UNIVERSITÀ DEGLI STUDI
DI TRENTO

DEPARTMENT OF INFORMATION ENGINEERING AND COMPUTER SCIENCE
ICT International Doctoral School

DOCTORAL THESIS

From Edge Computing to Edge Intelligence:

exploring novel design approaches to intelligent IoT applications

Mattia Antonini

Advisor
Ing. Fabio Antonelli
Fondazione Bruno Kessler

Co-Advisor
Prof. Dr. Massimo Vecchio
Fondazione Bruno Kessler

XXXIII Cycle - June 2021

To my parents, my sister,
and my brother.

Acknowledgments

Every time I reach this point, I realize that many people have deeply contributed to my training as a young scientist and believed in me as a person. To all of you goes my deepest gratitude.
First, I would like to thank my advisor Ing. Fabio Antonelli and my co-advisor Prof. Dr. Massimo Vecchio for their constant and outstanding support, guidance, and belief in me even when everything was going pretty bad. I hope when I will advise my future students to be half good as you were with me.
Second, my gratitude goes to my Ph.D. referees, Prof. Dr. Pietro Ducange and Prof. Dr. Javier Del Ser, for their time spent providing me feedback and improvements to deliver a better version of this thesis.
Following, my gratitude and thanks go to Dr. Fahim Kawsar and the Pervasive System research group at Nokia Bell Labs in Cambridge (UK) for hosting me in summer 2019. I enjoyed discovering how you conduct research in an industrial environment, how to deal with a multidisciplinary team, and how to enjoy time at conferences.
I would like to thank the librarians at Biblioteca Universitaria Centrale (BUC) to provide me feedback to improve the style of this dissertation in the last days before the thesis's deposition.
To all my colleagues and friends at FBK, thanks for your support, friendship, many coffees, and chats we shared in these years. Even if we did not meet so much in the last year, I always feel you near.
To whom this thesis is dedicated, my family. Even if we have been living the darkest chapter of our lives, you gave me all the support and love that I needed to gain this degree. I hope I made you proud of me.
To my old and new friends, you have given me the strength to keep my feet on the path even when giving up was the only choice. I am here because

of you. Thanks from the heart.
Last, but for sure not least, I would like to thank Prof. Alessandro Pettoruti (a.k.a. *il Petto*). You have always been my guide, my friend, and my mentor. I hope I made you proud of me again. Knowing you, I believe so. Miss you.

Mattia Antonini

The work contained in this thesis was partially supported by the AGILE project[1]*, within the Horizon 2020 programme of the European Union, grant number 688088.*

[1] `https://cordis.europa.eu/project/id/688088`

Abstract

The Internet of Things (IoT) has deeply changed how we interact with our world. Today, smart homes, self-driving cars, connected industries, and wearables are just a few mainstream applications where IoT plays the role of enabling technology. When IoT became popular, Cloud Computing was already a mature technology able to deliver the computing resources necessary to execute heavy tasks (*e.g.,* data analytic, storage, AI tasks, etc.) on data coming from IoT devices, thus practitioners started to design and implement their applications exploiting this approach. However, after a hype that lasted for a few years, cloud-centric approaches have started showing some of their main limitations when dealing with the connectivity of many devices with remote endpoints, like high latency, bandwidth usage, big data volumes, reliability, privacy, and so on. At the same time, a few new distributed computing paradigms emerged and gained attention. Among all, Edge Computing allows to shift the execution of applications at the edge of the network (a partition of the network physically close to data-sources) and provides improvement over the Cloud Computing paradigm. Its success has been fostered by new powerful embedded computing devices able to satisfy the everyday-increasing computing requirements of many IoT applications. Given this context, how can next-generation IoT applications take advantage of the opportunity offered by Edge Computing to shift the processing from the cloud toward the data sources and exploit everyday-more-powerful devices? This thesis provides the ingredients and the guidelines for practitioners to foster the migration from cloud-centric to novel distributed design approaches for IoT applications at the edge of the network, addressing the issues of the original approach. This requires the design of the processing pipeline of applications by considering the system requirements and constraints imposed by embedded devices. To make

this process smoother, the transition is split into different steps starting with the off-loading of the processing (including the Artificial Intelligence algorithms) at the edge of the network, then the distribution of computation across multiple edge devices and even closer to data-sources based on system constraints, and, finally, the optimization of the processing pipeline and AI models to efficiently run on target IoT edge devices. Each step has been validated by delivering a real-world IoT application that fully exploits the novel approach. This paradigm shift leads the way toward the design of Edge Intelligence IoT applications that efficiently and reliably execute Artificial Intelligence models at the edge of the network.

Keywords: Edge Computing, Cloud-to-Thing continuum, IoT application, Edge Intelligence, Edge AI, Artificial Intelligence, Embedded Intelligence

Activity Report

Doctoral Candidate	Mattia Antonini
Cycle	XXXIII
Thesis title	From Edge Computing to Edge Intelligence: exploring novel design approaches to intelligente IoT applications
Advisor	Ing. Fabio Antonelli (Fondazione Bruno Kessler, Italy)
Co-advisor	Prof. Dr. Massimo Vecchio (Fondazione Bruno Kessler, Italy)

List of Publications

- Peer-reviewed publications

1. **Mattia Antonini**, Massimo Vecchio, Fabio Antonelli, Pietro Ducange, and Charith Perera. "Smart audio sensors in the internet of things edge for anomaly detection." *IEEE Access 6* (2018): 67594-67610.
   DOI: 10.1109/ACCESS.2018.2877523
2. Luca Davoli, **Mattia Antonini**, and Gianluigi Ferrari. "DIRPL: A RPL-based resource and service discovery algorithm for 6LoWPANS." *Applied Sciences 9*, no. 1 (2019): 33.
   DOI: 10.3390/app9010033
3. Zaffar Haider Janjua, Massimo Vecchio, **Mattia Antonini**, and Fabio Antonelli. "IRESE: An intelligent rare-event detection system using unsupervised learning on the IoT edge." *Engineering Applications of Artificial Intelligence 84* (2019): 41-50.
   DOI: 10.1016/j.engappai.2019.05.011

4. **Mattia Antonini**, Massimo Vecchio, and Fabio Antonelli. "Fog computing architectures: A reference for practitioners." *IEEE Internet of Things Magazine 2*, no. 3 (2019): 19-25. DOI: 10.1109/IOTM.0001.1900029
5. **Mattia Antonini**, Tran Huy Vu, Chulhong Min, Alessandro Montanari, Akhil Mathur, and Fahim Kawsar. "Resource characterisation of personal-scale sensing models on edge accelerators." *In Proceedings of the First International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things*, pp. 49-55. 2019. DOI: 10.1145/3363347.3363363
6. **Mattia Antonini**, Andrea Gaiardo, and Massimo Vecchio. "Meta-NChemo: A meta-heuristic neural-based framework for chemometric analysis." *Applied Soft Computing 97* (2020): 106712. DOI: 10.1016/j.asoc.2020.106712

- Non peer-reviewed publications

1. Raffaele Giaffreda and **Mattia Antonini**. "IoT Technologies and Privacy in a Data-Bloated Society: Where Do We Stand in the Fight to Prepare for the Next Pandemic?." *IEEE Internet of Things Magazine 3*, no. 4 (2020): 2-3 (*Magazine column*). DOI: 10.1109/MIOT.2020.9319621
2. **Mattia Antonini** and Massimo Vecchio. " IoT-Panic: The Cloud Just Disappeared!" *IEEE IoT Newsletter (January 2021).* URL: https://iot.ieee.org/newsletter/january-2021/iot-panic-the-cloud-just-disappeared

Research & study activities

Research period spent abroad

During the second year (A.Y. 2018-2019), I spent more than 4 months (15th June to 20th October 2019) as visiting student in the Pervasive System research group at Nokia Bell Labs in Cambridge (UK), under the supervision

of Dr. Fahim Kawsar.
During my stay, we developed an end-to-end benchmarking platform to understand the performances of AI edge accelerators (more details in). The outcome of the study has been published as a paper in the First International Workshop on Challenges in Artificial Intelligence and Machine Learning for the Internet of Things (AIChallengeIoT 2019).
After this work, I joined the development of a multi-device AI system that was under development by Nokia Bell Labs.

Participation in research projects

- Participation in the research activities of the EU H2020 AGILE project (grant agreement n. 688088);
- Participation in the proposal write-up and research activities of the EU EIT RM SAFEME4MINE project (grant agreement n. 19036);
- Participation in multiple project proposal writing activities;
- Participation in a research project related to Edge AI and multi-device scenarios with the Pervasive Systems group at Nokia Bell Labs (Cambridge, UK);
- Participation in several internal research projects related to condition monitoring, anomaly detection, digital industry, and smart-agriculture with the OpenIoT research unit at FBK Digital Industry center (formerly FBK ICT center, formerly FBK CREATE-NET).

Presentations and seminars

- Paper presentations
  - **Mattia Antonini**, Tran Huy Vu, Chulhong Min, Alessandro Montanari, Akhil Mathur, and Fahim Kawsar. "Resource characterisation of personal-scale sensing models on edge accelerators." *Oral presentation at the First International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things*, New York City, USA (10 November 2019).

- Seminar talks
  - "Introduction to Artificial Intelligence" *Seminar for 4AI - I.T.T. "G. Marconi"*, Rovereto (TN), Italy (16 April 2020).
  - "Resource characterisation of personal-scale sensing models on edge accelerators." *End of internship presentation,* Nokia Bell Labs, Cambridge, UK (19 September 2019);
  - "Smart Audio Sensors in the Internet of Things Edge for Anomaly Detection" *INW2019*, Bormio (SO), Italy (17 January 2019);
  - "Machine Learning on IoT" *Quinto Seminario de Ingenierìa de Software*, Università del Bìo-Bìo, Chillan, Chile (23 November 2018).

Conferences and workshops attended

- INW2019 - Bormio, Italy (16-18 January 2019);
- UBICOMP2019 - London, UK (9-13 September 2019);
- SenSys2019 - New York, USA (10-13 November 2019).

Reviewer services

- IEEE Internet of Things Journal;
- IEEE Transactions on Systems, Man, and Cybernetics: Systems;
- IEEE Internet of Things Magazine;
- Elsevier Engineering Applications of Artificial Intelligence;
- Elsevier Applied Soft Computing;
- Elsevier Soft Computing Letters.

Editorial services

- Co-Guest Editor "Social Sensing Applications in Edge Computing Systems" - *MDPI Sensors Special Issue* (November 2020 – Present);
- Co-Guest Editor "Blockchain-based soft computing tools and methodologies" - *Elsevier Soft Computing Letters Special Issue* (March 2021 – Present).

TPC membership

- SMARTCOMP2020 - Industry Track.

Teaching activities

- Middleware for IoT (Prof. Massimo Vecchio). Laboratory teaching assistant and mentor (Fall 2018, A.Y. 2018-2019) – 14 hours

Contents

List of Tables

List of Figures

Chapter 1

Introduction

We are living in a fast-evolving world that is, every day, more connected and intelligent. The physical world is embracing what we call "the virtual world" enabling new immersive and exciting applications. Thinking about smart speakers empowered with vocal assistants, connected houses, self-driving cars, and many others, we are surrounded by many things that allow us to interact with virtual entities as we interact with other humans, transforming our devices into real commodities. Behind this technological advancement, there is one of the Internet of Things' promises: "*connect anything, anywhere, anytime*".

1.1 The origin of the Internet of Things

The term *The Internet of Things* is not a new expression and its first appearance is dated back to September 1985 when Peter J. Lewis [1] gave a talk at the Congressional Black Caucus Foundation 15^{th} Annual Legislative Weekend. He stated

> *The Internet of Things, or IoT, is the integration of people, processes, and technology with connectable devices and sensors to enable remote monitoring, status, manipulation, and evaluation of trends of such devices.*

In 1999, the second definition of IoT [2] was given by Dr. Kevin Ashton, co-founder of the Auto-ID Laboratory at the Massachusetts Institute of Technology (MIT, Cambridge MA, US), during a presentation at

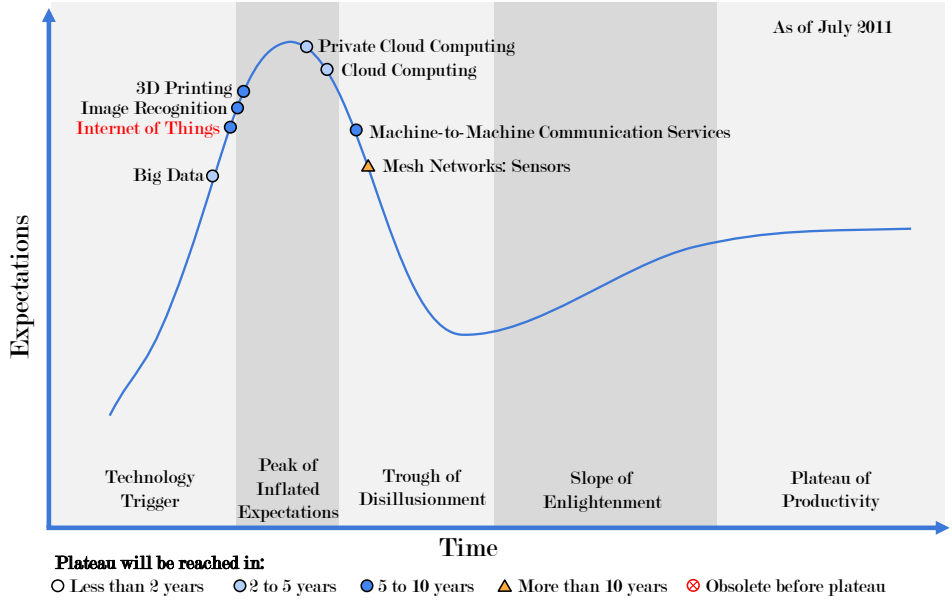
Procter&Gamble to emphasize the power of Radio-Frequency Identification (RFID) technology to support supply chains and tracking goods without the need of humans in-the-loop. He stated:

> *Today computers—and, therefore, the Internet—are almost wholly dependent on human beings for information. [...] Conventional diagrams of the Internet include servers and routers and so on, but they leave out the most numerous and important routers of all: people. The problem is, people have limited time, attention, and accuracy—all of which means they are not very good at capturing data about things in the real world.*
> *And that's a big deal. We're physical, and so is our environment. Our economy, society, and survival aren't based on ideas or information—they're based on things. [...] Ideas and information are important, but things matter much more. Yet today's information technology is so dependent on data originated by people that our computers know more about ideas than things.*

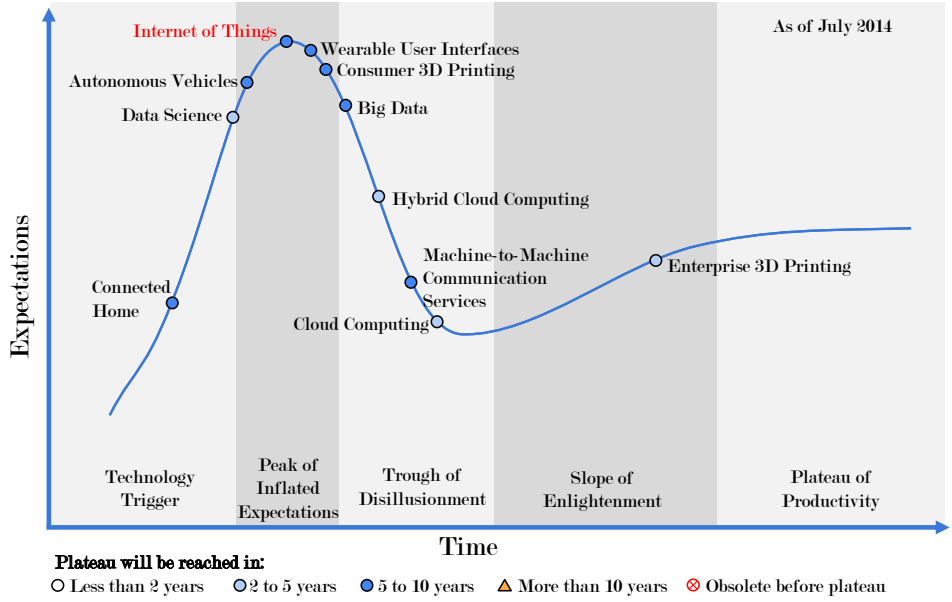
A third definition has been provided, in 2014, by the International Telecommunication Union (ITU) and by the IoT European Research Cluster (IERC) [3] and states

> *A dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual "things" have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network.*

Many definitions have been provided in the last 35 years, however, we have to come back to 2008/2009 to find the actual birth of the IoT. According to the Cisco Internet Business Solutions Group (IBSG) [6], the IoT era started when the number of Internet-connected devices (*e.g.,* computers, smartphones, etc.) was greater than the number of humans on Earth. The technological hype started a few years later, in 2011, when Gartner inserted the term Internet of Things in its Hype Cycle of Emerging Technologies [4] (Figure 1.1a) and IoT hit the top of the curve in 2014 [5] (Figure 1.1b). At the same time, the need to process data, coming from IoT devices



(a) Hype Cycle 2011 (adapted from [4]).



(b) Hype Cycle 2014 (adapted from [5]).

Figure 1.1: Hype Cycles for Emerging Technologies related to IoT in 2011 and 2014. Red labels indicate the "Internet of Things".

and the strong limitations imposed by devices constraints, has pushed research effort to design IoT applications by moving the computation to remote entities by relying on the Cloud Computing paradigm [7], which ideally provides infinite computational power, memory, and storage capa-

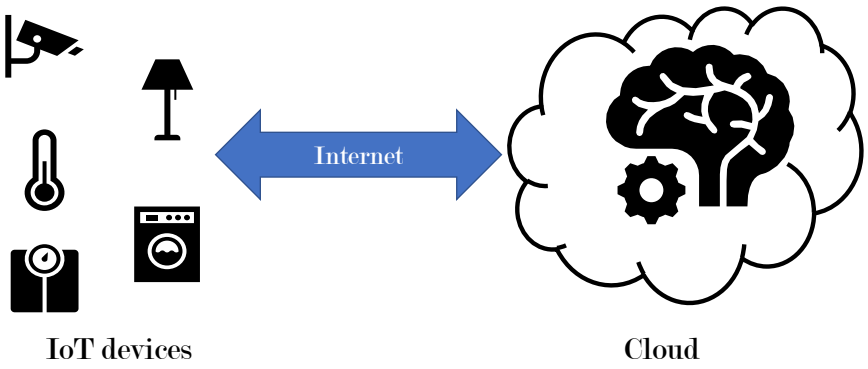


Figure 1.2: Two-tier architecture of a Cloud-centric IoT application.

bilities. In such a scenario, as depicted in Figure 1.2, the typical approach is to collect data from the field using connected sensors or smart devices and, then, send data to remote endpoints for computationally heavy tasks. Cloud Computing allows executing computationally intensive routines, like AI tasks (*i.e.,* inference, training, validation, etc.), processing, data visualization, asset monitoring, data storage, etc. Many research and industrial players have invested a lot of effort to develop cloud products and solutions to reliably collect data, process information, and extract knowledge. This approach has enabled new classes of applications that may support geographically distributed deployments, even across multiple continents, of actuators and sensors. Moreover, the combination of Cloud Computing with AI has enormous economical potential since it has been forecast that AI on the cloud will reach a market size of 6.4 billion USD by 2026 [8].

1.2 Cloud Computing is not the answer for IoT

Cloud Computing based approaches, called Cloud-centric, are affected by several issues related to the communication between IoT devices and cloud endpoints. First, the distance between the devices and the cloud endpoints, which can be even thousands of km, introduces an unavoidable latency that may become dangerous for people, the environment, or simply makes latency-sensitive applications unresponsive (*e.g.,* AR/VR applications). On the other hand, the data volume produced by devices may be too big to be sent to and processed by a cloud endpoint. It has been forecast that overall IoT devices will produce 73.1ZB (1ZB, 1 zettabyte, is

1.000.000.000.000 GB) of data[1] by 2025. Moreover, sensible data, *e.g.,* real-time health data, should not be transmitted over the Internet to prevent any possible interference, *e.g.,* destruction, tampering, or eavesdropping of information. Last, but not least, devices require an always-on connection with the remote entity to work properly. If the cloud counterpart is unavailable, these connected devices become only useless "smart" commodities. We usually assume that the cloud is always present and available, however, a disaster (*e.g.,* earthquake, fire, internal bug, etc.) may happen. And it happened. Indeed, between December 2020 and March 2021, two cloud platforms experienced major incidents that caused big outages. These have risen many concerns about our strong dependency on the cloud.
The first incident happened on 14th December 2020, around 12 pm UTC, when the UserID Service of Google, a service used to authenticate OAuth requests, started to reject incoming requests due to an internal error in a quota system[2]. This error had a cascade effect on all main Google services that rely on OAuth, *e.g.,* the Google Workspaces apps (Drive, Meet, Docs, GMail, YouTube, Hangouts, Calendar, etc.), making them unavailable. This turns to a huge economical impact since people, all around the globe, could not access their data for almost 1 hour. Even if this problem looks only affected people, this had a huge impact also on the Google Home ecosystem, since smart devices use OAuth to authenticate their requests. Many users reported on Twitter[3] that they could not turn on their light, their heating system, and they were reconsidering the full automation of their houses based on products that require an always-on connection to cloud services to work. The interested reader can find more details here [9].
A second main disaster happened on 10th March 2021, at 00.47 am CET, when a fire started in the OVH SBG2 datacenter[4], in Strasbourg, destroying hundreds of servers and affecting thousands of websites, platforms, services, and so on. This incident was probably due to a failure of two UPSes (Uninterruptible Power Supplies), one of them was under maintenance and pushed back to service the day before SBG2 burnt down. As a

[1] https://www.idc.com/getdoc.jsp?containerId=prAP46737220
[2] https://status.cloud.google.com/incident/zall/20013
[3] https://twitter.com/alexdunsdon/status/1338461046785368067
[4] https://www.ovh.ie/news/press/cpl1786.strasbourg-datacentre-latest-information

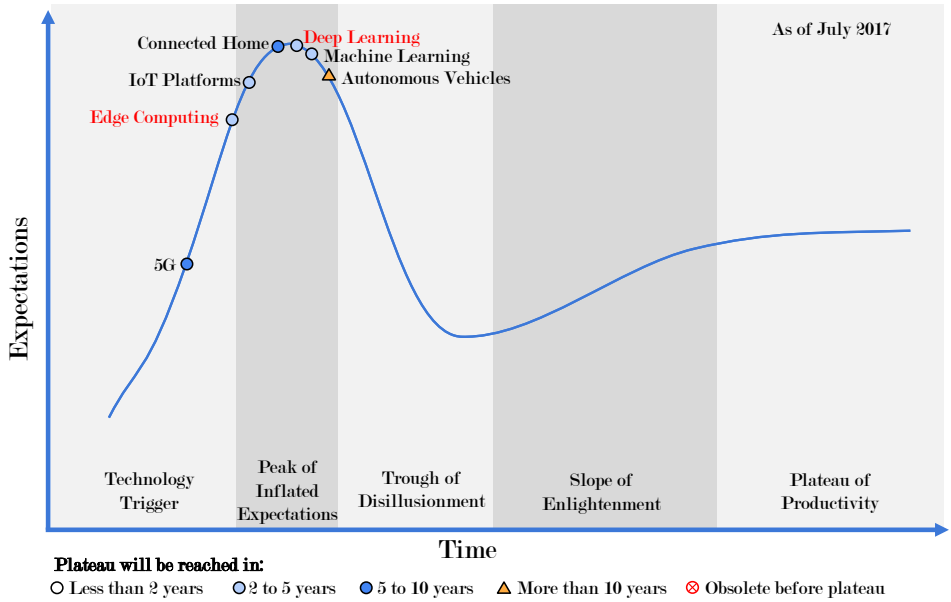


Figure 1.3: Hype Cycle for Emerging Technologies related to IoT in 2017 (adapted from [12]). Red Labels indicate the"Edge Computing" and "Deep Learning" topics.

consequence, also part of the SBG1 datacenter was affected by the fire. These incidents demonstrate that the idea/dogma "*too big to fail*", when referred to the cloud, does not work anymore. We need a strong change of direction on how we, as practitioners, design and implement IoT applications since it is not reliable to completely rely on remote endpoint to deliver main their functionalities. For instance, we can tolerate if we cannot listen to our favorite playlist or get the latest weather forecast but we should always able to heat-up our houses or dim our lights, even if the connection to the cloud is not available. This can be extended also other domains like transports, industry, e-health, or any other domain where the IoT plays the role of enabling technology. For these reasons, a strong push is required toward the design of applications that exploit more reliable, by design, computing paradigms. A candidate paradigm is Edge Computing [10], even if it has been a niche for more than a decade [11], it has become popular when entered in the Gartner Hype Cycle [12] (Figure 1.3). Edge Computing has been defined by the Industrial Internet Consortium[5] as

[5] `https://www.iiconsortium.org/IIC-OF-faq.htm`

> *Distributed computing that is performed near the edge, where the nearness is determined by the system requirements. The Edge is the boundary between the pertinent digital and physical entities, delineated by IoT devices.*

Talking about "*the edge*", we refer to the "*edge of the network*", which is a partition of the network close to data sources. Edge Computing, by design, provides some improvements over Cloud Computing with respect to communication latency, privacy, reliability, autonomy, and bandwidth usage [13]. This approach reduces the communication latency required to send data to the processing entity (*i.e.,* an edge device) providing fast responses, improves the user experience enabling new classes of applications (*e.g.,* augmented reality and virtual reality applications), and reducing the migration costs like the bandwidth usage. Since the computation is performed at the edge of the network, a high level of privacy is guaranteed by design even if the application manages sensible data (*e.g.,* e-health data). Moreover, the application should guarantee a minimum level of functionalities if the cloud counterpart is not available.
The edge of the network is populated by "edge devices", which are any network resource or computing entity that stands between a cloud endpoint and a data source. It can be a home gateway and connected devices in a smart home scenario, a cloudlet that supports heavy computations for latency-sensitive applications (AR/VE), a smartphone and a smartwatch in a wearable scenario, or the sensing device itself if powerful enough. Moreover, an edge device can play two roles [14]: it can be a data producer and a data consumer at the same time. It can continue to receive or send data to a cloud entity but it can also execute tasks from the cloud. Indeed, edge devices can run data processing, AI tasks, storage, caching, off-loading, and so on. This is also possible to the recent development of new embedded computing platforms that support fully-fledged operating systems (*e.g.,* Raspberry Pi) or MCU-based devices that offer enough computing resources for heavy tasks (*e.g.,* ESP32). However, this requires a proper design of the whole system to fulfill the application's requirements.

1.3 Toward new IoT application design approaches

In 2017, when the research for this thesis was started, Edge Computing was getting popular and big Internet players were releasing their first versions of platforms to start design and implement IoT applications in the network edge. However, applications were just a transposition of cloud-scale functionalities (*i.e.,* serverless functions like lambda functions) at the edge of the network on fully-fledged devices, thanks also to the migration and orchestration functionalities offered by the Fog Computing [15] paradigm[6]. Simultaneously, Artificial Intelligence (AI) was experiencing its second spring and, especially, Deep Learning was at the very top of the Hype Cycle [12] in 2017 (Figure 1.3). As a natural consequence, also IoT practitioners have embraced the possibility to exploit AI [16] to extract knowledge from sensory signals, however, the strong computational requirements have mainly forced the execution of AI algorithms in the cloud. At this point, a few questions arise:

- *How can the next-generation of IoT application take advantage by passing from a cloud-centric approach to new design approaches where the processing is not concentrated anymore in the cloud?*
- *How can processing (including AI tasks, firmware update, data-analytic, storage, caching, etc.) be distributed between the cloud and the data-sources by exploiting everyday-more-powerful computing devices?*

This thesis answers the above questions by proposing and discussing a paradigm shift from cloud-centric design approaches to more distributed approaches exploiting the edge of the network, with the final goal to execute as much as possible computations at the edge. However, a drastic change from a cloud-based scenario to a fully distributed scenario at the edge of the network requires, almost, a complete redesign of the application from scratch. Thus, to simplify this transition, this thesis guides a potential IoT practitioner to smoothly migrate toward better design approaches where

[6]Hereafter, the terms Edge Computing and Fog Computing will be used to indicate the same paradigm, however, they are slightly different. The former focuses on the system design point of view, the latter focuses on the infrastructure aspects of the edge of the network. More details about the difference are available in Chapter 2.

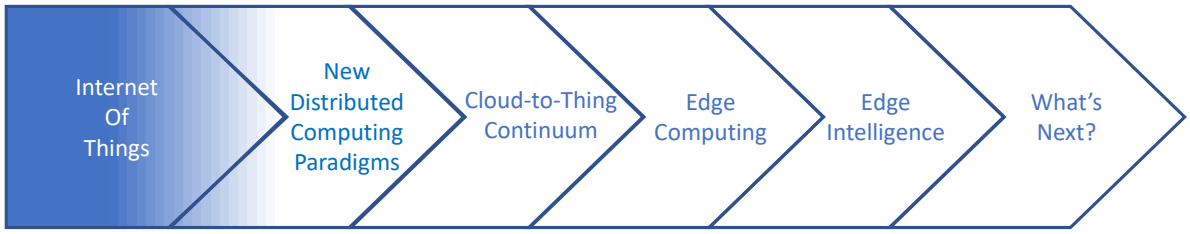


Figure 1.4: Schema of the transition toward Edge Intelligence applications. The introduction of the thesis presented the IoT, cloud-based applications, and the opportunity offered by the edge.

the application core runs at the edge. This migration will take several steps which are mapped on the chapters of this thesis.

1.4 Structure of the thesis

Every chapter opens with a "progress bar", like Figure 1.4, that provides an insight into the current position in the transition toward IoT applications that fully exploit the edge of the network. Moreover, each chapter provides the ingredients and the guidelines to build new applications at the edge and, then, uses the novel approach to implement a real-world IoT application as validation. More in detail, this thesis is structured as follows: Chapter 2 provides a better overview of new distributed computing paradigms (*i.e.,* Edge Computing and Fog Computing) by presenting the architectural, standard, product, and open-source community landscape. Chapter 3 presents how to design an IoT application that embeds data pre-processing routines and an AI algorithm on an edge device, *i.e.,* an IoT gateway, to provide its functionalities. Here, the improvement is on the moving of cloud-scale routines to the edge, indeed, IoT devices act as in a cloud-centric scenario with a different data-sync endpoint (*i.e.,* the IoT gateway). Then, Chapter 4 improves the previous design approach by moving part of the data pre-processing routines from an edge device (*i.e.,* the IoT gateway) to another (*i.e.,* the sensing device) by fulfilling the system requirements in terms of latency and bandwidth usage. As result, the processing is distributed over multiple entities at the edge of the network. Chapter 5 presents a novel class of chips, known as Edge AI accelerators, properly designed to execute cloud-scale deep learning models at the edge.

The chapter provides some key insight on how to design systems and the AI models to fully exploit these new devices, which are an enabler of Edge Intelligence or Edge AI. Chapter 6 presents an end-to-end framework to design Edge Intelligence IoT applications, starting from data sampling to the selection of the best AI model, passing through data pre-processing and AI model optimization. It uses a set of bio-inspired AI methods to identify the best parameters of the AI model that best tackle system requirements and constrained. Finally, Chapter 7 concludes this thesis by providing a recap on the contributions to the state-of-the-art and depicting some future works and scenarios that will take advantage of Edge Intelligence in the following years.

The research presented in this thesis is based on the contributions presented and discussed in the following publications:

- *Chapter 1:* **Mattia Antonini** and Massimo Vecchio. "IoT-Panic: The Cloud Just Disappeared!" *IEEE IoT Newsletter (January 2021), Non-peer reviewed (Newsletter).*
  URL: https://iot.ieee.org/newsletter/january-2021/iot-panic-the-cloud-just-disappeared

- *Chapter 2:* **Mattia Antonini**, Massimo Vecchio, and Fabio Antonelli. "Fog computing architectures: A reference for practitioners." *IEEE Internet of Things Magazine 2*, no. 3 (2019): 19-25.
  DOI: 10.1109/IOTM.0001.1900029

- *Chapter 3:* Zaffar Haider Janjua, Massimo Vecchio, **Mattia Antonini**, and Fabio Antonelli. "IRESE: An intelligent rare-event detection system using unsupervised learning on the IoT edge." *Engineering Applications of Artificial Intelligence 84* (2019): 41-50.
  DOI: 10.1016/j.engappai.2019.05.011

- *Chapter 4:* **Mattia Antonini**, Massimo Vecchio, Fabio Antonelli, Pietro Ducange, and Charith Perera. "Smart audio sensors in the internet of things edge for anomaly detection." *IEEE Access 6* (2018): 67594-67610.
  DOI: 10.1109/ACCESS.2018.2877523

- *Chapter 5:* **Mattia Antonini**, Tran Huy Vu, Chulhong Min, Alessandro Montanari, Akhil Mathur, and Fahim Kawsar. "Resource characterisation of personal-scale sensing models on edge accelerators." *In Proceedings of the First International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things*, pp. 49-55. 2019.
  DOI: 10.1145/3363347.3363363

- *Chapter 6:* **Mattia Antonini**, Andrea Gaiardo, and Massimo Vecchio. "MetaNChemo: A meta-heuristic neural-based framework for chemometric analysis." *Applied Soft Computing 97* (2020): 106712.
  DOI: 10.1016/j.asoc.2020.106712

Additional contributions, and not included in this thesis, have been published in the following publications:

- Luca Davoli, **Mattia Antonini**, and Gianluigi Ferrari. "DIRPL: A RPL-based resource and service discovery algorithm for 6LoWPANS." *Applied Sciences 9*, no. 1 (2019): 33.
  DOI: 10.3390/app9010033

- Raffaele Giaffreda and **Mattia Antonini**. "IoT Technologies and Privacy in a Data-Bloated Society: Where Do We Stand in the Fight to Prepare for the Next Pandemic?." *IEEE Internet of Things Magazine 3*, no. 4 (2020): 2-3, *Non-peer reviewed (Magazine column).*
  DOI: 10.1109/MIOT.2020.9319621

Chapter 2

From simply connected to fully autonomous devices

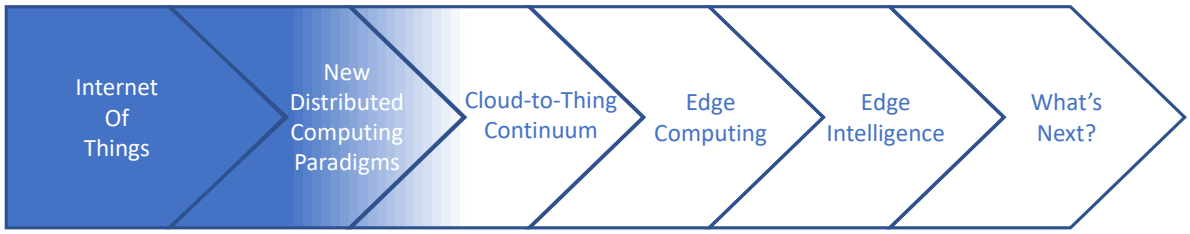


Figure 2.1: This chapter presents how novel computing paradigms, like Fog Computing and Edge Computing, help to migrate from cloud-centric applications toward IoT applications with Edge Intelligence.

