Moreover, the cloud can be responsible for storing the reporting and business information, such as dashboards and other BI tools, by using the SQL commands in the KSQL framework.

In this master's thesis, we tested the proposed Edge AI architecture in different Industrial IoT (IIoT) scenarios. Figure 13 describes the architecture used in the IIoT experiments. The architecture design and data flow follow the MLOps (KREUZBERGER;

KÜHL; HIRSCHL, 2023) methodology, where ML models used can be monitored and retrained.

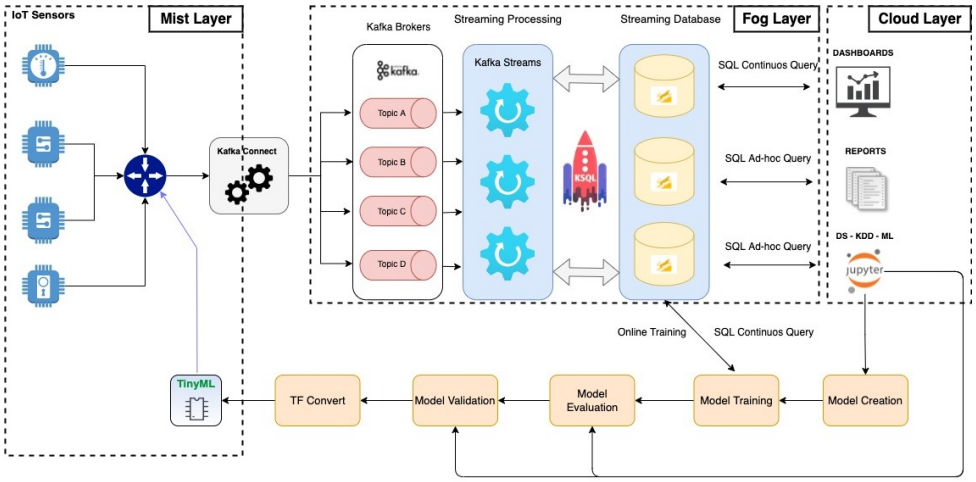

Figure 13 – The proposed IIoT architecture.

The main blocks of the architecture in Figure 13 are: (a) Message-Oriented Middleware (MOM), e.g: Kafka. Responsible for asynchronously forwarding the data produced by IoT devices to the Edge nodes; (b) Streaming Processing framework, e.g: Kafka Streams. Responsible for applying transformation functions on the streaming data; (c) Streaming Database, e.g: RocksDB. Responsible for storing the data; (d) Machine Learning steps, starting by querying the data in the streaming database, training a model, evaluating it and deploying into an edge node.

4.2 CONSENSUS ALGORITHM

One of the goals of our architecture is to ensure that the distributed system running several lightweight ML models can ensure good accuracy and low latency. However, the lighter models that are commonly used in nodes in the edge layer have low accuracy compared to more robust models, such as those that run on servers in the cloud. Using such robust models is impractical for many hardware proposed in the architecture: even if it is possible to run only the inference of the model, the training will still need to be carried out in the cloud.

Our objective is to guarantee that both training and prediction can be carried out in the edge layer. To do so, we propose a consensus algorithm that aims to transform the prediction results of multiple models into one. The function used to aggregate the data is called the *majority function*, which can range from a simple *max* or *min* function to a *weighted average*.

We consider that a network of computational nodes in the edge layer can be defined as a graph $G(V, E)$, where the vertices are embedded hardware located in one of the possible sub-layers in the edge (e.g., mist or fog). In the architecture, we assume that:

- The graph is complete, i.e., all nodes are connected to each other;
- A reliable broadcast communication protocol;
- The location of the vertices/nodes could be in different layers, i.e., mist nodes or fog nodes could belong to the same graph;
- A synchronous system with a defined maximum time for the consensus algorithm;
- Nodes can perform the model inference operation without any failure and send the result through a reliable broadcast to the other nodes;
- Nodes interact in a peer-to-peer fashion (instead of the common master-slave structure) to avoid a single point of failure, and each node can send the result of the model inference to the end device.

The algorithm consists of 3 main phases and a pre-configuration phase. The pre-configuration phase is characterized by two different scenarios that can be chosen in advance:

1. **Scenario 1:** Each node in the edge layer will train a different ML model but with the same training dataset; and
2. **Scenario 2:** The training dataset is divided into n distinct subdatasets, and each one is used to train the same ML model for every node. The entire data can be obtained by aggregating all subdatasets.

In this pre-configuration phase, all nodes are available to perform the inference of the selected model. To ensure a near-real-time response, the total processing time of the group of nodes is given by the higher inference time of a chosen ML model plus the time of sending the result to the other nodes (this time cannot exceed that latency time defined by the application). The first phase begins when the IoT devices send the same data to all nodes in the graph.

Figure 14 illustrates the first phase in a scenario where we have 3 nodes with the same hardware running different ML models (Scenario 1) on a single layer, in this case, the mist. Differently, in Figure 15, the nodes are hierarchically distributed in different layers, where nodes in the mist layer have less powerful hardware than in the fog layer, hence, running lightweight models.

The second phase consists of sending the result of the inference performed in each node via reliable broadcast to the other nodes. Each node N_i contains a List $V[N_0, N_1, \ldots, N_k]$, shown in Figure 15, where i goes from 0 to k, where k is the number of nodes and $V[N_i]$ is the result of the inference for node i. Figure 16 illustrates a node sending its response from the inference of a classification model (binary response) to all nodes in the group, where each one will save the result in its own position in array $V[N_i]$.

The third phase begins when a node N_i has all values in V. It comprises the computation of the consensus algorithm, which can be defined as a decision-making process where a group of nodes expresses their individual results from the ML inference to

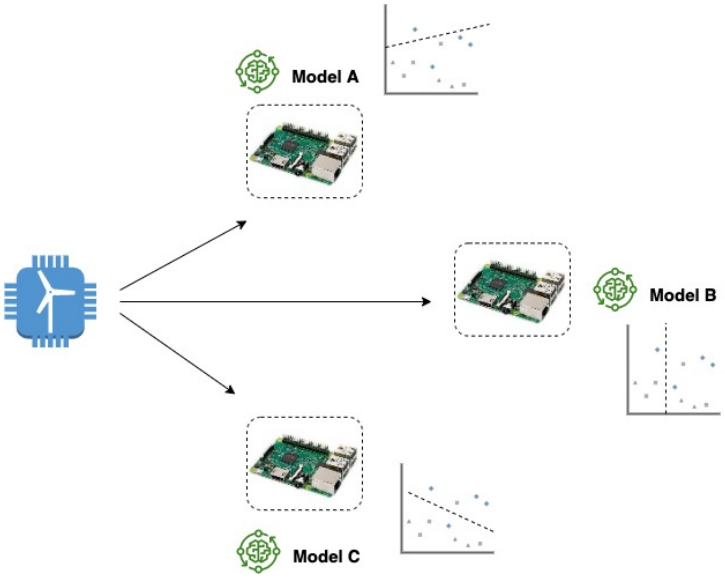

Figure 14 – First phase using a group of nodes in the mist layer.

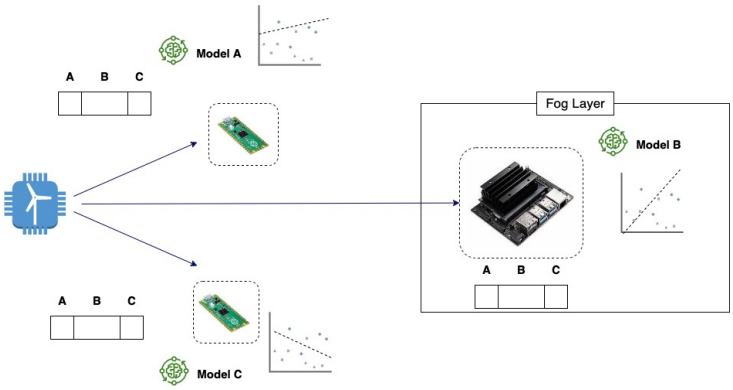

Figure 15 – First phase using a heterogeneous group of nodes in different layers.

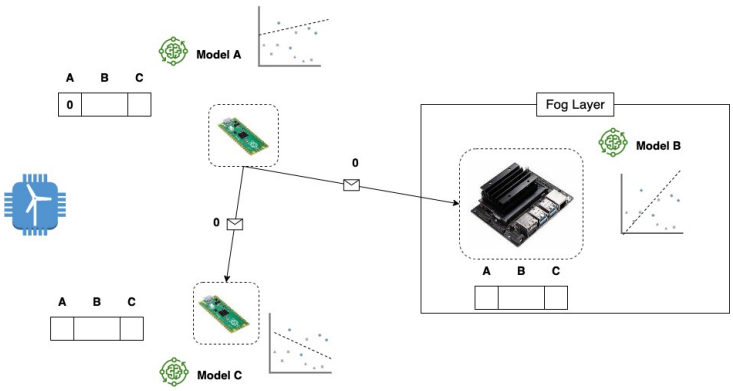

Figure 16 – Second phase using a heterogeneous group of nodes in different layers.

construct the decision which provides the best estimate of a process or system (BHARDWAJ; DATTA, 2020). The consensus function, called *majority function*, depends on the nature of the ML application. Usually, for classification problems, the majority function is defined as a boolean *AND* operation, whereas for regression problems, the function computes the average of the inference results from the nodes. The consensus function is

calculated on every node that received the last message or after exceeding the maximum time limit, using its local V array.

The behavior of each node is described by Algorithm 1. Each node starts with the array V with all empty values ($None$). In addition, they initialize variable $init_t$ to the current timestamp to calculate the algorithm timeout. Each node executes two functions, each one executed by a separate thread. The first function, ModelPredict(), is responsible for calculating the local inference based on the data received from the stream. The inference result in N_i is stored in $V[N_i]$, and this result is sent via reliable broadcast to the other nodes. The second function, ListenNodeValues(), is responsible for receiving the message that contains the node number and the result of its inference. Then, the node that received the message updates its local $V[N_i]$ value and checks if it has all values, i.e., if it has no $None$ value in the V array. If it has any $None$ value, it keeps waiting. Otherwise, it calculates the majority function (consensus) and sends the result to the actuator and to the other nodes.

Figure 17 illustrates a scenario with 3 nodes (2 nodes in mist and 1 node in fog layers), each one running a different ML model (A, B or C). Nodes that ran models A and C (mist layer) are waiting for the last node to send the response (node in fog layer). When the node in the fog layer finishes the prediction function, it will update its local $V[N_i]$ and then will broadcast its output to the other nodes, completing its own array. In that case, the node in the fog layer is able to calculate the majority function and send the signal to an actuator device that uses the obtained response to perform any operation.

$init_t \leftarrow now()$
$V[N_i] \leftarrow None$
$data \leftarrow get.streaming()$
function ModelPredict($model$,$data$)
    $V[N_i] \leftarrow model.predict(data)$
    $send.broadcast(\text{“}node\text{”} : N_i, \text{“}value\text{”} : V[N_i])$
function ListenNodeValues()
    repeat
        $node, value \leftarrow listen.broadcast()$
        $V[N_{node}] \leftarrow value$
        if $None$ is not in V **then**
            $response \leftarrow majority(V)$
            $send.back(response)$
    until $init_t < timeout$ *or* $None$ is not in V

Algorithm 1 – Consensus algorithm.

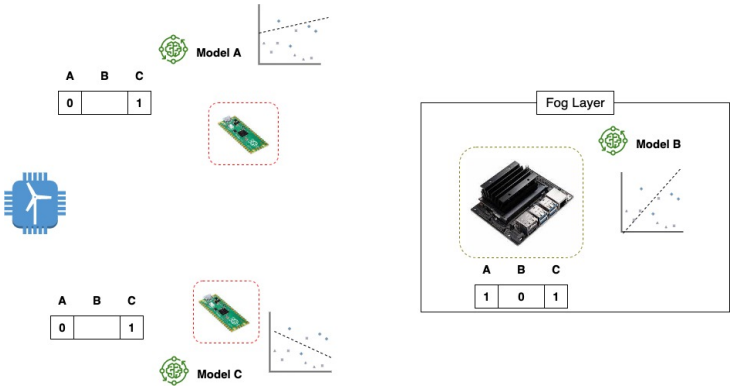

Figure 17 – Last phase in a heterogeneous group of nodes in different layers.

4.3 FEATURE SELECTION WITH CONSENSUS ALGORITHM

Most of the unsupervised ML models trained using a dataset with a large dimensionality tend to be not very accurate. For these cases, techniques of dimensionality reductions or feature selection should be applied to decrease the number of less important features, leaving only the most important ones and thus increasing the accuracy of the model.

Feature selection methodologies fall into three general classes: intrinsic (or implicit) methods, filter methods, and wrapper methods. Intrinsic methods have feature selection naturally incorporated into the modeling process. On the other hand, filter and wrapper methods work to combine feature selection approaches with modeling techniques (BUTCHER; SMITH, 2020). The main downside to intrinsic feature selection is that it is model-dependent. If the data are better fit by a non-intrinsic feature selection type of model, then predictive performance may be sub-optimal when all features are used. Filters are simple and tend to be fast. In addition, they can be effective at capturing large trends (i.e., individual predictor-outcome relationships) in the data. However, they are prone to over-selecting predictors. Wrapper methods use iterative search procedures that repeatedly supply predictor subsets to the model and then use the resulting model performance estimate to guide the selection of the next subset to evaluate. In case of success, a wrapper method will iterate through a smaller set of predictors that has better predictive performance than the original predictor set. Wrapper methods can take either a greedy or non-greedy approach to carry out feature selection.

The consensus algorithm can be used in a greedy wrapper method called Recursive Feature Elimination (RFE). The RFE algorithm is based on using a single ML model M, all the features set F and a metric O. The goal of RFE is to find the best set of features $Fb \subset F$ that will minimize the error, i.e.: $E = 1 - O_{Fb}$ and thus maximize the metric value. The algorithm starts using all features set and it iterates over and over by decreasing the set of predictors where the least important features have been removed. This process continues down a prescribed path (based on the ranking generated by the

importance) until the error stops decreasing, reaching an optimal value.