2.1 Introduction

The first need of connected devices was to stream sensed data to powered-enough entities able to process incoming data and deliver responses and commands. As a natural answer, the cloud was the first option thanks to theoretically infinite computing power, infinite storage capacity, and easily reachable (since the only requirement is an internet connection, *e.g.*, ADSL subscription). This approach, known as the *cloud-centric* approach,

Part of this chapter appears in the following publication that I co-authored:
M. Antonini, M. Vecchio, and F. Antonelli, "Fog Computing Architectures: A Reference for Practitioners". *IEEE Internet of Things Magazine*, vol. 2, no. 3, pp. 19-25, Sep. 2019. DOI: 10.1109/IOTM.0001.1900029.

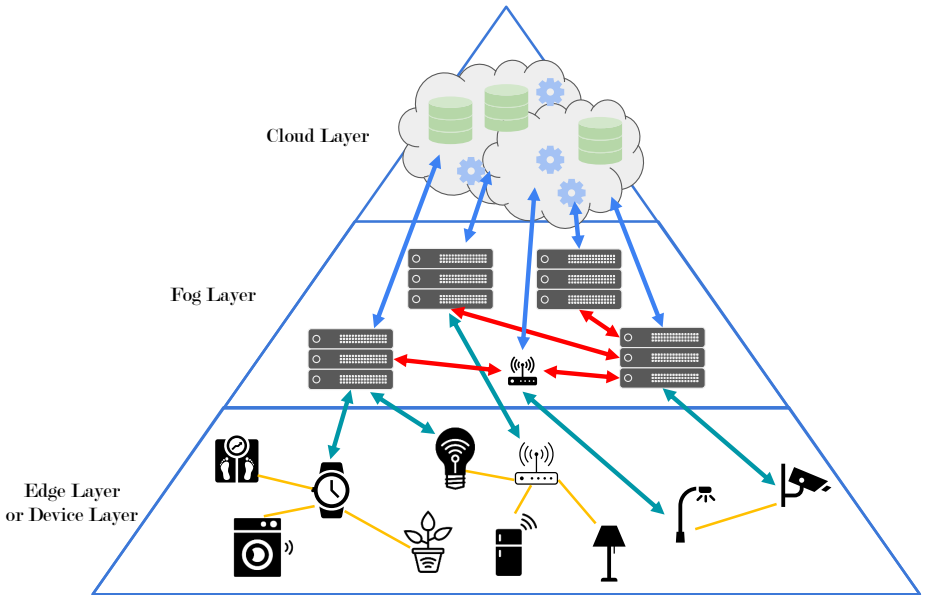


Figure 2.2: The Fog Computing architecture presented in [15].

has been applied to smart-agriculture, smart-homes, smart-cities, and connected industries, just to cite a few domains; and has been widely demonstrated by Internet big players (*e.g.,* Google, AWS, Microsoft, etc.). However, this approach suffers from many drawbacks as we presented in the introduction of this thesis. A possible solution to alleviate these issues was proposed by Flavio Bonomi (Cisco) in 2012 when he first introduced the Fog Computing paradigm as an extension of the Cloud Computing capabilities to the edge of the network [15].
In this chapter, we will give a wide overview of the state-of-the-art [17] of architectures, paradigms, standard definitions, and software platforms (both commercial and open-source) that support the extension of the Cloud toward the edge of the network. We will present these concepts from the perspective of practitioners that use these "tools" to design, build, and deploy their own IoT applications at the edge. Finally, Figure 2.1 depicts how this chapter is related to the transition from the design of cloud-centric IoT applications to applications that fully exploit the concept of Edge Intelligence.

2.2 Fog Computing architectures

Fog Computing [15] aims to move the execution of tasks from the Cloud closer to the data sources, by exploiting networking entities like gateways, access points, and routers. The Fog Computing architecture that he originally proposed comprised three main layers, as depicted in Figure 2.2: the bottom layer, containing IoT devices able to sense and act in the surrounding environment; a middle layer, that is the Fog layer, responsible for processing data locally and, if needed, able to forward them to a remote cloud system; and the upper layer, that is the Cloud itself. Fog Computing introduces the concept of *Cloud-to-Thing continuum* since the data processing from the sensing device to a cloud endpoint can be distributed along the way without discontinuity. This paradigm addresses, with a by-design approach, some of the major issues of today's cloud-based solutions with respect to privacy, reliability, latency, and bandwidth, at the additional cost of increasing the local computing power. This approach has the potential to enable new classes of applications; just as an example, consider a condition monitoring application, where some dedicated sensors sample the current state of an engine and a software application takes suitable actions based on the data acquired from the field. Here, we can have two alternative approaches: a cloud-based and a fog-based approach. In the first case, we deploy some sensors in the field that collect information about the physical phenomenon and forward it to a cloud endpoint, in charge of executing the required processing. If an anomalous condition is met, the cloud application sends a command to some actuators to make an action on the environment (*e.g.,* halting the monitored engine). In this case, the bandwidth required to stream the data might be too large, or the latency too high to react sufficiently fast to avoid a dangerous event. Moreover, if the connectivity fails and an anomalous event happens, then the end-to-end application might not be able to react, causing even serious consequences. The orthogonal possibility is to develop an application leveraging the Fog Computing paradigm. In this case, the computation is done closer to the sensors (*e.g.,* on a gateway) and the application does not need to send all the information to the cloud to react to anomalous events. In this sense, the system is faster and more reliable with respect to the previous scenario.

As we write, many efforts have been paid by different communities, both academic and industrial communities, to define a standard architecture for Fog Computing. Among all the definitions, the IEEE has adopted the OpenFog architecture [18] as a reference architecture, as we will discuss in Section 2.3. It is worth mentioning that, in the last few years, research and development efforts have been focused on identifying the correct number of layers comprising the perfect Fog Computing architecture. More in detail, based on the implementation details, the literature boasts of architectures composed of three [15, 19, 20], four [21], five [22], six [23], and even eight [24] layers. Arkian et al. [21] proposed a four-layer architecture, where the first three layers are those initially proposed in [15], while the fourth one is a vertical layer, namely the Data Consumer layer, used to make requests to the other layers. Dastjerdi et al. [22] introduced a five-layer stack, where the IoT applications and the Software-defined resource management layers are located on top of the three traditional ones. Aazam et al. [23] defined a six-layer structure by highlighting the functionality that should be implemented in the Fog. Recently, Naha et al. [24] described a detailed and fine-grained architecture, where components are divided into eight different groups based on their functionality that defines the layer. Specifically: physical (sensors, actuators), Fog device (configuration and connectivity), Monitoring, Pre- and Post-Processing, Storage, Resource Management (resource allocation, scalability, reliability), Security (encryption/decryption, privacy, authentication), and Application layers. Another approach to Fog Computing has been recently proposed by Sinaeepourfard et al. [25] that uses Fog Computing as a building block for a distributed-to-centralized data manager for smart cities. They identify three main layers: the Fog layer that contains IoT devices and performs some on-site processing. The Cloudlet layer is the mid-layer, it is located in the same city of the fog layer and it is used as a communication layer among the different and distributed entities in the fog layer. The third layer is the (traditional) Cloud layer. Notwithstanding this plethora of system architectures essentially reveals that a wide effort has been needed to fill the gap of consensus among researchers, software architects, system integrators, and IoT practitioners in general on a unified reference architecture for Fog Computing. It is important to take stock of the current, most mature, implementations from

standardization, commercial, and open-source communities' perspectives. Indeed, this is what the remaining of this chapter will be all about.

2.3 Standard Initiatives

Some big names of industry and academia have already joined efforts, forming consortia to formalize possible architectures for Fog Computing. One of the main initiatives in this respect was the OpenFog consortium, established in November 2015 by ARM, Microsoft Corp., Intel, Cisco, Dell, and Princeton University and now incorporated into the Industrial Internet Consortium (IIC). OpenFog Consortium defined Fog Computing as *"a horizontal, system-level architecture that distributes computing, storage, control, and networking functions closer to the users along a Cloud-to-Thing continuum"* [26]. This consortium formally aimed to fill the gap [15] present in the design of IoT applications that are built using a "cloud-only" architecture. More in details, OpenFog had identified some "pillars" to distinguish Fog Computing from Cloud Computing, namely: 1) low latency, deployments, and computations near the data-sources (*i.e.,* IoT devices); 2) avoid migration costs (*i.e.,* bandwidth); 3) local communications instead of communications with remote end-nodes; 4) management, network configuration and measurement deployed in fog nodes; 5) support for telemetry and analytics that should be sent to a remote system for orchestration and additional analytics. The proposed architecture follows Bonomi's (Cisco) [15] one: a three-layer stack where the Fog layer, composed of nodes called Fog Nodes, is split into four main sub-layers, namely Platform Hardware, Node Management, and Software Backplane, Application Support, and Application Services. The lower layer is the Platform Hardware that is the physical hardware of the Fog device. The Node Management and Software Backplane layer is in charge of the general management of nodes and communications among endpoints (*e.g.,* remote cloud systems, edge devices, other Fog nodes). The Application Support layer is a collection of micro-services that are not application-specific. These modules comprise databases, storage managers, networking stacks, security modules, message/event buses, runtime engines, analytic tools, etc.

The last module is the Application Services layer that offers many services to applications like Fog Connector services, Core services, Supporting services, Analytic services, Integration and User Interface services.
It is important to notice that this architecture had been conceived for quite powerful devices that are capable to offer both reactive and predictive capabilities. More in detail, reactive capabilities analyze the incoming data (*e.g.,* vibrational signals) to discover if something is happening in the surrounding environment. This set of capabilities includes, for instance, Anomaly Detection, Rule Engines, Event processing, Sensor fusion and meta-sensors, supervisory control. On the other hand, predictive capabilities, also referred to as forecasting capabilities, comprise Artificial Intelligence and Machine Learning based techniques that can identify patterns and forecast future behaviors based on the incoming data. These predictive models may be directly inferred locally by the Fog node or, if required, in a hybrid fashion among different Fog nodes and Cloud systems. Since some operations over such models can be more demanding in terms of computational and memory power (*e.g.,* model training) than others, these can be executed on more powerful remote cloud systems; after the training phase, such models are downloaded from the Cloud and deployed in the Fog nodes.
In August 2018, the Institute of Electrical and Electronics Engineers (IEEE) adopted the OpenFog Architecture as the reference architecture for Fog Computing for the IEEE 1934-2018 standard [18]. This should allow developers and companies to build their own Fog-oriented applications using a standardized approach. However, as we write, an open-source project implementing the whole IEEE 1934-2018 architecture, or that is at least fully compliant with all its specs, does not exist yet.
Then, in January 2019, the OpenFog Consortium and the Industrial Internet Consortium[1] (IIC) announced that they had finalized the agreement to join forces and merge the OpenFog consortium (and all of its working groups) under the umbrella of the IIC. This has been done since the two consortia were working on the common objectives and, in this way, they could boost the development and the deployment of Fog Computing applications for the Industry 4.0 scenarios. Interestingly, as part of the

[1] `https://www.iiconsortium.org/`

agreement were the formal definitions of the terms Fog Computing and Edge Computing [27]. In more detail, Fog Computing, as defined by the OpenFog Consortium, is "*a system-level horizontal architecture that distributes resources and services of computing, storage, control and networking anywhere along the continuum from Cloud to Things*". On the other hand, the IIC defined Edge Computing as "*distributed computing that is performed near the edge, where the nearness is determined by the system requirements. The Edge is the boundary between the pertinent digital and physical entities, delineated by IoT devices*". So, while the terms Edge and Fog Computing are often used interchangeably, the above definitions shed light on their conceptual differences. Specifically, Edge Computing leverages on processing resources *already located* at the edge of the network (*i.e.,* closer to end-users and IoT devices), while Fog Computing *shifts* typical cloud capabilities toward the edge of the network, leveraging on the edge's resources (*e.g.,* gateways, local servers, etc.), also facilitating the distribution of application logic in a *Cloud-to-Thing continuum.*

On another front, in January 2019, the Linux Foundation has started a new initiative, known as LF Edge[2]. In this case, the declared objective is "*to establish a unified open-source framework for the edge [...] contributing a new agnostic standard edge architecture*" [28]. This subfoundation aims to create an open and interoperable ecosystem of software frameworks for fog/Edge Computing platforms, constrained to be vendor-neutral, hardware-independent, and technology-agnostic (*i.e.,* cloud- and OS- independent). This would create a unified and aligned vision for the fog/Edge Computing paradigm by creating communities that will drive a better and more secure development of applications at the edge of the network. As declared in the "The State of the Edge Report 2018" [29], the LFEdge identified four pillars that we report in the following list: 1) Edge is not a thing, it is a location; 2) there is not only one edge, however, the LF Edge is now focusing on the last-mile network; 3) the edge is the combination of infrastructure and devices; 4) edge will continue to work coordinately with centralized cloud counter-part. At the time of writing, the LF Edge Foundation counts nine different projects[3] under its umbrella

[2] `https://www.lfedge.org`

[3] `https://www.lfedge.org/projects/`

partitioned into three different maturity stages.
At Large Stage belongs projects that have the potential or the Technical Advisory Council (TAC) believes that they will become important for other more mature projects or the whole edge ecosystem. *At Growth Stage* we find projects that aim to reach the higher stage and have found the plan to do that. They receive direct mentorship from the TAC and they have to develop their governance and community of developers, contributors, supporters, and so on. Finally, *at Impact Stage* we find projects that have achieved their growth goals and now have a self-sustaining life-cycle for maintenance, release, and so on. More details about the development stages of LF Edge projects can be found here [30]. Starting from the "*at Large Stage*" projects, we find

1) *Baetyl*[4], originally known as *OpenEdge* by Baidu [31], is an open-source project that enables developers to build edge applications extending the Cloud Computing counterpart seamlessly. It relies on the design concepts of serverless and containerized applications, it abstracts different hardware capabilities into a built-in set of APIs and a runtime environment, reducing the difficulties of developing applications that can run on a wide range of platforms, from embedded IoT devices to clusters of cloud machines.

2) *Open Horizon*[5], originally known as *Blue Horizon* by IBM, enables the autonomous management of machine learning assets (*e.g.,* trained models) and of the containerized workloads over fleets of Edge Computing nodes, which might contain even more than 10.000 devices. Moreover, the management of fleets does not require administrators close to the deployments allowing the automatic hand-free management of edge nodes. Developers that adopt Open Horizon can deploy and replace capabilities of single-purpose devices also based on policies and negotiated agreements.

3) Secure Device Onboard[6], or SDO, was originally released by Intel Corp. as an open-source project. SDO aims to provide an automatic

[4]`https://www.lfedge.org/projects/baetyl/`
[5]`https://www.lfedge.org/projects/openhorizon/`
[6]`https://www.lfedge.org/projects/securedeviceonboard/`

and secure (*i.e.,* without using unsafe default passwords) zero-touch method to provision new edge IoT devices, reducing deployment costs and simplifying the installation procedures (*i.e.,* plug&play). This should also enlarge the Total Available Market (TAM) since it enables high volume production of IoT devices and, consequent, wider adoption by customers.

Then, "*at Growth Stage*" we find:

4) Edge Virtualization Engine[7], or EVE project, aims to build the EVE-OS, a Linux distribution for distributed Edge Computing. The objective is to simplify the design, deployment, and orchestration of edge nodes and at the same time guarantying a high level of security by keeping a technology-agnostic approach (*i.e.,* any combination of hardware, applications, and cloud providers). Access to hardware capabilities, orchestration, and other applications is guaranteed via EVE APIs.

5) Fledge[8] is an open-source framework developed to build industrial edge IoT applications. The aim is to tackle verticals like predictive maintenance, critical operations, safety (*e.g.,* hazardous environment), and many others. Fledge has been designed to eliminate data silos, usually found in many industrial settings, by supporting legacy and industrial-level ecosystems and equipment like SCADA, PLC, DCS. The fragmentation issues and complexity of applications are overtaken by implementing a well-defined set of APIs for applications and administrative purposes. This project is usually accompanied by the EVE project, which provides orchestration and other services.

6) Home Edge[9] is a Fog/Edge Computing framework for home automation, offering an open-source, robust, flexible, and interoperable environment where devices can be simply integrated through a set of APIs, libraries, and runtimes. Home Edge, thanks to its orchestrator, allows flexible and agnostic deployments given the usage of Docker as

[7] `https://www.lfedge.org/projects/eve/`
[8] `https://www.lfedge.org/projects/fledge/`
[9] `https://www.lfedge.org/projects/homeedge/`

containerization technology. Containers can, then, be easily deployed and off-loaded (Home Edge support dynamic load balancing) over the infrastructure.

7) The State of the Edge (SOTE) [10] is an ambitious vendor-neutral platform to foster the development of open research on fog and Edge Computing. The research developed within the project is freely shared among partners and used to propose, discuss, and define solutions that will drive the evolution of Edge Computing and next-generation internet (NGI). Under the umbrella of this project, three assets have been developed: the SOTE reports, the open glossary of Edge Computing, and the landscape of Edge Computing.

Finally, "*at Impact Stage*" we find:

8) Akraino[11] is an open software stack that enables edge-cloud infrastructures (with VMs, containers, microservices, ...) by keeping the focus on the pillars of Edge Computing like low latency and local processing. Akraino delivers a set of application blueprints created and tested by its community that tackles edge use-cases for both enterprise and provider domains. These can be used as-is or to derive other blueprints. Akraino blueprints have been created following different principles: design principles (availability, continuity, security, and capacity), build principles (low-latency deployment and processing, and plug&play architecture), run principles (zero-touch provisioning, lifecycle, and operations), and community principles(organization and oversight of the project by the community that also pushes the development of blueprints over sponsored hardware).

9) EdgeX Foundry[12] is an open-source and modular framework for Fog/Edge Computing applications. EdgeX Foundry enables the interoperability of applications and heterogeneous devices directly in the IoT Edge, along with a strong foundation for security, management, and so on. It allows to plug modules and create custom functional blocks in a

[10] `https://www.lfedge.org/projects/stateoftheedge/`
[11] `https://www.lfedge.org/projects/akraino/`
[12] `https://www.edgexfoundry.org/`

simplified way. Given its maturity, this project will be separately presented with more details in Section 2.4.1.

The nature and the philosophy behind the LF Edge Foundation drive the open-source soul of this consortium. It is open to contributors, thus everyone can contribute to existing projects and can ask to incubate new ideas, while only the membership to the group requires the payment of an annual fee.

Other standardization initiatives are currently under development by different working groups in the Internet of Things ecosystem.
The Alliance for the Internet of Things Innovation (AIOTI) has proposed the AIOTI High Level Architecture (HLA) [32], with the aim to propose a deployment- and technology-agnostic architecture oriented to IoT applications. It is possible to host AIOTI HLA functionalities on top of Fog nodes. However, this architecture does not target Fog Computing but more in general IoT applications, thus we will not further discuss this initiative.
In 2012, the ITU-T has published the "Recommendation Y.2060" [33], renumbered as Y.4000, that describes the general architecture for IoT applications. The document aims to define how an IoT application should be designed by defining a set of layers and corresponding capabilities. However, it assumes that the devices have simple and limited capabilities related to sensing, node management, and networking; a device may overlap with the gateway and it assumes that there is a remote entity that manages and runs the IoT application.
The European Telecommunication Standard Institute (ETSI) has proposed the oneM2M Architecture [34], which defines a software architecture for IoT and Machine-to-Machine (M2M) applications. This document splits the architecture into three different layers (Application Layer, Common Service Layer, and Network Service Layer), it defines many different types of nodes and their role in the architecture. It is possible to map some functionalities along the *Cloud-to-Thing continuum*. Moreover, ETSI has recently published a technical report [35] in which they are evaluating possible changes of the oneM2M architectures to introduce the concepts of Fog and Edge Computing.

2.4 Fog Computing Platforms

In addition to the effort paid to define a standard Fog Computing architecture, many communities and private companies are also actively involved in designing and developing full-fledged software platforms able to cope with the unique requirements of this challenging computing paradigm. In the remainder of this section, we introduce some of the main actors playing on this stage.

2.4.1 Frameworks for Fog/Edge Computing

This section introduces two frameworks that are currently used in many applications, namely Apache Edgent and EdgeX Foundry.
Apache Edgent[13] is a Java-based framework for edge stream analytics incubated by the Apache Foundation. Mainly, it enables a data flow-based programming model suitable for fog/edge devices. Moreover, it provides a lightweight micro-kernel run-time environment embeddable in several off-the-shelf gateways and on other constrained devices able to execute a Java Runtime Environment (JRE). It also supports local and real-time analytics on data streams from the surrounding environment, such as vehicles, appliances, equipment, and so on. More in details, a fog application can integrate Apache Edgent in the fog layer and the framework uses analytics (*e.g.,* split, union, filters, windowing, aggregations, etc.) to identify which data have to be streamed from the edge of the network to another computing entity (*i.e.,* a cloud endpoint). This reduces the overall network bandwidth (and the associated cost of transmission, especially high in IoT contexts) and storage needs, also guaranteeing faster feedback toward local devices. Here, a developer can easily decide how data streams are managed inside his application and which computations have to be applied to which data. However, Apache Edgent provides only a few more capabilities than a stream manager, thus it does not come with a complete architecture to design an entire application. It can be easily integrated using the provided SDK and it can communicate with the outside world using well-known protocols such as, for instance, MQTT.

[13] `https://edgent.apache.org`

A more complete and mature platform for Fog Computing is EdgeX Foundry. The latter is a project originally donated by Dell Technologies to the open-source community in October 2017 and since then hosted under the umbrella of the LF Edge Foundation. Currently, EdgeX Foundry is supported and developed by more than 60 members coming from academia and industry. Since this project is constrained to be vendor-neutral, it provides a software solution that is not tied to any specific hardware or software supplier. The project aims to accelerate the deployment of IoT solutions by creating a unified and plug-and-play ecosystem that relies on an interoperability framework. This framework is implemented through an OS- and hardware-agnostic software platform for Fog and Edge devices that allows developers and companies to design new interoperable applications by combining standard connectivity interfaces (*e.g.,* Wi-Fi, Bluetooth, BLE, etc.), common software modules, and proprietary extensions. Moreover, the project leaderboard aims to contribute to create a common standard for IoT interoperability and to create a certification program for hardware and software components to guarantee compatibility and interoperability.

The EdgeX Foundry Architecture relies on the well-known three-layer Fog Architecture [15], as depicted in Figure 2.3, where edge devices and cloud systems are located below ("southbound", or edge layer) and above ("northbound", or cloud layer) the EdgeX Foundry software architecture, respectively. More in detail, the EdgeX Foundry software architecture is located in the middle layer (*i.e.,* fog layer), being composed of many sub-layers, as described in the following. Such architecture has been conceived starting from the micro-service paradigm [37] that enables modular, scalable, secure, and technology-agnostic applications. Specifically, the chosen approach is the loosely-coupled micro-services architecture that requires a common layer to enable communications and data exchanges among modules using Inter-Processes Communications (IPC) APIs (*i.e.,* REST APIs). This layer may be also distributed over more than one device if different services run on many Fog nodes. EdgeX Foundry can run on any edge/fog device such as gateways, routers, industrial PCs, servers, hubs, etc.
The EdgeX Foundry software architecture [38], which is located in the fog layer (see Figure 2.3), is composed of four horizontal sub-layers for

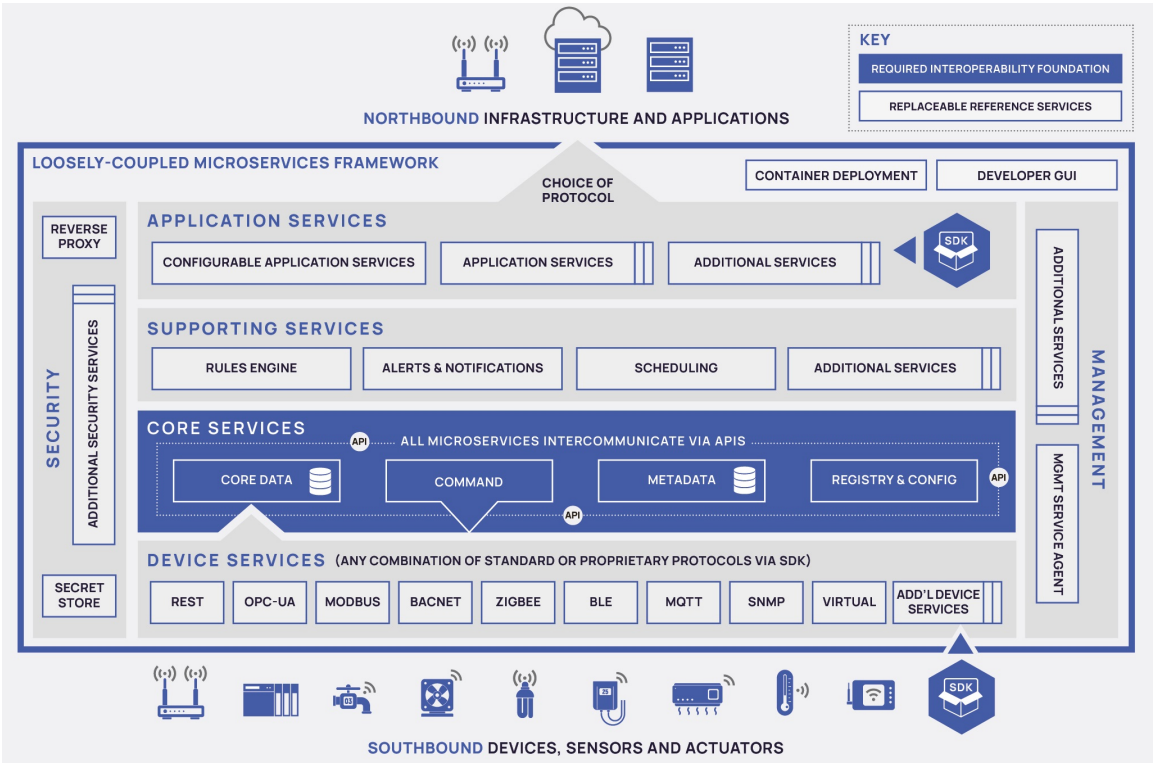


Figure 2.3: The EdgeX Foundry platform architecture (source [36], license: CC BY 3.0).

the business logic definition, and two vertical sub-layers for security and management functionalities. The horizontal sub-layers are Device Services, Core Services, Supporting Services, and Application Services. The Device Services layer comprises communication protocols and schema to interact with heterogeneous IoT Edge devices (*e.g.,* BLE, MQTT, BACNET, MODBUS, REST, and so forth). It is also possible to integrate any missing protocol, by developing a micro-service that integrates such protocol through libraries. This allows companies to integrate their own proprietary technologies and protocols within EdgeX Foundry. Core Services layer implements the Interoperability Foundation that is a set of micro-services (Core Data, Command, Metadata, Registry&Configuration) required to build an application. The Supporting Services layer exposes functionalities that are useful for all the applications defined at higher layers. It comprises Rule Engine, Scheduling Engine, Logging services, and Alert&Notification system. Similar to protocols, it is possible to add new modules and services by developing a micro-service exposing the functionalities through an API. The Application Services layer provides the functionalities to manage,

process, deliver data from EdgeX to other endpoints or processes. These services are designed by following the idea of "function pipelines", it is a sequence of routines that process data one function after the other. The first element of each pipeline is a trigger, *e.g.,* a message from a device, that initializes the data processing. An SDK is currently available to enable developers to build their own services, *e.g.,* a module to connect to a specific cloud provider. Finally, the two vertical sub-layers, the Security and the Management layers, expose micro-services that interact with all four horizontal layers. The Management layer contains a micro-service to deploy new modules in the system and the Security module manages all the security operations like encryption, decryption, access, policies, and so on.
The EdgeX Foundry project and Architecture has been designed to create Edge/Fog applications for the Industrial IoT. Some of the main contributors (*e.g.,* Dell) of this project have already commercialized some industrial computers that natively run EdgeX Foundry.
At the time of writing, the latest version of EdgeX Foundry is 1.3.0-Hanoi. A new version, 2.0-Ireland, is expected in Spring 2021.

2.4.2 Nebbiolo Technologies

Commercially, some companies are offering products that implement the Fog Computing architecture. One of them is Nebbiolo Technologies that has been co-founded by Bonomi, *i.e.,* one of the Fog Computing pioneers. Nebbiolo Technologies produces a complete Fog solution composed of nodes (*i.e.,* fogNode[14]), an operating system (*i.e.,* fogOs[15]), and a system manager (*i.e.,* fogSM[16]). Focusing on fogNodes, these are powerful machines built for industrial environments and equipped with powerful CPUs (Intel i5, i7, or Atom), solid-state disks (from 32 GB up to 512 GB), large RAM banks (from 4 GB to 16 GB), and networking interfaces (Ethernet, WiFi, and 3G/LTE). These machines support many functionalities and

[14] https://www.nebbiolo.tech/wp-content/uploads/NFN-300-Datasheetv1.8FINAL-C-2018-Pantone.pdf
[15] https://www.nebbiolo.tech/wp-content/uploads/fogOS-Datasheetv1.5a-2018-Pantone.pdf
[16] https://www.nebbiolo.tech/wp-content/uploads/NFN-300-Datasheetv1.8FINAL-C-2018-Pantone.pdf

address many requirements of Fog Computing. However, these products are quite expensive and they are not as flexible as an open-source project could be, as all the code is developed by the company and it does not exist a community that supports and improves these products.

2.4.3 Fog/Edge Computing-as-a-Service

Another approach to Edge/Fog Computing is offered by Internet big players like Amazon Web Services (AWS), Microsoft, and so on. They are proposing solutions to implement Fog nodes with a cloud back-end and with support for computing capabilities like machine learning algorithms. Using this approach, providers can implement and offer an edge/Fog Computing infrastructure as a service (IaaS). This allows developers and customers to put effort only on their fog/edge application removing the management, orchestration, and deployment complexity.

AWS offers Greengrass (AWS-GG) [39], a software that extends the cloud capabilities of AWS Cloud closer to edge devices by directly enabling data collection and analysis on the edge of the network. At the time of writing, AWS has recently rolled-out Greengrass V2 that delivers new capabilities and functionalities with respect to Greengrass V1. Developers can create, deploy and manage, via AWS-GG Cloud APIs, software components on edge devices for local execution. AWS Greengrass runs on edge devices that may be full-fledged computers, servers, virtual machines, but even single-board mini-PC like Raspberry Pis. Moreover, these devices can securely communicate among them, using authentication and authorization mechanisms, on the same network without any mediation with the remote cloud back-end. Running applications can continue their execution even in absence of connectivity. Furthermore, AWS-GG caches outbound and inbound messages using a local publish/subscribe message manager, based on MQTT, to preserve undelivered messages.
AWS-GG is composed of a modular core software, namely the *nucleus* and other optional components provided by AWS, an SDK to implement edge nodes, an SDK to manage data streams, cloud APIs to manage devices and orchestrate deployments, and the integration of many other AWS products/services like Machine Learning Inference, Shadows implementation,

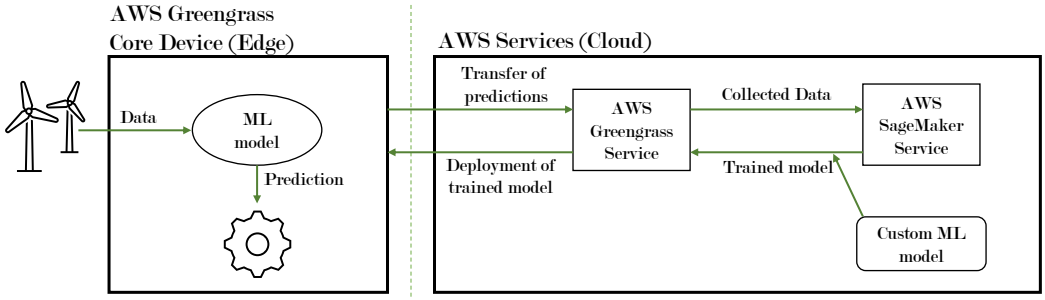


Figure 2.4: The AWS Greengrass application architecture with machine learning functionalities (adapted from [39]).

group management, Lambda runtime, message manager, secure over-the-air updates, local resource access and so forth. Other components may run on the edge device and implement functionalities. Each component is composed of three entities: the recipe, which contains the parameters, the configuration, metadata needed to run the component; the artifact that is the source code/binaries of the component, which might be Lambda functions, Docker containers, and so on; the dependencies that describe how different component are dependent one with respect to the others. For instance, Lambda functions can be used to build IoT devices that can be triggered by events, messages from the cloud, or other components.

Additionally, AWS Greengrass offers the possibility to embed and infer machine learning models in two different flavors: using Amazon SageMaker Neo DLR components and models for computer vision, or embedding TensorFlow or other frameworks in the component that will run on the edge device. Models, embedded within components, are automatically deployed by Greengrass. This approach (Figure 2.4) enables more intelligent edge applications that can reduce latency, costs (*i.e.,* bandwidth and energy), and exploit powerful cloud systems to train models. Models can be built and trained using AWS SageMaker[17], a cloud service developed to train deep-learning models using common frameworks like Tensorflow[18], Keras[19], PyTorch[20], Caffe2[21], and so on. Furthermore, components, which infer

[17] `https://aws.amazon.com/sagemaker/`
[18] `https://www.tensorflow.org/`
[19] `https://keras.io/`
[20] `https://pytorch.org/`
[21] `https://caffe2.ai/`

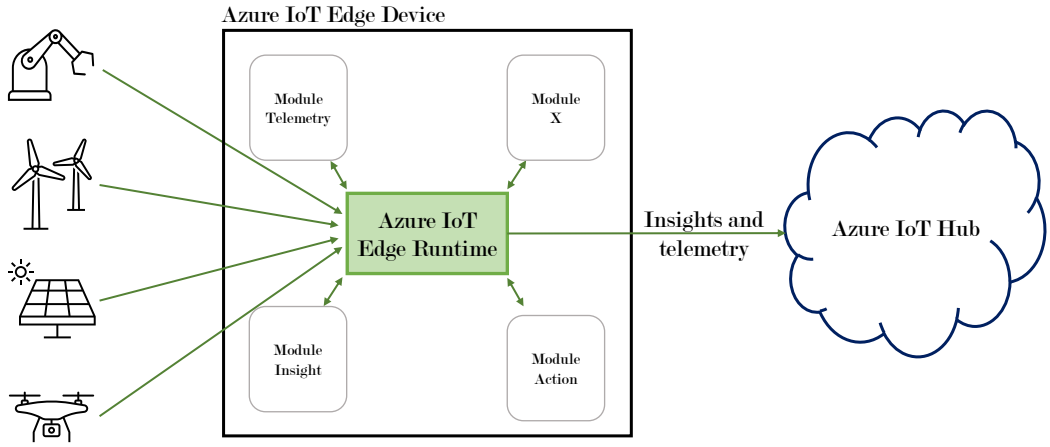


Figure 2.5: The Azure IoT Edge application architecture (adapted from [40]).

models, can forward back incoming data to AWS S3[22] (AWS cloud storage service) to provide new data samples to update or re-train models.