A new version of an RFE adopting the consensus algorithm was developed using the same theory of the greedy algorithm. The distributed RFE algorithm can be described as follows:

- We start using all the sensor data as our feature set, and for each feature, we train different ML models. We end up with N features $\times$ M chosen models, an N to M matrix called **MF**, where the values MF_{ij} of the matrix correspond to the evaluated metric chosen, for example, F1 score, Recall, AUC, etc;
- Next, we apply a rank to sort all the possible combinations, or pairs, of models and sensors in the matrix according to the M_{ij} values;
- The first iteration uses all the models for all the features and combines them by applying the consensus algorithm. The consensus algorithm needs to be evaluated by the chosen metric. The less important combinations of features and models based on the rank are discarded;
- The iterative method continues, discarding the less important features/model combinations until we reach an optimal subset of features and model combinations.

The proposed architecture uses the consensus algorithm in the edge layer nodes. The consensus algorithm plays a fundamental role that allows distributed nodes to use several lightweight ML models. To evaluate the performance of the architecture, it is necessary to take into account the different types of sub-layers at the edge and also the performance of the proposed consensus algorithm. Our goal is to achieve a good system accuracy in low response latency.

5 EVALUATION METHODOLOGY

To validate the proposed architecture with the distributed consensus algorithm, we designed a set of experiments to evaluate its performance, taking into account the main requirements imposed by low-latency IoT applications. The following questions guided our evaluation methodology:

- (Q1) How much time the architecture has saved running in the edge layer?
- (Q2) What is the improvement of the consensus algorithm compared to running lightweight ML models alone?
- (Q3) How robust is the architecture to different scenarios with low and high dimensions?
- (Q4) What is the processing time running the proposed architecture in a real distributed scenario using embedded hardware?

In this chapter, we first discuss the architecture used for all the experiments. Then, we discuss the proposed experimental scenario that emulates an Industrial IoT application that aims to answer the aforementioned research questions.

5.1 EXPERIMENTAL ARCHITECTURE

Figure 18 presents an overview of the architecture for the proposed experiments, showing the embedded hardware used. The grey nodes represent the: (a) Raspberry Pi Pi Pico with RP2040 chip with two ARM Cortex-M0+ cores clocked up to 133 MHz, 256 KB of RAM, 2 MB of Flash memory and (b) Raspberry Pi model 4B, featuring a quad-core Cortex-A72 (ARM v8) 64-bit SoC, clocked up to 1.5 GHz, 8GB DDR4 RAM coupled with an external 128 GB SSD. The green nodes represent the NVIDIA Jetson Nano with 4 GB RAM. Figure 19 shows the consensus algorithm in action for the same architecture.

5.2 EXPERIMENTAL DESIGN

The experiments seek to evaluate the performance of our Edge AI architecture for IoT applications. For that, we considered a very common and important task for many industrial IoT applications that require the use of ML models with low latency response and high accuracy. That task is *predictive maintenance*, which seeks to anticipate and find the root of problems in machines and pieces of equipment. To achieve that, an artificial intelligence model will be used to detect anomalies in the machine, stopping it or alerting the engineers before a possible failure occurs. The following sections describe the two scenarios we carried out for the experiments and the ML models chosen to test our architecture and the proposed consensus algorithm.

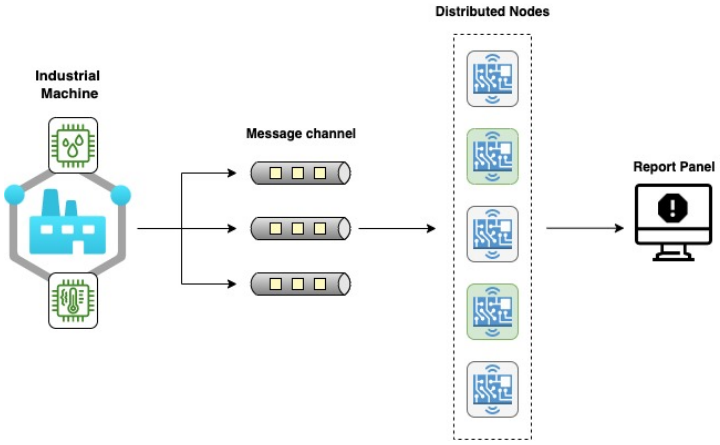

Figure 18 – The Architecture overview for the experiments.

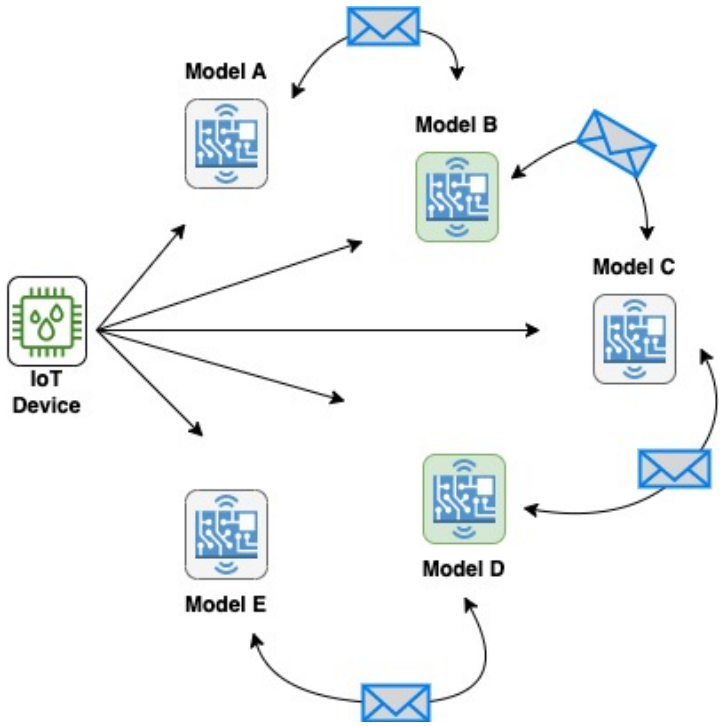

Figure 19 – Consensus algorithm running in the architecture.

5.2.1 Single Sensor: Temperature Sensor Dataset

This scenario is composed of a temperature sensor that is linked to an industrial machine. A dataset for this scenario can be found in the Numenta Anomaly Benchmark (NAB) (LAVIN; AHMAD, 2015), a novel benchmark for evaluating algorithms for anomaly detection in streaming, real-time applications. The NAB dataset provides real-time data collected from the IoT devices along with a set of labels that indicates when an anomaly occurs. The goal of this experiment is to determine whether an anomaly or failure will occur by using ML models capable of operating in the nodes of the edge layer. In this dataset, there are 4 main labels, where the first one is used to train any machine

learning model and the others for the test dataset. All the labels indicate when a machine failure occurs, where in the three last labels, the first anomaly is a planned shutdown, the second anomaly is a subtle but observable change in its behavior, and the third anomaly is a catastrophic system failure.

Figure 20 shows the time series data from the temperature sensor along with the labels (red dots) indicating an anomaly behavior of the machine.

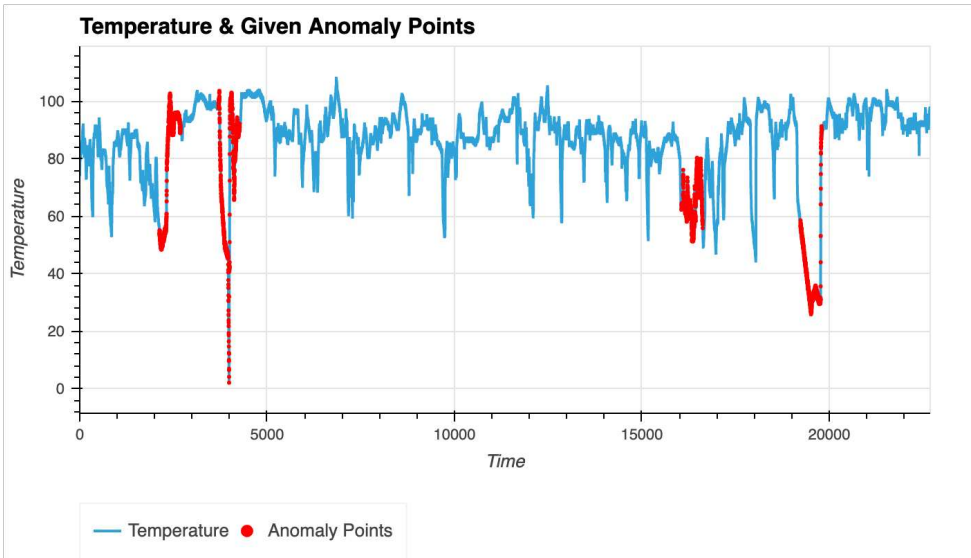

Figure 20 – Temperature and anomaly points.

5.2.2 Multiple Sensors: Water pump Dataset

The second scenario describes multiple sensors of different natures attached to a water pump. The goal is to use a subset of the total sensor data for predictive maintenance. The dataset contains: (i) an index column to identify every single event; (ii) a timestamp column that indicates the time when the event was collected (note that the difference between each timestamp observation is approximated 1 minute); and (iii) other 52 columns representing the raw data from different sensors. As discussed earlier, it is crucial to select the best subset of sensors because most of them are irrelevant to solving the problem. Figures 21, 22 and 23 show the raw data obtained from some sensors.

5.2.3 Machine Learning Models

A set of unsupervised models were previously selected, and for each experiment, a different subset of the models was chosen. The models selected were the following:

- One-Class Support Vector Machines (OCSVM) (SCHÖLKOPF et al., 2001);
- Rotation-based Outlier Detection (ROD) (ALMARDENY; BOUJNAH; CLEARY, 2020);

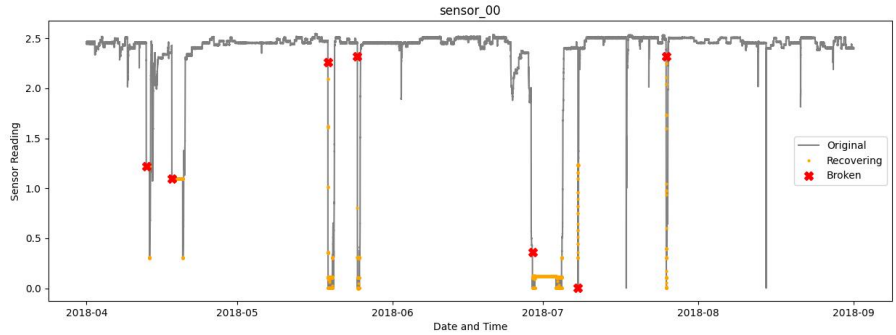

Figure 21 – Raw sensor data.

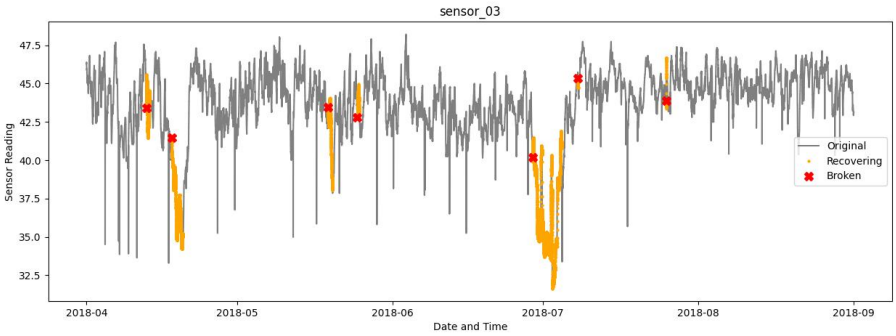

Figure 22 – Raw data from sensor 03.

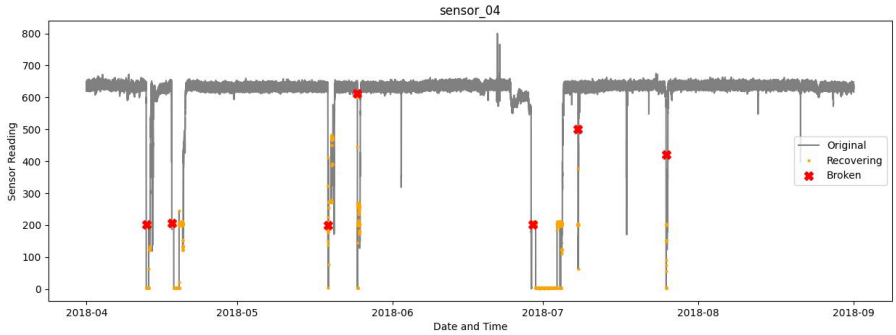

Figure 23 – Raw data from sensor 04.

- Deviation-based Outlier Detection (LMDD) (ARNING; AGRAWAL; RAGHAVAN, 1996);
- Extreme Boosting Based Outlier Detection Supervised (XGBOD) (ZHAO; HRYNIEWICKI, 2018);
- Local Outlier Factor (LOF) (BREUNIG et al., 2000);
- Angle-based Outlier Detection (ABOD) (KRIEGEL; SCHUBERT; ZIMEK, 2008);
- K-Nearest Neighbors (KNN) (RAMASWAMY; RASTOGI; SHIM, 2000);
- Connectivity-Based Outlier Factor (COF) (TANG et al., 2002);
- Local Outlier Factor (LOF) (BREUNIG et al., 2000);
- Principal Component Analysis (PCA) (SHYU et al., 2003);

- Stochastic Outlier Selection (SOS) (JANSSENS et al., 2012);
- Support Vector Machine (SVM) and OCSVM (SCHÖLKOPF et al., 2001);
- iForest (LIU; TING; ZHOU, 2008);
- Minimum covariance determinant (MCD) (HARDIN; ROCKE, 2004);
- Deviation-based Outlier Detection (LMDD) (ARNING; AGRAWAL; RAGHAVAN, 1996);
- Rotation-based Outlier Detection (ROD) (ALMARDENY; BOUJNAH; CLEARY, 2020); and
- Copula-Based Outlier Detection (COPOD) (LI et al., 2020).