Another suite of products for Fog Computing has been developed by Microsoft, namely Azure IoT Edge [40]. As AWS Greengrass, it allows developers to deploy their own business logic closer to data sources in order to reduce latencies, bandwidth, and increase reactiveness. Figure 2.5 depicts the typical architecture of an Azure IoT Edge application. Azure IoT Edge is a service developed over Azure IoT Hub and it is composed of three main components: containers that locally run Azure, user, or 3rd parties modules; Azure IoT Edge runtime that is executed on each device and manages all the other modules; a cloud-based interface to remotely monitor and manage a fleet of Azure IoT Edge devices. Azure IoT Edge runs on many different hardware architectures including Raspberry Pi, full-fledged computers, industrial computers, servers, and so on. It has been designed by following the micro-service architecture [37] to implement modules, which are Docker-compatible containers. Modules can be interconnected through pipelines to exchange data. Additional modules may add new classes of supported devices by exploiting the edge node's networking interfaces. Moreover, it enables machine learning applications by combining user-developed containers to run custom code, *i.e.,* Feature Extraction, and Azure Machine Learning.
Concurrently, Microsoft is also developing a new set of tools for data an-

[22] `https://aws.amazon.com/s3/`

alytics at the edge of the network known as Azure SQL Database Edge[23]. As we write, this toolkit enables local data processing on devices that support containerization, *e.g.,* Linux- or Windows-based systems. It offers the possibility to execute graph-based local analytics, data-stream, and time-series processing (*e.g.,* data filtering, windowing, aggregation, etc.) before forwarding information to another entity, like a cloud endpoint, optimizing bandwidth and money. Moreover, it can locally execute in-database machine learning algorithms to identify patterns, anomalies, classify instances, and so on. The model might be trained in the cloud and then locally deployed to be able to execute it offline and to minimize the inference latency. Finally, other Azure services, *i.e.,* Azure Streaming Service (ASA)[24], can be easily deployed on edge devices to support new functionalities and deliver a better service.

2.5 Comparisons and Considerations

All of the described solutions deal with the Fog Computing paradigm with very specific approaches, sometimes offering peculiar services and/or capabilities and hitting the market with different sales models and prices.

On the one side, Internet giants like Amazon, Microsoft, and Google provide cloud-assisted digital products targeting vertical domains like industry, home automation, buildings, vehicles, and so on. Essentially, they are comfortable playing the role of enablers for several novel applications within this framework, given the unique degree of integration with other cloud services they are already offering to their millions of users. However, Fog/Edge Computing has also the clear mandate to embed Artificial Intelligence closer to the data sources, meaning (among the other things) enabling the integration of some machine learning services and modules inside tinier IoT devices. On the contrary, these companies' products are not always available for embedded architectures (*e.g.,* ARM-based boards) and sometimes they are not even publicly available yet. Moreover, even if such products are in general very flexible, they are also closed-source and often based on a cloud-centric approach. This means that the management

[23] `https://azure.microsoft.com/en-us/services/sql-database-edge/`

[24] `https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-edge`

and the orchestration of users' resources, models, and data are performed at the cloud side. As said, this approach suffers from major issues, such as reliability, privacy, bandwidth, and so forth. Since these frameworks depend on a cloud back-end, the system may have unpredictable behavior if connectivity fails. Furthermore, data coming from devices are usually streamed to cloud endpoints to build a historical database and to update their predictive models. The amount of data might be too high to be streamed by the producers and/or stored by the cloud service (because of its volume, cost of the service, etc.). Typically, the billing of these products is based on batches of messages or per bunch of data. Finally, depending on the application, data might be sensitive, hence it should not be streamed.

On the other side, solutions like those offered by EdgeX Foundry and Nebbiolo tackle Fog Computing with a different approach: in principle, applications do not need to rely on any cloud infrastructure or remote service to implement their functionalities, but they can occasionally use cloud-based support to more efficiently tackle specific operations (usually the most demanding ones, in terms of computational and memory power). In this way, fog/edge devices may be empowered with high-performance computing units and big storage devices that allow applications where computation is pushed as much as possible closer to the edge of the network. For example, it is possible to train and execute machine learning and Artificial Intelligence algorithms directly on the edge. The developers can thus create modular applications keeping full control on the entire data-flow processing and of the devices. The typical approach is to exploit the micro-service paradigm [37] as a reference methodology.

Concluding, this latter type of solutions are most suitable to meet requirements like 1) low latency and reactiveness (*e.g.,* anomalies and faults are detected as fast as possible); 2) reliability and cloud-independence (*e.g.,* the system is not dependent on any specific cloud endpoint or service provider); 3) guarantee privacy (*e.g.,* sensitive data like machinery vibrational data or e-health data are processed only locally); 4) reduced bandwidth (*e.g.,* it is to be considered unfeasible to stream all raw data generated by all sensing devices, thus aggregation, feature extraction, and fusion are tasks to be resolved locally within the edge of the network). It goes without saying that this approach initially has higher economic costs

with respect to cloud-centric approaches. However, these costs pay off later when no periodical payments of fees based on data volumes or transactions will be required.

2.6 Remarks

The rise of Fog Computing is remarkably changing the way IoT applications are designed. In this chapter, we have provided an overview of this novel computing paradigm from different perspectives. We have started our journey from the origins, trying to reconstruct the most pioneering steps made by the research community in this field. Then, we have dived into the main standardization initiatives, the most mature open-source solutions and the most advanced products/services already available on the market. In this latter case, we focused on the Fog/Edge Computing paradigm offered with an as-a-Service model. As we will present in the next chapters of this thesis, we expect that Fog and Edge Computing will play a key role in the development of the IoT solutions of the future, mainly because of its by-design capabilities of enabling lower-latency, more secure, more cost-effective, and more complex applications, hence unleashing the true potential of the AI.

Chapter 3

Design and deployment of an IoT application into the Cloud-to-Thing continuum

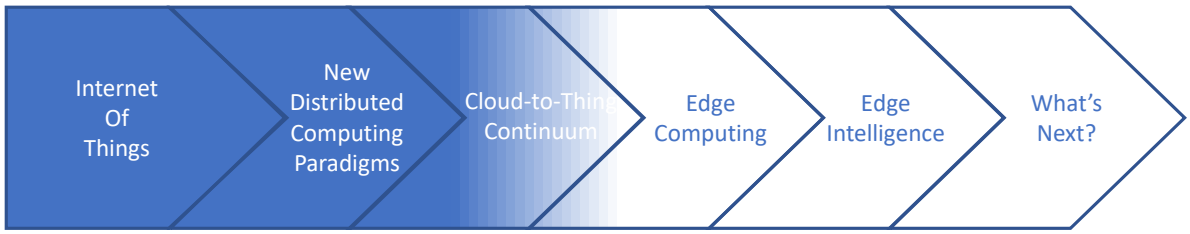


Figure 3.1: This chapter describes how to design, implement, and deploy an intelligent IoT application along the *Cloud-to-Thing continuum*. More in detail, this application makes use of a machine learning technique to perform data stream analytics at the edge without relying on remote cloud endpoints. This is a further step toward Edge Intelligence.

3.1 Introduction

The *Cloud-to-Thing continuum* and the Edge Computing paradigm [10] have gained much attention in recent years due to improved resources and

Part of this chapter appears in the following publications that I co-authored:
- Z. H. Janjua, M. Vecchio, M. Antonini, and F. Antonelli, "IRESE: An intelligent rare-event detection system using unsupervised learning on the IoT edge. *Engineering Applications of Artificial Intelligence*, vol. 84, pp. 41-50, Sep. 2019. Copyright Elsevier (2019). DOI: 10.1016/j.engappai.2019.05.011.
- M. Antonini, M. Vecchio, F. Antonelli, P. Ducange, and C. Perera, "Smart Audio Sensors in the Internet of Things Edge for anomaly detection". *IEEE Access*, vol. 6, pp. 67594-67610, 2018. DOI: 10.1109/ACCESS.2018.2877523.

increased processing power of edge devices available on the market. Today, this paradigm is extensively adopted in various applications such as smart homes, smart cities, smart health, and smart transportation. In these applications, data are directly processed by edge devices (*i.e.,* IoT gateways) with the aim to extract meaningful information, gain knowledge, and, finally, take the right action at right time without the support of any other remote entity. A real example is represented by Boeing 787 aircraft that generate around 5 GB of data every second [14]: it is unfeasible to move such an amount of data to cloud endpoints, indeed, these data are directly processed by the on-board avionics. Only a small amount of such data are streamed to remote entities (*e.g.,* health status of components). Therefore, one can assert that Cloud Computing alone is not efficient enough to handle the enormous amount of IoT data that will be generated in the coming years [41]. It is delineating the necessity to perform processing on edge IoT devices; near the source. The edge devices are becoming more powerful and resource-friendly with optimal utilization of resources such as memory and energy.
In many IoT applications, devices generate data streams that may contain interesting information thus data stream analytics plays a crucial role to discover interesting hidden patterns in *disorganized* and *unbounded* data streams. One of the most demanding tasks is to discover patterns reflecting short-duration abrupt changes in a data stream, which may indicate an unusual situation or event [42]. In literature, different terms are used for such short-duration abrupt changes including rare-event, anomaly, or outlier. Summarizing various definitions of these terms given in the literature [43, 44, 45], we can formally define a rare-event as follows:

Definition 1 *A rare-event, or an outlier, is an observation (or set of few observations) that occurs infrequently, deviates or, is inconsistent with respect to other observations so much that becomes suspicious. It may indicate an irregularity or an anomaly in the given set of observations.*

It is extremely important to detect rare-events occurring in data streams, as it may be helpful in detecting potentially hazardous situations. For example, a microphone is deployed in an outdoor environment (*e.g.,* a park), receiving typical city-related sounds (*e.g.,* cars, horns, birds, etc.). All of a

sudden, a siren is heard; this sound is very different from the background audio, and for this reason, it is considered a rare-event with respect to its background environment. Consider another example, a vibration sensor is deployed on a machine located in an industrial plant to continuously measure the vibrations generated by its motor(s); when the machine will start malfunctioning, an abnormal vibration pattern may be registered by the attached sensor, representing this a rare-event in the context of normal working conditions of that machine. Again, due to the *network bandwidth vs. data production rate* bottleneck, Cloud Computing may not be the right solution since it has limitations in rare-event detection, especially in time-critical applications. It may not be able to generate timely alerts. On the other hand, in Edge Computing, IoT data can be locally processed by an intelligent gateway.

In this chapter, we continue the migration (Figure 3.1) toward Edge Intelligence by presenting how to design a data streaming IoT application, which usually runs in the cloud, by executing almost all the required processing at the edge of the network without relying on remote cloud instances by exploiting the concept of *IoT gateway* [46]. We will describe the design process of IoT applications that need to locally process data streams, execute machine learning algorithms, infer AI models, and take actions. This will include also how to choose the main parameters of the application.
In order to support our design approach and provide a complete *proof-of-concept*, we propose an intelligent rare-event detection system suitable for the IoT Edge, which we called *IRESE*[1] [47]. The system uses unsupervised machine learning techniques to detect rare-events occurring in the incoming data streams. Figure 3.2 illustrates the overall concept of *IRESE*: the IoT devices continuously sense the environment, while an edge device (intelligent gateway) processes the incoming data streams with the goal of detecting rare-event instances and then transmit them to a cloud endpoint. As we will highlight also later, all the intelligence and the processing can be concentrated over one or more edge devices.

[1]The name *IRESE* recalls the name of the flower *Iris*, which is also the name of one of the most famous data-set used by entry-level data scientists to learn multivariate analysis techniques.

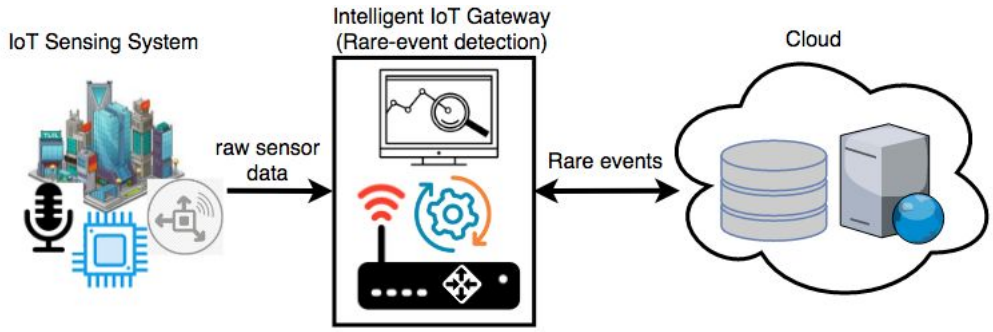


Figure 3.2: The conceptual diagram of *IRESE* (source: [47], p. 43).

3.1.1 Additional Contributions

Besides the main contribution to the design of IoT applications along the *Cloud-to-Thing continuum*, this chapter provides additional contributions as follows:

- IoT data stream analytics: we have strictly considered the limitations of IoT data stream analytics in the proposed model since they are continuous, high-speed, and unbounded. Given these unique characteristics, data streams need quick data processing without storing the data. We have used data stream processing machine learning algorithms that work in two stages: micro-clustering and macro-clustering [48]. In a nutshell, micro-clustering enables an edge device to quickly get summaries of the high-speed incoming data stream in real-time without storing it, whereas macro-clustering further processes micro-clusters to discover separate clusters of rare-events and normal events. We have practically deployed *IRESE* on an IoT gateway that continuously receives audio streams from microphones and detects rare-events that happened in an environment, *e.g.,* a gunshot.

- Detecting rare-event without prior knowledge: *IRESE* relies on a combination of unsupervised machine learning techniques. One of the challenging tasks in machine learning is to label data and provide it as training examples to a supervised machine learning algorithm. To overcome this problem and save the effort to manually label data, we adopted unsupervised machine learning techniques, which do not require labeled data and, once a rare-event is detected, we can further investigate its type. Unsupervised machine learning techniques

allow us to automatically detect hidden patterns of interest in data without having prior knowledge about these patterns. Given the potentially enormous amount of data generated by IoT and edge devices, this feature is quite appealing. We applied two techniques: BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) algorithm [49] to get micro-clusters and Agglomerative Clustering [50] is used to get macro-clusters from the input data stream. More details on the applied techniques are presented in Section 3.3.4.

Finally, this chapter is structured as follows: we present a literature review about anomaly detection, both in IoT and traditional contexts, with a focus on audio events in Section 3.2. We describe the principles behind *IRESE* in Section 3.3. Section 3.4 presents the experiments conducted to validate *IRESE*. Finally, we draw some final remarks in Section 3.5.

3.2 Literature review on anomaly detection

Several Anomaly Detection (AD) techniques have been proposed in the literature using different machine learning approaches based on unsupervised, supervised, and semi-supervised training algorithms.

Generally speaking, when dealing with unsupervised training algorithms, no labeled training datasets are adopted for building the models. Usually, only patterns describing normal behavior are available, while additional information regarding anomalies is not available. In these situations, algorithms based on clustering, data density and proximity, and one-class detection models may be adopted [51, 52, 53]. Among these algorithms, the one-class support vector machine (1-SVM) algorithm still continues to be one of the most adopted for unsupervised anomaly detection [54, 55]: a kernel model, namely a decision function, is derived by using normal data patterns, while new patterns projecting too far from the model are marked as anomalies. Moreover, also frequent pattern and association rule mining algorithms have been adopted for unsupervised anomaly detection: normal data vectors can be considered as transactions and frequent patterns and association rules can be mined. New transactions (*i.e.,* new measured data) that cannot be projected into the frequent patterns or the mined

rules are marked as anomalies [56, 57].

When a sufficient number of labeled training patterns is available, including both normal and anomalous situations, supervised learning algorithms can be employed. More in detail, multi-class classification models can be trained using a fully labeled training set. Classification models, such as decision trees [58], multi-class SVM [59], and Bayesian classifiers [60], have been successfully employed for different kinds of anomaly detection. A recent survey [61] compares a number of classification algorithms for a specific anomaly detection framework, namely intrusion detection in networks.

Finally, there exist specific situations in which only a low number of anomalous labeled patterns are available. In such cases, a semi-supervised training algorithm can be employed for learning the models for detecting anomalies. These kinds of algorithms usually are based on the hybridization of unsupervised and supervised algorithms [62]. As recently discussed by the authors of [63], a classification model (specifically, a neural network) is trained using the labeled patterns. Then, the unlabeled patterns, appropriately pre-elaborated using a fuzziness function, are classified and exploited for reinforcing the structure of the models. The work in [64] discusses the use of the so-called deep auto-encoder (DAE), a kind of deep belief network followed by an ensemble of K-NN classifiers. In particular, unlabeled data are used for training the DAE in order to reduce the dimensionality of the data [65]. The subset of labeled data, transformed by using the DAE, is used for training the ensemble of K-NN classifiers. New patterns are filtered by the DAE and then classified by means of the ensemble.

3.2.1 Audio anomaly detection and rare event detection

Focusing on audio anomaly and rare event detection, several novel techniques have been proposed within *Task 2* of the *DCASE 2017 Challenge* [66]. Many submitted techniques adopt deep neural network architectures [67, 68, 69, 70] to create classifiers able to detect the on-set time instant of rare-events (*e.g.,* gunshots, glass breaks, baby cries) over background audio. However, supervised algorithms can be adopted if and only if a labeled

dataset is available. Usually, these datasets are manually generated (*i.e.,* labeled) by researchers, but this is arduous and tedious work. Apart from being not affordable from time and money perspectives, this approach is not always feasible because some events are either extremely rare or unknown. For this reason, wherever possible, unsupervised approaches (*e.g.,* learning algorithms that can be trained using unlabeled datasets, because they can identify, extract, and learn patterns directly from data) are always advisable.

Oh et al. [71] propose an AD strategy based on an auto-encoder to detect audio anomalies produced by a Surface-Mounted Device (SMD) machine that places components on top of a Printed Circuit Board (PCB). The algorithm creates an auto-encoding manifold able to measure differences among instances and the manifold, signaling an anomaly if such distances are too large. Koizumi et al. [72] propose a similar AD approach based on an auto-encoder. They trained the unsupervised algorithm by optimizing an objective function formulated by starting from the Neyman-Pearson lemma. In order to pursue this way, they assumed that the AD task was a statistical hypothesis test.

Recently, Bose et al. [73] proposed a novel approach to Anomaly Detection on the IoT Edge. There, the authors describe a new computing schema, called Anomaly Detection based Power Saving (ADEPOS), to adaptively update an anomaly detector, through time, without losing detection accuracy. The authors validated their approach by implementing a system to detect anomalies and failures of rotating bearing equipment by analyzing some time-based features extracted from vibrations. This technique consists of a group of one-class classifiers, which detect if an anomaly happened or not, followed by a majority voting strategy. ADEPOS is used to vary the number of detectors in the ensemble. Moreover, they evaluated the power saving of ADEPOS by simulating it in a Very Large Scale Integration (VLSI) hardware architecture. However, ADEPOS and *IRESE* have two different targets: the former aims to create adaptive anomaly detection systems, based on edge devices, that require a small amount of energy. *IRESE* aims to create an Audio Rare-Event Detection system (Audio Anomaly Detection system), based on unsupervised machine learning, that runs on an IoT Gateway.

Another class of techniques that allow anomaly detection in audio streams is the semi-supervised learning algorithms. Aurino et al. [74] propose a 1-SVM approach within an automatic surveillance framework to detect burst-like audio events, namely screams, gunshots, and glass breaks. Such an approach uses a two-stage classification scheme: the first stage classifies short audio segments (200 ms) through an ensemble of 1-SVM classifiers, while the second stage composes and re-classifies the first stage's decisions using a majority strategy, in order to take one decision per second. Elizalde et al. [75] present a framework to train audio event detectors using a semi-supervised self-training approach. Audio Event Detectors have to be firstly trained on the UrbanSound8K dataset [76], then have to run on unlabeled audio streams extracted from YouTube videos. If the detector recognizes a known sound with a high level of confidence, it uses that sound to re-train the model. This approach helps to train models with acoustic diversity even if the original dataset is relatively small.

3.2.2 Audio Anomaly Detection in IoT contexts

Anomaly Detection algorithms have been adopted also in IoT contexts, by creating more intelligent, reactive, and secure environments. Hilal et al. [59] present and describe a Sensor Management framework called *IntelliSurv*. It realizes an acoustic surveillance system that follows the pervasive IoT paradigm, being able to detect and localize anomalous audio events using different kinds of distributed devices: smart sensors for environmental monitoring, and delegate sensors devoted to sensor management, localization, and identification of anomalous events. Moreover, all the smart sensors have enough computing capabilities to locally execute the abnormality detection. At the classification stage of events, the authors adopted SVM and LDA models.
Socoró et al. [77] propose an *Anomalous Noise Event Detector (ANED)* algorithm to map the traffic noise in urban and sub-urban areas using low-cost wireless sensor networks. These networks are composed of smart devices that perform simple signal pre-processing, then execute event detection using machine learning algorithms and, finally, they send labels to a central server that updates and draw noise maps. The authors there

adopted a two-class classification scheme to distinguish the anomalous traffic noise (*e.g.,* jammed or semi-jammed traffic) from the normal traffic noise. They discovered that this approach performs better than the one using the one-class classifier, but they had to manually annotate the dataset. This system has been conceived using some outcomes from the European Project called DYNAMAP[2].
Alsina-Pagés et al. [78] present an Ambient Assisted Living (AAL) system, called *homesound*, that is able to detect and recognize different audio rare events happening in an everyday environment. This system uses a wireless sensor network to record audio from the environment; then the sensors forward the sampled audio streams to a GPU-based central device, which has two roles: first, it performs feature extraction from the raw audio stream, by computing 48 Mel Frequency Cepstral Coefficients (MFCC) and considering only the first 13 coefficients; then, it executes the inference of data using the trained model that is based on a classification algorithm (SVM) and clustering algorithm. The model response is finally sent to a remote system, where the medical staff can monitor the patient status.
Finally, even though the work is not properly focused on IoT architecture, it is worth discussing the issues regarding AAD arisen in [79]. This work deals with important challenges in AAD, namely intra-class variations, such as the different duration for the same sound type, and spectral-temporal properties across classes, which include impulse-like sounds, tonal events, and noise-like events. Among the latter types of sounds/events, we can found, respectively, door slams, phone rings, and printer sounds. In particular, the authors propose the use of both contextual information and prior knowledge of the event category and the event boundary. Random forests are employed as models for anomaly detection.

3.3 The *IRESE* framework

In principle, the proposed model involves IoT devices that are deployed in an environment to measure signal energy through its transducer. The considered environment could be indoor or outdoor and it should have uniform

[2] `http://www.life-dynamap.eu/`

characteristics and does not suppose to have frequent abrupt changes. We considered sensors that can produce a continuous waveform from measured quantities such as acoustic events, vibrations, or accelerations. Measured quantities must be represented as a waveform as *IRESE* performs complex spectrum analysis to detect rare-events. Figure 3.3 shows the overall architecture of the rare-event detection system which is deployed on an edge device. IoT devices (for example, sensing devices) generate a data stream $D[n]$ sampled at sampling frequency f_s, where f_s satisfies Nyquist-Shannon sampling theorem: $f_s \geq 2f_{max}$, f_{max} represents the maximum frequency that occurs in the signal. An unbounded time-series data stream is represented as a discrete signal: $D[n] = x_n, x_{n-1}, ..., x_{n-t}, ...$, where x_n is the current sample, and x_{n-t} is the $t - th$ recorded sample. Since the data stream is unbounded, we need to buffer it to hold it for a small duration for further processing.

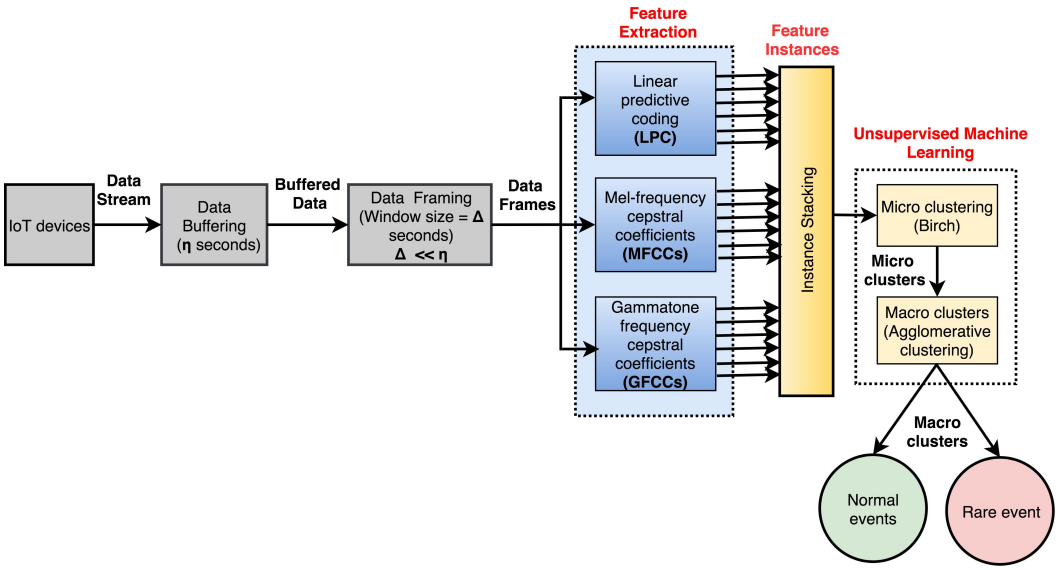


Figure 3.3: Logical schema of the rare-event detection system proposed in the *IRESE* framework (source: [47], p. 44).

3.3.1 Data Buffering

The incoming data stream $D[n]$ is periodically buffered in the local memory of an edge device. Each cycle has a fixed duration, in which data are buffered during a short interval of η seconds, for example, 60 or 120 seconds.

Data buffering is required because it allows applying *IRESE* on a longer time window instead of on each sample data. Single sample processing may not be feasible since *IRESE* requires some time to process samples. We will deeply discuss this problem in Section 4.4.
The buffering time η could vary according to the type of data generated by IoT devices, however, it remains fixed for a given setup. The buffered data stream is, then, supplied to the *Data Framing* module that splits data into small frames suitable for subsequent feature extraction techniques.

3.3.2 Data Framing

The data framing module takes buffered data and breaks them into smaller frames of duration Δ seconds, where $\Delta << \eta$, for example, Δ is 1 second when η is 60 seconds. For data framing, we defined a fixed-length rectangular window of Δ seconds. The rectangular window function is represented in Formula (3.1). It is a *tumbling* window that moves over the buffered data stream in a way that two consecutive windows do not overlap with each other. For example, the buffer holds data for 60 seconds then the data framing module breaks this buffered data into 60 equal-sized frames by using a fixed window of size $\Delta = 1$ second.

$$\Pi(n) = \begin{cases} 1 & \text{if } 0 \leq n < i \cdot \Delta \cdot f_s \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

By multiplying the data stream $D[n]$ with the rectangular window function $\Pi(n)$ of Formula (3.1) (element-wise multiplication), we obtain the i^{th} frame $F_i[n]$, also represented as:

$$F_i[n] = D[n] \cdot \Pi\left(n - (i-1) \cdot \Delta \cdot f_s\right) \tag{3.2}$$

where $F_i[n] = x_{n-(i-1)\cdot\Delta\cdot f_s}, ..., x_{n-i\cdot\Delta\cdot f_s+1}$ contains a sequence of samples selected during the interval starting at $n - i \cdot \Delta \cdot f_s + 1$ and ending at $n - (i-1) \cdot \Delta \cdot f_s$ time instant. If we consider the first frame ($i = 1$), $F_1[n] = x_n, ..., x_{n-\Delta\cdot f_s+1}$. Clearly, each frame contains $\Delta \cdot f_s$ samples, where f_s is the sampling frequency.

3.3.3 Feature Extraction

We have considered both time and frequency domain features to effectively and accurately detect abrupt changes visible in time or frequency domains. In order to preserve the time-domain envelope of the signal, we have used Linear Predictive Coding (LPC) [80], which is a well-known technique used for feature extraction from audio and speech signals [81]. For the frequency domain analysis, we selected Mel-frequency cepstral coefficients (MFCCs) [82, 83] and Gammatone frequency cepstral coefficients (GFCC) [84]. MFCC and GFCC filter banks uniquely characterize the input signal to detect a rare event. Thus, the feature vector ν is a tuple that is composed of subset features: $L_p(LPC)$, $M_f(MFCC)$, and $G_f(GFCC)$, which can be represented as

$$\nu = \{L_{p1}, L_{p2}, ..., L_{pI}, M_{f1}, M_{f2}, ..., M_{fJ}, G_{f1}, G_{f2}, ..., G_{fK}\}$$

where I, J, and K are the numbers of LPC, MFCC, and GFCC coefficients in each tuple ν.

3.3.4 Unsupervised Machine learning

In Section 3.2, we have discussed in detail the application of machine learning techniques in anomaly detection. However, in this chapter, we want to design a data stream IoT application that able to automatically extract patterns of rare-events occurring in an IoT data stream using an edge device. If we would apply a supervised machine learning technique, we should individually label patterns of rare-events exhibited by extracted features. However, data labeling is a difficult, tedious, and time-consuming task since it requires a domain expert who closely observes incoming instances and assigns them meaningful labels [85].

In order to avoid wasting the effort involved in data labeling and to automatically find the patterns of rare-events hidden in a data stream, we have used a two-stage rare-event detection strategy that relies on a combination of state-of-the-art unsupervised machine learning techniques. As shown in Figure 3.3, the unsupervised machine learning module takes stacked feature instances as input and processes them using a two stages technique to detect the occurrence of a rare event. Here it is important

to highlight the working of an unsupervised machine learning technique, which basically aims to partition data instances in a way that similar instances are assigned to the same cluster [86]. On the other hand, dissimilar instances belong to different clusters. Exploiting the fact that patterns of rare events are reasonably different from normal events, *IRESE* tries to find two separate clusters in the incoming data stream. One cluster, *i.e.,* the *normality cluster*, should contain instances of *normal* events whereas the other cluster, *i.e., the outlier cluster*, should contain instances associated with rare events.

The two-stage strategy is used due to the one-pass constraint of a high-speed incoming data stream [87]. It is not possible to store such a high-speed data stream due to a lack of resources and the amount of data produced. The incoming data stream is processed in two stages: the online micro-clustering stage and the subsequent offline macro-clustering stage [48, 88]. In the first stage, online micro-clustering, the high-speed data stream is processed in real-time to quickly extract statistical information from it in the form of micro-clusters. Micro-clusters could indicate the presence of rare-event patterns in the data stream. Therefore, instances belonging to such clusters are further processed in the second stage, *i.e.,* offline macro-clustering, that extracts rare-events from the incoming data stream. As mentioned above, the final output is in the form of two clusters: *the normality cluster* is dense and containing data points reflecting normal behavior, whereas *the outlier cluster* containing a rare-event (if it exists) which is an outlier and different from other events occurring in that specific interval of buffered data. Further detail of both stages is described in the following subsections.

Micro-Clustering

Since data streams are unbounded and having a large amount of data, an efficient method is required to extract important statistics from the data in real-time. An online micro-clustering [48, 88] technique considers the *one-pass* nature of streaming data and attempts to quickly and efficiently collect the useful summary of data. One-pass means that it is not suitable to store raw data and it must be efficiently processed in the first attempt

to get meaningful information from it. The outcome of micro-clustering is several small clusters having unique properties due to the similarity between instances observed during the small time duration. There are several stream clustering techniques available for online micro-clustering which are compared in [88]. We have used the BIRCH (acronym of Balanced Iterative Reducing and Clustering using Hierarchies) algorithm, which is a tree-based stream clustering algorithm proposed in [49]. The algorithm constructs a *clustering feature (CF)* tree for incoming data instances, in which leaf nodes are micro-clusters. BIRCH is a fast and memory-efficient algorithm and these characteristics make it suitable to be executed by an edge device.

Macro-Clustering

In the offline macro-clustering phase [48, 88], micro-clusters are further processed and merged to produce bigger clusters. The merging of clusters is based on the distance between the cluster centroids. Hence, the clusters having centroids close to each other are merged to form a single bigger cluster. Keeping in mind that a rare-event has distinctive features, the outlier cluster remains isolated from the other clusters. We have used the *Agglomerative Clustering* [87, 89] method and the *Ward method* [50] to recursively merge micro-clusters by minimizing variance among them.

Figure 3.4 shows the overall process of cluster merging. Following the Ward algorithm, note that d is the squared Euclidean distance between the centroids of any two given micro-clusters. A low value of d shows that two micro-clusters are close to each other having similar characteristics, whereas a high value of d means that two clusters are far from each other due to their varying characteristics. The algorithm recursively merges any two given micro-clusters at each step while optimizing the *objective function*, which tries to minimize the total with-in cluster variance. The algorithm continues the merging process until only two clusters remain: the normal events (*normality cluster*) and rare-events(*outlier cluster*), which may not exist. The objective function considers a threshold value Th that decides whether two micro-clusters are close enough to be merged into a bigger cluster or not. Theoretically, increasing the value of Th expands

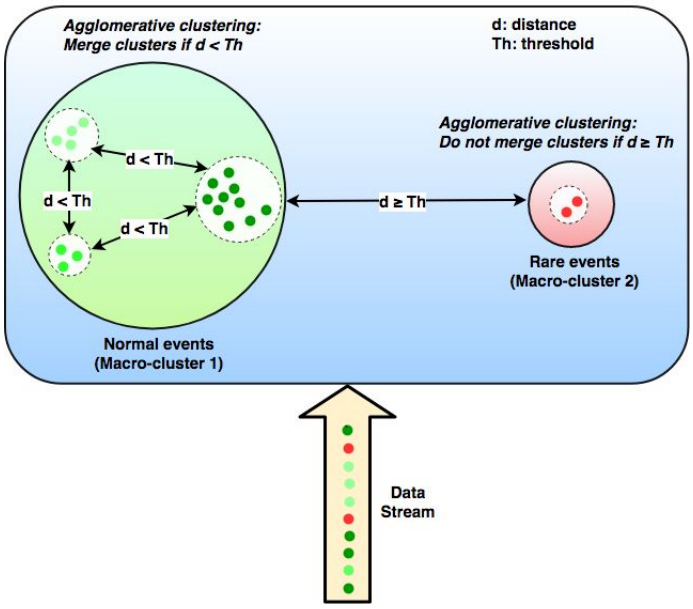


Figure 3.4: Macro-cluster formation in *IRESE* (source: [47], p. 46).

the size of a recursively merged cluster while reducing the detection rate of a rare-event, whereas decreasing the value of Th results in recognizing a normal event as a rare-event. The Th value is strictly related to the environment where the algorithm is deployed and the type of rare-event we want to detect. We have performed (Section 3.4) some empirical analysis of received data in order to identify the best value of Th for different rare-events: gunshot, scream, siren, and glass breaking.

3.4 Experimentation

In this section, we quantitatively assess our proposed approach, thus we have conducted experiments with a typical use case involving the processing of audio data containing rare-events. We can safely extend our hypothesis that *IRESE* can be applied in other similar use cases that involve data streams from IoT devices having similar temporal and spectral characteristics. For example, another suitable scenario is the detection of faults in the machines using vibration and acoustic sensors. This section presents experiments conducted to detect various types of rare-events.

3.4.1 Experimental setup

In this chapter we are trying to deploy a data stream IoT application along the *Cloud-to-Thing continuum* where, typically, devices have constrained computing and networking capabilities, allowing to reduce costs and energy consumption since they are often battery-powered. In many scenarios, IoT devices need an external entity deployed at the edge of the network, called IoT gateway, that is able to execute computing- and networking-intensive operations, *e.g.,* bridging different networking stacks like Bluetooth and Wi-Fi. One of the players in the open-source landscape is the Adaptive Gateway for dIverse muLtiple Environment (AGILE) [46]. AGILE is a modular software framework for IoT gateways with wide support for many network stacks and devices. Moreover, AGILE has been designed by following the micro-service paradigm [37] that was initially conceived for distributed systems. This paradigm defines that all modules of the system are independently designed and implemented and they are able to interact among them using a well-defined set of Application Programming Interfaces (APIs). This enables strong modularity, resiliency against failures, scalability, reliability, and simpler maintenance of the whole system. The paradigm has been successfully applied to different domains (*e.g.,* Cloud Computing). The AGILE software framework will be presented in detail in Section 4.3.2.

The framework for audio rare-event detection presented in this chapter has been implemented as an independent micro-service within the AGILE gateway framework. Since this module requires a raw audio stream in order to extract features, the AGILE gateway board, *i.e.,* a Raspberry Pi, is connected to a USB microphone. This microphone is recognized as a classic microphone by the gateway operating system. The micro-service records the audio stream from the microphone, then it performs data buffering (Section 3.3.1) and windowing (Section 3.3.2). Thus, it extracts features over a temporal frame by computing MFCC, LPC, and GFCC coefficients. Consequently, the module feeds the algorithm with the feature vector and finally verifies if the anomaly happened or not by checking which cluster, normality or outlier, contains the audio frame.

This module can be used with two different data sources: recorded (from

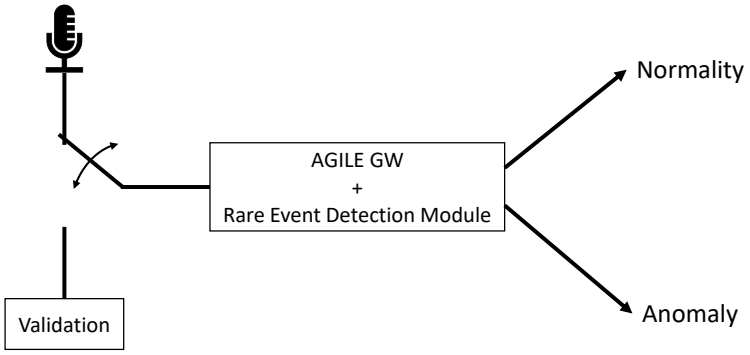


Figure 3.5: Functional schema, implementation of *IRESE* with AGILE (source: [47], p. 48).

microphone) audio stream and validation audio stream. Figure 3.5 shows how it is possible to choose the data source. If we select the *recorded audio stream*, the system behaves as presented above. If we choose the latter stream, the module loads the audio stream from WAV files stored in the SD card of the gateway. Using this data source, we can evaluate the system performances as will be described in Section 3.4.4.

3.4.2 Software tools

The overall system is implemented in Python. Three types of features are extracted using three python libraries: 1) LPC features are extracted using audiolazy[3] python library; 2) MFCC are extracted using librosa [90]; 3) GFCC are extracted using gammatone python library[4]. We have adopted the implementations of BIRCH and Agglomerative Clustering available within the scikit-learn [91] library.

3.4.3 Dataset

There are several datasets available for various audio events — UrbanSound8K [76], TUT Sound Events [92], and Audio set by Google [93], just to cite a few. In [93], authors have provided a resourceful compilation of various audio datasets which include tagged and mixed audio events. Since we are using unsupervised machine learning to detect rare-events, we need a dataset with labeled time stamps of various rare-events over

[3] `https://pythonhosted.org/audiolazy/`
[4] `https://github.com/detly/gammatone`

normal background audio signals. In DCASE 2017 Challenge [66], authors produced a dataset with various backgrounds for three events: gunshot, glass break, and baby cry. However, in our understanding, their mixture model is not suitable for our case study, as we are looking for relatively more prominent rare-events from different sources and having varying characteristics that could highlight the seriousness of the situation. For this purpose, we crafted a dataset by mixing various rare-events, from multiple sources, with different backgrounds. To ensure the relatedness of this chapter with the state-of-the-art research happening in the domain, we rely on already published datasets to produce our mixture models. Therefore, we collected background sounds from DCASE 2017 Challenge dataset [66], and selected a subset, containing several variations, of rare-events from UrbanSound8K [76] or downloaded directly from the Freesound[5] search engine. We have considered *four* types of rare-events: *gunshot*, *glass break*, *scream*, and *siren*. Furthermore, the sounds in each type of rare-event are also different from each other. We created a dataset of 160 samples per rare event resulting in an overall dataset of 640 audio samples. We used the Pydub[6] library to create each sample by mixing rare-events and background sounds. Each sound clip contains exactly one rare-event and the insertion point of the rare event is randomly selected. We resampled samples at 44.1 kHz, which meets the standard audio sampling rate of compact-disks (CDs). Since we are simulating an IoT environment, we can safely assume that these sounds are similar to those received by a microphone deployed in the environment. As illustrated in Figure 3.3, the data received from the IoT devices are temporarily stored in a buffer for few seconds. The buffer size is variable, however, it remains fixed for a given environment. In these experiments, we have considered the buffer size equals 30 seconds, which is simulated by taking 30 second sound clip each time. The 30 seconds sound is further split into frames, and for each frame, features are extracted. We already discussed in detail the feature extraction method in Section 3.3. However, here it is important to mention the number of coefficients, we have considered for each of three types of feature extraction methods: LPC, MFCC, and GFCC. We have taken 10 coefficients of LPC,

[5] `https://freesound.org/`
[6] `https://pydub.com`

40 MFCCs, and 40 GFCCs. Thus, in total, the length of the feature vector is 90 and each value is represented as a floating-point variable.

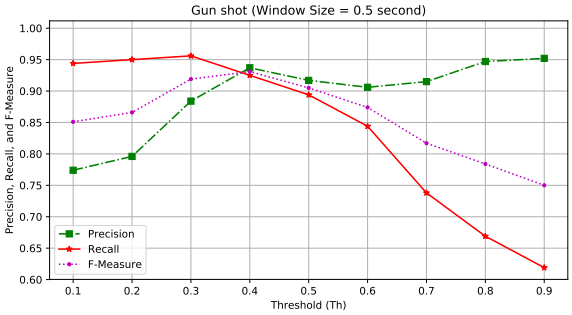
3.4.4 Experimental results

In this section, we will explain empirical results obtained while conducting experiments using *IRESE* on the dataset described above. The two-staged unsupervised machine learning strategy of *IRESE* ultimately produces two clusters: a cluster of normal events, and a separate cluster of rare-event, if it exists. As mentioned in the previous section, we have synthetically constructed the dataset, in which we have added a rare-event sound at a random time instant in a background sound of relatively longer duration. For the sake of the evaluation, we recorded the time instant at which we added a rare-sound in a background sound clip. The recorded information is used to evaluate the performance of *IRESE* by comparing the time instant, called "On-Set" time, at which *IRESE* detects a rare-event to the actual time instant when the rare-event occurred according to records.

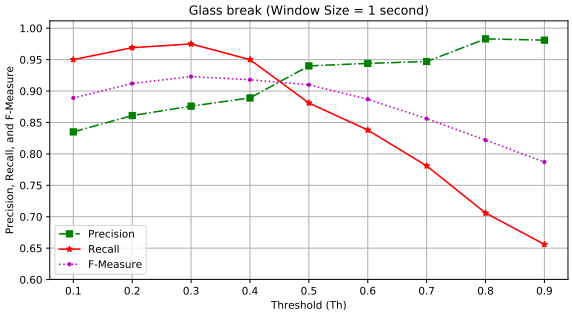
In order to evaluate the model, we have used matching matrix values: True Positive (TP), False Positive (FP), and False Negative (FN). In these experiments, a *TP* occurs when *IRESE* correctly separates a rare-event observation from the rest of the observations. A FP occurs when *IRESE* wrongly detects a background sound or a normal event as a rare-event, whereas a FN occurs when *IRESE* fails to distinguish between a rare-event and background sounds. Additionally, we have also calculated precision (P), recall (R), and f-measure (F1) values, where P gives us the positive predictive value, R gives us true positive rate, and F1-score gives us the harmonic mean of P and R values. In conclusion, the value of P decreases with an increase in the number of FP and, similarly, the value of R decreases as the number of FN increases. Following equations are used to calculate these measures:

$$P = \frac{TP}{TP + FP} \tag{3.3}$$

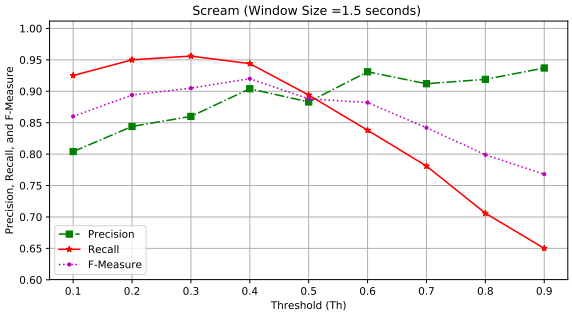
$$R = \frac{TP}{TP + FN} \tag{3.4}$$



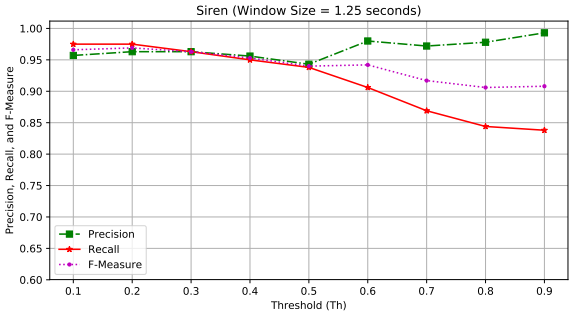
(a) Rare event: gunshot.



(b) Rare event: glass break.



(c) Rare event: scream.



(d) Rare event: siren.

Figure 3.6: These plots present the precision, recall, and F-measures computed at different value of Th for four different rare-events: gunshot, glass break, scream, and siren (source: [47], p. 48).

$$F1 = 2 \cdot \frac{P \cdot R}{P + R} \tag{3.5}$$

Figure 3.6 shows different plots of P, R, and F1-score values against the threshold (Th) values discussed in Section 3.3.4 using a specific window size. It is clearly observable that as threshold Th increases the precision increases and recall decreases. It confirms the trend that the rare-event detection rate decreases with the increase in Th value, whereas more false predictions are produced with low values of Th. The reason is that the boundary of the cluster defining normal events grows with the value of Th. Consequently, at a certain point, the size of the normal cluster grows so much that even an anomalous observation (occurring at a relatively larger distance) becomes part of the normal cluster which increases the number of FN. We have selected an optimum value of Th, which could be observed in

the graphs, where the combination of all three values (P, R, and F1-score) is highest. Thus, for gunshot the optimum value of Th is 0.4 by using a window size of 0.5 seconds, for glass break the optimum value of Th is 0.45 by using a window size of 1 second, for siren the optimum value of Th is 0.3 by using a window size of 1.25, and for scream event the optimum value of Th is 0.4.

Table 3.1: Detection of different rare-events against different window sizes performed by *IRESE* over the dataset (source: [47], p. 48).

	Gun shot (Th=0.4)			**Glass break (Th=0.45)**			**Siren (Th=0.3)**			**Scream (Th=0.4)**		
Window Size (s)	TP	FP	FN	TP	FP	FN	TP	FP	FN	TP	FP	FN
0.25	151	21	9	156	36	4	156	11	4	157	46	3
0.5	**148**	**10**	**12**	153	32	7	155	11	5	157	28	3
1	143	11	17	**149**	**12**	**11**	154	6	6	157	20	3
1.25	125	27	35	136	18	25	**154**	**6**	**6**	151	19	9
1.5	116	28	44	124	27	36	152	7	8	**151**	**16**	**9**
2	103	26	57	102	23	58	152	6	8	145	13	15

Table 3.1 shows the values of TP, FN, and FP metrics for the four types of events. Each row in the table represents the results obtained for the given window size. Notice that window size is the size of an individual frame, as defined in Equation (3.1). We can observe a trend in values that TP decreases as window size increases, and it is true for all the cases. Consequently, FN increases as the window size increases, whereas FP does not follow a specific trend; it is probably due to using different background sounds which may contain some sounds similar to the rare-events.

Table 3.2: Performance evaluation results using *IRESE* for rare-event detection (source: [47], p. 49).

	Gun shot (Th = 0.4)			**Glass break (Th =0.45)**			**Siren (Th=0.3)**			**Scream (Th=0.4)**		
Window Size	P	R	$F1$	P	R	$F1$	P	R	$F1$	P	R	$F1$
0.25	0.87	0.94	0.91	0.81	0.97	0.88	0.93	0.97	0.95	0.77	0.98	0.86
0.5	**0.93**	**0.92**	**0.93**	0.82	0.95	0.88	0.93	0.96	0.95	0.84	0.98	0.91
1	0.92	0.89	0.91	**0.92**	**0.93**	**0.92**	0.96	0.96	0.96	0.88	0.98	0.93
1.25	0.82	0.78	0.8	0.88	0.85	0.86	**0.96**	**0.96**	**0.96**	0.88	0.94	0.91
1.5	0.8	0.72	0.76	0.82	0.77	0.79	0.95	0.95	0.95	**0.9**	**0.94**	**0.92**
2	0.79	0.64	0.71	0.81	0.63	0.71	0.96	0.95	0.95	0.91	0.9	0.91

While looking at Table 3.2, we can estimate a suitable window size to detect a particular type of rare event and using an optimum value of Th. The precision (P) increases as the number of FP decreases, whereas recall (R) increases as the number of FN decreases. In general, we can observe that the suitable window size varies from one event to another and it

depends on the duration of occurrence of a particular event. In summary, window size 0.5 seconds show the optimum detection performance with $P = 0.93$, $R = 0.92$, and $F1 = 0.93$. For the glass break, the optimum window size is 1 second with a $P = 0.92$, $R = 0.93$, and $F1 = 0.92$. Detection of sirens performs better with a window size of 1.25 seconds with all P, R, and F1-score equals to 0.96. The largest suitable window size is observed for scream which is 1.5 seconds with a $P = 0.9$, $R = 0.94$, and $F1 = 0.92$.

Table 3.3: Performance evaluation using only macro-clustering (no *IRESE*) for rare-event detection (source: [47], p. 49).

	Gun shot			**Glass break**			**Siren**			**Scream**		
Window Size	P	R	$F1$	P	R	$F1$	P	R	$F1$	P	R	$F1$
0.25	0.66	0.96	0.79	0.63	0.98	0.77	0.9	0.97	0.93	0.66	0.96	0.78
0.5	**0.76**	**0.94**	**0.84**	0.72	0.97	0.83	0.95	0.97	0.96	0.78	0.96	0.86
1	0.76	0.91	0.83	**0.83**	**0.95**	**0.89**	0.94	0.97	0.95	0.81	0.98	0.89
1.25	0.63	0.84	0.72	0.76	0.88	0.82	**0.95**	**0.97**	**0.96**	**0.83**	**0.96**	**0.89**
1.5	0.64	0.81	0.72	0.73	0.86	0.79	0.96	0.95	0.95	0.79	0.92	0.85
2	0.68	0.74	0.71	0.71	0.78	0.75	0.96	0.95	0.95	0.8	0.9	0.84

In our understanding, this variation in optimum window sizes is due to the duration of rare events. For example, a gunshot sound is sudden and exists for a very short duration such as between 0.5 seconds to 1 second. On the other hand, the sound of a scream normally lasts longer (up to a few seconds) such as 1.5 seconds or 2 seconds, which is also obvious from the results.

In order to prove the significance of *IRESE*, we have also calculated the results of rare-event detection using only the Agglomerative Clustering technique (*i.e.,* macro-clustering stage). Note that the two-stage strategy, micro-clustering followed by macro-clustering, improves the rare-event detection rate, which is obvious by comparing the results presented in Table 3.2 and Table 3.3. While using *IRESE*, we can see an improvement in all three calculated (P, R, and F1-score) values for different rare-events with different window sizes. Besides this improvement, the major benefit we achieve with *IRESE* is its suitability to be used in data stream IoT applications deployed along the *Cloud-to-Thing continuum*. The micro-clustering stage of *IRESE* is able to quickly extract the statistical information from an incoming high-speed data stream, without storing the data. Later on, this statistical information is further processed to make macro-clusters, which eventually indicates the presence of rare-events in the in-

coming data stream. Hence, *IRESE* is an effort to provide a solution to detect rare-events without storing incoming data on an edge device and it also empowers an edge device with Artificial Intelligence to reduce the burden on a cloud; sending only the patterns of interest to the cloud.

3.5 Remarks

This chapter shows how to deploy intelligence along the *Cloud-to-Thing continuum*, or better close to the data sources at the edge of the network. Moreover, it enables faster and more reliable IoT applications that consume high-speed data streams. In this context, rare-event detection using an edge device is a promising research area. We proposed *IRESE* that has shown a significant performance to detect various types of rare-events: gunshot, glass break, siren, and scream. We have used a two-stage unsupervised machine learning strategy, deployed on an IoT gateway, that allows the system to detect interesting patterns in the form of rare-events without having any prior knowledge.
This is just an example of what is possible to design and implement on the *Cloud-to-Thing continuum*. This approach allows to create new applications and tackle different issues that usually affect cloud-based applications. First of all, considering that we are dealing with an audio stream, privacy is preserved by design since all the processing is locally done. Eventually, only the information 'a rare-event happened' may be streamed to a cloud endpoint instead of the entire audio stream/sample. Moreover, given the local processing, the application does not require to send data to another entity and this saves bandwidth, energy and reduces the latency. For instance, the gateway may be connected to the Internet via LoRAWAN technology that permits streaming only a few bytes per day, *e.g.,* the rare event flag or timestamp. Additionally, intelligence deployed at the edge of the network can be fine-tuned to the target environment and specific deployment.
This chapter does not aim at providing all the details about the design of intelligent IoT applications at the edge of the network but only to establish the foundations for the next chapters. Indeed, this is just the tip

of the iceberg. In the next chapter, we will make a further step toward Edge Intelligence. We will present how to design an IoT application that will perform processing also at the very edge of the network, *i.e.,* on the device/sensor. This will allow to fine configure data-sampling, processing, and networking parameters (*e.g.,* sampling frequency, windowing, number of features, I/O baud-rate, etc.). In this way, we will save even more energy, bandwidth, reduce the latency. To provide scientific validation of our proposed framework, we will validate our methodology on another audio rare-event application where the processing is more spread at the edge.

Chapter 4

Design of an IoT device for Edge Computing applications

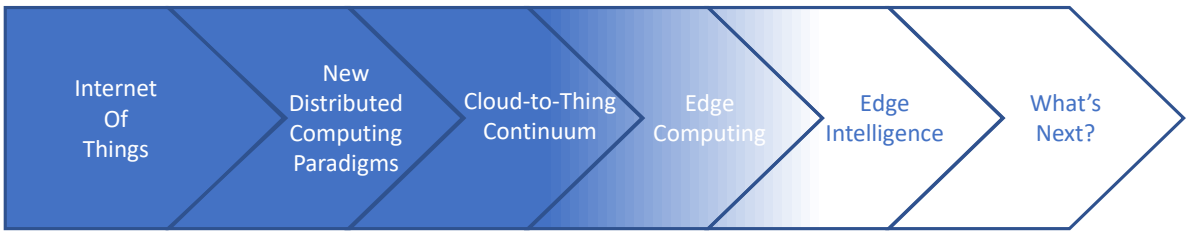


Figure 4.1: This chapter describes how to design an IoT application at the very edge of the network. Here, the processing is pushed even toward the very edge of the network, *i.e.,* the device. This unlocks new dimensions of design since allows to distribute the workload on many entities based on constraints, with subsequent savings of energy and bandwidth, the possibility to reconfigure the device processing algorithms without any human intervention, enabling also deployment in hazardous (*e.g.,* industrial) or sparse environments (*e.g.,* agriculture). Here, the intelligence is still resident on the gateway, however, small portions of it may be deployed on the device itself.

4.1 Introduction

Edge Computing is revolutionizing the Internet of Things and how we interact with the physical and virtual, world. Unlocking new design dimensions, like the possibility to deploy intelligence closer to the data sources, Edge

Part of this chapter appears in the following publication that I co-authored:
M. Antonini, M. Vecchio, F. Antonelli, P. Ducange, and C. Perera, "Smart Audio Sensors in the Internet of Things Edge for anomaly detection". *IEEE Access*, vol. 6, pp. 67594-67610, 2018. DOI: 10.1109/ACCESS.2018.2877523.

Computing enables new scenarios and new applications even if it is not a novel paradigm [10].
In previous chapters, we introduced the definitions of Edge Computing, Fog Computing, and *Cloud-to-Thing continuum*; the technological landscape (Chapter 2) from the academic point of view, standard definitions perspective, commercial products, open-source projects, etc.; and how to design an IoT data stream application at the edge of the network (Chapter 3). We also presented how to empower the edge with AI by deploying a machine learning technique on an edge device to perform anomaly detection over an audio stream proving. In such an application, the edge device, *i.e.,* an IoT gateway, was in charge of all the processing and acting as the core of the application, instead, other IoT devices were simply data collectors pushing data to a sink. Up to now, we have moved the computation from the cloud to the edge of the network.
In this chapter, we push forward the design of Edge Computing IoT applications by spreading the computation on multiple entities: devices, gateways, and so. We also move toward the concept of "embedded intelligence", the deployment of AI methods on embedded devices. Figure 4.1 provides a snapshot of where we stand with respect to Edge Intelligence.
To support our transition, we consider three new different aspects:

- the possibility to design and deploy processing algorithm on the very edge of the network, *i.e.,* on IoT devices and not only the gateway (which is still at the edge);
- the distribution of the computational workload (processing algorithm, AI algorithms, etc.) on different entities (*i.e.,* devices and gateways) based on the computational (CPU, memory, storage, ...), networking, and energy constraints. For the sake of this chapter and to provide the foundations, we will consider a static allocation based on some system constraints;
- the possibility to simply reconfigure IoT devices in order to change and adapt their behavior to the application's requirements (*e.g.,* level of accuracy or detection rate, energy budget, etc.).

Again, the aspects presented in previous chapters, like reduction of latency,

improvement of privacy, flexibility, reliability, saving of bandwidth, and so on are still valid and we will build our new concepts on top of these pillars. Here, we will develop the key elements to move a step in the directions presented above by presenting a framework to design Edge IoT devices for Edge IoT applications. To support our framework, we introduce a cost-effective, smart IoT device to autonomously perform anomaly detection on-site, called Smart Audio Sensor or SAS [41]. This device samples an audio stream from a transducer, *i.e.,* a microphone, executes some algorithms over incoming data (*e.g.,* compress information) before sending them to an edge device, *i.e.,* a gateway, which is in charge to execute an anomaly detection algorithm [94]. In the architecture we are going to present, the AI algorithm will still be resident on the gateway, however, part of the feature extraction processing is deployed over multiple devices, allowing scalable and resilient deployments. Only pre-processed data will be streamed to the IoT gateway.

This chapter, whose main goal is to prove the effectiveness of distributing the intelligence of a real Edge IoT application comprising several technological entities (*e.g.,* sensors, embedded devices, gateways, software platforms, etc.), provides the following main contributions that can be summarized as follows:

- *Design Framework for Smart Audio Sensors*: in this chapter, we analyze the bandwidth constraints imposed by a Bluetooth-UART interface. Since one of the main features of a Smart Audio Sensor (SAS) device is wireless connectivity, we introduce a theoretical framework to tune its parameters, taking into consideration both latency and bandwidth constraints. Thus, based on the specific requirements of the application scenario at hand, the SAS can be dynamically reconfigured to meet these requirements, without breaking any technology constraint.

- *Distribution of processing at the edge*: SASs autonomously sample the surrounding audio environment using cheap onboard microphones. Then, instead of transferring the digitized version of the acquired signal as is (as a "dumb" data producer would do), they first preprocess the signal in real-time by transforming it into the frequency domain

and by computing some state-of-the-art features in this domain. The processing output is a more compact, though unrecoverable, representation of the original signal in the feature domain. This approach has several advantages with respect to sending the raw audio stream over the network: the signal is compressed to a factor of 8 with respect to the original time-domain representation. At the network level, this means reduced network traffic, reduced radio utilization, less complex access policies to the wireless channel, reduced number of re-transmissions, etc. [95]. At the application level, since the signal transformation is partially irreversible (*e.g.,* once transformed into the feature domain, the original signal cannot be completely reconstructed), reasonable levels of privacy are guaranteed by design.

- *Full integration with the AGILE ecosystem*: the SAS and the whole design framework have been fully integrated within the AGILE IoT gateway ecosystem [46], in order to validate their applicability. More in detail, we propose a proof-of-concept scenario that describes how SASs can interact with the AGILE gateway and which modules we developed. Such modules allow developers and users to simply connect, configure and use the SASs, without much integration effort.

- *Modular, maintainable and scalable software architecture*: obeying the AGILE's software design and architectural principles, the developed modules follow the micro-service philosophy [37], thus they were independently implemented. Given that every module is independent of each other, the system allows simple updates. Then, if a module fails or crashes for some reason, the system is resilient and remains up, eventually with a limited set of capabilities. Moreover, the system is scalable by design: if several devices have to be managed, then the system can dynamically instantiate resources and modules, in order to cope with the increased load, and vice-versa. Last, but not least, we are distributing the computational load among different devices. In fact, the SASs perform feature extraction, while the IoT gateway only runs the trained model.

- *Detection of audio anomalies in an office environment*: as a proof of

concept and to provide scientific validation of our approach in a real use case, we deployed a SAS in an office environment to detect anomalies using a cheap microphone. The SAS is connected via Bluetooth to an AGILE gateway that infers, using an Anomaly Detection algorithm, over the features stream. Detection models have been trained using a pre-recorded audio stream coming from the same SAS. We evaluated two different aspects: firstly, the detection delay from the first recorded sample to the algorithm output, and then the CPU load of our module with respect to the AGILE CPU load.

Finally, given that this chapter will describe how to build an anomaly detection application at the edge of the network, Section 4.2 reviews the recent works related to Anomaly Detection in different IoT architectures, with also a few examples on audio anomaly detection. Section 4.3 presents the key elements that we use to build our design framework for Edge IoT applications, while our IoT edge devices are presented in Section 4.4. We describe the Proof-of-Concept scenario in Section 4.5, then we outline some remarks in Section 4.6.

4.2 Anomaly Detection in IoT-based architectures

In the following, we summarize some recent contributions in the framework of anomaly detection carried out considering an IoT-based architecture.
Islam et al. [96] discuss the application of a novel association rule-based approach for handling uncertain, vague, and noisy data in anomaly detection. Authors envision that data from sensors feed a web-based expert system able to predict flooding using rainfall and temperature measurements. The expert system employs a database of belief association rules that allows to identify the anomalies in the level of rainfall and temperature and to predict a flooding event.

Trilles et al. [97] present a classical cloud-based methodology for handing, in a real-time fashion, heterogeneous sources of data streams. As a proof-of-concept, they discuss the application of their methodology for environmental anomaly detection using meteorological data. The proposed methodology considers a logical architecture that comprises three layers,

namely content, services, and application layers. The content layer is composed of different sets of heterogeneous sensors which are deployed in a specific scenario and send streams of information to the service layer. The services layer includes 1) connectors for handling the streams coming from different data sources, 2) a brokering system for allowing access to data coming from sources that use different communication protocols and message encoding techniques, and 3) algorithms that elaborate data. The application layer includes all users' applications. The different layers communicate by means of real-time massage services. In the discussed proof-of-concept, a simple CUSUM (cumulative sum) statistical algorithm has been adopted, which is a nonparametric and univariate method for anomaly detection.

Lyu et al. [98] discuss a Fog Computing architecture for anomaly detection. Raw data are collected by the end nodes and sent to the fog nodes which are in charge of building the model for detecting the anomalies, performing both sensor layer and fog layer clustering. The results of these clustering steps are sent to the cloud in order to be merged. The cloud layer sends the merged clusters to the fog layers which carry out the anomaly detection steps. The identification of the clusters is based on a hyper-ellipsoidal clustering algorithm that adopts a scoring mechanism for distinguishing normal and anomalous events [99]. The proposed architecture is compared with both a centralized architecture and a distributed architecture in the same Fog Computing framework. Similar architectures have been previously introduced by Rajasegarar et al. [99] considering a multi-level hierarchical topology of Wireless Sensor Networks (WSNs). In the centralized architecture, sensor nodes just sent data to the cloud server that carries out all the data elaboration. In the distributed architecture, end nodes conduct a clustering step at the sensor layer, send the results of the clustering to the fog and cloud layers, and receive the results of fog and cloud layers clustering. Fog and cloud layers receive the clustering results from the lower layers, merge the clusters and send back to the end nodes the results of the merging. The fog layers are also in charge of the anomaly detection task. As an application scenario, the authors present a smart traffic light system, where the traffic light acts as a fog node and receives signals from different devices such as sensors mounted on cars and pedes-

trians, flashing lights of the ambulances. The smart traffic light process all the data coming from the street in order to maintain a smooth and safe traffic flow.

A Fog Computing architecture for anomaly detection in smart cities has been recently presented in [52]. Pereira dos Santos et al. [52] discuss the application of their approach for monitoring the air quality in the city of Antwerp, Belgium, considering Low Power Wide Area Network (LPWAN) technologies for communications. Even in this case, end nodes send raw data to the fog resources which perform the anomaly detection in a distributed fashion. Fog layers may send alerts to both end nodes and cloud servers, whenever anomalies have been detected. The cloud layer combines the results of the anomaly detection in order to update in real-time the status of the entire network. Moreover, the cloud layer can also perform global anomaly detection operations and show the results to the central dashboard of a control room of the smart city. The anomaly detection procedure has been carried out using an unsupervised approach based on clustering algorithms.

Rathore et al. [100] discuss a real-time geo-visualization framework. The framework is fed by micro-climate data sensed and transmitted by low-cost multi-sensors. These sensors send raw data to the gateway using the Zig-Bee transmission protocol. Then, by means of a 3G/4G wireless modem, data are transferred to a cloud server for persistent storage. The anomaly detection process and the interactive geo-visualization have been implemented as a web application. The application gets data by means of SQL queries toward the data cloud server. Hyper-ellipsoidal clustering algorithms have been adopted for anomaly detection.

Finally, a few papers have been published about IoT architectures and acoustic anomaly detection. Hilal et al. [59] introduce a pervasive IoT-based indoor acoustic surveillance architecture that detects and localizes anomalous sounds associated with abnormal events. The anomaly detection process is carried out in a distributed fashion: the proposed architecture includes both smart sensors, devoted to monitoring the environment, and delegate sensors, which are in charge of assisting the management of the sensors, the identification and the localization of anomalous events. Smart sensors are equipped also with resources capable to carry out local

abnormality detection. Both SVM and LDA models have been adopted for the classification stage of the sound events. Other results have been achieved in the context of an EU project called DYNAMAP[1], aimed at developing low-cost sensor networks for real-time noise mapping, which have been reviewed in [77]. In particular, the authors show the results achieved by their anomalous noise events detector algorithm for mapping the traffic noise, considering both urban and suburban areas. The algorithm is based on a two-class anomaly detection model which discriminates between normal road traffic noise and anomalous noise events. A set of smart and low-cost acoustic sensors have been deployed along the roads to be monitored. These sensors perform simple signal pre-processing and also event classification exploiting acoustic models based on machine learning algorithms. All the classification labels are sent to a centralized server that is in charge of updating the noise maps. A third Edge Computing-based architecture for audio event detection has been discussed in [78]. Wireless audio sensors are deployed in indoor environments and raw data are sent to data concentrator devices, which are equipped with graphics processing units (GPUs). Such devices are in charge of data pre-processing, including feature extraction and anomaly detection tasks. The detection model is based on both clustering and classification algorithms. The data concentrators send the results of their elaboration to a remote server that takes care of the needs of people living in the considered scenario.

4.2.1 Gap analysis

Table 4.1 summarizes the main features of the aforementioned recent approaches to anomaly detection in the IoT context. Observing this table, we notice that the majority of the recent related literature sticks to classical technological approaches (*e.g.,* web/cloud-based frameworks [96, 97, 100], and distributed WSN deployments [99, 59, 77]). Indeed, only a few works attempt to rely on the Fog/Edge Computing paradigm to distribute the execution of the different tasks of an environmental sensing IoT application along the *Cloud-to-Thing continuum* [98, 52, 78].

However, it is worth noticing that the fog node used in [98] is a full-

[1] `http://www.life-dynamap.eu/`

Table 4.1: Summary of the main features of the recent methods for anomaly detection in IoT architectures (source: [41], p. 67598).

Reference	Year	Technological approach	Anomaly Detection Model	Type of Training	Anomaly Detection Level	Data from sensors
[96]	2018	Web service	Association Rule	Unsupervised	Centralized	Raw
[97]	2017	WSN + Data brokering platform	CUSUM	Unsupervised	Centralized	Raw
[98]	2017	Fog	Clustering	Unsupervised	Distributed	Raw
[99]	2014	hierarchical WSN	Clustering	Unsupervised	Distributed	Pre-elaborated
[52]	2018	Fog	Clustering	Unsupervised	Distributed	Raw
[100]	2018	Cloud	Clustering	Unsupervised	Centralized	Raw
[59]	2018	Collaborative WSN	SMV and LDA	Supervised	Distributed	Pre-elaborated
[77]	2017	hierarchical WSN	Classification	Supervised	Distributed	Pre-elaborated
[78]	2017	WSN + GPU-powered Edge gateway	Clustering and Classification	Semi-Supervised	Distributed	Raw

fledged server PC equipped with an Intel i7-4790 quad-core processor and 16 GB of RAM while, in [78], even though the authors propose a Mobile Edge Computing (MEC) [101] approach, they rely on a high-performance General Purpose Graphics Processing Unit (GE-GPU) directly installed on a Jetson TK1 development board[2], that is yet a quite powerful (and expensive) edge gateway device. On the other hand, as described in Section 4.3.2, the full gateway framework adopted in our experimentation is deployed on a very cheap Raspberry Pi 3[3] single-board computer.