The common metrics used to evaluate the models in the experiments were the following:

- *Accuracy*: the ratio of correct predictions over the total number of instances evaluated:
$$Accuracy = \frac{TP + TN}{(TP + TN)} \tag{5.1}$$
- *Precision*: measures the positive patterns that are correctly predicted from the total predicted patterns in a positive class:
$$Precision = \frac{TP}{(TP + FP)} \tag{5.2}$$
- *Recall*: the fraction of positive patterns that are correctly classified:
$$Recall = \frac{TP}{(TP + FN)} \tag{5.3}$$
- *F1 score*: the harmonic mean of precision and recall (HOSSIN; SULAIMAN, 2015):
$$F1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} \tag{5.4}$$
- *ROC curve*: a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied:
$$axisX = \frac{FP}{(FP + TN)} \tag{5.5}$$
$$axisY = \frac{TP}{(TP + FN)} \tag{5.6}$$

5.3 EVALUATED METRICS

The obtained results were evaluated with different metrics. Below is a description of each evaluated scenario and its correlation with the research questions presented in Section 5:

- **Latency time (Q1)**. This analysis compares the response time for different computation layers, edge (fog and mist) and cloud, using the same architecture. The response time is the time for data to be sent to a node in a particular layer, plus the processing time for a particular ML model and the time to send a signal back to the IoT device.
- **Machine Learning evaluation (Q2)**. It compares many different metrics for a set of different ML models used for anomaly detection in streaming data. The goal here is to find the best ML model to be used in the architecture.
- **Distributed Machine Learning evaluation (Q2)**. It aims to compare the metrics between different combinations of ML models. This analysis focuses on finding the best set of ML models that maximize the overall accuracy. In our case, the accuracy is measured by the number of detected points found before a true anomaly happens, discarding the maximum number of false anomalies (i.e., a high rate for the true positive and true negative points and a low rate for true negative and false negative points).
- **Dimensionality contribution (Q2, Q3)**. It evaluates the performance of the distributed consensus algorithm for multiple sensors. The analysis focuses on how to reduce dimensionality by applying a feature selection technique that uses the consensus algorithm to find the best set of features that maximize a particular evaluation metric. This is particularly important for scenarios that have multiple sensors' data, which are hard to be processed by lightweight ML models without increasing the processing time.
- **Consensus processing time (Q3, Q4)**. It assesses the time for the consensus algorithm to run in a set of embedded hardware applied in the edge layer. This analysis is done using the set of ML models previously chosen in the analysis step of the combinations of models that led to the best performance of the system in finding the anomalies. With the chosen set, each hardware will process a single model at a time, and the total time taken for the consensus algorithm to run is the sum of the time that each node spends processing, plus the time it takes for messages to be delivered and sent to the nodes, and the time spent computing the majority function.

6 EXPERIMENTS

In this chapter, we demonstrate the application of the proposed architecture in different simulated scenarios of IoT applications and discuss the experimental results. For all experiments, we first analyze the ML model metrics. Then, we examine the processing time for performing the scoring phase of those models. Each experiment aims to answer a specific set of our research questions.

6.1 INFERENCE RESPONSE TIME IN DIFFERENT LAYERS

A preliminary evaluation focusing on the network performance of the proposed architecture involving an Edge AI application was carried out to answer question Q1 of the evaluation methodology. Overall, the following sub-questions guided our experiment:

Q1.a How the response latency is affected by the amount of data used in the streaming window?

Q1.b What is the ML model that shows the best trade-off between accuracy and processing time?

We considered a scenario that simulates the streaming of data coming from an IoT device. Data is sent as a windowing data format, i.e., in micro-batches, to an API responsible for processing the inference of the ML model. The API is hosted in different layers of the proposed architecture (mist, fog, and cloud). For each layer, a specific hardware was used to process an ML inference task: Raspberry Pi Pico (mist), Raspberry Pi 4B (fog), and a SaaS service based on an IBM VMC instance with 8GB of memory and 4vCPUs (cloud). Each hardware hosts the same model application that will try to identify an anomaly, and it will send back a signal to the IoT device in order to stop the machine operation if an anomaly is detected. Figure 24 shows the diagram of the experimental scenario. The data produced by the streaming simulation was coming from the temperature sensor dataset described in Section 5.2.1.

The proposed scenario is composed of (a) a Docker container hosted on embedded hardware that simulates an IoT machine that streams data through the network; (b) an ML inference API hosted in the mist, fog, and cloud layers; and (c) 5 different ML algorithms chosen from the trained set of models discussed in Section 5.2.3.

The subset of models chosen for these experiments are: OCSVM, ROD, LMDD, XGBOD, and LOF. The ROC, precision, and inference processing time metrics were used to choose the best model. In these experiments, we fixed the window size to 100 values. Table 3 presents the obtained results.

As can be noticed, the ROD model achieved the best tradeoff between accuracy and inference processing time for this scenario, thus answering question Q1.b. Based

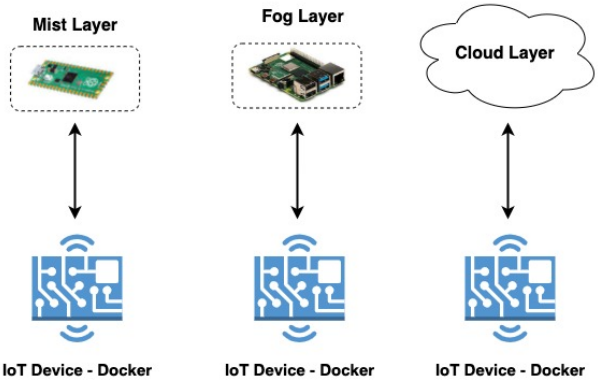

Figure 24 – Experiment configuration.

Model Evaluation			
Model	ROC Metric	Precision Metric	Inference Time (s)
OCSVM	0.8099	0.5635	1.021
ROD	0.807	0.5847	0.936
LMDD	0.8033	0.5847	0.859
XGBOD	0.8774	0.5908	9.760
LOF	0.5073	0.1151	0.00269

Table 3 – Model metrics.

on this experiment, the ROD model was chosen to perform inferences on the streaming data. To obtain results with good accuracy, it is necessary to perform a windowing in the data stream to allow the model to run the prediction with good accuracy. The proposed experiment aims to show the inference response time in each computation layer for a given window size. Figure 25 shows the elapsed response time for the fog, mist, and cloud layers and the respective window size. The results shown are an average of 100 executions.

As can be observed, the response time of the chosen model (ROD) is higher using the cloud due to the time required for transmitting data through the Internet. We noticed that the time for processing the inference on the API does not change so much with the increase in the windowing size. This is due to the high processing power provided by the cloud infrastructure.

In the fog layer, where the computation is performed closer to the IoT devices, the response time is shorter than in the cloud due to the proximity of the cluster to the producing source. However, increasing the amount of data in the window affects the response time of the model since the hardware used in this layer is less robust compared to the hardware available in the cloud layer. However, avoiding to send data over the Internet ends up being more advantageous even when increasing the window size.

In the mist layer, where both the hardware and the API are located next to the IoT devices, generally being connected directly to them, the response time approaches near real-time. However, since the infrastructure used in this layer is less powerful than

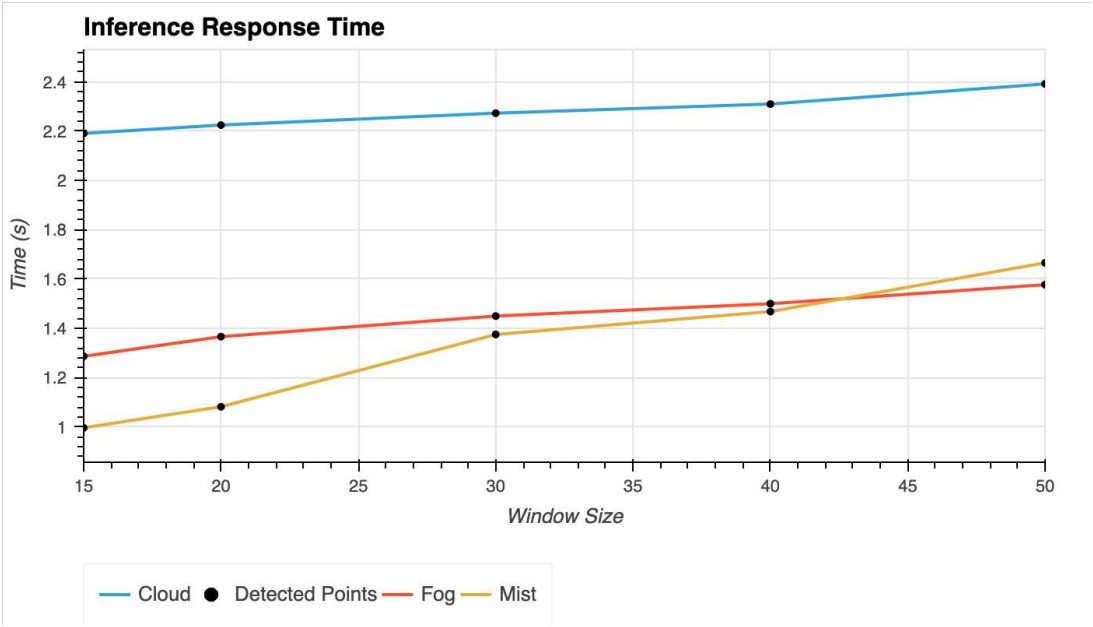

Figure 25 – Response time of the machine learning inference in each computing layer.

those used in other layers, we noticed an impact when increasing the data in the streaming window. A window size equal to or greater than 43 increased the response time considerably, making it worse than the latency obtained using fog nodes.

6.2 CONSENSUS ALGORITHM

We now evaluate the distributed learning mechanism for our proposed architecture using the proposed consensus algorithm in two different ways. First, we analyze the metrics of the aggregation result of several different models running on different nodes. Our goal with this experiment is to find the best set of ML models that leads to better accuracy than using a single model. Then, we evaluate the response latency of the consensus algorithm performed in physical hardware. This experiment seeks to answer question Q2.

6.2.1 Accuracy

For these experiments, we used the temperature sensor dataset (Section 5.2.1), where the first 10.000 points from the temperature time series data are used to train a subset of ML models described in Section 5.2.3 and the rest of the data are used to validate those models. A host machine equipped with an AMD Ryzen 5 processor operating at 3.30 GHz with 16 GB of RAM and 500 GB of SSD was used to train the models and to stream the data to the embedded hardware located in the edge layer, as illustrated in Figure 19. In this experiment, the edge layer is composed of 3 Raspberry Pi 4B nodes.

The chosen models were ABOD, KNN, COF, LOF, PCA, SOS, SVM, OCSVM,

iForest, MCD, LMDD, ROD, and COPOD. To evaluate them, we considered the following metrics: accuracy, precision, recall, and F1 score. Table 4 presents the results for each individual ML model.

Table 4 – Metrics for each model.

Model	F1	Accuracy	Recall	Precision
ABOD	0.481	0.920	0.960	0.500
COF	0.491	0.958	0.729	0.501
IForest	0.485	0.935	0.967	0.501
KNN	0.480	0.918	0.959	0.500
LOF	0.476	0.900	0.704	0.500
PCA	0.491	0.954	0.729	0.501
SOS	0.488	0.949	0.724	0.501
SVM	0.479	0.914	0.957	0.501
MCD	0.481	0.922	0.961	0.501
LMDD	0.462	0.855	0.927	0.500
ROD	0.470	0.882	0.941	0.500
OCSVM	0.477	0.907	0.953	0.501
COPOD	0.487	0.941	0.970	0.501

The results show that the ML models evaluated in this experiment did not present a good precision value and, consequently, the F1 score was also not good. Even with good accuracy and recall, the positive patterns that are correctly predicted do not present a good hit rate. This metric is very important, especially for IoT anomaly classification applications, where a high hit rate is a mandatory requirement.

In order to observe if the proposed consensus algorithm could improve the results obtained with individual ML models, we considered a mix of different ML models in 4 different scenarios. Each scenario is composed of N nodes, where each node runs a certain model different from the others. The inference output at node i for a given input j is given by p_{ij}, where p_{ij} can only take the possible values $P = \{0, 1\}$. The majority function applied in each scenario is defined in Equation 6.1:

$$\prod_{n=1}^{N} p_i \tag{6.1}$$

The proposed consensus algorithm was evaluated using an unseen dataset, shown in Figure 26, containing two anomaly points. Each scenario is a combination of a set of ML algorithms that will work together with the proposed consensus algorithm to reach a final decision. The ML models included in each scenario are presented below:

- Scenario I: IForest, ABOD, and KNN;
- Scenario II: IForest, ABOD, KNN, COF, and LOF;
- Scenario III: IForest, ABOD, KNN, COF, LOF, PCA, SOS, and OCSVM; and

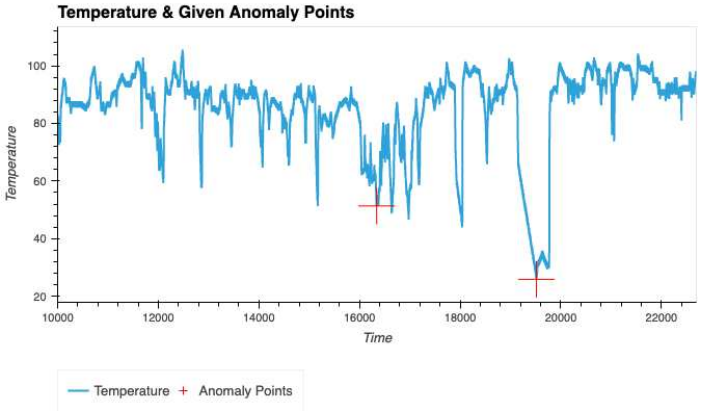

Figure 26 – Unseen dataset used to evaluate the consensus algorithm.

Table 5 – Metrics for the consensus algorithm for each scenario.

Scenario	F1	Accuracy	Recall	Precision
I	0.491	0.953	0.976	0.501
II	0.541	0.998	0.749	0.522
III	0.666	0.999	0.749	0.624
IV	0.672	0.999	0.752	0.633

- Scenario IV: all models.

Table 5 shows the metrics obtained when applying the ML consensus algorithm in each scenario. Figures 27, 28, and 29 show the points where the anomalies were labeled for Scenarios I, II, and III, respectively. As can be noticed, the number of wrong predictions (false positives for anomalies) decreases as we increase the number of ML modes used in the consensus algorithm. In Scenario III, few false positives were detected, resulting in a high accuracy of the final predictions.