Moreover, the sensor nodes of [98] simply collect environmental data and transmit, using a wireless connection, the digitized values to the fog node, which is responsible for the full processing and analysis of all raw data flows coming from (possibly various) sensor nodes. The same approach is also followed in [52] to detect anomalies in slowly changing environmental phenomena (*e.g.,* 3 particle matter indicators enriched with GPS locations). Conversely, as described in Section 4.4, in our framework the sensor nodes are able to locally pre-process high-definition raw audio streams in real-time, hence only transmitting the extracted features to the near gateway. In this case, the intelligence of the application is truly distributed at the edge of the network (*e.g.,* from tiny sensing terminals, through cheap IoT gateways, until web/cloud endpoints), as also the most constrained devices of the chain can bear non-negligible computational overheads.

[2] `https://developer.nvidia.com/embedded/jetson-tk1-developer-kit`
[3] `https://raspberrypi.org`

4.3 Technological Background

This section introduces and describes the main Edge technologies adopted to develop and assess the proposed framework.

4.3.1 The Teensy-based Smart Audio Sensor

A Smart Audio Sensor (SAS) was originally defined in [102] as an embedded device equipped with one or more microphones that is able to record and perform some computations *directly* on the audio stream, without losing real-time capabilities. In this way, it is possible to have devices that are able to monitor the surrounding environment, even in some extreme situations directly from the edge of the network. For instance, a SAS deployed in a street may detect gunfire events, even in absence of light or illumination. Moreover, by using an array of microphones, it may be also possible to identify the direction of the event source.
However, SASs should be developed adopting powerful computing units able to execute heavy operations directly on the audio stream (*e.g.,* ARM Cortex-M4). In the prototyping landscape, one of the most promising boards for embedded intensive computations is the Teensy board[4]. Teensy, developed by PJRC, is a versatile and powerful development platform for embedded projects compatible with the Arduino Environment, thanks to the Teensyduino libraries, thus most of the Arduino sketches run on Teensy boards. Teensy is available in eight different flavors, based on the requirements of the project. One of the most powerful board is the Teensy v3.6, that is equipped with a 32 bit ARM Cortex-M4F processor, with the DSP instruction set, working at 180MHz with a floating-point unit (FPU), 256KB of RAM, 1MB of Flash memory, 58 Digital I/O, 6 UART interfaces, 4 I^2C buses, 3 SPI interfaces, 2 I^2S Digital Audio buses, 1 micro-SD card slot and the possibility to connect one Ethernet shield at 100Mbps.

Teensy boards have been designed by following the Arduino philosophy *"Easy to mount, cheap to produce"* in order to extend the board capabilities. Shields can be easily developed and plugged, thanks to the high

[4] `https://www.pjrc.com/teensy/`

availability of I/O pins. Even Arduino shields can be plugged into a Teensy board using the Teensy Arduino Shield Adapter[5].
A powerful extension shield for Teensy boards is the Audio Shield[6]. This board, created by PJRC, is able to add I/O audio capabilities to Teensy. It is powered by the powerful SGTL5000 Low Power Stereo Codec[7] and allows 16 bits high-quality audio recording at 44.1 kHz (CD quality), using either the on-board mono-channel microphone or the stereo line-level input. Moreover, it supports stereo line-level output and stereo headphones through the 3.5mm jack soldered on the board. A Teensy board can be physically connected to an Audio shield using the 14x2 extension header and the audio stream is transferred from one device to the other one, using the I^2S Digital Audio bus, which is a special communication bus designed for audio streams supporting up to 2 different audio channels. The audio data transfer is managed by the Teensy Audio library[8], a software library that is also able to execute various operations on audio streams. Furthermore, if the library is executed on Teensy 3.x boards, it can run real-time, computationally-intensive operations, like FFTs, using the DSP instruction set provided by the ARM Cortex-M4 processor. In addition, another library, OpenAudio for Teensy[9], has been developed on top of the Teensy Audio library and provides additional features and operations for real-time audio processing. This enables developers to create sound-reactive projects with reduced costs.
More in general, Teensy boards can be an interesting starting point to prototype edge IoT devices.

4.3.2 The AGILE-based IoT Gateway framework

In general, SASs represent a versatile kind of embedded devices that can be adopted within almost every IoT scenario, spanning from industrial plants to smart city contexts, simply by connecting them to a gateway, either through a wired or wireless radio communication technology. Gate-

[5] https://www.sparkfun.com/products/15716
[6] https://www.pjrc.com/store/teensy3_audio.html
[7] https://www.nxp.com/docs/en/data-sheet/SGTL5000.pdf
[8] https://www.pjrc.com/teensy/td_libs_Audio.html
[9] https://github.com/chipaudette/OpenAudio_ArduinoLibrary

ways are able to bridge different networking stacks, *e.g.,* IP and non-IP worlds, and execute computationally intensive operations closer to the deployed devices. In the context of gateway devices suitable for the IoT, one opportunity is offered by the Adaptive Gateway for dIverse muLtiple Environments (AGILE) [46]. AGILE is an open-source modular software framework for IoT gateways that supports a wide range of components. Hence, it enables developers, users, and companies to develop their own solutions and products on top of its stack. The AGILE architecture has been designed by following the micro-service paradigm, which was originally proposed for distributed systems. Nowadays, this paradigm is well-recognized and adopted in several different fields (*e.g.,* Cloud Computing), as it enables strong modularity, maintainability, scalability, reliability, and resiliency against failures of systems [103]. Indeed, if a service fails, the whole system remains alive, with reduced capabilities in the worst case. The AGILE project consortium adopted these concepts and ideas to design a robust framework for IoT gateways. This framework consists of different modules, where each module is implemented as an independent service within a Docker container[10], which creates a sandbox where the component executes. Within a container, it is possible to use very specific technologies and programming languages, without suffering any compatibility issue with other modules. Interactions among modules exploit a well-defined set of Application Programming Interfaces (APIs) defined in the framework. APIs are available in two different manners: using a common Inter-Process Communication (IPC) bus (*e.g.,* DBus[11]) or using RESTful interfaces.
The AGILE gateway framework runs on many different hardware architectures, including x86 and ARM. In particular, AGILE has been successfully tested and deployed on affordable Raspberry Pi 3 (RPi3) computers[12], which is a single-board computer. Despite its reduced cost, an RPi computer can rely on several analog and digital input/output lines to extend its basic capabilities (*e.g.,* connecting sensors and actuators). For instance, the Libelium company (which is a partner of the AGILE consortium) has

[10] `https://www.docker.com/`
[11] `https://www.freedesktop.org/wiki/Software/dbus/`
[12] `https://www.raspberrypi.org`

developed an extension shield for the RPi, called the Maker's Shield[13,14], equipped with sensors, buttons, LEDs, and two XBee sockets. This board can help makers and developers to create complete IoT solutions in an easy and straightforward way, by just plugging the shield on top of the RPi GPIO header. Moreover, considering the presence of two XBee-compliant sockets, it is possible to extend the networking capabilities of the AGILE gateway by adding new families of supported devices and radio technologies.
Finally, the AGILE framework has been designed to be suitable in many scenarios and applications. To this aim, the consortium identified five different pilot tests: open field and cattle monitoring, enhanced retail services, port area monitoring for public safety, air quality and pollution monitoring, and self-tracking. All these pilots are implemented by exploiting the modularity and the fine-granularity of the whole framework. In Section 4.5, we will introduce a new showcase for the AGILE IoT gateway, which is the rare sound event detector in an office environment, based on anomaly detection algorithms.

4.4 The proposed Wireless Smart Audio Sensor

In this chapter, to describe how to design and deploy intelligence on Edge IoT devices, we propose a tiny and affordable wireless smart audio sensor (SAS) for indoor environments. Our wireless SAS has been developed using commercial boards and components. As depicted in Figure 4.2, the device comprises four different modules: a mono-channel electret microphone capsule (Figure 4.2a), a Teensy Audio Shield (Figure 4.2b) for audio recording at 44.1 kHz with 16-bit resolution, a Teensy 3.6 board (Figure 4.2c) for local audio processing, and an HC-05 Bluetooth module (Figure 4.2d) for serial data transfer.
During the initial phase of the design of our Edge IoT device or SAS (Figure 4.3a), the device was conceived to record an audio stream with a sample rate of 44.1 kHz and 16-bit resolution and to transmit the signal over a UART interface using the Bluetooth module. Another device, e.g.

[13] https://github.com/Agile-IoT/agile-makers-shield-hardware
[14] https://github.com/Agile-IoT/agile-makers-shield-software

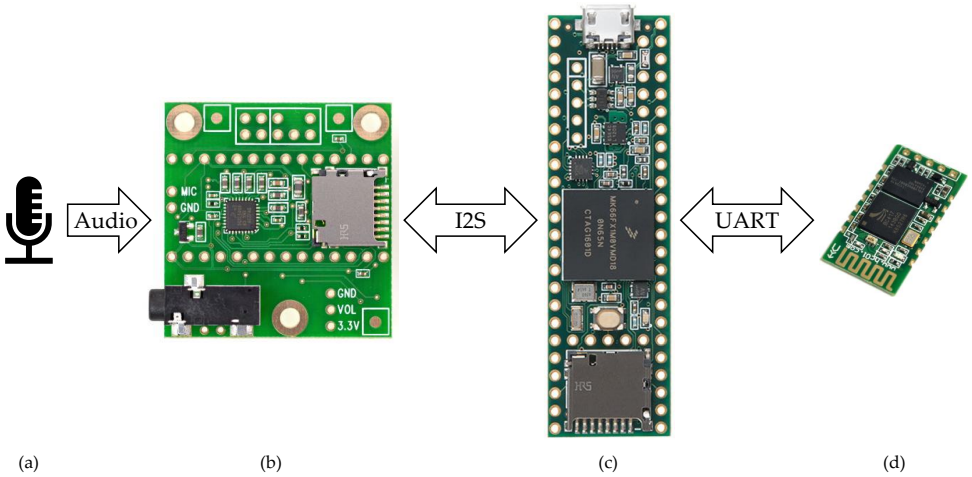


Figure 4.2: Hardware schema of the proposed Wireless Smart Audio Sensor. A Teensy 3.6 (c) is connected, via I^2S bus, to a Teensy Audio Shield (b) that records the audio signal coming from a mono-channel microphone (a). Then, the Teensy MCU (c) computes the FFT and Mel coefficients of a recorded audio frame. Finally, the Teensy board (c) sends all the Mel coefficients to a UART endpoint using a UART interface provided by an HC-05 Bluetooth 2.0+EDR module (d) (source: [41], p. 67600).

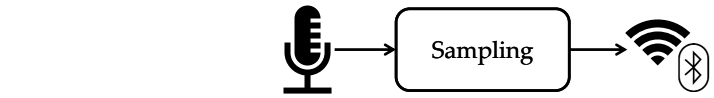
gateway, was in charge to perform some additional computations on the sampled signal. The radio module is able to establish a Bluetooth 2.0 + EDR connection (up to 3 Mbps) and supports UART baud-rates up to 460800 baud (from specifications). The original idea was to simply make the microphone a wireless device over Bluetooth. From the technological point of view, it was almost the same scenario we described in Chapter 3. However, we conducted some tests and figured out that the module guarantees good performances only up to 230400 baud using the 8N1 mode. Such baud-rate is not enough to transmit the audio stream since it requires (in this case 1 bps = 1 baud)

$$\begin{aligned} Sample\_Rate \cdot Bit\_Resolution &= 44100 \cdot 16 \\ &= 705600\ bps, \end{aligned} \tag{4.1}$$

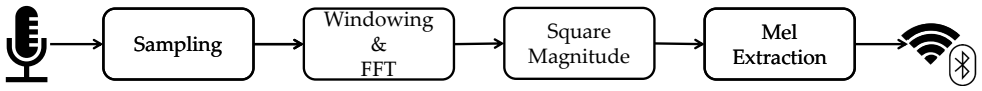
where each sample is represented with 16 bits. A possible solution was to reduce the sample rate to a value that was small enough to realize a bit rate that fits in the available bandwidth. Starting from the maximum baud-rate, the maximum achievable sample rate is calculated by using Formula (4.2).

$$\frac{Maximum\_baudrate}{Bit\_Resolution} = \frac{230400}{16} = 14400\ Hz. \tag{4.2}$$

Since 14400 Hz is not a recommended sample-rate (is not an integer division of 44100), the first smaller acceptable rate is 11025 Hz. The Nyquist-Shannon sampling theorem tells that the maximum theoretical signal bandwidth is one-half (5512.5 Hz) of the sampling-rate, thus the computed sample-rate is too small to create a Smart Audio Sensor because high-frequency components are required in many applications.
The above bandwidth bottleneck pushed a re-design of the Teensy board behavior. In the beginning, the DSP pre-processing was not performed on the SAS but on the other side of the Bluetooth link, e.g., gateway. SASs are commonly adopted in use-cases in which the raw audio stream is transformed to other more meaningful quantities such as Mel-Frequency Cepstral Coefficients (MFCC) or Mel coefficients. The former is mainly used as features for speech-based applications, the latter are frequently adopted as features in non-speech scenarios. Both of them require the preliminary computation of the frequency spectrum from the raw audio stream. Teensy's libraries provide efficient DSP operations exploiting the DSP capabilities offered by the ARM M4F MCU. Using such libraries, we have implemented the whole software flow to compute the Mel coefficients within the Teensy firmware. Figure 4.3b shows the sequence of operations required to calculate such coefficients. After the sampling performed by the Audio Shield at $f_{sr} = 44100Hz$, the Teensy MCU applies the Hanning window to the audio stream, in order to reduce discontinuities in the signal, and calculates the Fast Fourier Transform (FFT) at N_{FFT} points using a temporal window large N_{FFT} audio samples overlapped by $\delta_{overlap}$ with the previous window. Overlapping is used to maintain a high correlation between two successive windows. The FFT implementation returns N_{FFT} complex samples made of two *float32* numbers. We compute the square magnitude of the FFT output in order to have an instantaneous estimation of the power distribution over frequency. Consequently, we apply the Mel-Filtering using N_{mel} bins. This filtering is performed using a



(a) Early design.



(b) After re-design.

Figure 4.3: Software flowcharts of the implemented SAS. (a) is the early design of the SAS with only recording capabilities. (b) is the re-designed version of the SAS with embedded feature extraction (source: [41], p. 67601).

filter-bank, composed of N_{mel} filters, that implements the mel-scale, the non-linear perception scale of the human ear [104]. Every Mel-filter has a triangular shape, has response 1 at the central frequency and it linearly decreases down to 0 when it reaches the central frequencies of neighbor filters. The frequency response of the i-th filter is described in Formula (4.3).

$$H_{mel_i}(k) = \begin{cases} 0 & k < f(i-1) \\ \frac{k-f(i-1)}{f(i)-f(i-1)} & f(i-1) \leq k < f(i) \\ 1 & k = f(i) \\ \frac{f(i+1)-k}{f(i+1)-f(i)} & f(i) < k \leq f(i+1) \\ 0 & k > f(i+1) \end{cases} \tag{4.3}$$

$f(i)$ is the central frequency of the i-th filter. To calculate these frequencies, two different techniques have been proposed in the literature: the Slaney's formulation [105] and the HTK's formulation [106]. In this chapter, we adopt the former formulation that splits the frequency domain into two different regions: a linear region for frequencies within the range 0-1000 Hz and a log region for frequencies greater than 1000Hz. Formula (4.4) and Formula (4.5) describe how to pass from hertz to mel and vice-versa, respectively.

$$m = \begin{cases} \frac{f}{200/3} & f \leq 1000Hz \\ \frac{1000}{200/3} + \frac{log(6.4)}{27} \cdot log\left(\frac{f}{1000}\right) & f > 1000Hz \end{cases} \tag{4.4}$$

$$f = \begin{cases} \frac{200}{3} \cdot m & m \leq \frac{1000}{200/3} \\ 1000 \cdot exp\left(\frac{log(6.4)}{27} \cdot \left(m - \frac{1000}{200/3}\right)\right) & m > \frac{1000}{200/3} \end{cases} \quad (4.5)$$

In order to compute the central frequencies, we have first to calculate the mel representations of the lower and higher frequencies of the audio bandwidth, *e.g.,* 0 Hz and 22050 Hz, using Formula (4.4). Then, we compute N_{mel} linearly spaced values in the range of the previously computed lower and higher mel representations. Now, applying Formula (4.5) to every value, we obtain the central frequencies $f(i)$ of our filters.

After the mel-filtering, the smart audio sensor transmits the N_{mel} coefficients, represented as *float32*, over a Bluetooth channel using an 8N1 UART interface working at *br* baud.

4.4.1 Design Framework for Smart Audio Sensors

In this section, we present and develop our design framework for Smart Audio Sensors that perform the computations presented above. Before starting, we have to define four parameters that will be tuned in order to correctly implement the SAS. Since the SAS operates on frequency representations of the audio stream, the first two parameters that we present are the length of the temporal window on which we compute the FFT (N_{FFT}) and the overlapping fraction ($\delta_{overlap}$) between two successive audio windows. These parameters influence the maximum length of the preprocessing window, the length of the temporal window, and the granularity of the audio transformation. The third parameter is the number of Mel coefficients (N_{mel}) computed over the frequency spectrum. It affects the transmission delay and the granularity of filters over the spectrum. The last parameter is the baud-rate (br) of the Bluetooth module. It affects only the transmission delay.

Given that we are designing an Edge IoT device, we need to minimize the latency introduced by recording (t_{rec}), pre-processing (t_{pre}), and data-transmission (t_{tx}) phases. We assume that the maximum latency for

real-time response is 150ms. This particular value is the maximum Mouth-to-Ear latency for VoIP systems, as defined in the G.114 ITU Recommendation [107]. We formalize this condition, called the *Real-Time condition*, in Formula (4.6).

$$t_{rec} + t_{pre} + t_{tx} < 0.15[s]. \tag{4.6}$$

Moreover, during the design phase of a SAS, we have to set the processing and the transmission parameters in order to satisfy another condition that we call *Buffering-Processing condition* and it is formalized in Formula (4.7).

$$t_{pre} + t_{tx} < t_{buffering}. \tag{4.7}$$

Since the FFT has to buffer N_{FFT} samples before it is able to compute the frequency spectrum and the window overlap fraction is $\delta_{overlap}$, a full pre-processing buffer is available every

$$t_{buffering} = (1 - \delta_{overlap}) \frac{N_{FFT}}{f_{sr}}[s], \tag{4.8}$$

where f_{sr} is the sample rate. The SAS has to complete all the pre-processing operations and the transmission phase within the buffering window $t_{buffering}$ in order to not overlap with other adjacent ones.

On the left-hand side of Formula (4.7), we have two terms: t_{pre} and t_{tx}. The former can be rewritten as

$$t_{pre} = t_{FFT} + t_{mag} + t_{mel}, \tag{4.9}$$

where t_{FFT}, t_{mag}, and t_{mel} are the temporal duration of FFT, square magnitude, and mel-coefficients extraction operations, respectively. More in detail, the time required to compute an FFT operation t_{FFT} depends on the number of points N_{FFT} and it has form

$$t_{FFT} = 4.3 \cdot 10^{-8} \cdot N_{FFT} \cdot log_2(N_{FFT}) \quad [s], \tag{4.10}$$

where the coefficient $4.3 \cdot 10^{-8}$ has been empirically found by interpolating the time duration required to compute FFTs with different values of N_{FFT}.

The second term (t_{mag}) of Formula (4.9) describes the time required to compute the square magnitude of the FFT spectrum. Even this term depends on the number of FFT coefficients N_{FFT} and we have empirically found that it has shape

$$t_{mag} = 2.46 \cdot 10^{-8} \cdot N_{FFT} + 2.5 \cdot 10^{-6} \quad [s] \tag{4.11}$$

The last term (t_{mel}) of Formula (4.9) represents the time required by the Teensy MCU to compute the mel-coefficients starting from the squared magnitude of the FFT coefficients. Formula (4.12) shows the experimental formula to compute t_{mel}.

$$t_{mel} = 5.1 \cdot 10^{-7} \cdot N_{mel} + 3 \cdot 10^{-7} \cdot N_{FFT} - 1.9 \cdot 10^{-5} \quad [s] \tag{4.12}$$

All empirical coefficients have been found by running 5000 times every single operation in the pre-processing loop with different values of parameters $N_{FFT} \in \{256, 512, 1024, 2048, 4096\}$ and $N_{mel} \in \{40, 64, 128\}$. Moreover, we used the ARM functions available in the Teensy *arm_math.h* library.

The left-hand side of the *Buffering-Processing condition* (Formula (4.7)) contains a second term, t_{tx}, that keeps track of the time spent to transmit the N_{mel} mel coefficients through a UART interface over the Bluetooth channel. This term (Formula (4.13)) depends on three different parameters: the number of coefficients N_{mel}, the effective bit-rate br_{eff}, and the number of bits (N_{bits}) used to represent each coefficient, *e.g.,* $N_{bits} = 32$ if we use *float32* numbers.

$$t_{tx} = \frac{N_{mel} \cdot N_{bits}}{br_{eff}} \quad [s]. \tag{4.13}$$

It is important to note that the effective bit-rate br_{eff} used in Formula (4.13) is not equivalent to the baud-rate br configured on the serial connection, *e.g.,* 230400 baud. We have to scale the configured baud-rate by factor 0.8, so $br_{eff} = br \cdot 0.8$. This scale factor comes from the serial mode configured. Since we are using the 8N1 mode, we transmit 1 start bit, 8

Table 4.2: Configured baud-rates Vs Effective bit-rates with 8N1 mode (source: [41], p. 67602).

Configured baud-rate br	**Effective bit-rate** br_{eff}
57600	46080
115200	92160
230400	184320

data bits, 0 parity bits, and 1 stop bit. Resuming, we are actually transmitting 10 bits every 8 information bits, thus the efficiency is 0.8. Effective bit-rates are shown in Table 4.2.

4.4.2 A possible parameters tuning

As described above, a SAS designer has to tune only four parameters ($\delta_{overlap}$, N_{FFT}, N_{mel}, br), which are free variables. Typically, the overlap fraction $\delta_{overlap}$ is set to 0.5 in order to keep a good correlation between consequent windows. A possible set of parameters is the following:

$$\delta_{overlap} = 0.5 \ , \ N_{FFT} = 4096$$
$$N_{mel} = 128 \ , \ br = 230400.$$

Now, we prove that this set of parameters verifies both the design conditions. First, using Formula (4.8), we compute the inter-arrival time between two consequent windows and we get

$$t_{buffering} = 46.44 \cdot 10^{-3} \quad [s].$$

Hence, we compute the time required for pre-processing operations using Formula (4.9), Formula (4.10), Formula (4.11), Formula (4.12), thus we obtain

$$t_{pre} = (2.11 + 0.1 + 1.3) \cdot 10^{-3} = 3.51 \cdot 10^{-3} \quad [s].$$

Then, using Formula (4.13), we compute the time required to transmit N_{mel} coefficients as *float32* numbers, and we get

Table 4.3: Other possible parameter combinations assuming $\delta_{overlap} = 0.5$ (source: [41], p. 67603).

N_{FFT}	N_{mel}	br
4096	128, 64, 40	230400
4096	64, 40	115200
4096	40	57600
2048	64, 40	230400
2048	40	115200
1024	40	230400

$$t_{tx} = 22.2 \cdot 10^{-3} \quad [s].$$

Immediately, we can see that the *Buffering-Processing condition* (Formula (4.7)) holds, since

$$\begin{aligned} t_{pre} + t_{tx} &= (3.51 + 22.2) \cdot 10^{-3} \\ &= 25.71 \cdot 10^{-3} \\ &< 46.44 \cdot 10^{-3} = t_{buffering}. \end{aligned}$$

Now, we have to verify if the *Real-Time condition* (Equation Formula (4.6)) holds. The recording delay t_{rec} is simply the time required to record N_{FFT} samples at 44100 Hz and, in this case, it has value

$$t_{rec} = 92.9 \cdot 10^{-3} \quad [s].$$

Using this result we can compute the overall delay introduced by recording, pre-processing, and transmission phases. We obtain

$$t_{rec} + t_{pre} + t_{tx} = 118.61 \cdot 10^{-3} < 150 \cdot 10^{-3} \quad [s].$$

The *Real-Time condition* holds and our set of parameters can be used to build a SAS or an edge device. Other possible parameter combinations are shown in Table 4.3.

4.4.3 COTA: Configuration Over The Air

In the previous section, we have presented a framework that requires the tuning of four parameters to design an Edge IoT device that performs some

local computation. As we also depicted at the beginning of this chapter, these devices may be deployed in hazardous or inaccessible locations, thus the device's parameters should be remotely set or changed from the other side of the Bluetooth link. These devices can be connected to an AGILE gateway and the gateway framework offers this functionality to remotely configure parameters. It is called Configuration Over The Air (COTA) and is implemented as a small micro-service that pushes parameters over the serial link used to communicate with the target Edge device, a SAS in this case. Parameters are sent to the remote device as a text string with the following format

$$\#\delta_{overlap}; N_{fft}; N_{mel}; br\$ \quad .$$

An example of a possible string is

$$\#0.5; 4096; 128; 230400\$ \quad .$$

The SAS replies with an acknowledgment message, which is *OK*, for a valid configuration or with *ERROR* if it rejects the configuration. After the acknowledgment message, the remote device reboots with the new configuration.

Moreover, the COTA module exposes a UI, accessible through the AGILE web-based UI, that allows users to configure parameters. The interface is user-friendly and ready to use. The title shows the device family and ID, the body contains four different fields to set the value of parameters. The UI controller is able to verify if the proposed configuration verifies our two design conditions. If one condition is not met, the interface prompts an error. Figure 4.4 shows an example of the COTA UI interface.

4.5 Proof of Concept: an Edge IoT application with SAS devices

In order to demonstrate the applicability of the proposed Edge IoT device, *i.e.,* Smart Audio Sensor, introduced in Section 4.4, we have designed and implemented an Edge IoT application. It has been deployed in an IoT Smart Office environment that exploits a SAS and an AGILE gateway to

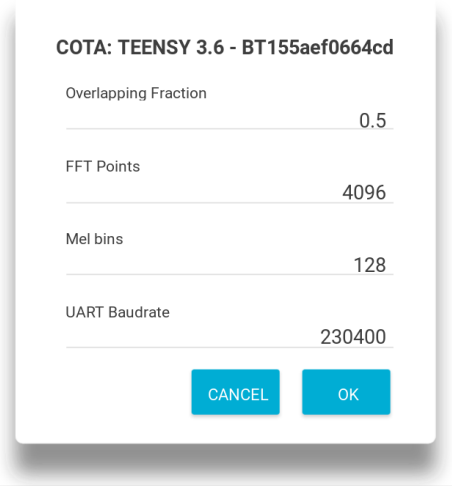


Figure 4.4: COTA UI (source: [41], p. 67603).

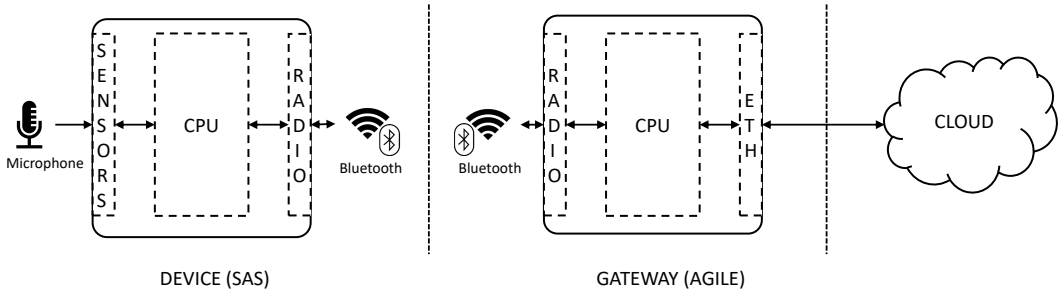


Figure 4.5: Block diagram of our technological framework, which is deployed at the edge of the network, and the corresponding interactions between computing entities (source: [41], p. 67604).

detect audio anomalies. In this context, anomalies may be screams, door slams, or every event that is not a normal keyboard typing, a call, or a talking. Figure 4.5 provides a block diagram depicting how the different technological entities and comprising components interact with each other.

The SAS implements the pre-processing technique previously described and it is configured as follow:

$$\delta_{overlap} = 0.5 \ , \ N_{FFT} = 4096$$
$$N_{mel} = 128 \ , \ br = 230400.$$

The device has been deployed in a top corner of a rectangular room, which has a surface of 25 square meters. The SAS records the audio-stream from

an electret microphone, then pre-processes it by extracting mel coefficients and, finally, sends computed features to an AGILE gateway using the Bluetooth interface. On the other side of the radio link, the gateway runs a python-based micro-service that collects mel coefficients, through the serial interface, and feeds an Anomaly Detection algorithm to detect if an anomaly occurred or not in the current time frame. If yes, the micro-service sends a notification to the user showing an error in the logging console.

Our objective is threefold: 1) we want to prove the possibility to spread the intelligence on multiple computing entities at the edge of the network (*i.e.,* the AI model on the gateway and the feature extraction on the Edge IoT devices); 2) we want to evaluate the impact over the CPU of Anomaly Detection algorithms running on the AGILE gateway; 3) we aim at evaluating the delay between the anomalous event to the event detection. In this way, we have considered two different anomaly-detection algorithms among all the algorithms offered by the Scikit-Learn [91] toolbox: Elliptic Envelope (EE) [108] and Isolation Forest (IF) [109, 110].
The EE algorithm assumes that the training set has a Gaussian distribution, thus it models a robust covariance estimator over data. Giving the estimation of the inlier location and covariance, the algorithm uses the Mahalanobis distance to measure the outlyingness of unseen data points.

The Isolation Forest algorithm exploits random forests to isolate outliers from inliers. IF randomly chooses a feature and then splits in a point between the maximum and the minimum values of the selected feature. The number of splitting operations that are required to isolate an instance is equal to the path from the root node of a tree to the leaf node. This is due to the recursive nature of splitting that can be modeled by a tree structure. We obtain a detection rule and a measure of normality by averaging path lengths over a forest of trees. This is due to an intrinsic property of IF that generates short paths for anomalies since they are isolated from not-anomalous, or inlier, values.

4.5.1 Training of the AI models

Since Anomaly Detection techniques belong to the Unsupervised Learning family of machine learning algorithms, a labeled dataset is not required to train our models. However, such algorithms require an unlabeled dataset that describes the normality condition of the office environment. In order to realize a reliable and real dataset, we conducted an audio recording campaign using a Teensy Audio Shield and a Teensy board that was programmed only as a recorder and stream forwarder over a UART interface. As stated in Section 4.4, the bandwidth available over a Bluetooth interface is not sufficient to transmit the audio stream recorded at 44.1 kHz with 16-bit resolution (Formula (4.1)), thus we configured the UART interface over USB that can reach a higher baud-rate, up to 4.608 Mbaud. We recorded the office audio stream for 4 hours saving a WAV file every 60 seconds. The obtained dataset was split into two different subsets: 2 hours reserved for the training set and 2 hours assigned to the test set. Then, the training phase comprises two steps: feature extraction and the training of a model. Feature extraction calculates the mel coefficients starting from audio files using APIs offered by LibROSA [90], a python package for audio analysis. The sequence of operations is the same one explained in Figure 4.3. Anomaly Detection algorithms were implemented in Python3 using the Scikit-Learn framework [91], a powerful machine learning python toolbox. As we stated above, we selected two different algorithms among all the tens of available algorithms: Elliptic Envelope (EE) and Isolation Forest (IF). These algorithms require a few hyper-parameters to configure the training process. Since we want to study the CPU load of the model inference, we used the default value of parameters for both of the algorithms: we used 0.1 as contamination fraction and 100 as the number of adopted trees in IF. Firstly, we have trained our algorithms on just the first half of the set. Then, we retrained them on the entire set.

4.5.2 Performance evaluation of AI methods on an Edge device

As we stated above, we want to evaluate the proposed solution by considering the CPU load due to an Anomaly Detection algorithm running on an AGILE gateway and the *event-detection latency*, the delay between the

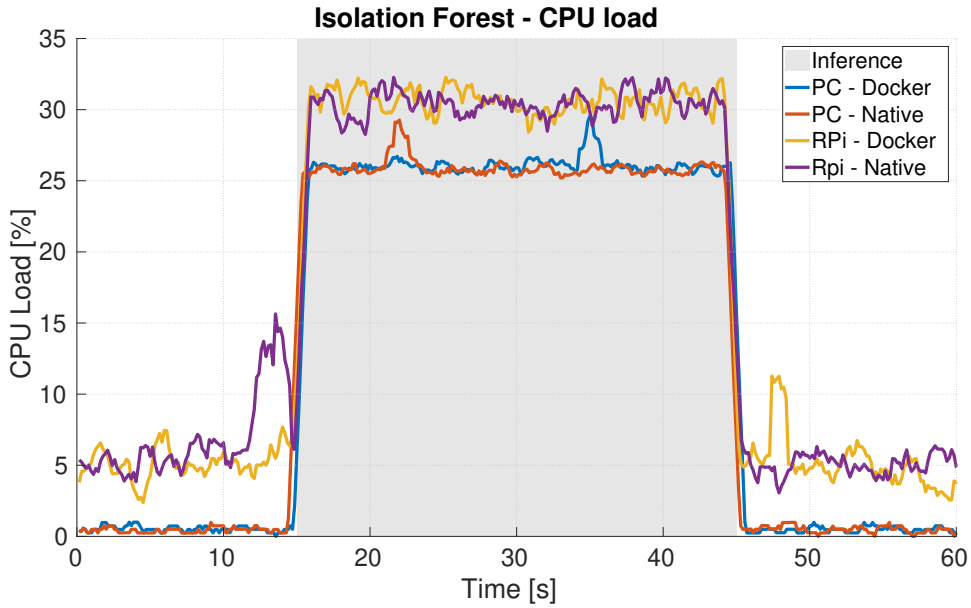


Figure 4.6: CPU load of Isolation Forest (source: [41], p. 67605).

anomalous event, and the event detection.
Since the AGILE gateway has been designed to run over a Raspberry Pi 3 computer, we conducted the load evaluation by running the AGILE gateway modules and the AD algorithm on the same device in two different fashions: embedded within a Docker container and executed as a native Python script. We recorded the user CPU load, since the module and AGILE run in the user-space, using the `top`[15] utility. Each experiment comprises three different phases: in the first 15 seconds, the system runs all the AGILE gateway modules and AD module without inferencing the feature stream, since it is not attached to the detection module. Then, we connect the feature stream to the AD algorithm for 30 seconds. Finally, we detach the data stream from the module and we continue to record the CPU load for the other 15 seconds. We profiled the CPU load for both the AD algorithms. Moreover, in order to provide a baseline, we also repeated the same experiments over an x86 machine that run the AD algorithm and the AGILE gateway framework.

Figure 4.6 shows the CPU load due to the execution of the Isolation Forest algorithm. As we can see, the average CPU load on a Raspberry

[15] `http://man7.org/linux/man-pages/man1/top.1.html`

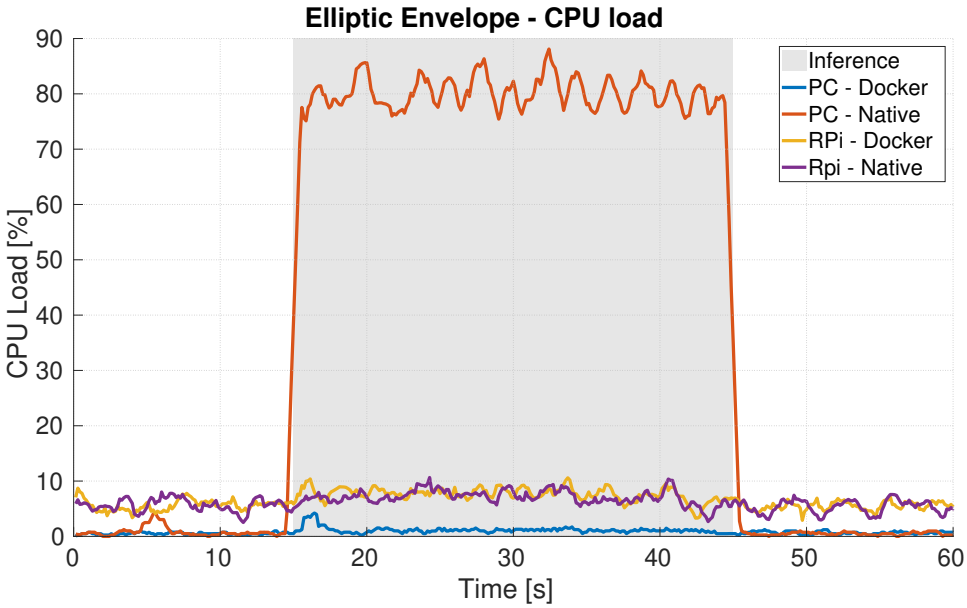


Figure 4.7: CPU load of Elliptic Envelope (source: [41], p. 67605).

Pi increases from 5.2% to 30.6% when the module is embedded within a Docker container and from 6.8% to 30.4% when the module is *natively* executed. We can see that when the algorithm is running the Docker overhead is 0.2%. For IF, the Docker overhead on an x86 machine is the same (0.2%) since the CPU load is 26.1% (*dockerized*) and 25.9% (*native*), respectively. Figure 4.7 depicts the computing load due to the Elliptic Envelope algorithm. The graph shows that the Raspberry Pi CPU load passes from 5.6% to 7.9% when the algorithm is embedded within a container and from 5.6% to 7.3% when the script is natively executed. The load over a x86 machine is 1.18% when the module is containerized and 80.3% when the Python script is run natively. This behavior might be due to how the Python interpreter distributes the load over the CPU.
In general, except for the EE run as a native script over an x86 machine, the CPU load of IF is higher than the load required to compute EE. This is due to the nature of the running algorithm. EE has to check if the input sample is inside or not the region defined by the learned decision function. If the sample is outside, the algorithm notifies an anomaly. The IF algorithm has to compute the output of all the trees present in the ensemble (*e.g.,* 100) and then it takes a decision. It is also possible to evince

Table 4.4: Inference delays t_i in [ms] (source: [41], p. 67606).

Setup	Elliptic Envelope	Isolation Forest
RPi - Docker	1.7	222.9
RPi - Native	1.7	234.4
x86 - Docker	0.18	19.3
x86 - native	0.18	15.7

this behavior from the delay introduced by the algorithm to compute the output.

Table 4.4 shows that the IF is much slower than EE since IF has to compute the output of 100 trees.

Using the inference delays (t_i), we can compute the average overall latency ($t_{detection}$) from the event to the detection. Formula (4.14), which describes this latency, is the sum of 4 contributions: t_{rec}, the time required to record an audio frame, t_{pre}, the delay introduced by Teensy to preprocess the audio frame, t_{tx}, the transmission delay, and t_i, the time required to infer the data in the AD model.

$$t_{detection} = t_{rec} + t_{pre} + t_{tx} + t_i. \tag{4.14}$$

Applying Formula (4.14), we get $t_{detection} = 120.3[ms]$ when we infer with an EE model and $t_{detection} = 341.5[ms]$ when we adopt IF as an AD algorithm. Since EE is much faster and also lighter than IF, we decide to adopt the Elliptic Envelope as the AD algorithm on our gateway.

Moreover, we conducted an evaluation campaign in order to profile the performance of the adopted algorithm in terms of F1-score [111] and Error Rate (ER) [112]. F1-score is defined as

$$F1 = \frac{2 \cdot P \cdot R}{P + R}$$

where P is the precision ($P = \frac{TP}{TP+FP}$, TP = True Positive and FP = False Positive) and R is the recall ($R = \frac{TP}{TP+FN}$, TP = True Positive and FN = False Negative). Error Rate (ER) is defined as

$$ER = \frac{S + D + I}{N}$$

where S is the number of substitution (correct time instant but wrong class), D is the number of deletions (event not detected but present in the ground truth), I is the number of insertions (event detected but not present in the ground truth), and N is the number of events in the ground truth. ER can have a value greater than 1.

We tested the adopted algorithm using a self-made dataset generated using the mixture generation engine [66] developed for the DCASE2017 Challenge. We created a test dataset made by 501 examples: 167 with a gunshot, 167 with a glass-brake, and 167 with a baby-cry. Each sample is 30 seconds long and was obtained by adding the rare event (*e.g.,* a gunshot, a baby-cry, or a glass-break) with probability one over the recorded office audio stream. We applied this dataset to our system and we computed the evaluation metrics using the *sed_eval* toolbox [113] using a time-collar of 500 ms. The EE algorithm trained with the 60 minutes dataset obtained an F1-score of 50.55% and an ER of 0.71. The same algorithm trained with the 120 minutes dataset got an F1-score of 50.42% and an ER of 0.71. We evaluated also the IF algorithm on the same datasets and we obtained an F1-score of 56.07% and an ER of 0.63 training with the 60 minutes dataset and an F1-score of 53.49% and an ER of 0.65 using the 120 minutes training dataset. Even if IF performs better than EE, we use EE since it is faster and lighter.

4.5.3 Deployment on an AGILE gateway

In the previous section, we discovered that the Elliptic Envelope is the best algorithm for our proof of concept, even if it has lower performance than the Isolation Forest algorithm. The Elliptic Envelope algorithm is 2 orders of magnitude faster and much lighter than IF, thus the system may be able to respond earlier and waste less computational resources. We deployed the EE algorithm and the model trained with 60 minutes of office audio recordings in the AGILE Anomaly Detection micro-service. Moreover, we bound a serial port to the module and we connected the Teensy board programmed as a SAS with the parameters presented before.

The system receives the features computed by the embedded board and can to detect Audio Anomalies (*e.g.,* hand claps). Once an anomaly happens, the micro-service notifies the user by logging detection messages in the terminal output. However, it is important to remember the nature of the incoming signal and the source of events. Since we are working with a transformed audio stream, we have many feature frames per second (21.5 frames per second) and the algorithm can erroneously detect (False Positive) an anomaly. In order to reduce isolated False Positives, a possible solution was to apply a median filter, with window length N (N has to be an odd number), to the algorithm output. This filter considers the $N-1$ past samples and the current one, then it sorts all the predictions. If at position $\frac{N+1}{2}$ there is an anomaly, the filter declares that an anomaly occurred. We have to choose a small value of N (*e.g.,* 5, 7,...) such that it does not introduce delays or hide anomalies. A good value of N is 5. Moreover, another aspect that affects the accuracy of the deployed system is the quality of the dataset. The training dataset should contain all the audio events that define the condition of *normality* in the considered environment. The anomaly model was trained using a real audio stream recorded in an office environment during the working time, thus the audio stream contained normal office sounds, *e.g.,* talking, phone ringing, typing, and clicking.

This kind of scenario suffers weaknesses related to the physical deployment of the SAS. Since the audio sensor embeds an electret microphone, which has a sort of directivity even if it is omnidirectional, should be positioned in the direction of the audio source in order to have better transduction from the audio signal to the electric signal with reduced reverberation. Moreover, the raw audio signal exiting from the microphone is really small and it requires a pre-amplification before the elaboration. The pre-amplification gain should be fine-tuned in order to guarantee a good quality of the signal with a high Signal-to-Noise Ratio (SNR). Noise may be introduced by the microphone cable (depends on the cable quality and length) and by the amplifier itself. Another weakness is related to the power consumption since the SAS is always active and continuously streams mel features over the Bluetooth channel. It should be deployed

closer to a power source like an electric plug or connected to a high-capacity battery. Future works will characterize the power consumption of this device.

In the considered PoC, we deployed just one SAS, which streams the extracted features. The proposed system is able to deal with multiple streams coming from different SASs. This is possible since the Anomaly Detection module supports more than one serial port binding. Using a multiplexing technique, it is possible to choose one of the streams and then infer on it. Another possibility is to deploy one AD micro-service per device, but this approach requires more CPU power.
More in general, we proved the possibility to spread the computation required to build an IoT application, which uses AI methods, across different entities, how to allocate different functionalities based on the constraints (statically in these cases), and how remotely configure and reconfigure a device. On this last point, the injected configuration has to be validated in order to satisfy the application requirements such as latency, accuracy, privacy, and so on.

4.5.4 Adoption in other domains: an industrial scenario

The proposed framework can be easily accommodated in different verticals, especially within the Industrial IoT (IIoT) domain. In this case, the sensing devices may be directly deployed on-machine, so as to locally perform their computations before sending data to the gateway, which, in this way, has only to execute the machine learning algorithm. This enables new vertical use-cases focused on diagnostics, prognostics, and predictive maintenance, which reduce expenses and optimize the machine life-time.

However, industrial scenarios often require a higher number of sensing devices in order to effectively monitor different points of a plant. In these situations, this framework can easily scale up, providing a gateway that is able to manage large numbers of devices and related data streams. Since we keep deploying an AGILE gateway instance on a cheap Raspberry Pi 3 computer, we prefer not exceeding 80% of CPU load. Given this constraint and extrapolating the CPU load of the Elliptic Envelope container

from Figure 4.7, we assert that more than 30 parallel Elliptic Envelope containers can be concurrently hosted on a single gateway. Then, given the harsher conditions of a typical industrial environment in terms of electromagnetic interference, for our preliminary campaign we have changed the radio interface from Bluetooth to Wi-Fi as the latter, besides offering higher resistance against electromagnetic interference, allows for higher data-rates. We conducted a wide performance test over the Wi-Fi channel using IPerf[16], obtaining stable data-rates of 4Mbps, at the UDP level. Considering that our gateway is able to support up to 31 devices, this means that every device can stream data up to a maximum data-rate of 129 Kbps.

We have successfully applied this framework to a real industrial plant available at the Micro-Nano Facility (MNF) of Fondazione Bruno Kessler (FBK, Trento, Italy)[17]. This facility allows researchers to study, develop and build microdevices by processing raw silica wafers in an extremely clean and fine-controlled environment, also known as *Clean Room*. We will talk again about the research developed in this facility in Chapter 6. The Clean Room is composed of different modules that keep the environment suitable for silica processing. One of the most important modules is the air treatment system that controls both the injection and expulsion of the air from the room, while keeping constant the air pressure and the relative humidity. These systems are extremely critical since a malfunction could have disastrous effects on processes and equipment. Each system has two electric engines, while each engine is connected with a belt to a shaft that rotates a fan to push or extract the air from the Clean Room. Since these engines do not have on-board sensors, we are developing retrofitting kits to sense vibrations (using MEMS accelerometers) and temperature sensors. These kits sample the physical dimension, perform time-frequency feature extraction and then stream data using a Wi-Fi radio chip to the AGILE gateway. More in detail, the kit computes 32 features, expressed as 32-bit float numbers, starting from a 4 seconds time window sampled at a frequency of 1 kHz. It follows that each kit has to transmit 1024 bits every 4 seconds. Given that the maximum data-rate for each kit is as large as

[16] `https://iperf.fr/`
[17] `https://mnf.fbk.eu/`

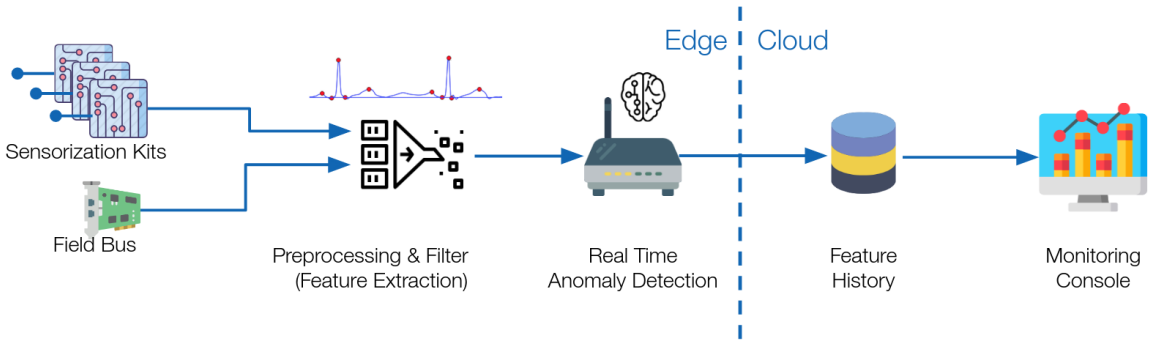


Figure 4.8: Pipeline of deployment in an industrial scenario.

129 Kbps, then the effective bandwidth allocation for each kit is only 0.2%. Such bandwidth requirement allows for radio chips to stay in a low power consumption mode (sleep mode) for more than 99% of the time, which increases the life-time of battery-powered kits.

We have installed 2 kits for each machine (one on top of the engine chassis and one directly on the shaft bearing) for a total of 4 machines (thus, 8 kits in total). The gateway can manage all the data streams coming from these kits and, contextually, it can infer anomalies per stream. The inference output (*i.e.*, 0 if the anomaly is absent; 1 otherwise) is used to tag data streams that are sent to a remote cloud data broker. The remote system stores data and it allows for historical analysis through a Grafana-based dashboard[18]. An overall view of the processing pipeline is depicted in Figure 4.8.

Last but not least, since the gateway supports the Configuration Over The Air (COTA), we are able to dynamically reconfigure devices based on the actual needs: if an anomaly occurs, we can easily reconfigure the device in order to have a finer analysis of the system. Future developments of this scenario comprise the possibility to simply and smoothly deploy new firmware on sensor kits via OTA updates.

4.6 Remarks

Edge Computing has deeply changed the IoT. Moving computation to the edge of the network, or closer to the data sources, has enabled many novel

[18] `https://grafana.com/`

applications and scenarios that were not possible before. Starting from the design of an IoT application (Chapter 3) that runs all the processing, including an AI model, on an edge device, in this chapter we moved a step forward. Here, we consider again an Edge IoT application that runs all the processing (including AI) at the edge of the network, however, we moved the computation, even more, closer to the data sources, *i.e.,* on the IoT devices. We literally spread the computation across multiple computing entities: gateways, devices, and so on. This allows building Edge IoT applications that can fully exploit the power of the paradigm. This requires allocating the processing algorithms on edge devices based on the resources offered by each device. However, IoT devices, given their embedded nature, usually introduce strong system, networking, and energy constraints that have to be considered during the design phase of the application.
In this chapter, to practically demonstrate how to design an Edge IoT application and its devices, we presented a simple framework to design a special class of IoT devices called Smart Audio Sensors (SASs), which are audio devices able to autonomously record audio streams, locally perform computations on the recorded streams and send the results of these computations over a wireless link. In the beginning, we designed a device that was a simple audio recorder, with Bluetooth connectivity to transmit the raw audio stream. However, we had to deal with a Bluetooth link that does not have enough bandwidth to carry an audio stream sampled at 44.1KHz with 16-bit resolution. Therefore, we decided to migrate most of the computations directly into the audio device firmware. More in detail, we implemented the entire software flow (Figure 4.3) to extract the mel-coefficients from the raw audio stream. We defined a mathematical framework to design parameters of the mel extraction software flow. Such framework is based on two conditions, namely the *Real-Time condition* (Equation Formula (4.6)) and the *Buffering-Processing condition* (Equation Formula (4.7)), that have to be satisfied by the flow parameters.

After that, we have integrated the SAS into the AGILE gateway ecosystem and we have developed an ad-hoc module, called Configuration Over The Air (COTA), to remotely configure and push the mel-flow parameters without direct intervention on the device firmware. This module offers a user interface (UI) through the AGILE UI, and it is also able to detect if

the inserted configuration is valid or not.

The proposed solution has been validated by deploying the proposed Edge IoT device (*i.e.,* SAS) in a real smart office scenario. This device is responsible for collecting and locally computing the mel-coefficients while transmitting the computed features to an AGILE gateway instance. The latter simply receives the mel-coefficients from its wireless radio interface and runs a micro-service that executes a purposely trained anomaly detection algorithm, in charge of detecting anomalous events in the received feature-transformed stream. We compared two different options, namely Elliptic Envelope and Isolation Forest, in terms of average computing latency and user CPU load at the gateway level. We observed that, on the AGILE gateway instance, the best model is the Elliptic Envelope, being it two orders of magnitude faster and one order of magnitude lighter than the Isolation Forest counterpart.

The approach presented in this chapter has been validated considering a static allocation of processing algorithms on different entities based on the system constraints imposed by the application, the hardware devices, and the selected networking interface. We partially overcame the problem of static allocation by introducing the concept of Configuration-Over-The-Air, which allows us to reconfigure the main parameters of deployed algorithms. However, the possibility to dynamically re-allocate the computation on the different entities (*i.e.,* gateways, devices, etc.) allows having more resilient and powerful applications where, for instance, components can be migrated or updated or removed or introduced. This may give the flexibility to unleash even more power of the Edge Computing paradigm. In the direction of Edge Intelligence, the same approach may be adopted to retrain and redeploy machine learning models based on conditions or when a model is not valid anymore (*e.g.,* accuracy below a threshold). Considering the industrial scenario presented in Section 4.5.4, a possible improvement may be the automatic retraining and redeployment of anomaly detection models based on the evolution of the system (*e.g.,* replacement of a bearing). In this perspective, in the next chapter, we present a novel class of hardware chips, known as Edge AI Accelerators, specially designed to efficiently execute cloud-scale AI models at the edge, *e.g.,* deep learn-

ing models. We will provide a wide characterization of them in order to understand their pros and cons, the system requirements, and the effort that practitioners have to invest in order to adopt these devices in their applications.

Chapter 5

Novel hardware platforms for Edge Intelligence

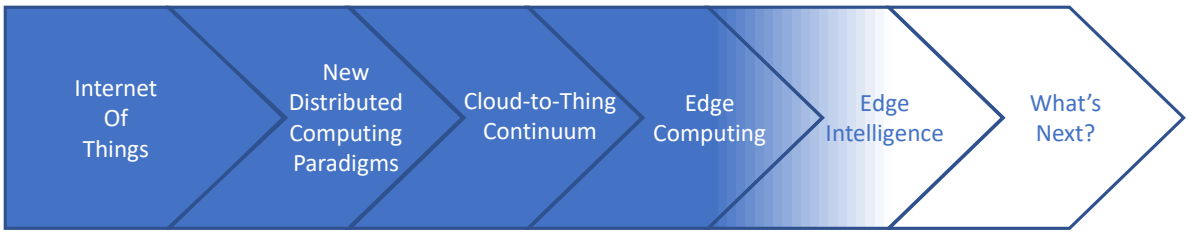


Figure 5.1: This chapter presents a systematic overview of Edge AI accelerator chips available on the market by describing their requirements, constraints, and performances in Edge IoT applications. These chips are a fundamental building block for Edge Intelligence IoT applications since enable the efficient execution of AI algorithms at the edge from the computational and energetic points of view. Practitioners can design and implement intelligent Edge IoT applications based on these hardware platforms and removing the dependency from Cloud Computing to infer complex AI models.

.

5.1 Introduction

Edge IoT applications, *e.g.,* personal-scale sensing applications, are increasingly pushing the inference part of AI models to edge devices such as IoT, smartphones, wearables, sensors, etc. This transition offers attractive

Part of this chapter appears in the following publication that I co-authored:
M. Antonini, T. H. Vu, C. Min, A. Montanari, A. Mathur, and F. Kawsar, "Resource Characterisation of Personal-Scale Sensing Models on Edge Accelerators". In *Proceedings of the First International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things - AIChallengeIoT'19.* New York, NY, USA: ACM Press, 2019, pp.49-55. DOI: 10.1145/3363347.3363363

benefits concerning privacy, performance, and cost. In the last 18 months, this shift has resulted in the emergence of a brand-new class of neural chips aimed at inferences at the edge. The proposition is remarkable; for the first time, we can move away from software accelerators and push cloud-scale AI models into edge devices without compromising accuracy. Naturally, these edge accelerators are uncovering exciting opportunities for building powerful Edge Intelligence IoT applications with complicated learning objectives and demanding computations, and, as Figure 5.1 shows, we are posing the pillars for Edge Intelligence.
There have been several attempts to understand the performance characteristics of human sensing models on smart devices like smartphones and commodity devices [114, 115]. However, the characterization of Edge AI accelerators is still at a very early stage. To this end, we take a systematic look at a set of edge accelerators, their working principles, and their performance in executing a variety of human sensing models.

This chapter presents a systematic characterization [116] of seven different accelerator configurations (Google Coral Dev Board, Google Coral Accelerator with Raspberry Pi (hereinafter, RPi) 4B and 3B+, NVIDIA Jetson Nano with native TensorFlow GPU and TensorRT, and Intel Neural Compute Stick with RPi 4B and 3B+) running eight different deep learning models with three tasks (motion, audio, and image). We report on their execution performance concerning memory, execution time, and energy overhead and share observations that lay an empirical foundation for both the evolution of these accelerators and their usage in Edge IoT applications.

	Coral Dev Board	**Coral Accelerator + Raspberry Pi 3B+**	**Coral Accelerator + Raspberry Pi 4B**	**NVIDIA Jetson Nano**	**Intel NCS2 + Raspberry Pi 3B+**	**Intel NCS2 + Raspberry Pi 4B**
CPU	Quad-Core Cortex A53	Quad-Core Cortex A53	Quad-Core Cortex A72	Quad-Core Cortex A57	Quad-Core Cortex A53	Quad-Core Cortex A 72
Memory	1 GB LPDDR4	1 GB LPDDR2	2 GB LPDDR4	4 GB LPDDR4	1 GB LPDDR2	2 GB LPDDR4
AI Chip	Google EdgeTPU	Google EdgeTPU	Google EdgeTPU	128 Core Maxwell GPU	Intel Movidius Myriad X VPU with 16 SHAVE cores	Intel Movidius Myriad X VPU with 16 SHAVE cores
On-Chip Memory	8 MB	8 MB	8 MB	Shared with CPU	512 MB LPDDR4 + 2.5 MB Centralized	512 MB LPDDR4 + 2.5 MB Centralized
AI Chip Interface	PCIe	USB 2.0	USB 3.0	PCIe	USB 2.0	USB 3.0
AI Chip OPs	4 TOPs	4 TOPs	4 TOPs	472 GFLOPs	1 TOPs	1 TOPs
Network Interfaces	ETH, WiFi, BT	ETH, WiFi, BT	ETH, WiFi, BT	ETH	ETH, WiFi, BT	ETH, WiFi, BT

Table 5.1: Specification of the hardware platforms used in the study (source: [116], p. 50).

The rest of the chapter is structured as follows: Section 5.2 describes the different AI accelerators, their characteristics, and the steps required to adapt a cloud-scale deep learning model, *i.e.,* a TensorFlow or Keras model, to run an edge AI accelerator. We describe the deep learning sensing models considered in this chapter in Section 5.3. Section 5.4 describes the benchmarking framework and outlines the systematic report on performance metrics of the accelerators for the models. Finally, we present some final remarks and key insights in Section 5.5. It is worth mentioning that the design of the benchmarking framework and characterization experiments were carried out during a visiting period abroad at Nokia Bell Labs (Pervasive Systems research group) in Cambridge (UK) from June 2019 to October 2019.

5.2 Edge AI Accelerators

Many big tech companies have proposed different hardware solutions to accelerate the execution of deep learning algorithms at the edge of the network. This opens many different new application markets where complex AI can be executed on devices. In this study, we consider seven different hardware/software configurations with three types of edge accelerators. Table 5.1 reports their hardware specifications; we consider two TensorFlow frameworks for Jetson Nano, TensorFlow GPU[1] and TensorRT[2].

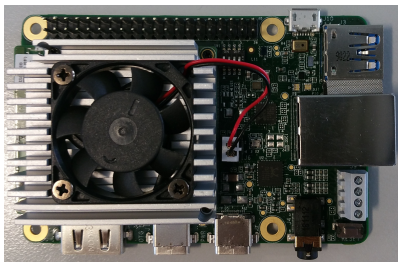
Google Coral[3]: In summer 2018, Google announced the edge version of its Tensor Processing Unit (TPU) platform known as EdgeTPU under the brand name Coral. The EdgeTPU is an application-specific integrated circuit designed to deliver up to 4 Tera OPerationS (TOPS) per second using a power budget of 2 watts (2 TOPS/watt). This chip supports only signed integer operations at 8 and 16 bits and it comes with approximately 8 MB of on-chip RAM used to cache the model's parameters. Since this board has been strictly designed for optimal inference, it currently supports only TensorFlow Lite models that meet specific requirements [117] (*e.g.,* parameter quantization). The EdgeTPU is offered by Google in ten

[1] `https://www.tensorflow.org/install/gpu`
[2] `https://docs.nvidia.com/deeplearning/frameworks/tf-trt-user-guide/index.html`
[3] `https://coral.ai/`

different flavors[4] grouped in two main categories: Prototyping products and Production products. The former group contains four different products: two dev-kit called *Coral Dev Board* with respectively 1 and 4 GB of RAM, a *Coral Dev Board mini*, and a USB dongle called *Coral Accelerator*. The Production products comprise four PCIe boards, which mount the EdgeTPU chip, with different form factors (Mini PCIe, M.2 A+E key with single or dual edge TPU, and M.2 B+M key), a System-on-Module (SoM) ready to be plugged in the target device, and the spare solderable module. Additional commercial devices, which embed the Coral TPU, are offered by ASUS that offers the ASUS AI Accelerator PCIe Card card, which supports up to 8 Coral M.2 cards, and the ASUS Tinker Edge T, a single-board computer based on the Coral SoM board.
For the sake of this chapter, we consider only two Coral devices: the 1 GB Coral Dev Board and the USB Coral Accelerator. The former dev-kit (See Figure 5.2a) is a single board computer that hosts onboard RAM, storage, and other peripherals. The second device is a USB device that requires a host device, thus, we use Raspberry 4B (See Figure 5.2b) and 3B+. The biggest difference between Raspberry Pi (hereinafter, RPi) 4B and 3B+ regarding our benchmark study is the AI chip interface. RPi 4B supports USB 3.0 (with a maximum rate of 5 gigabits per second), whereas RPi 3B+ supports USB 2.0 (with a maximum rate of 480 megabits per second). We will evaluate the impact of the interface later in this chapter.

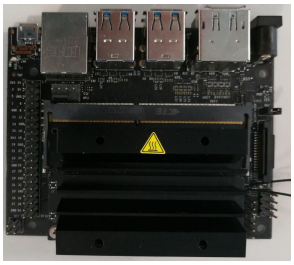
NVIDIA Jetson Nano: In March 2019, NVIDIA has announced and made available a new GPU-powered board, known as Jetson Nano, targeting the maker community (See Figure 5.2c). This board hosts a 64-bit quad-core Arm Cortex-A57 CPU and an NVIDIA Maxwell GPU with 128 CUDA-cores able to deliver up to 472 GFLOPs running float operations. CPU and GPU share a common bank of 4 GB of LPDDR4 RAM, which requires the tuning of the memory reservation between CPU and GPU. Since this board runs a full-fledged operating system derived from Ubuntu, the board natively supports TensorFlow 1.x compiled with GPU support and TensorRT 5. In October 2020, NVIDIA announced and made available a new version of Jetson Nano with 2 GB of RAM.

[4] `https://coral.ai/products/`

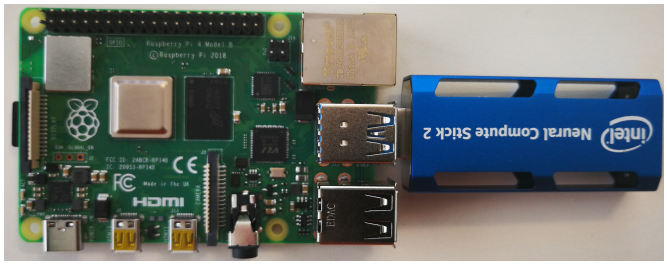


(a) Google Coral Dev Board

(b) Coral Accel. with RPi 4B



(c) NVIDIA Jetson Nano



(d) Intel NCS2 with RPi 4B

Figure 5.2: Hardware platforms used in the study (source: [116], p. 51).

Intel NCS2: Intel has made available the new Intel Movidius Myriad X Vision Processing Unit (VPU), a low-power System-on-Chip (SoC) designed to accelerate deep-learning deployments and computer vision applications. This chip includes several processors and computing units optimized for high parallelism and DNN inference making it capable of running up to 4 TOPS with a power budget of 1.5 watts. The VPU is available in two different in-package configurations: without in-package additional RAM and with 4GBits (512 MBytes) in-package RAM. Intel has released a USB 3.0 dongle known as Intel Neural Compute Stick 2 (NCS2) that hosts the Movidius Myriad X VPU with 4GBit of RAM. This USB stick can be plugged as a co-processor to speed-up the inference of neural networks. NCS2 requires the model to be optimized using the OpenVINO framework[5]. We also consider Raspberry Pi 4B (See Figure 5.2d) and 3B+ as the mainboard for benchmarking NCS2.

5.2.1 Model compilation workflow

Since edge accelerators have different constraints and requirements, different optimizations need to be applied to fully exploit the hardware acceler-

[5] https://software.intel.com/content/www/us/en/develop/tools/openvino-toolkit.html

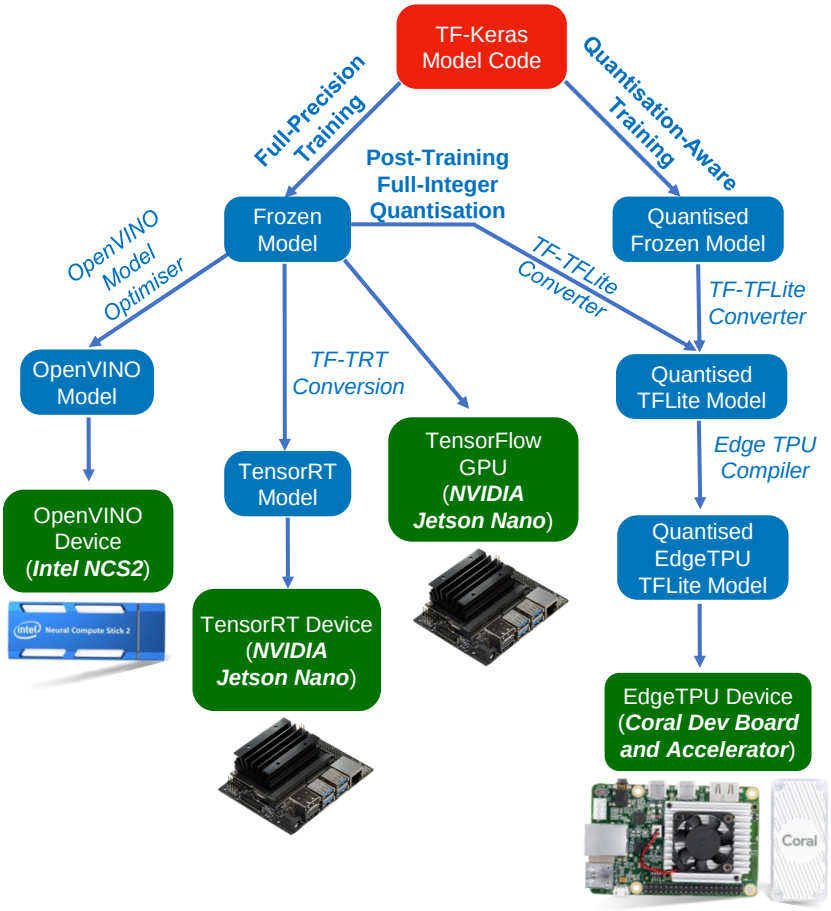


Figure 5.3: Compilation workflow (source: [116], p. 51).

ation. Figure 5.3 presents the required steps for the edge accelerators we use for our benchmark study. In this chapter, we consider deep learning models that have been implemented with native TensorFlow or with Keras with TensorFlow as backend.

NVIDIA Jetson Nano: The first step is to train the algorithm by applying full-precision training which outputs a model with parameters expressed as 32bit floating-point numbers. Then, the model needs to be frozen to convert all the inner variables to constant and make the model ready for the inference phase and further optimization. The frozen model can natively run on the Jetson Nano using native TensorFlow with GPU support. Jetson Nano also supports TensorRT, a library that optimizes the execution of neural networks by replacing the implementations of some layers with more efficient ones. TF-TRT converter needs information including input tensor name and shape, precision mode (FP16 or FP32),

size of the inference batch, and size of the reserved execution memory. The output is a TensorFlow-TensorRT frozen model ready to be deployed.

Intel NCS2: Intel NCS2 also needs the full-precision frozen model to generate a model compatible with it. Then, the model is converted using the OpenVINO model optimizer[6], a cross-platform tool that runs static analysis and adjustments of the model. The optimizer needs only the shape of the input tensor and the floating number precision (*e.g.,* FP16). It returns a set of files, known as Intermediate Representation (IR), that are used by the Inference Engine API to run the model over the Movidius Myriad X VPU.

Google Coral: Since EdgeTPU does not support floating-point parameters, it is essential to represent the model weights as signed-integer numbers, *i.e., quantization*. The EdgeTPU run-time supports quantization-aware training [118, 119] which performs parameter quantization at training time. The model is frozen after this step and then converted to TensorFlow Lite format. As an alternative, from the v12 of the EdgeTPU run-time, it supports post-training full-integer quantization [120]. This procedure quantizes all the parameters and activations without re-training the model. It requires a small and representative dataset, which might be a part of the training set, to define the quantization range. Note that, while quantization-aware training requires the additional cost for re-training, higher accuracy is achievable as it is generally more tolerant to lower precision values. The last step is to feed the quantized TensorFlow Lite model to the EdgeTPU compiler[7], which has been dockerized by us[8]. The compiler verifies if the model meets the requirements [117]. It statically defines 1) how weights are allocated in the edge TPU on-chip memory 2) the execution of the TensorFlow Lite graph on the acceleration hardware.