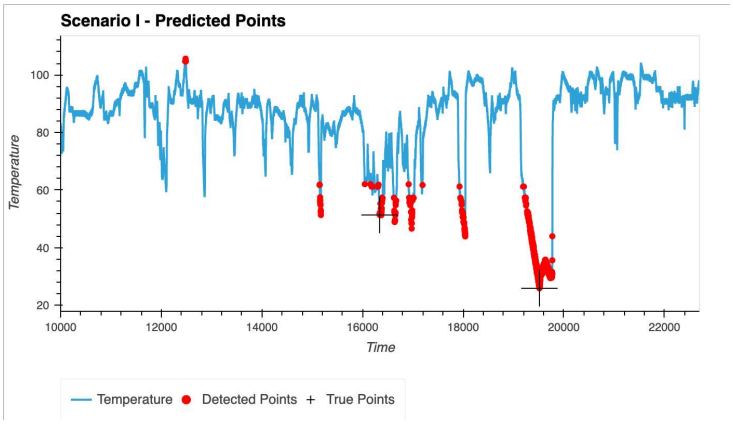

Figure 27 – Scenario I using 3 nodes and the respective models: IForest, ABOD, and KNN

As it can be noticed in Table 5, the technique of distributing the inference of different ML models using the consensus algorithm presented better results than individ-

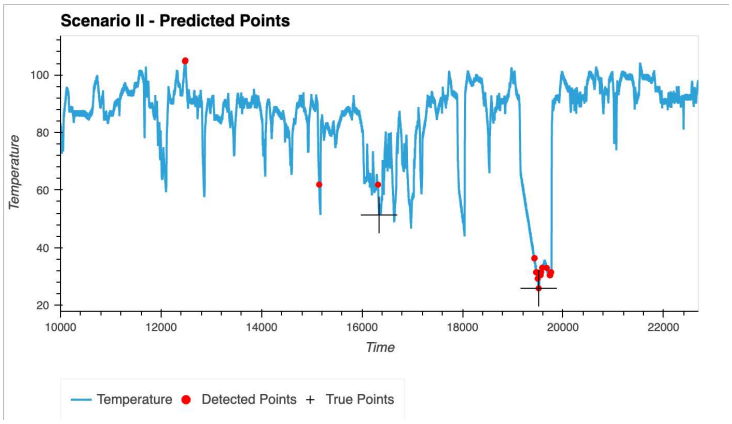

Figure 28 – Scenario II using 5 nodes and the respective models: IForest, ABOD, KNN, COF, and LOF

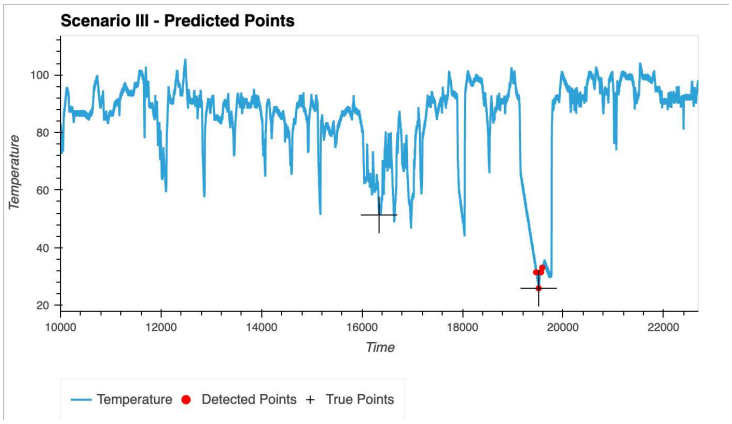

Figure 29 – Scenario III using 8 nodes and the respective models: IForest, ABOD, KNN, COF, LOF, PCA, SOS, and OCSVM

ual ML models. Overall, as we increase the number of models in each scenario, the final result of the consensus algorithm is improved. However, the difference in metrics between scenarios III and IV is not too expressive, meaning that the addition of more models in the consensus algorithm was unable to further increase the quality of the results in this case. This result shows that it is possible to achieve near-perfect accuracy by using different ML models along with the proposed consensus algorithm.

6.2.2 Latency

Although ML-related metrics are important in assessing the quality of ML predictions, another important aspect that must be taken into consideration is the inference processing time for each model running in real hardware located in the edge layer. Table 6 shows the mean and standard deviation of the total response time for each model in a node located in the fog layer. In this experiment, we measured the response time, which includes the time required for the data sent from the host to arrive at the computational node ($Time_1$), the processing time by the selected model prediction function ($Time_2$), and the time for the model response to arrive at the host ($Time_3$). Equation 6.2 presents

the formula used to calculate the response time:

$$ResponseTime = Time_1 + Time_2 + Time_3 \tag{6.2}$$

Table 6 – Response Time for each scenario.

Scenario	**Inference response time (s)**		
Model	**Mean**	**STD**	**Nodes**
IForest	0.132	0.016	1
KNN	0.029	0.010	1
LOF	0.021	0.009	1
PCA	0.021	0.018	1
SOS	0.068	0.076	1
OCSVM	0.048	0.0133	1
MCD	0.019	0.011	1
COPOD	0.033	0.008	1
LMDD	0.123	0.0416	1
ROD	0.032	0.218	1
ABOD	0.045	0.015	1
COF	0.043	0.013	1
Consensus			
ABOD, LOF	0.2226	0.04335	2
PCA, IForest	0.309	0.0541	2
ABOD, IForest, KNN	0.411	0.0371	3
ABOD, LOF, PCA	0.373	0.0444	3

The results in Table 6 show that the inference time for each model running on a single node is quite satisfactory for applications that require low response latency. With the use of a distributed architecture using the consensus algorithm, the inference time becomes longer, as it is necessary to account for the time for exchanging messages between nodes plus the time that the majority function takes to be processed. However, experiments carried out with an architecture using 2 and 3 Raspberry Pi nodes, with each one running a different model, show that the consensus algorithm overhead, i.e the message exchange time, was around 0.134s for 2 nodes and 0.240s for 3 nodes, on average. As new nodes are added, this overhead time increases, but the accuracy improves in return. The decision on an acceptable timeout for the ML service response time will vary for each IoT application and, therefore, will influence the maximum number of nodes that can be used with this approach.

6.3 DIMENSIONALITY REDUCTION USING THE CONSENSUS ALGORITHM

This experiment focuses on dimensionality reduction using the consensus algorithm, answering our final research questions (Q3 and Q4). The multiple-sensor water pump scenario was used to carry out our analysis. The set of ML models chosen were: K-Nearest Neighbors (KNN), Local Outlier Factor (LOF), Principal Component Analyses (PCA), Rotation-based Outlier Detection (ROD), Multi-dimensional Concept Discovery (MCD), IForest and Angle-based Outlier Detection (ABOD). The sensors were considered as independent features and trained separately with all the chosen models, and precision and recall were used as the evaluation metrics. We discarded[1] 14 out of 52 sensors since their data had a high level of noise.

For each sensor and model combination, we generated a $N \times M$ matrix, where the lines correspond to the features and the columns to the models, and the value of each cell corresponds to a specific metric. Table 7 shows the obtained matrix where the M_{ij} position corresponds to the precision metric. Table 8 is similar to the previous one, but the M_{ij} position corresponds to the recall metric.

Table 7 – Features x Models: precision values.

Sensor	ABOD	IForest	MCD	ROD	PCA	LOF	KNN
sensor_26	0.49989	0.49988	0.49989	0.49989	0.50019	0.50016	0.50002
sensor_17	0.49989	0.49989	0.49989	0.49989	0.49989	0.49988	0.49989
sensor_30	0.49989	0.49989	0.49989	0.49989	0.49989	0.49987	0.49987
sensor_23	0.49989	0.49988	0.49989	0.49989	0.50012	0.49988	0.49988
sensor_48	0.49989	0.49981	0.50003	0.49989	0.49987	0.49987	0.49988
sensor_36	0.49989	0.49988	0.49989	0.49989	0.49989	0.49988	0.49988
sensor_27	0.49989	0.49988	0.49988	0.49988	0.49988	0.49988	0.49987
sensor_16	0.49989	0.49989	0.49989	0.49989	0.49987	0.49988	0.49988
sensor_29	0.49989	0.49989	0.49989	0.49989	0.49989	0.49988	0.49988
sensor_31	0.49989	0.49989	0.49989	0.49989	0.49989	0.49989	0.49989
sensor_32	0.49989	0.49989	0.49989	0.49988	0.49989	0.49988	0.50006
sensor_04	0.49989	0.50581	0.49988	0.49988	0.49988	0.49971	0.50019
sensor_33	0.49989	0.49989	0.49989	0.49989	0.49989	0.49988	0.49988
sensor_28	0.49989	0.49988	0.49989	0.49989	0.49989	0.49988	0.49985

The experiment was divided into two parts. The first uses the precision metric to reduce the number of sensors and models, while the second part uses recall as a metric to be optimized. Our intent was to find a combination of features and models able to maximize the precision or the recall metrics (i.e., maximize the number of true positives and minimize false positives, or maximize the number of true positives and minimize false negatives). For both parts of the experiment, the window technique was performed on the data in the same way as we performed in the latency experiment, described in section

[1] Discarded sensors: sensor_04, sensor_27, sensor_16, sensor_28, sensor_26, sensor_36, sensor_48, sensor_29, sensor_31, sensor_23, sensor_17, sensor_32, sensor_33, sensor_30.

Table 8 – Features x Models: recall values.

Sensor	ABOD	IForest	MCD	ROD	PCA	LOF	KNN
sensor_04	0.5	0.981	0.477487	0.477487	0.477498	0.196333	0.694596
sensor_16	0.5	0.49507	0.5	0.499733	0.425119	0.475552	0.490939
sensor_17	0.5	0.49596	0.5	0.5	0.497509	0.4664	0.493992
sensor_23	0.5	0.494462	0.499972	0.499989	0.508981	0.461742	0.469005
sensor_26	0.5	0.634932	0.5	0.499728	0.697229	0.628753	0.71762
sensor_27	0.5	0.459189	0.467949	0.48063	0.486899	0.470843	0.421113
sensor_28	0.5	0.480675	0.5	0.5	0.5	0.474304	0.387445
sensor_29	0.5	0.496096	0.5	0.5	0.5	0.472505	0.456222
sensor_30	0.5	0.493305	0.499977	0.5	0.499972	0.428353	0.446849
sensor_31	0.5	0.498372	0.498218	0.49819	0.498218	0.494389	0.497901
sensor_32	0.5	0.495257	0.498621	0.486888	0.49861	0.473793	0.561985
sensor_33	0.5	0.497078	0.499972	0.494122	0.499972	0.47065	0.491592
sensor_36	0.5	0.479053	0.498196	0.5	0.5	0.47078	0.460647
sensor_48	0.5	0.533815	0.533214	0.476335	0.426628	0.430282	0.460437

6.2.2. First, we calculated the RFE using individual points collected by each sensor. This means that, for each sensor and model chosen, only a single point collected in time was used to predict the occurrence of an anomaly. Then, we increased the window size from 2 up to 9. Our first assumption was that larger windows would help improve the metrics of the model.

6.3.1 Precision as the Main Metric

Table 9 illustrates a sample of the matrix created by ranking the precision value for each combination of sensor and model individually.

Table 9 – Sensor and model combination sorted by Precision

Sensor	Model	Precision	Rank
sensor_04	LOF	0.49971	98
sensor_48	IForest	0.49981	97
sensor_28	KNN	0.49985	96
...	...	...	...
sensor_26	PCA	0.50019	3
sensor_26	KNN	0.50020	2
sensor_04	IForest	0.500459	1

The RFE starts using all sensor and model combinations in the consensus algorithm and calculates the new precision of each combination, excluding at the end the less important model and feature combination. In each iteration, the RFE excludes the less important sensor and model combinations, dropping one at a time until it finds the

optimal precision value. The precision value calculated in each iteration corresponding to the use of the top N combinations in the consensus algorithm is illustrated in Figure 30. Table 10 shows the precision value for combinations 12 to 2.

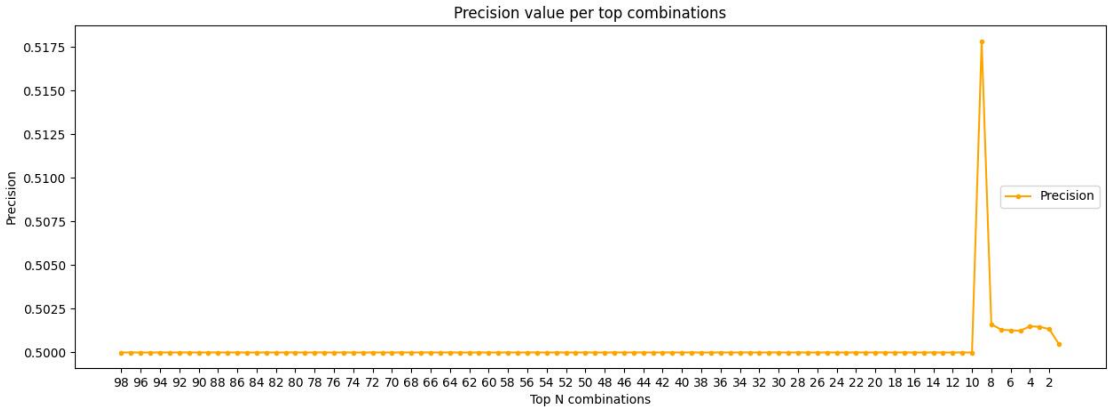

Figure 30 – Precision value calculated using the top sensor and model combinations.

Table 10 – Sensor and model combinations sorted by precision.

Combinations	Precision
12	0.499989
11	0.499989
10	0.499989
9	**0.517851**
8	0.501607
7	0.501303
6	0.501260
5	0.501238
4	0.501488
3	0.501464
2	0.501459

The optimal precision value is achieved in iteration 89, which uses the top 9 model and sensor combinations. However, to ensure that this optimal combination is capable of detecting anomalies before they actually happen, we analyzed the predicted points and the true anomaly points. Figure 31 shows the obtained result. The first anomaly detected was wrong (false positive) on timestamp 2018-07-22 03:26:00, and the algorithms continued to detect for every event until 03:30:00 an anomaly situation. The second detection occurs on timestamp 2018-07-25 13:54:00. Differently from the first detection, this was a true positive point since the anomaly will happen at 14:00:00 (in this case, the model was able to detect it 6 minutes in advance). The consensus algorithm will continue to detect an anomaly until 17:39:00, 10 minutes before the machine did the last recovery. Therefore, among the 2 anomalies that occurred, the model missed the first one and hit the last one.

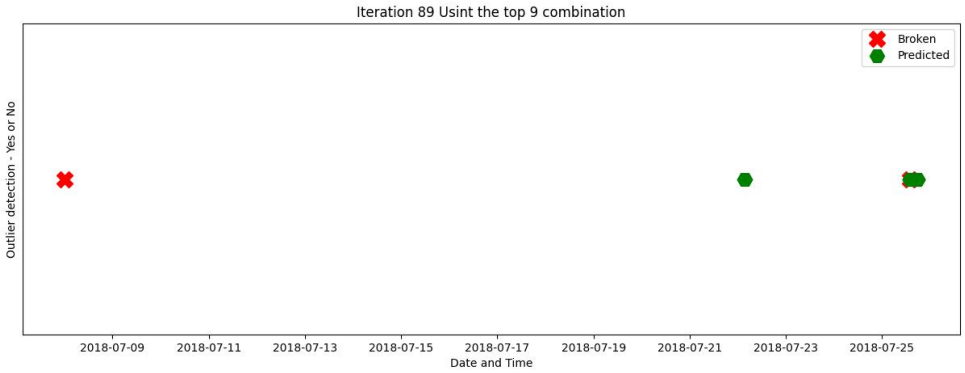

Figure 31 – Anomaly points predicted using the Top 9 combinations vs True anomaly points.

Table 11 shows the top 9 sensor and model combinations, with their precision values. We can see that if each combinations is taken separately, the precision value will be below the value obtained using the consensus algorithm.

Table 11 – Top 9 most important sensor and model combinations.

Sensor	Model	Precision	Rank
sensor_04	IForest	0.500459	1
sensor_26	KNN	0.500020	2
sensor_26	PCA	0.500019	3
sensor_04	KNN	0.500019	4
sensor_26	LOF	0.500016	5
sensor_26	IForest	0.500015	6
sensor_23	PCA	0.500012	7
sensor_32	KNN	0.500006	8
sensor_48	MCD	0.500003	9

Table 12 illustrates the best precision values for the same set of sensor and model combinations using different window sizes. Figure 32 confirms the values from the previous table. The best precision value was achieved using a window size equal to 3 with the top 9 combinations shown in Table 11. Figure 33 shows that the second anomaly was predicted before the anomaly happens, and no other false negative point was detected. However, the first anomaly was not detected, confirming that even with larger windows, the best set of combinations chosen by the RFE algorithm using precision as the main metric was not enough to predict the two anomalies in the dataset.

6.3.2 Recall as the Main Metric

We now proceed with the same analysis using the recall metric. The recall value calculated in each iteration corresponding to the use of the top N sensor and model

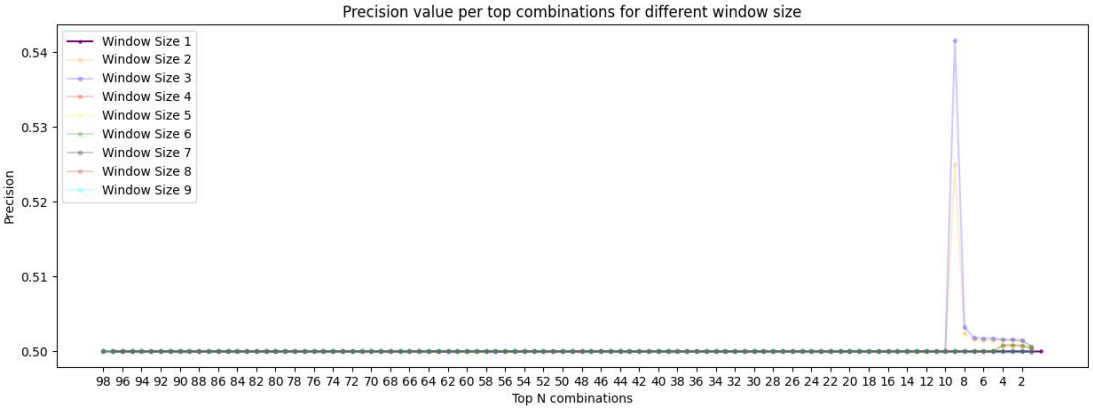

Figure 32 – Precision value per iteration changing window size.

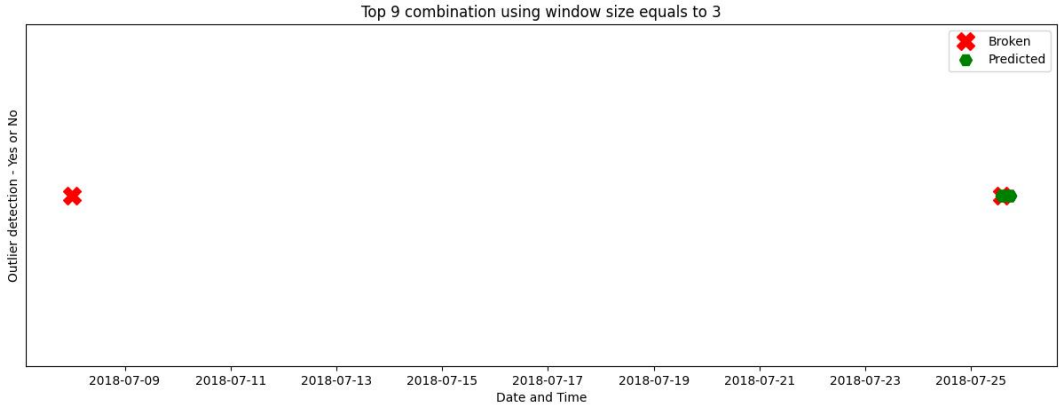

Figure 33 – Anomaly points predicted using the top 9 sensor and model combinations and a window size equal to 3.

Table 12 – Best precision values per sensor and model combinations when varying the window size.

Top Combination	Window Size	Precision
9	**3**	**0.541661**
9	2	0.524994
9	1	0.517851
8	3	0.503262
8	2	0.502375
7	3	0.501806
5	3	0.501767
6	3	0.501767
7	2	0.501567
3	3	0.501565
4	3	0.501565

combinations in the consensus algorithm is shown in Figure 34. Table 13 shows the recall value obtained using the consensus algorithm for combinations 12 to 2.

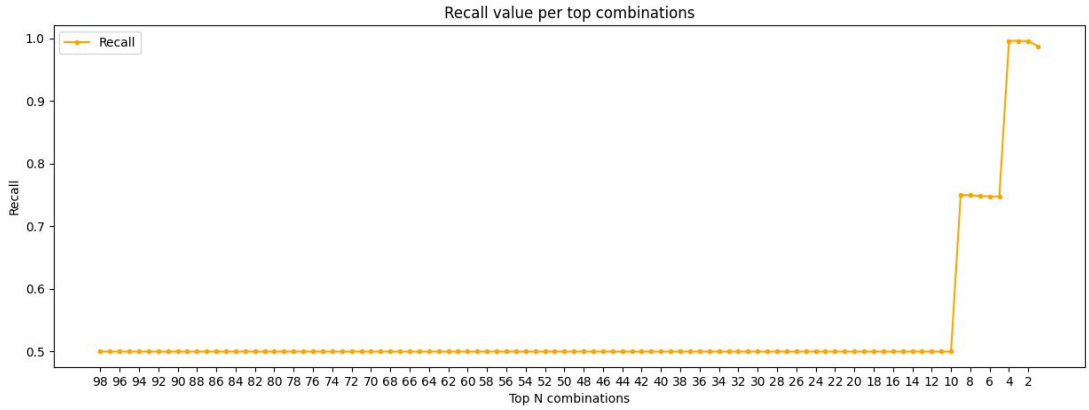

Figure 34 – Recall value calculated using the top sensor and model combinations.

Table 13 – Sensor and model combinations sorted by recall.

Combination	Recall
12	0.50000
11	0.50000
10	0.50000
9	0.749847
8	0.749847
7	0.749847
6	0.747765
5	0.747725
4	**0.996199**
3	0.996136
2	0.995733

In this scenario, the optimal precision value was achieved in iteration 93, which used the top 4 sensor and model combinations. As can be noticed in Figure 35, the consensus algorithm has predicted the two positive anomalies. However, it has also detected 5 false positive points, the first at 2018-07-04 18:58:00, the second at 2018-07-04 23:59:00, the third at 2018-07-17 15:29:00, the fourth at 2018-07-25 22:11:00 and the last at 2018-08-12 21:55:00. The first predicted anomaly was detected at 2018-07-08 00:05:00 6 minutes before the true anomaly point (2018-07-08 00:11:00), and the algorithms stopped detecting an anomaly at 2018-07-08 00:52:00. The second predicted anomaly was detected at 2018-07-25 13:54:00, also 6 minutes before the true anomaly point (2018-07-25 14:00:00).

Table 14 shows the top 4 sensor and model combinations with their recall values. Again, the results for each ML/sensor combination alone are worse than the one obtained using the consensus algorithm.

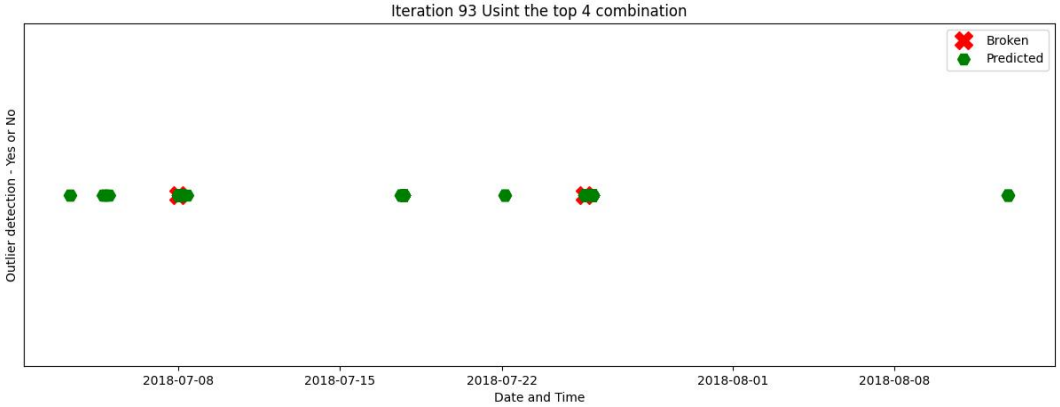

Figure 35 – Anomaly points predicted using the top 4 sensor and model combinations vs. true anomaly points.

Table 14 – Most important features.

Sensor	Model	Recall
sensor_04	IForest	0.9476
sensor_26	KNN	0.7176
sensor_26	PCA	0.6972
sensor_04	KNN	0.6945

Figure 36 shows the recall value for different window sizes over different sensor and model combinations. The best recall value was achieved using a window size of 3, and the best 3 combinations showed in Table 15. Figure 37 shows the analysis to ensure that this optimal combination detects the anomaly before it happens. The results are similar to the top 4 using a window size equal to 1, although the number of false positive points has decreased.

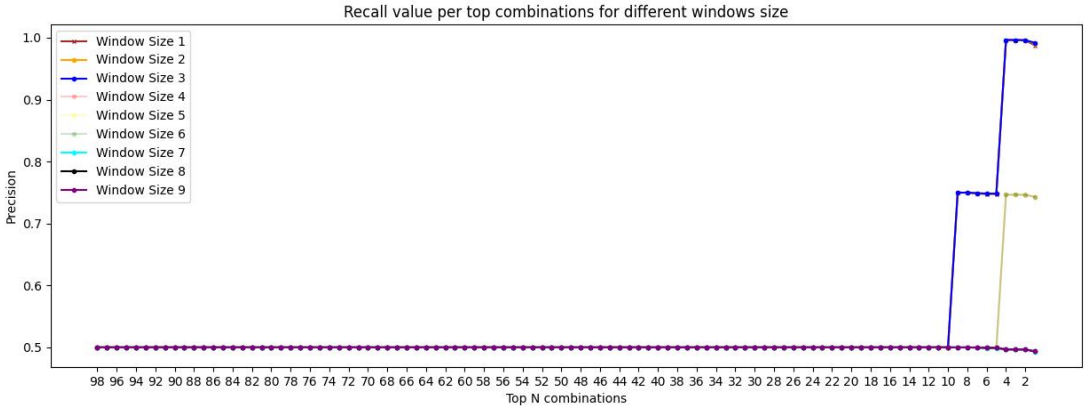

Figure 36 – Recall value per iteration changing window size.

Table 15 – Top 3 most important sensor and model combinations.

Sensor	Model	Recall
sensor_04	IForest	0.4876
sensor_26	KNN	0.7176
sensor_26	PCA	0.6972

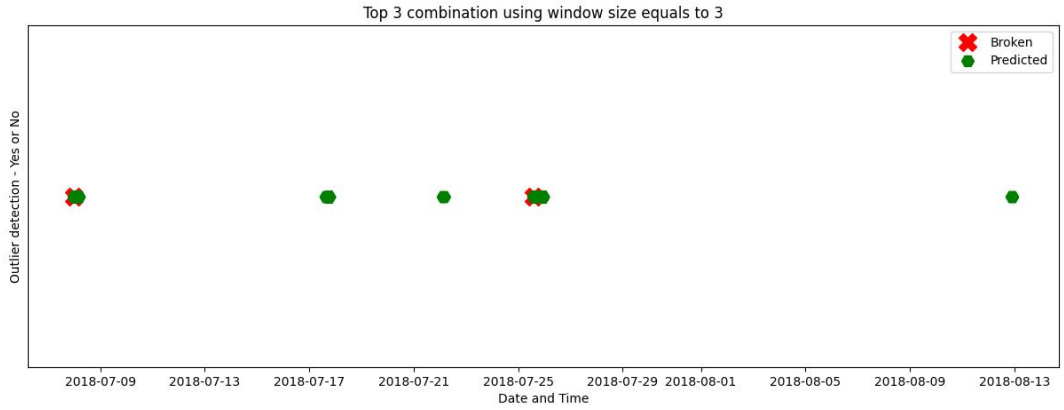

Figure 37 – Anomaly points predicted using the top 3 sensor and model combinations and window size equal to 3.

6.3.3 Impact on Latency

Due to the limited number of physical nodes, only the top 5 and top 3 sensor and model combinations were evaluated. For the experiment involving 5 nodes, all the nodes described in Figure 19 were used, and for the scenarios with 3 nodes, only the Raspberry Pis were used. This experiment seeks to answer our research question Q4.