Although the model meets the requirements, it is possible that some operations could not be supported by the EdgeTPU run-time. Then, the compiler tags them as *unsupported* and forces the execution of those and subsequent operations on the CPU instead of TPU. It is also possible that

[6] https://docs.openvinotoolkit.org/latest/openvino_docs_MO_DG_Deep_Learning_Model_Optimizer_DevGuide.html

[7] https://coral.ai/docs/edgetpu/compiler/

[8] https://github.com/mattiantonini/edgetpu-compiler-container

Task	Model	Architecture	Characteristic	Parameters (Millions)
Motion	Aroma (**A**) [121]	CNN+DNN + Residual Connections	Excluding LSTM	0.385
Audio	Audio Classification (**E**) [122] (Emotion Only)	CNN + FC	No MFCC extraction	0.249
	DKWS (**D**) [123]	CNN + FC	No MFCC extraction	0.923
Vision	SqueezeNet V1.0 (**S**) [124]	CNN + FC	Fire modules	1.235
	MobileNet V1 (**M**) [125]	CNN + FC	Traditional and depth-wise convolution layers	4.232
	EfficientNet-EdgeTPU (**E2**) [126]	CNN + FC	Traditional CNN	5.440
	Inception V1 (**I**) [127]	CNN + FC	Inception Modules	6.618
	DenseNet121 (**D2**) [128]	CNN + FC + Residual Connections	Each layer is connected to all the previous layers	7.978

Table 5.2: Specification of sensing models used in the study (source: [116], p. 52).

the model's weights do not fit in the TPU on-chip memory but the operations are still executed on TPU. In this case, the weights are dynamically streamed from off-chip memory, *e.g.,* RAM, to the on-chip memory, introducing additional latency. Even only a few tens of bytes stored in the off-chip memory may drastically degrade the device's performance.

5.3 Personal-scale Deep Learning Sensing Models

Given that we target Edge IoT applications, with a focus on personal-scale sensing applications, we select a broad range of deep learning sensing models tailored for motion, audio, and vision tasks. Table 5.2 summarizes the architectures and properties of the models we benchmark. They cover diverse types of convolutional neural networks (CNN) architecture, *e.g.,* with and without auxiliary branches, residual connections, depth-wise convolution, and fully connected layers.

Motion task: Motion sensors, *e.g.,* , accelerometer, gyroscope, and magnetometer, are crucial components in smart devices as they provide rich information about user context [129, 130]. One of the most desired applications of motion tasks is human activity recognition (HAR). For the

HAR model, we select Aroma [121]. It consists of two hierarchical classifiers. The first classifier exploits 8 convolution layers to automatically learn low-level features from the distribution of sensor data. These low-level features are then classified into different simple activities, *e.g.,* standing and walking, using a fully connected layer and a softmax classifier. On top of this classifier, a long short-time memory (LSTM) model is applied to learn and extract meaningful complex activities, *e.g.,* commuting, from temporal relationships in the low-level features over time. However, since the current accelerators do not support LSTM modules, we profile only the convolutional part of the model.

Audio task: Audio understanding is always on the front line of machine learning and enables a variety of sensing tasks. Using edge accelerators is promising to enable on-device audio processing, which provides clear benefits such as privacy assurance and low latency. In this chapter, we consider two different audio tasks, keyword spotting and emotion recognition. Keyword Spotting is a vital component in virtual assistant applications, *e.g.,* Google Assistant or Amazon Alexa. To this end, we use a deep keyword spotting (DKWS) model [123], which is a three-layer deep convolutional neural network. It is capable of detecting several spoken keywords, *e.g.,* yes and no. The goal of the emotion recognition task is to capture the human psychological state unobtrusively in daily lives using the speaker's utterance. We follow the implementation proposed in [122] which comprises of 3 CNN layers, a fully connected layer, and a softmax classifier to classify four different emotions, including neutral, upset, happy, and angry. For the benchmark, we focus only on the model execution and exclude the pre-processing steps such as the extraction of Mel-Frequency Cepstral Coefficients (MFCC) features.

Image task: Image recognition is one of the most active areas of machine learning with many applications [131]. Given the popularity of these models, we profile 5 different types of neural networks, including SqueezeNet V1.0 [124], MobileNet V1 [125], EfficientNet [126], Inception V1 [127], and DenseNet121 [128].

These models cover a variety of network architectures. SqueezeNet introduced Fire modules which makes use of 1x1 convolution to squeeze the number of input channels and a 3x3 filter to reduce the total number of

parameters. MobileNet V1 introduced depth-wise convolution, which applies convolutions on each channel before combining the filters to reduce the number of parameters. Recently Google has developed EfficientNet – a new family of CNN architecture which can be optimized for different platforms; we use EfficientNet-EdgeTPU, which is optimized for the Google EdgeTPU. EfficientNet utilizes architectural search (grid search on depth and width) to find a near-optimal architecture, which optimizes both depth and width of a neural network. Inception V1 includes 22 convolution layers with branches of 1x1, 3x3, and 5x5 convolutions and a fully connected layer. DenseNet121 contains both convolution layers and Dense blocks which maintain residual connections from one layer to all previous layers in the same block.

5.3.1 Scope of the benchmark

Our goal is to investigate the resource characteristics of edge accelerators under a range of deep learning sensing models usually adopted in IoT applications. However, there are several compilation and optimization parameters that affect the resource characteristics and inference accuracy as well. For example, for the precision mode, FP16 (half-precision point) could occupy less memory and lower inference latency compared to FP32 (full-precision point) but could result in accuracy degradation. In this chapter, as an initial step, we select personal-scale sensing models, which are widely used in IoT applications, as a key independent variable and aim at investigating their resource characteristics. To this end, we set the compilation and optimization parameters to the default values used in each edge accelerator. For example, for the precision mode, we used FP32 and FP16 for TensorFlow GPU and TensorRT on Jetson Nano, respectively. Intel NCS2 was also set to FP16. The investigation of the compilation and optimization parameters and their impact on the accuracy is left for future developments and works.

In this aspect, we do not include other operations into the benchmark, which are required for the entire sensing pipeline such as sensing, data transmission, and data management. It is because their resource characteristics are not affected by edge accelerators.

5.4 Performance Benchmarks

We conduct a set of benchmarks to characterize the resource usage of personal-scale sensing models on edge accelerators. We consider end-to-end model performance metrics of memory usage, execution time, and energy consumption. Given the proprietary nature of each accelerator and the limited availability of APIs, we could not include accelerator-related metrics such as TPU usage. We expect that the outcome of these experiments uncovers the feasibility of running sensing models and applications on edge accelerators with the final aim at build a pillar for future Edge IoT applications.

5.4.1 Experimental Setup

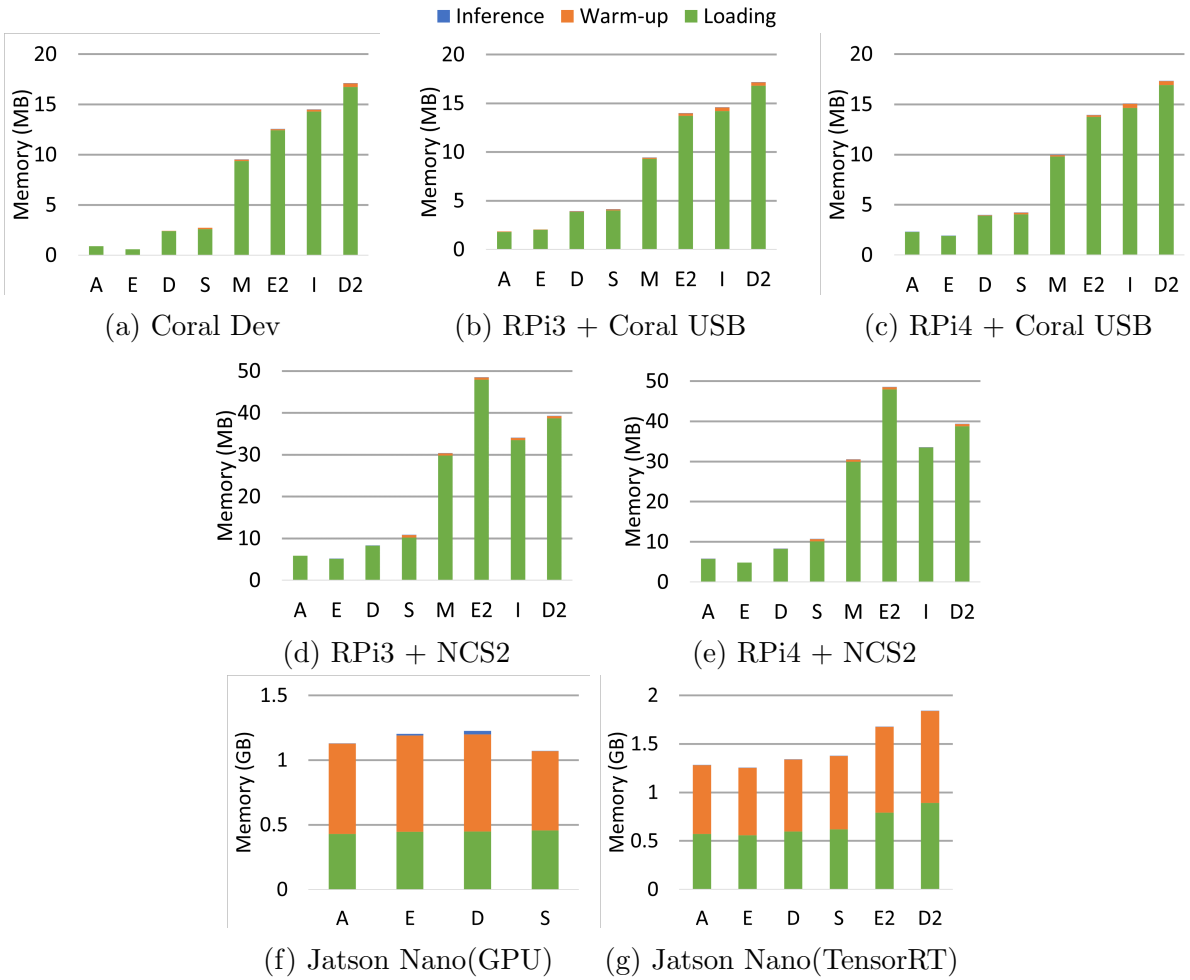
To systematically explore the resource characteristics, we develop a benchmark script that executes the sensing models repeatedly and measures memory usage and execution time. We perform 20,000 separate inferences for every model on each platform and report the average figures. For all the experiments, we use a batch size of 1 to consider applications where the models need to process and label sensory inputs as quickly as possible, without additional latency introduced by batching several data points. For energy measurements, we use a High-Voltage Monsoon Power monitor[9].

We consider three steps in the model lifetime: *loading*, *warm-up*, and *inference*. In the loading, the model is loaded into the accelerator's on-chip memory. The warm-up refers to the first execution of the model, and the inference is for the subsequent executions. We separate the warm-up from the inference since accelerator run-time completes hardware initialization (*e.g.,* model caching and memory allocation) upon the first request of the model execution.

5.4.2 Memory Usage

We investigate the memory footprint, which is known to be a key resource bottleneck in the processing of deep learning models on embedded devices due to a large number of parameters.

[9] `https://www.msoon.com/high-voltage-power-monitor`



(a) Coral Dev (b) RPi3 + Coral USB (c) RPi4 + Coral USB

(d) RPi3 + NCS2 (e) RPi4 + NCS2

(f) Jatson Nano(GPU) (g) Jatson Nano(TensorRT)

Figure 5.4: Host device memory footprint (notice the different scale for the y-axis) (source: [116], p. 53 for figures (a), (d), and (f)).

The first observation is that, when a model is executed on an accelerator, the memory is gradually allocated at three different times: (1) when the model is loaded, (2) at the first inference (warm-up), and (3) during subsequent inferences. It is important to notice that the loading and warm-up memory remains allocated for all subsequent inferences and it is deallocated only when the model is unloaded. We further discover that the way the memory is handled depends on the hardware architecture of the accelerator and also on its run-time software. For the accelerators with on-chip dedicated memory (memory isolated from system memory), *i.e.*, Coral Dev (Figure 5.4a), Coral Accelerator (Figure 5.4b and Figure 5.4c), and NCS2 (Figure 5.4d and Figure 5.4e), the compilation pipeline opti-

mizes the model to keep as much of the network as possible on the on-chip memory to ensure low latency access, thereby resulting in low utilization of the memory on the host device. Even for large models (*e.g.,* EfficientNet, Inception V1, and DenseNet), we observe that only 13–18 MB are allocated during the loading and the warm-up phases on the host memory of the Coral Dev Board (Figure 5.4a) and on the Raspberry Pi memory used with the Coral Accelerator (Figure 5.4b and Figure 5.4c). Also on the Intel NCS2 (Figure 5.4d and Figure 5.4e), the host memory allocated is a bit higher than the Coral devices but still marginal. For example, the largest amount of allocated memory is for the EfficientNet model with about 50 MB of memory allocated for loading and warm-up (for the same model only 14 MB are allocated when using the Coral devices). For the inference phase, the host memory used is even lower, we measure less than 10 KB across all models both on Coral devices and NCS2.

On the Jetson Nano, however, we notice that significantly more memory is allocated during loading and warm-up, as shown in Figure 5.4f and in Figure 5.4g for the TensorFlow GPU and TensorRT run-times, respectively. Only between 1 MB and 10 MB are used during inference instead. We hypothesize that this is because the TensorFlow and TensorRT runtimes are still not optimized for constrained devices with limited memory. The implication is that, since on Nano the memory is shared between the CPU and the GPU (*i.e.,* there is no dedicated memory for the GPU), the more memory is used for a deep learning model the less is available for the operating system and other processes running on the CPU. As a consequence, the Jetson Nano board running TensorFlow GPU could not run the large models (MobileNet, EfficientNet, Inception, and DenseNet), because the free memory is not enough to load and perform the warm-up phase of these models. On the other hand, the Jetson Nano board using TensorRT as runtime was able to run EfficientNet and DenseNet, however with a high memory demanding. This is important because a real system would need to execute other tasks in addition to the model inference (*e.g.,* communication, user interface, and data logging) requiring memory for each of these tasks. This might become impossible if most of the free memory is consumed by model execution, as on the Jetson Nano. Therefore we observe that devices with dedicated on-chip memory and with software

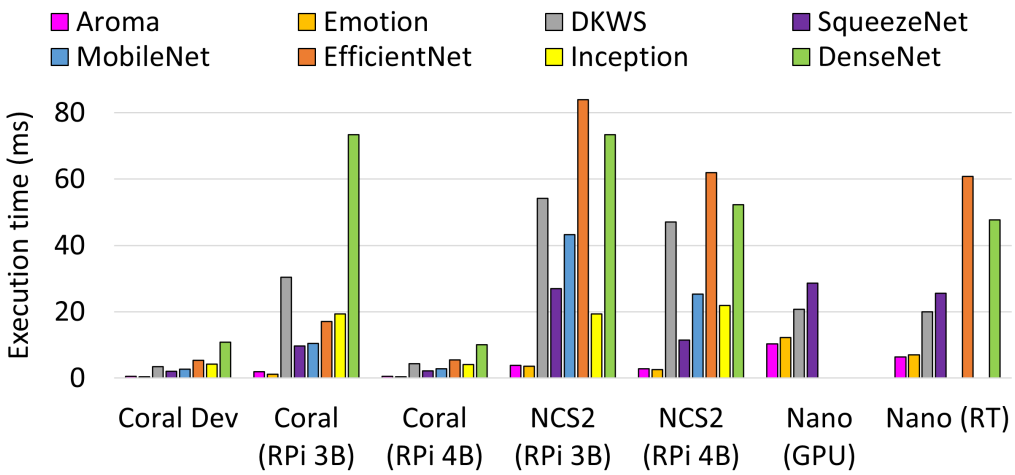


Figure 5.5: Execution time (inference time) on different platforms (adapted from [116], p. 53).

pipelines capable of optimizing the models' memory requirements, such as Coral Dev, Coral Accelerator, and NCS2, are preferable for systems that need to run processes in addition to the model execution. We remember that this was one of the first characterizations of these devices, thus further analyses are required to fully understand their potential.

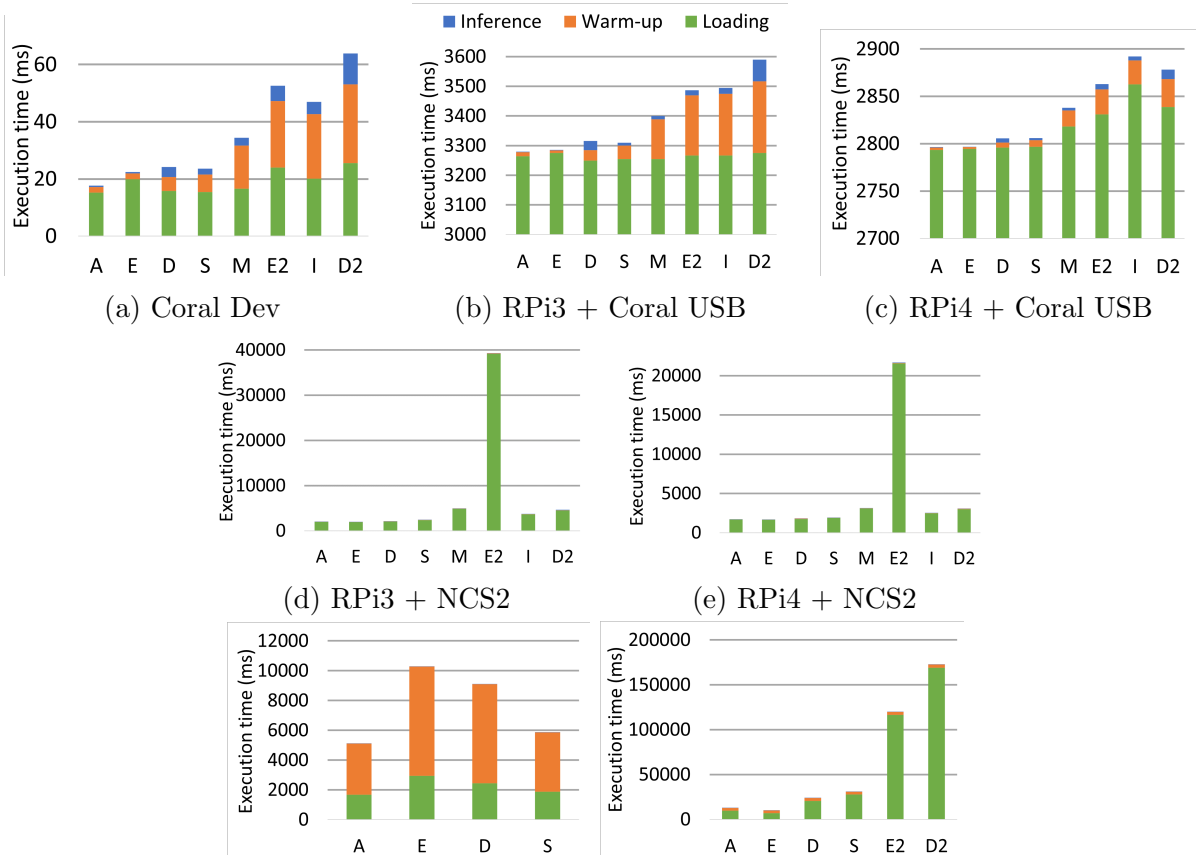
5.4.3 Execution time

We look into the execution time of the model inference, which is a key metric for IoT applications at the edge of the network that need to react quickly to input data. Figure 5.5 shows the inference time for different platforms. The results show a couple of interesting findings. First, the execution time is largely different, depending on the edge platform. In general, Coral Dev and Coral (RPi 4B) outperform other platforms. For example, one execution of the SqueezeNet model takes 2 ms both on Coral Dev and Coral (RPi 4B), whereas it takes 9.5 ms, 27 ms, 11 ms, 29 ms, and 26 ms on Coral (RPi3B), NCS2 (RPi 3B), NCS2 (RPi 4B), Nano (GPU), and Nano (RT), respectively.

Second, as expected, the inference is faster for simpler models. For example, the execution time on Coral Dev is 0.5, 0.5, 3.5, 2.1, 2.7, 4.2, and 10.9 ms for Aroma, Emotion, DKWS, SqueezeNet, MobileNet, Inception, and DenseNet (see Table 5.2 for the number of parameters). DenseNet is the slowest on the Coral devices because the entire model cannot be allo-

cated on the on-chip accelerator's memory (∼8 MB) and therefore part of the parameters was allocated on the host memory (1.9 MB). This causes additional latency because parameters need to be moved between the host memory and the accelerator on-chip memory. The trend showing that simpler models run faster is observable also on different platforms. However, NCS2 (RPi 4B) is an exception to this tendency. The execution time of DKWS is 47 ms, whereas that of SqueezeNet, MobileNet, and Inception is 11ms, 25ms, and 22ms. We hypothesize that this is because DKWS has an unusual kernel size in its first convolutional layer (*i.e.,* 8x20) which translates to heavy computation on the input data and possibly causes inefficiency because the Movidius chip is not optimized for this kernel size. We find a similar behavior on EfficientNet and DenseNet with the Intel NCS2. While the number of parameters of EfficientNet is lower than that of DenseNet, its execution time is much higher on the Intel NCS2. We conjecture that this is because EfficientNet has been designed and optimized for the EdgeTPU architecture.

We delve deeper into the execution time of the loading and warm-up steps as shown in Figure 5.6. Interestingly, edge platforms show a different tendency. We notice that the loading and warm-up times for all models on Coral Dev are always below 30 ms while the Coral accelerator, NCS2, and Jetson Nano take several seconds. Knowing loading and warm-up times of these accelerators is important in reactive systems where different models need to be executed on-demand to respond to certain sensory inputs. In this context, models are dynamically loaded to perform a few inferences and then unloaded. Large loading and warm-up times will reduce the performance of the system, making it difficult to promptly respond to input data. From our benchmarks, we can conclude that Coral Dev is suitable to support reactive systems where multiple models need to be loaded and unloaded over time while NCS2 and Jetson Nano are more suitable for applications where a single model is used for long periods of time, amortizing the loading and warm-up cost.



(a) Coral Dev (b) RPi3 + Coral USB (c) RPi4 + Coral USB

(d) RPi3 + NCS2 (e) RPi4 + NCS2

(f) Jatson Nano(GPU) (g) Jatson Nano(TensorRT)

Figure 5.6: Execution time for loading, warm-up, and inference operations. X-axis reports shortened model names as in Table 5.2 (source: [116], p. 53 for figures (a), (e), and (f)).

5.4.4 Energy

Energy is a precious resource in battery-powered Edge IoT devices empowered with edge accelerators. When we design a device, we have to guarantee that the battery life is enough to last at least the time between to rests, *e.g.,* 24 hours for a smartphone, 7 days for a wristband, and so on. Here, we define the energy overhead as the energy which is additionally consumed for the model execution. To obtain the net energy increase, we measure the difference between the average power consumed during the *model execution* and when the board is *idle* and, then, multiply it by the model execution time. These are estimations of the energy overhead using power consumption collected using an empirical approach.

Interestingly, as shown in Figure 5.7, the energy overhead varies much depending on the accelerator. For example, Coral Dev and Coral (RPi 4B)

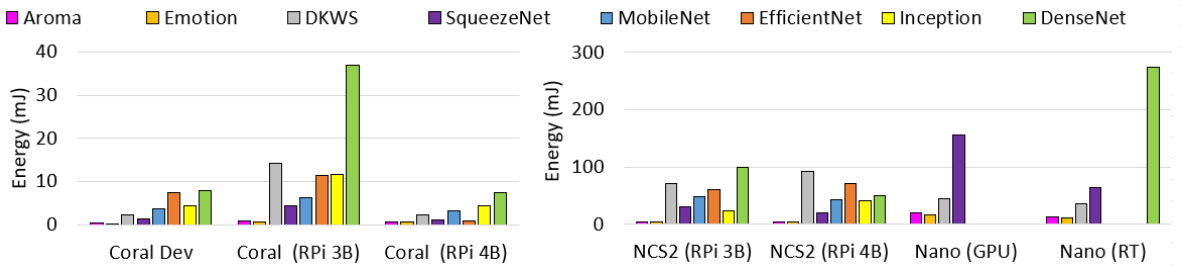


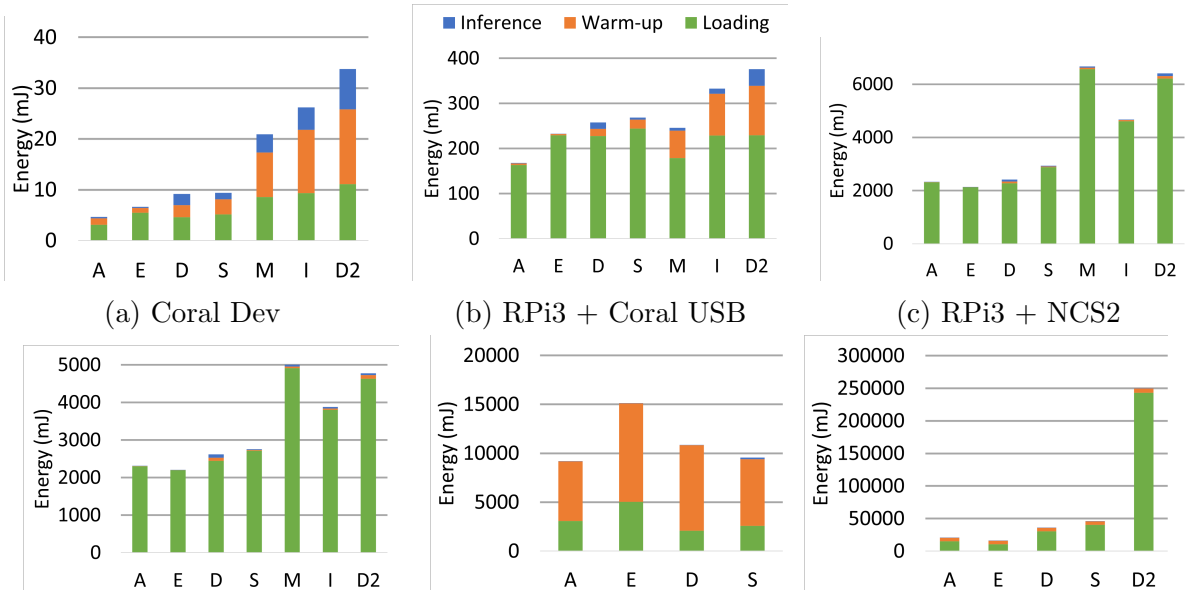
Figure 5.7: Energy consumption on different platforms at inference time (adapted from [116], p. 54)

Coral Dev	Coral (RPi 3B)	Coral (RPi 4B)	NCS2 (RPi 3B)	NCS2 (RPi 4B)	Jatson Nano
3.1	2.4	4.2	2.8	3.6	1.2

Table 5.3: Idle power of platforms (W) (adapted from [116], p. 54).

mostly consume less than 10 mJ for a single execution (inference) regardless of the model. On the other hand, the energy overhead ranges from 5 mJ to 274 mJ on NCS2 (RPi 4B) and Jetson Nano. We can also observe that the TensorFlow framework largely impacts the energy overhead, even with the same platform. On Jetson Nano, TensorFlow GPU generally consumes more energy than TensorRT. This is probably because TensorFlow GPU is not optimized for energy efficiency and it takes longer as well for the model execution.

Table 5.3 shows the power draw in the idle state, *i.e.,* when no operation is being executed. Interestingly, the power also varies much depending on the hardware specification. A Raspberry Pi 3B connected to an accelerator consumes less than 3 W, however, when it is connected to a Raspberry Pi4, the power draw is higher than 3.5W. This is due to the power draw of the Raspberry Pi 4B alone (without any accelerator), which is 2.9 W. Finally, Figure 5.8 reports the energy consumption of loading and warm-up phases for a subset of models and platforms. Even if these phases look to have a big impact on battery life, they do not deplete too much energy since these phases are needed to be executed only once, at the model loading phase. On the other hand, the inference phase may be repeated thousands of times.
More investigations are required on the energy consumption side considering also the pre-processing pipeline of data, *i.e.,* from sensors to the model.



(a) Coral Dev (b) RPi3 + Coral USB (c) RPi3 + NCS2

(d) RPi4 + NCS2 (e) Jatson Nano(GPU) (f) Jatson Nano(TensorRT)

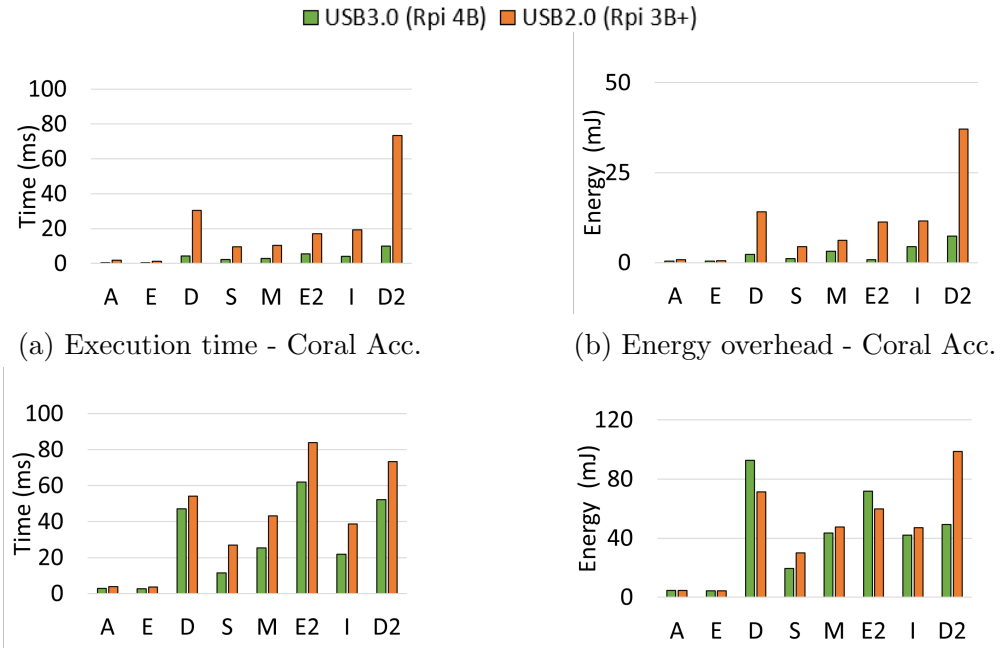
Figure 5.8: Energy overhead for loading, warm-up, and inference operations for a subset of models and platforms. X-axis reports shortened model names as in Table 5.2.

5.4.5 The impact of connection interfaces: USB3.0 (RPi 4B) vs USB2.0 (RPi 3B+)

According to the hardware specification, the main difference between RPi 4B and 3B+ is the interface with AI Chip as described in Table 5.1, *i.e.,* USB 2.0 on RPi 3B+ and USB 3.0 on RPi 4B. In this subsection, we quantify the impact of the interface on the performance of the model execution. Here, we focus on the latency and power consumption, which are mainly affected by the interface.

Execution time: Figure 5.9a and Figure 5.9c show the execution time (inference time) of Coral Accelerator and Intel NCS2, respectively. As expected, RPi 4B takes a shorter time than RPi 3B+ by virtue of its fast transmission via USB 3.0. Interestingly, the performance gap, *i.e.,* the difference of the execution time between RPi 4B and RPi 3B+, is different depending on the type of the accelerator. With the Coral accelerator, the ratio of RPi 3B+ to RPi 4B ranges from 3.1 (EfficientNet) to 7.3 (DenseNet). However, with Intel NCS2, the ratio mostly remains less than 1.7, except the SqueezeNet (2.4).

Energy: We also compare the energy overhead as shown in Figure 5.9b and Figure 5.9d. The results show that, in general, RPi 3B+ consumes



(a) Execution time - Coral Acc.

(b) Energy overhead - Coral Acc.

(c) Execution time - NCS2

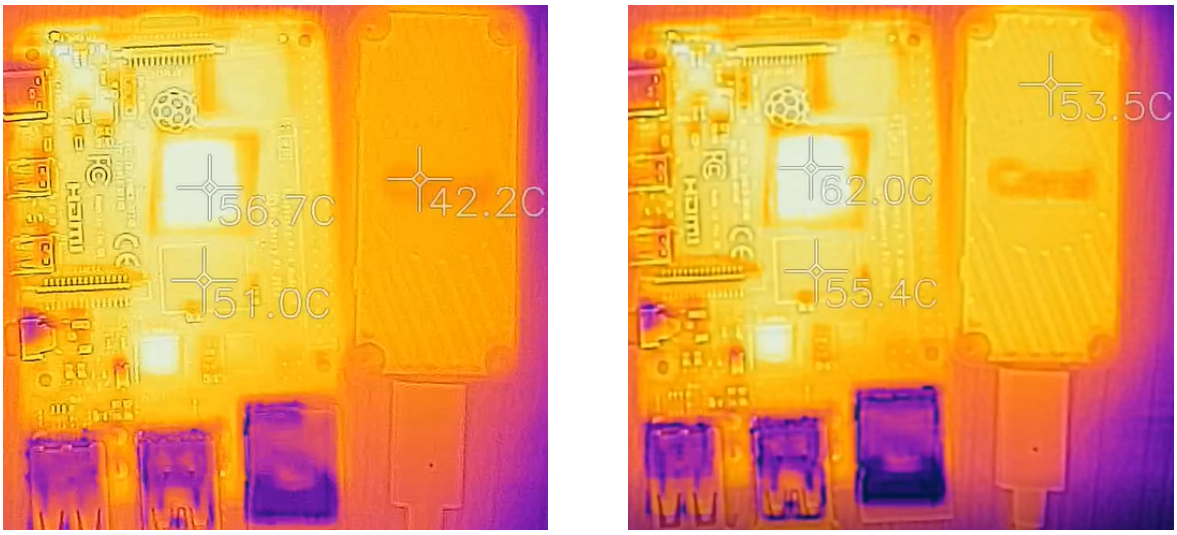
(d) Energy overhead - NCS2

Figure 5.9: Performance comparison between Edge AI Accelerators connected to host devices via USB3.0 (RPi 4B) and USB2.0 (RPi 3B+) (source: [116], p. 54 for figures (a) and (b)).

more energy for the model execution; the only exception is DKWS on Intel NCS2. The main reason is due to the increase in the execution time. However, the idle power of RPi 3B+ is much lower than RPi 4B. The idle power of RPi 3B+ is 2.4 W and 2.8 W with Coral Accelerator and Intel NCS2, respectively. This makes the average power of RPi 4B during the model execution (including the idle power) higher than that of RPi 3B+. For example, the average power of the Aroma model on RPi 4B is 5.0 W, whereas that on RPi 3B+ is 2.9 W. In this aspect, we estimate that the battery life of RPi 3B+ will be longer than that of RPi 4B assuming a battery with the same capacity.

5.4.6 Preliminary heating analysis

Besides the energy consumption or the time required to execute a model, another fundamental parameter that has to be considered when we design Edge IoT devices, especially for personal-scale applications, is the heat dissipated by chips. Usually, a running CPU is heated up by the heat produced in transistor junctions and this has to be dissipated into the sur-



(a) Idle (b) Continuous inference

Figure 5.10: Thermal images of a RaspberryPi 4 connected to a Coral Accelerator during idle phase (not inferencing) and running phase (continuous inference). Regarding the color scale, black/blue means cooler, yellow/white means warmer.

rounding environment, usually using heat-sinks, fans, or other strategies. Obviously, if the heat is not well dispersed, the device may become too hot and it may be damaged or, even, it may harm some people. Moreover, a hot device means that a lot of energy is "*wasted*".