The first step was to calculate the average time taken for a node to process the inference task of each chosen model. Then, the time taken to process the complete consensus algorithm with the best sensor and model combinations was carried out. Six different scenarios were proposed using a different number of nodes:

- **Scenario I**: sensor_26 + PCA | sensor_26 + LOF;
- **Scenario II**: sensor_26 + PCA | sensor_26 + IForest | sensor_26 + LOF;
- **Scenario III**: sensor_26 + PCA | sensor_04 + IForest | sensor_26 + KNN;
- **Scenario IV**: sensor_26 + PCA | sensor_26 + IForest | sensor_26 + KNN | sensor_04 + IForest | sensor_04 + KNN;
- **Scenario V**: sensor_26 + PCA | sensor_26 + IForest | sensor_26 + KNN | sensor_04 + IForest | sensor_48 + MCD.

The values for each scenario were collected and are illustrated in Table 16. As expected, the consensus algorithm increases the inference time due to the majority func-

tion that requires exchanging messages across the nodes. For the scenario using the top 3 models chosen in the RFE using recall as the main metric, the consensus time is still very low and suited for IoT applications.

Table 16 – Response time for each scenario.

Scenario	**Inference response time**		
Model	**Mean**	**STD**	**Nodes**
IForest	0.127	0.012	1
KNN	0.028	0.008	1
LOF	0.022	0.009	1
PCA	0.022	0.017	1
MCD	0.019	0.010	1
Consensus			
Scenario I	0.288	0.0519	2
Scneario II	0.352	0.0301	3
Scenario III	0.401	0.0222	3
Scenario IV	0.678	0.0331	5
Scneario V	0.622	0.0521	5

On the other hand, using 5 nodes results in a significant time increase compared to 3 nodes. Based on this observation, increasing to 8 nodes (the best result found using precision as the main metric) will lead to a significant increase in response time, which may not be accepted for many low-latency IoT applications. Therefore, the best value achieved for the experiments was using the windowing technique with the 3 best sensor and model combinations chosen by the RFE using recall shown in Scenario III.

6.4 DISCUSSION

The central goal of our study was to propose a new base architecture that could meet the requirements imposed by IoT applications that use ML models and need their prediction response as quickly as possible. In Section 6.1, we focused on answering question Q1 about the latency time of ML models executed either in the edge (subdivided into fog and mist) or in the cloud layer. The experiment showed that the cloud layer has a higher response time than in the edge layer, considering both fog and mist nodes, running the same ML model. The time to send and receive data through the internet from IoT devices to the cloud server is very high for low-latency applications. However, the experiment showed that the total processing time increases using the same model when sending more data in batches. In the edge layer, increasing the window size has more impact than the cloud layer, causing the processing time to increase significantly.

In Section 6.2, we focused on answering questions Q2 and Q4 using a single sensor scenario with the proposed architecture in the edge layer. We analyzed the accuracy of lightweight ML models, the accuracy of the consensus algorithm using a set of previously evaluated models, and finally, the processing time of the consensus algorithm. As shown in the results, using lightweight ML models has the following advantages: the possibility of training them on edge nodes without having to send data to the cloud and fast processing time for prediction. However, if the selected models were used independently and separately, the response for the system would have a low accuracy, since the models were not able to find the correct pattern of anomalies. On the other hand, when they were applied in the architecture using the consensus algorithm, the final accuracy of the system improved, and our final model could capture the anomalies, making our system suitable for the tested scenario. In addition to the high accuracy for our distribution algorithm, the processing time at the edge layer using the selected embedded hardware proved to be sufficiently low and suitable for many low-latency IoT applications.

Finally, in Section 6.3, we focused on answering questions Q3 and Q4 using a multiple-sensor scenario. The goal was to demonstrate the feasibility of the architecture using the consensus algorithm in a scenario where the number of sensors/features is high. Our results showed that the use of a single lightweight model in the edge layer in a scenario with multiple sensors proved to be inefficient, even using different sensor/feature combinations. However, our solution has proved to be highly efficient when using the consensus algorithm with the RFE feature selection algorithm. By selecting a subset of combinations of different features and models, the distributed approach was able to detect anomalies without having to use all the sensors. The experiment concluded that even in the face of a scenario with multiple sensors, the consensus algorithm applied with dimensional reduction techniques can increase the final accuracy of the system. We also showed that the processing time of the consensus algorithm for the small subset of features and model combinations was suitable for the low latency IoT applications.

7 CONCLUSION

Most IoT applications generate a large amount of data in a short time, and some of them need to use ML models to transform raw data into useful information. For most of these applications, the data generated by IoT devices is sent to the cloud, which has specialized services that can process complex ML models that lead to better accuracy. However, the high latency involved in data transfers between IoT devices and cloud servers makes this model unfeasible for IoT applications that require near real-time response. In this context, Edge Computing appears as a viable solution to solve this problem, allowing data to be processed near the IoT devices. Nonetheless, the hardware used in the edge layer is not capable of processing heavy ML models. Thus, the only alternative is to use lightweight models in the edge, but these models tend to have poor accuracy compared to more complex ones.

To allow the use of Edge Computing with IoT applications that require low response times, we proposed a new architecture that has a new way of distributing lightweight models and thus increasing the accuracy of the system. The architecture supports many kinds of scenarios and different ML models. We implemented our proposed architecture with a new distributed learning algorithm in different scenarios, and we evaluated its performance, proving that it can work in the edge layer meeting the requirements imposed by IoT applications. The experiments demonstrated that it is possible to process the ML models faster than processing in the cloud, with results comparable to more complex models. By using such lightweight models, our architecture has significant advantages in using only the edge layer since the number of communications with the cloud layer is significantly reduced. Overall, the main benefits of the proposed approach are: (i) the possibility of keeping the data closer to the source, ensuring protection and security for them; (ii) the possibility of training the models locally without the need for cloud servers; and (iii) great reduction in communications with the cloud.

7.1 FUTURE WORK

Due to limited time and a lack of more resources (hardware and data), we restricted the scope of the dissertation to evaluate a common IoT application: anomaly detection in Industrial scenarios. This restriction led us not to investigate some research aspects. In this context, future work can take the following directions:

- **Deal with malicious and Byzantine sensors.** The proposed scenarios were chosen to focus on a very specific configuration of sensors. However, in many real-world IoT applications, Byzantine effects and malicious sensors can hinder the consensus algorithm and reduce system accuracy. Thus, it would be interesting to investigate

how the proposed approach could be improved to consider the effects of malicious and Bizantine sensors.

- **Distributed learning using GPUs.** We trained the models and carried out ML inferences using only the CPUs available in the IoT devices. It would be interesting to investigate the performance benefits of using embedded hardware that feature accelerators such as GPUs or TPUs. By leveraging these accelerators new algorithms such as deep learning could be tested against the consensus algorithm.
- **Intelligent Control of Industrial Equipment** We focused on studying cases of anomaly detection in industrial machinery (a.k.a. predictive maintenance). However, new concepts of automation and intelligent control are present in the Industry 4.0, which use ML models to correct and adapt the state of the industrial equipment. The consensus algorithm could be directly applied to intelligent controllers in a distributed system composed by industrial machinery using the TinyMLOps to automate the entire workflow.
- **New scenarios** The proposed architecture is not limited to just the cases we studied in the present work. It is possible to explore other IoT scenarios, such as autonomous cars, smart cities, and smart grids for example.

7.2 PUBLICATIONS

The research work presented in this dissertation was partially reported in the XII Brazilian Symposium on Computing Systems Engineering (SBESC). The authors will publish a new extension of the work in the Springer Journal of Design Automation for Embedded Systems. More information about the published paper can be found below:

- FIDELIS, Samuel Amico; CASTRO, Márcia; SIQUEIRA, Frank. **Distributed Learning using Consensus on Edge AI**. In: 2022 XII Brazilian Symposium on Computing Systems Engineering (SBESC). IEEE, 2022. p. 1-8. Available at: *https://ieeexplore.ieee.org/abstract/document/9965153*

BIBLIOGRAPHY

ADI, E. et al. Machine learning and data analytics for the iot. **Neural computing and applications**, Springer, v. 32, n. 20, p. 16205–16233, 2020.

ALGHAMDI, R.; BELLAICHE, M. A deep intrusion detection system in lambda architecture based on edge cloud computing for iot. In: IEEE. **2021 4th International Conference on Artificial Intelligence and Big Data (ICAIBD)**. [S.l.], 2021. p. 561–566.

ALMARDENY, Y.; BOUJNAH, N.; CLEARY, F. A novel outlier detection method for multivariate data. **IEEE Transactions on Knowledge and Data Engineering**, IEEE, 2020.

ARNING, A.; AGRAWAL, R.; RAGHAVAN, P. A linear method for deviation detection in large databases. In: **KDD**. [S.l.: s.n.], 1996. v. 1141, n. 50, p. 972–981.

ASIF-UR-RAHMAN, M. et al. Toward a heterogeneous mist, fog, and cloud-based framework for the internet of healthcare things. **IEEE Internet of Things Journal**, IEEE, v. 6, n. 3, p. 4049–4062, 2018.

AZIMI, I. et al. Empowering healthcare iot systems with hierarchical edge-based deep learning. In: **Proceedings of the 2018 IEEE/ACM International Conference on Connected Health: Applications, Systems and Engineering Technologies**. [S.l.: s.n.], 2018. p. 63–68.

BANSAL, M.; CHANA, I.; CLARKE, S. A survey on iot big data: current status, 13 v's challenges, and future directions. **ACM Computing Surveys (CSUR)**, ACM New York, NY, USA, v. 53, n. 6, p. 1–59, 2020.

BASSETTI, E.; PANIZZI, E. Earthquake detection at the edge: Iot crowdsensing network. **Information**, MDPI, v. 13, n. 4, p. 195, 2022.

BELLAVISTA, P. et al. Machine learning for predictive diagnostics at the edge: An iiot practical example. In: IEEE. **ICC 2020-2020 IEEE International Conference on Communications (ICC)**. [S.l.], 2020. p. 1–7.

BHARDWAJ, R.; DATTA, D. Consensus algorithm. In: **Decentralised Internet of Things**. [S.l.]: Springer, 2020. p. 91–107.

BONOMI, F. et al. Fog computing and its role in the internet of things. In: **Proceedings of the first edition of the MCC workshop on Mobile cloud computing**. [S.l.: s.n.], 2012. p. 13–16.

BREUNIG, M. M. et al. Lof: identifying density-based local outliers. In: **Proceedings of the 2000 ACM SIGMOD international conference on Management of data**. [S.l.: s.n.], 2000. p. 93–104.

BRIK, B. et al. Towards predicting system disruption in industry 4.0: Machine learning-based approach. **Procedia computer science**, Elsevier, v. 151, p. 667–674, 2019.

BURKOV, A. **Machine learning engineering**. [S.l.]: True Positive Incorporated Montreal, QC, Canada, 2020. v. 1.

BUTCHER, B.; SMITH, B. J. **Feature Engineering and Selection: A Practical Approach for Predictive Models: by Max Kuhn and Kjell Johnson. Boca Raton, FL: Chapman & Hall/CRC Press, 2019, xv+ 297 pp., ISBN: 978-1-13-807922-9.** [S.l.]: Taylor & Francis, 2020.

BYERS, C. C. Architectural imperatives for fog computing: Use cases, requirements, and architectural techniques for fog-enabled iot networks. **IEEE Communications Magazine**, IEEE, v. 55, n. 8, p. 14–20, 2017.

CAO, J.; ZHANG, Q.; SHI, W. Challenges and opportunities in edge computing. In: **Edge Computing: A Primer**. [S.l.]: Springer, 2018. p. 59–70.

CHELLAPPA, R. Intermediaries in cloud-computing: A new computing paradigm. In: **INFORMS Annual Meeting, Dallas**. [S.l.: s.n.], 1997. p. 26–29.

CHEN, C.-H.; LIU, C.-T. Person re-identification microservice over artificial intelligence internet of things edge computing gateway. **Electronics**, MDPI, v. 10, n. 18, p. 2264, 2021.

CHIANG, M.; ZHANG, T. Fog and iot: An overview of research opportunities. **IEEE Internet of things journal**, IEEE, v. 3, n. 6, p. 854–864, 2016.

CHOPRA, U.; THAKUR, N.; SHARMA, L. Cloud computing: Elementary threats and embellishing countermeasures for data security. **globe**, v. 1, n. 2, p. 10–25, 2019.

CISCO, U. Cisco annual internet report (2018–2023) white paper. **Cisco: San Jose, CA, USA**, 2020.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. Distributed systems: Concepts and design edition 3. **System**, v. 2, n. 11, p. 15.

DIVYA, V.; SRI, R. L. Docker-based intelligent fall detection using edge-fog cloud infrastructure. **IEEE Internet of Things Journal**, IEEE, v. 8, n. 10, p. 8133–8144, 2020.

DONNO, M. D.; TANGE, K.; DRAGONI, N. Foundations and evolution of modern computing paradigms: Cloud, iot, edge, and fog. **Ieee Access**, Ieee, v. 7, p. 150936–150948, 2019.

DYBÅ, T.; DINGSØYR, T. Strength of evidence in systematic reviews in software engineering. In: **Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement**. [S.l.: s.n.], 2008. p. 178–187.

ERIKSEN, M. B.; FRANDSEN, T. F. The impact of patient, intervention, comparison, outcome (pico) as a search strategy tool on literature search quality: a systematic review. **Journal of the Medical Library Association: JMLA**, Medical Library Association, v. 106, n. 4, p. 420, 2018.

FIROUZI, F.; FARAHANI, B.; MARINŠEK, A. The convergence and interplay of edge, fog, and cloud in the ai-driven internet of things (iot). **Information Systems**, Elsevier, p. 101840, 2021.

GEETHA, V. et al. Deployment of computer vision application on edge platform. In: IEEE. **2021 IEEE 18th India Council International Conference (INDICON)**. [S.l.], 2021. p. 1–8.

GEORGOPOULOS, L.; HASLER, M. Distributed machine learning in networks by consensus. **Neurocomputing**, Elsevier, v. 124, p. 2–12, 2014.

GHAZAL, M. et al. Cloud-based monitoring of thermal anomalies in industrial environments using ai and the internet of robotic things. **Sensors**, MDPI, v. 20, n. 21, p. 6348, 2020.

GOKHALE, P.; BHAT, O.; BHAT, S. Introduction to iot. **International Advanced Research Journal in Science, Engineering and Technology**, v. 5, n. 1, p. 41–44, 2018.

GREENGARD, S. Ai on edge. **Communications of the ACM**, ACM New York, NY, USA, v. 63, n. 9, p. 18–20, 2020.

HARDIN, J.; ROCKE, D. M. Outlier detection in the multiple cluster setting using the minimum covariance determinant estimator. **Computational Statistics & Data Analysis**, Elsevier, v. 44, n. 4, p. 625–638, 2004.

HENSH, F.; GUPTA, M.; NENE, M. J. Mist-edge-cloud (mec) computing: An integrated computing architecture. In: IEEE. **2021 Second International Conference on Electronics and Sustainable Communication Systems (ICESC)**. [S.l.], 2021. p. 1035–1040.

HOSSIN, M.; SULAIMAN, M. N. A review on evaluation metrics for data classification evaluations. **International journal of data mining & knowledge management process**, Academy & Industry Research Collaboration Center (AIRCC), v. 5, n. 2, p. 1, 2015.

JAIN, K.; MOHAPATRA, S. Taxonomy of edge computing: Challenges, opportunities, and data reduction methods. In: **Edge Computing**. [S.l.]: Springer, 2019. p. 51–69.

JANSSENS, J. et al. Stochastic outlier selection. **Tilburg centre for Creative Computing, techreport**, v. 1, p. 2012, 2012.

JINDAL, M.; GUPTA, J.; BHUSHAN, B. Machine learning methods for iot and their future applications. In: IEEE. **2019 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)**. [S.l.], 2019. p. 430–434.

KANG, Y. et al. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. **ACM SIGARCH Computer Architecture News**, ACM New York, NY, USA, v. 45, n. 1, p. 615–629, 2017.

KAUR, N.; SOOD, S. K. Efficient resource management system based on 4vs of big data streams. **Big data research**, Elsevier, v. 9, p. 98–106, 2017.

KEELE, S. et al. **Guidelines for performing systematic literature reviews in software engineering**. [S.l.], 2007.

KELLERMEYER, L.; HARNKE, B.; KNIGHT, S. Covidence and rayyan. **Journal of the Medical Library Association: JMLA**, Medical Library Association, v. 106, n. 4, p. 580, 2018.

KHAN, W. Z. et al. Edge computing: A survey. **Future Generation Computer Systems**, Elsevier, v. 97, p. 219–235, 2019.

KHARE, S.; TOTARO, M. Big data in iot. In: IEEE. **2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)**. [S.l.], 2019. p. 1–7.

KITCHENHAM, B. et al. Protocol for systematic review of within-and cross-company estimation models. Unpublished, 2017.

KLAS, G. I. Fog computing and mobile edge cloud gain momentum open fog consortium, etsi mec and cloudlets. **Google Scholar**, v. 1, n. 1, p. 1–13, 2015.

KREUZBERGER, D.; KÜHL, N.; HIRSCHL, S. Machine learning operations (mlops): Overview, definition, and architecture. **IEEE Access**, IEEE, 2023.

KRIEGEL, H.-P.; SCHUBERT, M.; ZIMEK, A. Angle-based outlier detection in high-dimensional data. In: **Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining**. [S.l.: s.n.], 2008. p. 444–452.

KUBIAK, K.; DEC, G.; STADNICKA, D. Possible applications of edge computing in the manufacturing industry—systematic literature review. **Sensors**, MDPI, v. 22, n. 7, p. 2445, 2022.

LAVIN, A.; AHMAD, S. Evaluating real-time anomaly detection algorithms–the numenta anomaly benchmark. In: IEEE. **2015 IEEE 14th international conference on machine learning and applications (ICMLA)**. [S.l.], 2015. p. 38–44.

LEE, Y.-L.; TSUNG, P.-K.; WU, M. Techology trend of edge ai. In: IEEE. **2018 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)**. [S.l.], 2018. p. 1–2.

LI, E. et al. Edge ai: On-demand accelerating deep neural network inference via edge computing. **IEEE Transactions on Wireless Communications**, IEEE, v. 19, n. 1, p. 447–457, 2019.

LI, W. et al. Multimodel framework for indoor localization under mobile edge computing environment. **IEEE Internet of Things Journal**, IEEE, v. 6, n. 3, p. 4844–4853, 2018.

LI, Z. et al. Copod: copula-based outlier detection. In: IEEE. **2020 IEEE International Conference on Data Mining (ICDM)**. [S.l.], 2020. p. 1118–1123.

LIN, J. The lambda and the kappa. **IEEE Internet Computing**, IEEE Computer Society, v. 21, n. 05, p. 60–66, 2017.

LIU, F. et al. A survey on edge computing systems and tools. **Proceedings of the IEEE**, IEEE, v. 107, n. 8, p. 1537–1562, 2019.

LIU, F. T.; TING, K. M.; ZHOU, Z.-H. Isolation forest. In: IEEE. **2008 eighth ieee international conference on data mining**. [S.l.], 2008. p. 413–422.

MATT, C. Fog computing. **Business & information systems engineering**, Springer, v. 60, n. 4, p. 351–355, 2018.

MAXWELL, J. Understanding and validity in qualitative research. **Harvard educational review**, Harvard Education Publishing Group, v. 62, n. 3, p. 279–301, 1992.

MELL, P.; GRANCE, T. et al. The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National . . . , 2011.

MIRASHE, S. P.; KALYANKAR, N. V. Cloud computing. **arXiv preprint arXiv:1003.4074**, 2010.

MUNIR, A. et al. Artificial intelligence and data fusion at the edge. **IEEE Aerospace and Electronic Systems Magazine**, IEEE, v. 36, n. 7, p. 62–78, 2021.

MURSHED, M. S. et al. Machine learning at the network edge: A survey. **ACM Computing Surveys (CSUR)**, ACM New York, NY, v. 54, n. 8, p. 1–37, 2021.

NAIN, G.; PATTANAIK, K.; SHARMA, G. Towards edge computing in intelligent manufacturing: Past, present and future. **Journal of Manufacturing Systems**, Elsevier, v. 62, p. 588–611, 2022.

NATESHA, B.; GUDDETI, R. M. R. Fog-based intelligent machine malfunction monitoring system for industry 4.0. **IEEE Transactions on Industrial Informatics**, IEEE, v. 17, n. 12, p. 7923–7932, 2021.

NGUYEN, D. C. et al. Federated learning meets blockchain in edge computing: Opportunities and challenges. **IEEE Internet of Things Journal**, IEEE, v. 8, n. 16, p. 12806–12825, 2021.

OGORE, M. M.; NKURIKIYEYEZU, K.; NSENGA, J. Offline prediction of cholera in rural communal tap waters using edge ai inference. In: IEEE. **2021 IEEE Globecom Workshops (GC Wkshps)**. [S.l.], 2021. p. 1–6.

PARTO, M.; SALDANA, C.; KURFESS, T. A novel three-layer iot architecture for shared, private, scalable, and real-time machine learning from ubiquitous cyber-physical systems. **Procedia manufacturing**, Elsevier, v. 48, p. 959–967, 2020.

PETEIRO-BARRAL, D.; GUIJARRO-BERDIÑAS, B. A survey of methods for distributed machine learning. **Progress in Artificial Intelligence**, Springer, v. 2, n. 1, p. 1–11, 2013.

PI, R. Raspberry pi 3 model b. **online].(https://www. raspberrypi. org**, 2015.

PONCINELLI, C. F. et al. A systematic literature review on distributed machine learning in edge computing. **Sensors**, MDPI, v. 22, n. 7, p. 2665, 2022.

PRIYABHASHANA, H.; JAYASENA, K. Data analytics with deep neural networks in fog computing using tensorflow and google cloud platform. In: IEEE. **2019 14th Conference on Industrial and Information Systems (ICIIS)**. [S.l.], 2019. p. 34–39.

QIU, T. et al. Edge computing in industrial internet of things: Architecture, advances and challenges. **IEEE Communications Surveys & Tutorials**, IEEE, v. 22, n. 4, p. 2462–2488, 2020.

RAMASWAMY, S.; RASTOGI, R.; SHIM, K. Efficient algorithms for mining outliers from large data sets. In: **Proceedings of the 2000 ACM SIGMOD international conference on Management of data**. [S.l.: s.n.], 2000. p. 427–438.

RAUSCH, T.; DUSTDAR, S. Edge intelligence: The convergence of humans, things, and ai. In: IEEE. **2019 IEEE International Conference on Cloud Engineering (IC2E)**. [S.l.], 2019. p. 86–96.

ROSENDO, D. et al. Distributed intelligence on the edge-to-cloud continuum: A systematic literature review. **Journal of Parallel and Distributed Computing**, Elsevier, 2022.

SAHI, M.; SONI, M.; AULUCK, N. An intrusion detection system on fog architecture. In: IEEE. **2021 IEEE 18th International Conference on Mobile Ad Hoc and Smart Systems (MASS)**. [S.l.], 2021. p. 591–596.

SATOH, I. A framework for data processing at the edges of networks. In: SPRINGER. **International Conference on Database and Expert Systems Applications**. [S.l.], 2013. p. 304–318.

SATYANARAYANAN, M. et al. The case for vm-based cloudlets in mobile computing. **IEEE pervasive Computing**, IEEE, v. 8, n. 4, p. 14–23, 2009.

SAWALHA, S.; AL-NAYMAT, G. Towards an efficient big data management schema for iot. **Journal of King Saud University-Computer and Information Sciences**, Elsevier, 2021.

SCHMIDHUBER, J. Deep learning in neural networks: An overview. **Neural networks**, Elsevier, v. 61, p. 85–117, 2015.

SCHÖLKOPF, B. et al. Estimating the support of a high-dimensional distribution. **Neural computation**, MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info . . . , v. 13, n. 7, p. 1443–1471, 2001.

SENGUPTA, K.; SRIVASTAVA, P. R. Hrnet: Ai-on-edge for mask detection and social distancing calculation. **SN Computer Science**, Springer, v. 3, n. 2, p. 1–15, 2022.

SHA, K. et al. A survey of edge computing-based designs for iot security. **Digital Communications and Networks**, Elsevier, v. 6, n. 2, p. 195–202, 2020.

SHAHID, H. et al. Machine learning-based mist computing enabled internet of battlefield things. **ACM Transactions on Internet Technology (TOIT)**, ACM New York, NY, v. 21, n. 4, p. 1–26, 2021.

SHALEV-SHWARTZ, S.; BEN-DAVID, S. **Understanding machine learning: From theory to algorithms**. [S.l.]: Cambridge university press, 2014.

SHANTHAMALLU, U. S. et al. A brief survey of machine learning methods and their sensor and iot applications. In: IEEE. **2017 8th International Conference on Information, Intelligence, Systems & Applications (IISA)**. [S.l.], 2017. p. 1–8.

SHI, W. et al. Edge computing: Vision and challenges. **IEEE internet of things journal**, IEEE, v. 3, n. 5, p. 637–646, 2016.

SHYU, M.-L. et al. **A novel anomaly detection scheme based on principal component classifier**. [S.l.], 2003.

SU, W. et al. Ai on the edge: a comprehensive review. **Artificial Intelligence Review**, Springer, p. 1–59, 2022.

SUNYAEV, A. The internet of things. In: **Internet Computing**. [S.l.]: Springer, 2020. p. 301–337.

SZEPESVÁRI, C. Algorithms for reinforcement learning. **Synthesis lectures on artificial intelligence and machine learning**, Morgan & Claypool Publishers, v. 4, n. 1, p. 1–103, 2010.

TAHERKORDI, A.; ELIASSEN, F.; HORN, G. From iot big data to iot big services. In: **Proceedings of the Symposium on Applied Computing**. [S.l.: s.n.], 2017. p. 485–491.

TANG, J. et al. Enhancing effectiveness of outlier detections for low density patterns. In: SPRINGER. **Pacific-Asia conference on knowledge discovery and data mining**. [S.l.], 2002. p. 535–548.

TAURION, C. **Cloud computing-computação em nuvem**. [S.l.]: Brasport, 2009.

VAQUERO, L. M.; RODERO-MERINO, L. Finding your way in the fog: Towards a comprehensive definition of fog computing. **ACM SIGCOMM computer communication Review**, ACM New York, NY, USA, v. 44, n. 5, p. 27–32, 2014.

VERBRAEKEN, J. et al. A survey on distributed machine learning. **Acm computing surveys (csur)**, ACM New York, NY, USA, v. 53, n. 2, p. 1–33, 2020.

WANG, X. et al. **Edge AI: Convergence of edge computing and artificial intelligence**. [S.l.]: Springer, 2020.

WANG, X. et al. Distributed inference for linear support vector machine. **Journal of machine learning research**, v. 20, 2019.

WEI, J.; HAN, J.; CAO, S. Satellite iot edge intelligent computing: A research on architecture. **Electronics**, MDPI, v. 8, n. 11, p. 1247, 2019.

WHITE, G.; CLARKE, S. Urban intelligence with deep edges. **IEEE Access**, IEEE, v. 8, p. 7518–7530, 2020.

WOHLIN, C. et al. Successful combination of database search and snowballing for identification of primary studies in systematic literature studies. **Information and Software Technology**, Elsevier, v. 147, p. 106908, 2022.

YU, W. et al. A survey on the edge computing for the internet of things. **IEEE access**, IEEE, v. 6, p. 6900–6919, 2017.

ZHAO, Y.; HRYNIEWICKI, M. K. Xgbod: improving supervised outlier detection with unsupervised representation learning. In: IEEE. **2018 International Joint Conference on Neural Networks (IJCNN)**. [S.l.], 2018. p. 1–8.

ZSCHÖRNIG, T. et al. A fog-based multi-purpose internet of things analytics platform. **SN Computer Science**, Springer, v. 3, n. 3, p. 1–20, 2022.

8 APPENDICES

8.1 SYSTEMATIC REVIEW EXECUTION

8.1.1 Search string and search sources

The search phrase was created using the PICO (ERIKSEN; FRANDSEN, 2018) technique defined in Table 17 and the elaboration process was based on (KITCHENHAM et al., 2017) , which defines four steps for the elaboration of the string:

- **Step 1.** Derive major terms from the questions by identifying the population, intervention and outcome;
- **Step 2.** Identify alternative spellings and synonyms for major terms;
- **Step 3.** Use the Boolean OR to incorporate alternative spellings and synonyms;
- **Step 4.** Use the Boolean AND to link the major terms from population, intervention and outcome.

Table 17 – PICO method for Edge AI

PICO	Descriptions
Population	IoT Applications including: sensors, video streaming, healthcare etc
Intervetion	Architectures or framework models
Comparison	Latency, accuracy and energy efficiency of the nodes
Outcomes	Usability and Efficiency to ensure QoS for the IoT applications

The keywords used in the search queries are: IoT, edge, fog, mist, artificial intelligence, machine learning, Edge AI and architecture. Therefore, the search string in a generic format is:

(“IoT" OR “internet of things") AND (“edge" OR “fog" OR “mist") AND “edge ai" AND “architecture" AND (“machine learning" OR “ai" OR “artificial intelligence")

The search string was executed in the following sources: Science Direct, IEEEXplore, ACM Digital Library, Scopus and Springler. No filter was previously applied when performing the search in each of the sources. A total of 5.239 papers were found after running the search string on the sources. Table 18 shows the search query for each source.

Table 18 – Search string for each source

Source	Search String	Result
Science Direct	**Title, abstract, keywords:** (“IoT" AND (“edge" OR “fog" OR “mist") AND “architecture" AND (“machine learning" OR “artificial intelligence" OR “AI" OR "edge AI"))	422
IEEEXplore	(**“All Metadata"**: “IoT" OR “Internet of Things") AND (**“All Metadata"**: “edge" OR “mist" OR “fog") (**"All Metadata"**: “architecture" OR "architecture model") AND (**"All Metadata"**: “edge ai" OR “machine learning" OR “AI" OR “artificial intelligence")	737
ACM Digital Library	**Abstract**:(“IoT" OR “Internet of Things" AND (“edge" OR “fog" OR “mist") AND “architecture" AND (“edge ai" OR “machine learning" OR “AI" OR “artificial intelligence")) AND **Keyword**:(“IoT" AND (“edge" OR “fog" OR “mist") AND (edge ai OR "machine learning" OR artificial intelligence))	88
Scopus	**TITLE-ABS-KEY** (("IoT" OR "Internet of Things") AND ("edge" OR "fog" OR "mist") AND "architecture" AND ("edge ai" OR "machine learning" OR "ai" OR "artificial intelligence"))	274
Springler Link	("IoT" OR "internet of things") AND ("edge" OR "mist" OR "fog") AND "architecture" AND "edge ai"	74

8.1.2 Inclusion and Exclusion criteria

The inclusion and exclusion criteria used in this systematic review are shown in Table 19 for inclusion criteria and Table 20 for exclusion criteria. For a article to be excluded, it is necessary to meet only 1 exclusion criterion. For the process of including an article, it is necessary that it can meet all the criteria defined in Table 19.

Table 19 – Inclusion Criteria

Criteria
Article considers at least one of the following edge layers: Fog/Edge or Mist
Article describes an architecture for Edge AI
Article uses at least 1 framework in its architecture
Article was published, between, 2015 and 2022 (inclusive)
Article that describe and use some machine learning service or technique
Documents published in English.

Table 20 – Exclusion Criteria

Criteria
Article with restricted access to the full text
Duplicate (identical) articles found in more than one search base
Article that does not fit into an IoT application
Article that does not use any Machine Learning or AI services in the Edge layer

8.1.3 Quality assessment

According to (DYBÅ; DINGSØYR, 2008), the quality assessment of primary studies can be given by eleven criteria, which are summarized in 4 main aspects:

- **Reporting:** related to the quality of the reporting of a study's rationale, aims, and context;
  - Is the paper based on research (or is it merely a "lessons learned" report based on expert opinion)?
  - Is there a clear statement of the aims of the research?
  - Is there an adequate description of the context in which the research was carried out?
- **Rigor:** related to the rigor of the research methods employed to establish the validity of data collection tools and the analysis methods;
  - Was the research design appropriate to address the aims of the research?
  - Was the recruitment strategy appropriate to the aims of the research?
  - Was there a control group with which to compare treatments?
  - Was the data collected in a way that addressed the research issue?
  - Was the data analysis sufficiently rigorous?
- **Credibility:** related to the assessment of the credibility of the study methods for ensuring that the findings were valid and meaningful;
  - Has the relationship between researcher and participants been adequately considered?

  - Is there a clear statement of findings?

- **Relevance** related to the assessment of the relevance of the study for the software industry at large and the research community.

  - Is the study of value for research or practice?

The quality of a chosen primary work is given by the sum of the score obtained for each of the 11 questions of the quality criterion, being: 0.0 if the article does not satisfy the question, 0.5 if the article partially satisfies the question and 1.0 if the article completely satisfies the question. Finally, the quality of the study is defined as the mean of these scores. In the present work, quality assessment will not be used, only studied and practiced in some articles for didactic purposes.

8.2 EXECUTION

Using the protocol described in the previous section, the search and extraction of articles was performed in the databases described in Section 8.1.1 using the search strings shown in Table 18 on July 2, 2022. The software Rayyan (KELLERMEYER; HARNKE; KNIGHT, 2018) was used to manage the articles in each step of the execution process.

8.2.1 Selection process

The selection process of the primary works can be seen in Figure 38 and has the following eight stages:

- **Step 1.** The search string will be adjusted according to the formatting of each search engine, and in the sequence executed;
- **Step 2.** Identification and removal of duplicate articles
- **Step 3.** Each primary work will go through a filter pipeline. Starting with the analysis of the Title, Abstract and keywords, where the filtering by the exclusion and inclusion criteria will be done;
- **Step 4.** Articles that were not excluded were classified into Maybe and Included.
- **Step 5.** If the article was classified as Maybe another analysis of the Title, Abstract and keywords was performed. If the article was classified as Included another filtering step will be carried out on the entire text, where the inclusion and exclusion criteria will be analyzed again;
- **Step 6.** If the article was classified as Maybe and not excluded in the previously step another filtering step will be carried out on the entire text, where the inclusion and exclusion criteria will be analyzed again;

- **Step 7.** A quality assessment and data and information extraction will be carried out on each work, however the quality assessment is not considered as an exclusion filtering step.
- **Step 8.** The last step consists of applying the snowball technique, i.e. the using of the reference list of a paper or the citations to the paper to identify additional papers (WOHLIN et al., 2022). Only the Backward Snowballing technique will be applied.

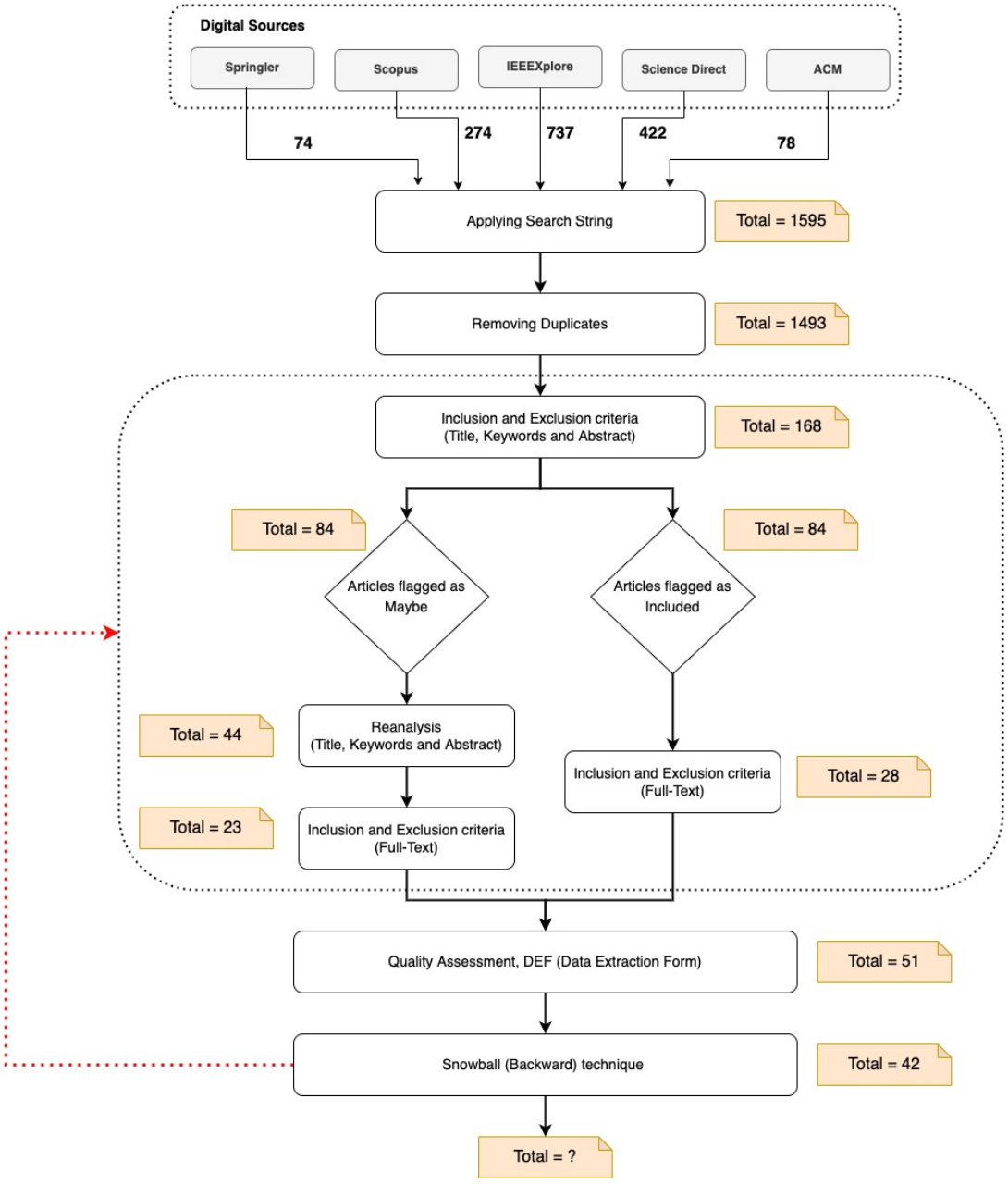

Figure 38 – Selection Process

In the stage 7, the data extraction process was carried out in the 51 articles, but during this stage nine articles were discarded because they did not meet all the inclusion

criteria and were not able to answer the research questions defined in section 3.1.2. So, only 42 articles are contained in the Data Extraction Form (DEF).

An example of the Data Extraction Form adopted in this paper can be seen in Table 21. The DEF was elaborated with the aim of answering the research questions listed in Section 3.1.2. The DEF can be visualised on https://docs.google.com/spreadsheets/d/e/2PACX-1vRU-sMdzefTqO5COwp8dEa_dHb3t2eYN2-8LqlBNu7BMnF3r1WExwhS-0oTHwaddFAi5xInA pubhtml.

8.3 VALIDITY THREATS

According to Maxwell (1992), a systematic mapping presents 4 different types of validity that must be described. The first is **Descriptive validity** and it is directly related to what and how the researcher described his observations, that is, he is not making up or distorting from the observations made. The other validity categories are dependent on this primary aspect according to Maxwell. To mitigate the threats in this validity category, the present work uses the following contingency measures: a) The DEF is organized to clearly and objectively categorize the primary studies; b) The use of consistent categories align with the research questions and the search string proposed.

The second category **Interpretive validity** is related to the researcher's bias, how he presented and wrote it from his perspective and the conclusions reached. To mitigate this, experts in the IoT field, like other researchers, are needed to review the results and interpretations taken from this mapping review.

The third category is **Theoretical validity**, and is related to the concepts and relationships of the theories, that is, the concepts related to how the researcher built his model that was applied to the phenomena studied, while the second describes the validity of how the concepts are placed together. Examples of theoretical validity and how these were mitigated are described below: a) Inappropriate or incomplete search terms in automatic search to model the research questions: The string was revised by a domain expert and a snowballing technique will be performed until an iteration no longer yields any additional included work; b) Researcher bias in the inclusion and exclusion criteria: A protocol with detailed guidelines was developed before the selection phase of the study, in addition to carrying out a review after the execution of the criteria with the help of an expert; c) The set of primary articles returned by the search string does not match the total found in the literature: explicit informed date when the extraction of primary works was carried out together with the realization of the forward snowballing technique.

The last category is **Generalizability validity** which refers to the ability of how the results of a particular situation or population can be extended to other cases, people and times. Where a possible validity in this category can be defined in the scope of work: The inability to generalize the results due to the wide range of possibilities that an architecture can be built upon. That threat was mitigated through a categorization, where we create a definition of layers and context of how the architecture can be used, however extreme cases were found and described.

Table 21 – Data extraction form (DEF)

DEF ID	Description	Example	RQ ID
Title	Article title	Earthquake Detection at the Edge: IoT Crowdsensing Network	N/A
Author	Author‘s name	Enrico Bassetti, Emanuele Panizzi	N/A
Journal	Journal name	Information (Switzerland)	N/A
Digital library	Source from where the article was extract	IEEE	N/A
Year	Publication year	2022	N/A
Language	language the article was written	English	N/A
Type of the Layer	Where the edge computing was done	Edge layer	RQ1,RQ21
Layers of the Architecture	Where the ML/AI tasks and the data storage was done	Edge-Cloud	RQ1,RQ21,RQ22
Evaluation Context	In which IoT application the architecture was proposed	Smart Home	RQ4,RQ3
AI Library / Framework / Model	What ML/AI library,model or framework was proposed and used in the Edge AI	Crowdquake CRNN - Tensorflow	RQ23
Purpose of the Edge AI	What is the goal of the proposed Edge AI layer and what challenges the architecture solved	The Edge AI layer moves the environment and process information from nearby to detect earthquakes locally. The approach tolerates multiple node faults and partial network disruption and keeps all data locally, enhancing privacy	RQ23,RQ3,RQ4
AI Library / Framework / Model	What ML/AI library,model or framework was proposed and used in the Edge AI	Crowdquake CRNN - Tensorflow	RQ23
Results of Edge AI	The proposed Edge AI architecture ensure the QoS of the IoT application	The Edge AI architecture showed the lowest mean (0.015 s) and median (0.015 s) latency compared to the only-Cloud architecture (average latency of 0.603 s with a median of 0.618 s.)	RQ22
Edge Hardware	Type o hardware used in the Edge/Fog/Mist layer to perform the experiment	Raspberry Pi +3B	RQ1