Here, we present a preliminary insight about the heating of a RaspberryPi4 connected to a Coral Accelerator. We used a FLIR One Pro thermal camera[10], connected to an Apple iPad, we collect the thermal spectrum of the devices. However, given that RaspberryPi4 mounts a metallic enclosure around chips that has a low emissivity of infrared radiation, we overcome the problem by placing some black electric insulating tape on top of the chips. Since the tape has a high emissivity, we were able to get an estimation of the temperature of the devices. We sampled the temperatures of three different points: the RaspberryPi CPU, the RaspberryPi RAM, and the Coral Accelerator. Figure 5.10 shows the exact positions of thermal sampling points. At idle, when the device is not running a model, the mean temperatures of RaspberryPi's CPU and RAM chips are 56.7 °C and 51 °C, respectively. The Coral Accelerator was at 42.2 °C. If we start to continuously infer a deep learning model, the average temperature of devices quickly raise and, after 30 seconds, it reached 62 °C (RPi CPU), 55.4 °C

[10] `https://www.flir.it/products/flir-one-pro/`

(RPi RAM), and 53.5 °C. Even if these values are not looking too high for a computing device, however, they may be dangerous if someone touches them. Different standards and guidelines have been defined to identify the suitable range of temperatures. For example, the ASTM C1055 [132] standard identifies the range of temperatures and contact duration after that people may be injured (first-, second-, and third-degree burns). For example, exposures longer than 15 seconds at 56 °C, or for 5 seconds at 60 °C, causes a third-degree burn. This opens the topic of thermal characterization and optimization of Edge AI accelerators and, more in general of IoT devices, not only from the energetic perspective but also from the safety point of view.

5.5 Remarks

We attempted to characterize the resource performance and suitability of personal-scale deep learning models on a wide variety of edge accelerators. Beyond the mere numbers, our study further offers useful insights for the development of Edge Intelligent IoT devices on top of the Edge AI accelerators. First, as described in Section 5.2.1, the execution path of deep learning models on edge accelerator is not optimized, yet. For example, on Google Coral platforms, if an operation in the model is tagged as *unsupported*, the execution path is statically determined by putting the whole subsequent operations (including the untagged one) to the CPU. It implies that the position of the untagged operation affects the performance of the model significantly. Second, the interface between CPU and AI chip is a critical bottleneck. As reported in Section 5.4.5, even with the same Coral accelerator, USB 3.0 accelerates the execution time by three to seven times compared to USB 2.0. Last, careful scheduling is needed to support multiple sensing models. This is because the dynamic change in sensing models incurs significant overhead, *e.g.,* as shown in the cost of loading and warm-up on Jetson Nano (Section 5.4.3).

For the automated and scalable benchmark, we prototyped an end-to-end benchmarking toolkit. As a core component for the resource benchmark, it takes a sensing model and a target platform as input. Then, it

converts the given model to the platform-specific model binary as described in Figure 5.3 and performs the benchmark. As future developments, we envision this toolkit as a full-fledged, comprehensive framework for edge accelerators. If practitioners provide the sensing model, the test dataset, and the execution requirements, *e.g.,* latency, accuracy, and energy budget, the toolkit can automatically test the given model on various edge accelerators in the background and recommend the most suitable platform providing an in-depth report on the expected performance. This might become a fundamental tool for Edge IoT practitioners to evaluate their future Edge Intelligence IoT applications.

However, an important aspect of Edge Intelligence applications is still missing: how can we design AI models to be executed at the edge? A few works have been published where authors try to optimize the architecture for neural networks for the target hardware platform. An example is EfficientNet [126], we evaluated it in this chapter, that has been designed using a grid-search approach to identify a good combination of network depth and width. In the next chapter, we will present an end-to-end (from data selection to the model hyper-parameters, passing through the feature extraction) framework to design an Edge IoT application that executes a small and performing neural network on embedded devices. Our framework, together with the benchmarking tool developed and presented in this chapter, may become part of an essential tool-set that practitioners should use in their every-day activity of application designers.

Chapter 6

AI-supported design of Edge Intelligence IoT applications

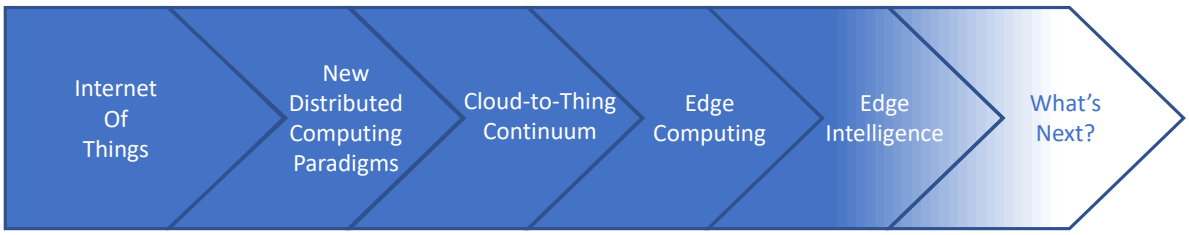


Figure 6.1: Edge Intelligence is almost here. Indeed, this chapter presents a framework to design the end-to-end processing pipeline of an Edge Intelligence IoT application, including the AI model that has to be executed at the edge of the network, *i.e.,* on an IoT device. We exploit different bio-inspired AI methods to identify the (sub)optimal pipeline, using state-of-the-art methodologies. A target pipeline includes data selection, feature extraction, hyper-parameters optimization, and selection of the best AI model. Even if our framework has been especially designed for a well-defined class of devices (*i.e.,* smart gas sensors), it can be easily adapted and adopted in other domains.

6.1 Introduction

Nowadays, smart devices [41] are present in almost every environment and they may become even more intelligent if combined with each-day-more-sophisticated AI algorithms. Using modern AI techniques, we can teach

Part of this chapter appears in the following publication that I co-authored:
M. Antonini, A. Gaiardo, and M. Vecchio "MetaNChemo: A meta-heuristic neural-based framework for chemometric analysis." *Applied Soft Computing*, vol. 97, p. 106712, Dec. 2020. Copyright Elsevier (2020). DOI: 10.1016/j.asoc.2020.106712.

devices what they should do based on data, context, and so on. Famous examples like self-driving cars, face recognition systems, and virtual personal assistants (*e.g.,* Amazon Alexa and Google Assistant) are examples of how AI is drastically changing the way we live and interact with the surrounding environment. However, most of the intelligence is still running in the cloud, as big players are still pushing, but we are seeing some movements toward the edge of the network. In the previous chapters, we have presented how to design and build Edge IoT applications that exploit the edge of the network to execute AI and deliver better services. We have also presented the current evolution of computing platforms that are now able to execute cloud-scale models at the edge of the network, shaping the design of future smart devices and applications.

In this chapter, we propose a multidisciplinary and AI-based end-to-end framework to support the design of an Edge Intelligence IoT application. In detail, this framework allows us to identify the entire processing pipeline from a sensor to the training of a small (in terms of the number of weights) and accurate (in terms of F1-score) neural network that can be inferred by a tiny embedded device. Given that we want to deliver a framework that can be used to design applications in real settings, our framework, which we call *MetaNChemo* [133], has been initially designed for smart chemometric systems. However, it is possible to adapt *MetaNChemo* to other application domains with a small effort.
MetaNChemo identifies, besides the pre-processing pipeline, the architecture and hyper-parameters of a neural network that is able to detect the right concentration of carbon monoxide (CO), within the range from 0 to 25 ppm in the air, at different relative humidity levels. Monitoring of CO concentration is extremely important due to the high toxicity of this gaseous compound, being CO poisoning the leading cause of fatal air poisoning [134] in several countries. Our neural network has to combine signals sampled from a sensor array made of up to 3 different chemoresistive sensors (SnO_2 [135], ZnO-1 [136], and ZnO-2 [137]). Such sensors have been designed and developed by the Micro-Nano Facility(MNF)[1] research unit

[1] `https://mnf.fbk.eu/`

of our institution, the Bruno Kessler Foundation (FBK, Italy). The interested reader can find additional details about the fabrication process in Appendix A. This synergy allows us to design and implement a complete end-to-end research pipeline using state-of-the-art and innovative techniques for sensor production and data analysis. Moreover, it reduces research costs to build and study smart gas sensors. In this chapter, we want to prove that the selection of the sensor array and hyper-parameters of the neural networks can be performed by combining state-of-the-art meta-heuristic tools and techniques, indeed our approach handles the optimization of these aspects at the same time. Many of the algorithms belonging to this family exploit bio-inspired mechanisms [138] to optimize parameters with respect to one or more fitness functions. For instance, genetic algorithms [139, 140] can be used as alternative methods to train models, even if we consider deep neural networks [140, 141]. Alternatively, meta-heuristic algorithms can be useful tools to optimize internal neural network architectures and internal connectivity patterns [142, 143, 144, 145]. Given the research hype around deep learning, meta-heuristic approaches have been successfully applied to discover deep neural network architectures [146], convolutional neural networks (CNNs) [147, 148] and their hyper-parameters optimizations, *e.g.,* the dropout rate [149]. Finally, meta-heuristic approaches are widely used to select features to feed machine learning models [150, 151, 152, 153].

In summary, the main contributions of this chapter are threefold:

- to introduce a multidisciplinary end-to-end framework, called *Meta-NChemo*, able to support the design of Edge Intelligence IoT applications, and more in particular smart chemometric systems, from the production of gas sensors to the assessment of AI models;

- to propose a data-driven approach that comprises the entire processing pipeline and exploits different state-of-the-art AI techniques (preprocessing, meta-heuristics, machine learning) to identify an AI model with strong design requirements imposed by embedded devices: low number of parameters (*i.e.,* 20–50 weights) and high detection capabilities ($F1score \geq 0.95$);

- to reduce the research gap and join forces between computer science and material science worlds.

The remainder of this chapter is structured as follows: Section 6.2 presents a quick overview of machine learning methods applied to gas sensors. The data collection setup from chemoresistive sensors and data-set construction are delineated in Section 6.3. Section 6.4 describes the meta-heuristic-based approach designed to identify the best sensor array and network architecture, which are the core of our Edge Intelligence IoT application. The results obtained are presented and analyzed in Section 6.5. Then, in Section 6.6, we draw some conclusions, outlining possible future works and the main challenges opened by *MetaNChemo*.

6.2 Machine learning methods for gas sensing

Historically, research around gas sensing devices has been driven by materials science communities and industries that have tried to identify the best sensing materials and procedures to build reliable sensors with high sensitivity and selectivity. Nowadays, given the opportunity offered by AI [154], and if properly combined with domain knowledge, it is possible to have better insights from data, *i.e.,* if there is a mixture of different gases and their concentration. Several works describe the combination of AI techniques with gas sensor data.
Vergara *et al.* [155] collected three years of data by injecting 6 different gases in a measurement chamber containing a 16 sensors array (metal-oxide sensors, TGS-type sensors from Figaro). They developed an ensemble method based on support vector machines (SVMs) to detect different gases with the aim to study the drift of sensors over time. Their data-set is publicly available and adopted, for instance, by Wang *et al.* [156] to train SVMs to detect the concentration of gases. Moreover, they exploited a genetic algorithm (GA) to find the best hyper-parameters of SVMs.
Other researchers [157] used an array of commercial sensors to collect responses to mixtures of gaseous compounds. They propose a methodology to identify and train the best neural network architecture with the aim to correctly identify the odor intensity and the hedonic tone.

Casey *et al.* [158] deployed sensor arrays, which comprise commercial sensors, and fed machine learning algorithms with mixtures of sensor output signals. They compared neural networks with linear models with the aim to understand if these methods may help to address the cross-sensitivity of sensors.
However, these examples use commercial sensors or public data-sets. First, commercial sensors may suffer from cross-sensitivity; moreover, they may be expensive and are often covered by the industrial secret that prevents research communities to acquire a full understanding. On the other hand, while research conducted over public data-sets and commercial sensors is essential to develop new methods, the main drawback is that we do not have full control of physical experiments since we cannot control the environmental conditions (*i.e.,* air temperature, relative humidity, etc.), the sensor array (*i.e.,* which sensors we want to study and their working temperatures), the injected gases (*i.e.,* compounds, concentrations, injection profiles, etc.), and how we sample signals (*i.e.,* sample frequency).

6.3 Data-set construction

6.3.1 Data Acquisition

In this chapter, to support the design of an Edge Intelligence IoT application, as use-case we investigate the gas sensing performances of SnO_2, ZnO-1, and ZnO-2 sensors against different concentrations of CO (0, 2, 5, 10, 15, 20, and 25 ppm), in the presence of various percentages of relative humidity (RH%), in order to develop and validate our approach in real conditions (even if in a controlled environment). Indeed, variations of the RH% values typically affect the response of MOX gas sensors [159]. The CO concentrations were chosen based on the CO threshold limit value [160], which is 25 ppm. Figure 6.2 shows a block diagram of the setup used for the gas sensing measurements. Eight sensors, *i.e.,* three SnO_2, three ZnO-1 (nanograins), and two ZnO-2 (nanorods), were placed in a dedicated gas test chamber with a volume of 0.5 dm^3. The arrangement of sensors in the gas test chamber is shown in Table 6.1. The electronic system of the gas test chamber was equipped with two separated electronic circuits

Table 6.1: Gas sensor arrangement in the test chamber and their working temperatures (source: [133], p. 5).

Channel	CH1	CH2	CH3	CH4	CH5	CH6	CH7	CH8
MOX Sensor Type	SnO_2	ZnO-1	ZnO-2	SnO_2	ZnO-1	SnO_2	ZnO-1	ZnO-2
Working Temperature	400°C	450°C	400°C	400°C	450°C	400°C	450°C	400°C

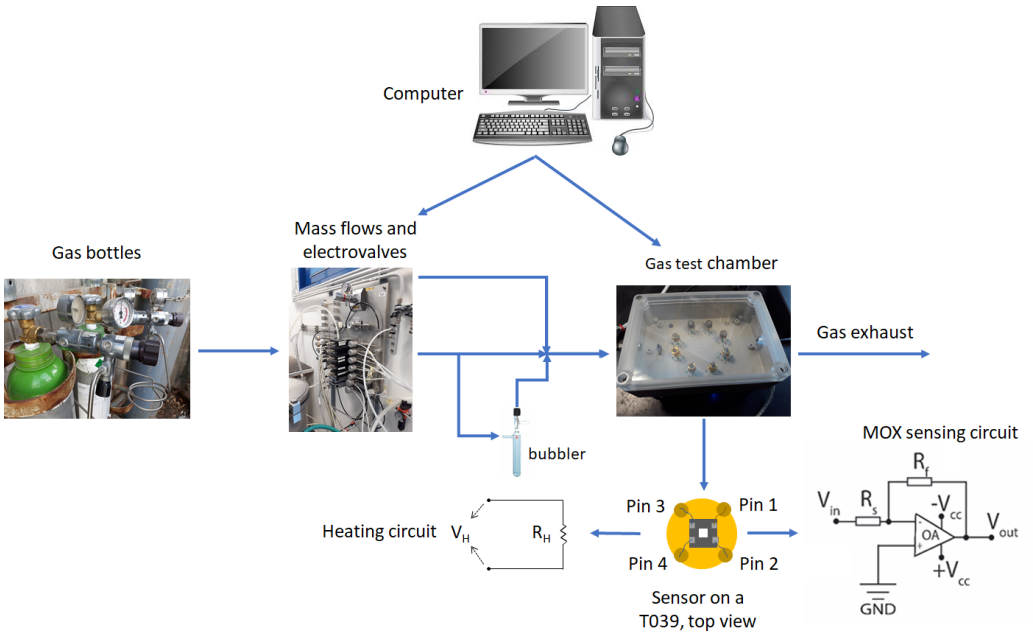


Figure 6.2: Schema of the gas sensing experimental setup (source: [133], p. 5).

for each sensor, namely one dedicated to the heater and one devoted to the collection of the sensing material signal. The sensing materials of all sensors were kept at 400 °C (SnO_2 and ZnO-1) and 450 °C (ZnO-2) during the experiments, which are the best working temperatures for these sensing materials [135, 136]. The device heating was obtained by applying a constant voltage to the sensor heaters. The MOX electrical resistance was collected by measuring the electrical current through the sensing material. One of the interdigitated electrode contacts was biased to -1.0 V, whereas the second was connected to the negative input of an operational amplifier. The amplifier worked in negative feedback mode through a set reference resistance (R_{ref}), and the amplifier output was measured with an analog to digital converter. The amplifier output is inversely proportional to the

sensing material electrical resistance with this configuration:

$$V_{out} = -\left(-1.0V\right) \cdot \frac{R_{ref}}{R_s}, \tag{6.1}$$

where R_s is the resistance of the sensing film. We sampled the amplifier output every 5 s. The experimental setup was equipped with certified cylinders of dry air (80% N_2, 20% O_2) and CO (100 ppm, mixed with dry air). The CO concentrations in the gas chamber were modified by mixing dry air and CO gas flows utilizing MKF mass-flow controllers. The humidity was injected into the test chamber passing an additional dry air flux through a bubbler containing deionized water. The temperature and the RH% values were collected with a digital temperature–humidity sensor (1.0% accuracy), located in the gas chamber. The MKF mass-flow controllers were driven with the LabVIEW software, which automatically saved in a file the mass-flow data at the end of each measurement. We performed several data collection campaigns, by modifying the RH% in the chamber. During each campaign, we kept constant the RH% and repeated different times the CO injections in the chambers, at different concentrations. Since we are using sensors that may be installed in house appliances, we selected 3 typical values of RH%: 18% (almost dry air), 36%, and 54% (both within the indoor humidity range defined as comfortable, 30%–60%).

6.3.2 Data pre-processing

As explained in Section 6.3.1, data are collected by running several daily campaigns with different CO concentrations and RH% values. However, such collected data are not suitable to train a neural network. In this section, we present the processing pipeline that produces the data-set that we will be used later in this chapter. Since data are mainly generated by two different sources, namely flow-meters and sensors, we need to apply different processing methods to the different signals.

Flow-meter data preparation

Flow-meter data describe the concentration of gases injected inside the measurement chamber. However, this signal suffers from noisy fluctua-

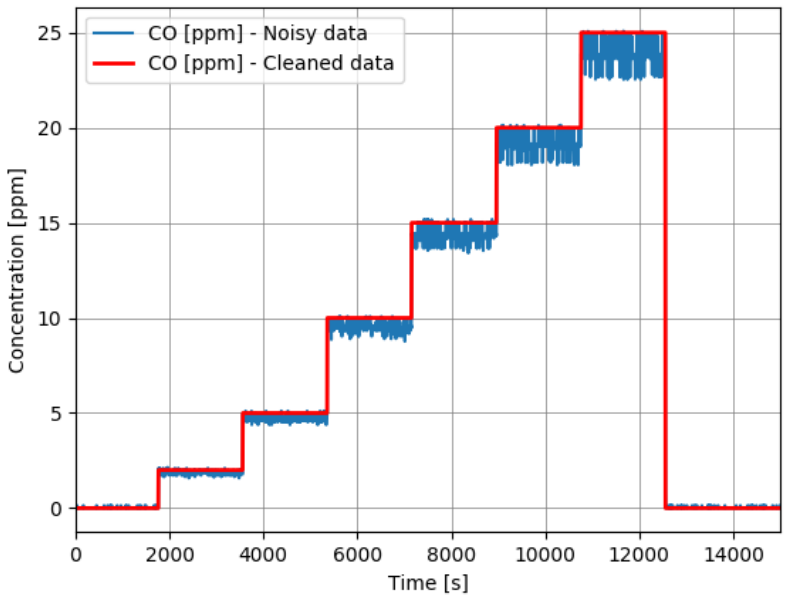


Figure 6.3: Example of flow-meter data (blue line) that describes the concentration of carbon monoxide (CO) injected in the measurement chamber. The red line is the same flow-meter data after the data cleaning procedure (source: [133], p. 5).

tions due to the non-ideal behavior of the equipment that does not keep constant the gas flow. Thus, we need a clean version of the signal (red line in Figure 6.3) by starting from the noisy sampled flow-meter data (blue line in Figure 6.3). We desire a signal that is constant when the concentration is constant, or when there are small fluctuations due to the non-ideal behavior of the flow-meter, and a step when there is a jump from a concentration level to another one. In order to denoise data, we need the sampled flow data ($flow\_data$) and the list of possible concentrations ($list\_concentrations$), apriori known from the experiment setup. In all our experiments, the values of concentrations used to clean the signal in Figure 6.3 are 0, 2, 5, 10, 15, 20, and 25 ppm. For our investigation, we assume that the concentration levels have only integer values.

Given the $flow\_data$ and $list\_concentrations$, we first need to apply a median filter to $flow\_data$ with a window length equal to 5 in order to smooth out some noise. Then, we round the output of the filter to the closest integer value, compute the gradient, and round it to the closest

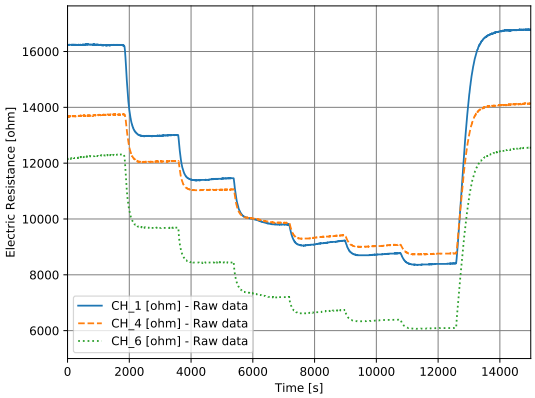
Algorithm 1 Procedure to clean the flow-meter data (source: [133], p. 6).

```
1: procedure CleanFlowMeterData(flow_data, list_concentrations)
2:     data ← median_filter(flow_data, 5)                ▷ filter with window length 5
3:     data ← round(data)                  ▷ round data to the closest integer value
4:     g_data ← compute_gradient(data)                              ▷ compute gradient
5:     g_data ← round(g_data)                                         ▷ round gradient
6:     g_zero_intervals ← identify_zero_intervals(g_data)      ▷ return boundaries of
   intervals with value zero, each element is a pair (start_interval, end_interval)
7:     for i ← 0 to length(g_zero_intervals) do
8:         start, end ← g_zero_intervals[i]
9:         data[start, end] ← max(data[start, end])   ▷ set all elements of the interval to
   the maximum value of the interval
10:    unique_values ← get_unique_values(data)
11:    for i ← 0 to length(unique_values) do
12:        if not (unique_values[i] in list_concentrations) then
13:            diff_vec ← (list_concentrations − unique_values[i])^2
14:            min_index ← get_index_minimum_value(diff_vec)
15:            data[data = unique_values[i]] ← list_concentrations[min_index]
16:    return data
```

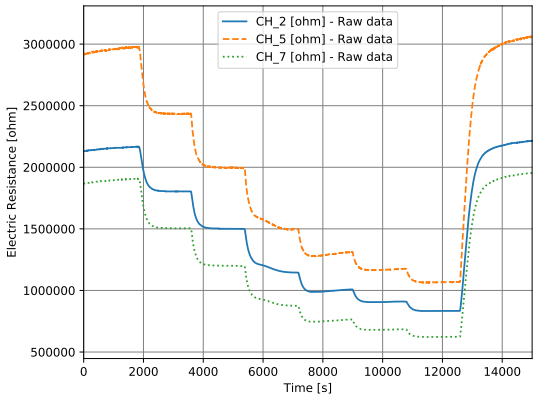
integer value. The desired effect is that small values of the gradient become 0. At this point, we have to identify the flat regions of the flow-meter data by looking for intervals where the gradient is 0, and we save them in a list. By iterating over this list, we substitute all the values of each interval with the maximum value of such interval. We obtain a signal that looks like a piece-wise function. However, some intervals may not have a value that belongs to *list_concentrations*. If this happens, we need to replace the values of these intervals with the closest value (*e.g.,* minimum absolute error) from *list_concentrations*. The output of this step returns a clean signal like the red signal in Figure 6.3. Algorithm 1 summarizes the procedure described above.

Sensors data preparation

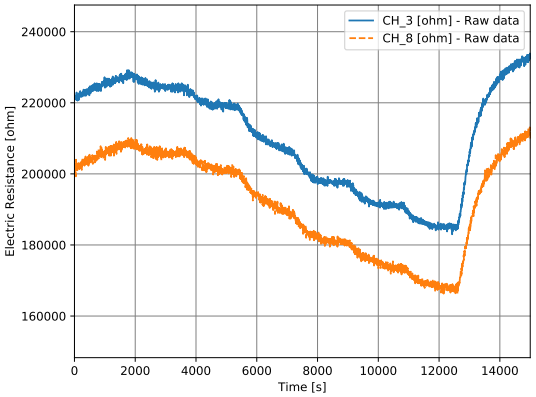
Raw signals sampled from sensors need to be processed before we can use them for our purposes. Such measurement values are affected by biases, measurement noise, and transitions between different concentrations that may lead to non-accurate readings. Figure 6.4 depicts an example



(a) Sensors with SnO_2 film.



(b) Sensors with ZnO-1 film.



(c) Sensors with ZnO-2 film.

Figure 6.4: Examples of raw signals sampled from sensors. Each signal has a different zero-concentration bias and a different excursion, at different concentration levels. Note that the vertical scales are different for the various types of sensors (varying from a few $k\Omega$ to $M\Omega$) (source: [133], p. 6).

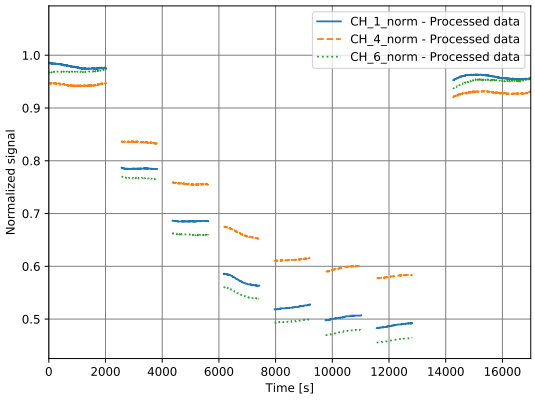
of raw signals sampled during one of our campaigns. In this section, we present the processing applied to each sampling campaign in order to have a clean and usable data-set. Each campaign is independent with respect to the others. The final composition of the data-set will be presented in the next section. Before starting, we need to process the flow-meter data as explained in Section 6.3.2. This step is required since the sensors' pre-processing depends on the concentration levels. We will refer to the cleaned flow-meter signal as $flow\_data$. First, we have to normalize the sensor signal with respect to the average value of the concentration at 0 ppm. This step reduces the bias of the signal that changes every time that the concentration returns to 0 ppm. Usually, the higher the bias, the big-

ger the excursion. At the beginning of each campaign, pure air is injected into the measurement chamber for several hours (*e.g.,* 6-12 hours) in order to clean the surface of the sensing paste and to stabilize the response of sensors. We take the response during the stabilization phase, we apply a median filter with window length 3, then we compute the mean over the last 60% of the stabilization response, and we get the zero-concentration value. We divide the entire signal by the zero-concentration value and we get the normalized signal around 1.0. Then, we truncate the beginning of the stabilization interval by keeping only the last 25% of the stabilization interval. This removes part of the signal that might be polluted by the stabilization process and we reduce the unbalanced ratio of instances associated with different concentrations. Moreover, the system suffers from different inertia due to the different components: the CO concentration reaches the desired level quickly since it is manipulated by the mass-flow controller, however, during a transitory condition, sensors' response is slow since the sensing material needs some time to interact with the gas and to reach a stable value of the resistance. Considering the interval between two different concentrations of CO in the chamber, we remove the initial 30% of the data from each interval. Then, we label the remaining data with the right concentration.

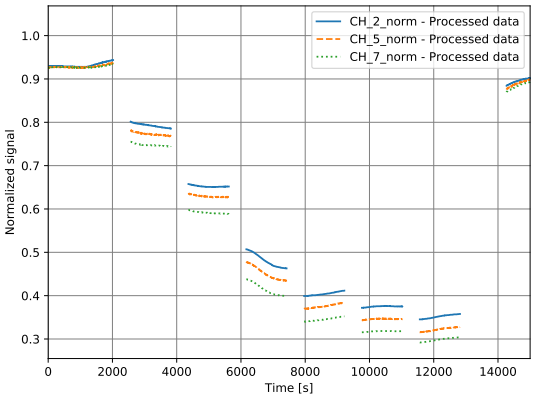
If we apply this procedure to signals drawn in Figure 6.4, we get the new signals depicted in Figure 6.5.

Data-set composition

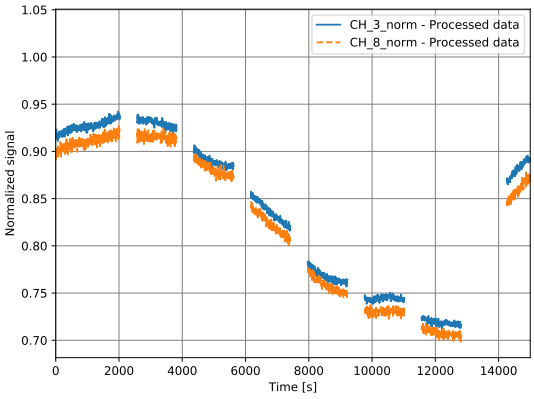
At this point, we have the tools to build our data-set. Since we want to train small neural networks that are able to classify the concentration levels from the sensors readings and should be executed by an Edge IoT device, our data-set should be built by keeping in mind that it targets a classification task. Each instance of the data-set contains the timestamp, $8M$ normalized sensor readings (M for each sensor), the environmental readings (relative humidity and temperature), and the concentration of injected CO (value obtained from the cleaning of the flow-meter signal) in the chamber. The $8M$ sensors' normalized readings are obtained by applying the procedure described in Section 6.3.2 to each sensor signal. M



(a) Sensors with SnO_2 film.



(b) Sensors with ZnO-1 film.



(c) Sensors with ZnO-2 film.

Figure 6.5: Processed signals using the procedure described in Section 6.3.2 and the signals of Figure 6.4 as input. All signals are normalized (source: [133], p. 7).

is the length of the time window we consider in the current instance. If $M = 1$, we only consider the current readings of the 8 sensors. If $M > 1$, we consider the current readings and the previous $M - 1$ (past) readings from each of the 8 sensors. We will identify the value of M during the execution of the optimization algorithm, but it will never be higher than 30. The environmental readings (relative humidity and temperature of the measurement chamber) are associated with the current reading and are scaled by 100 in order to have values in the range [0.0, 1.0]. Finally, the concentration of injected CO can only assume 7 distinct values (*i.e.*, 0, 2, 5, 10, 15, 20, and 25 ppm). Since these values are discrete, we identify each value as a class. Our NNs will classify 7 different concentration levels from sensors' readings.

From the data of each measurement campaign, we get a list of instances

Campaign	RH%	0 ppm	2 ppm	5 ppm	10 ppm	15 ppm	20 ppm	25 ppm	Total
July 1st	18	3855	1506	1510	1509	1510	1509	1507	12906
July 3rd	36	6434	1506	1508	1511	1509	1511	1506	15485
July 5th	54	4259	1506	1510	1509	1510	1508	1508	13310
July 13th	54	10737	1338	1339	1339	1341	1343	1340	18777
July 17th	18	2737	1002	1005	1006	1007	1005	949	8711
July 18th	36	2979	1002	1006	1006	1007	1005	1006	9011
July 19th	54	4410	1002	1006	1008	1005	1006	1005	10442
July 21st	18	4113	1506	1507	1508	1508	1510	1508	13160
July 24th	54	2816	502	503	503	503	502	504	5833
July 25th	54	4232	1506	1509	1510	1509	1509	1509	13284
July 27th	18	8397	1002	1005	1004	1004	1006	1004	14422
Total		54969	13378	13408	13413	13413	13414	13346	135341

Table 6.2: Details of the constructed data-set (source: [133], p. 8).

sorted by their timestamps. We identify each list with the day when the campaign was started. Table 6.2 shows some detail about the data-set constructed, including how many instances are associated per class and the relative humidity present in the measurement chamber. We want to make the reader aware that, due to a fault of the data collection system that happened between July 6th and July 12th 2018, such data could not be stored. However, we decided to use the data correctly collected, since the production of sensors (from the silicon wafer to the sensor packaging, presented in Appendix A) and their calibration campaigns are quite expensive processes.

This data-set allows us to train and assess our neural networks. To this aim, we split it into four disjoint sub-sets: *Training Set* (*TrSet*), used to train NNs, *Validation Set*(*VSet*), used to assess the performance models during the meta-heuristic research of hyper-parameters (Section 6.4), *Test Set*(*TSet*), used to run the selection strategy described in Section 6.4.2, and *Evaluation Set*(*ESet*), used to assess the performance in a different environmental condition. For our experiments, we distribute the sampled campaigns over the four sub-sets in two different ways:

- **5321-split**:

  TrSet: 5 campaigns: 2 at RH 18% (July 1st and 17th), 1 at RH 36% (July 3rd), and 2 at RH 54% (July 5th and 13th)

  VSet: 3 campaigns: 1 at RH 18% (July 21st), 1 at RH 36% (July 18th), and 1 at RH 54% (July 19th)

  TSet: 2 campaigns (July 24th and 25th) at RH 54%

ESet: 1 campaign (July 27th) at RH 18%

- **6221-split**:

  TrSet - Training Set: 6 campaigns: 2 at RH 18% (July 1st and 17th), 2 at RH 36% (July 3rd and 18th), and 2 at RH 54% (July 5th and 13th)

  VSet - Validation Set: 2 campaigns: 1 at RH 18% (July 21st) and 1 at RH 54% (July 19th)

  TSet - Test Set: 2 campaigns (July 24th and 25th) at RH 54%

  ESet - Evaluation Set: 1 campaign (July 27th) at RH 18%

In both cases, we train our networks with all the three values of RH, however, in the first case we use only one campaign at RH 36% and 2 campaigns in the second case. Moreover, we validate the performance of models using a campaign at RH 36%, besides the ones at RH 18% and RH 54%, in the first case, and no campaign at RH 36% in the second case.

Finally, we look for the best network to implement it by testing the models over two campaigns at RH 54%. Selected models are further tested against another campaign at RH 18% to check model performance on a really low value of RH.

It is important to highlight that the data-set we just built is used to discover, train, and evaluate the performance of the neural network that we want to deploy on a target device. At inference time, when a device will run the model, only the selected and pre-process data will be fed to the model. We will discuss the selection of sensors in the following section.

6.4 The proposed approach

Neural networks are machine learning (ML) tools able to mix knowledge and highlight patterns coming from different sources. Our goal is to identify the architecture and train a neural network that is able to run on low cost, constrained, and off-the-shelf computing platforms (*e.g.,* an ARM-based micro-controller with limited RAM and computing power, and costing a

few USD), like the ones that are usually used to build Edge IoT devices. Ideally, this ML model could be part of the computing pipeline of a smart gas sensor connected to an array of different sensors and that is able to provide feedback to a user if the concentration is too high. However, the complete design and characterization of such a smart device is out of the scope of this chapter. Here, we provide only a method to identify the AI model that should run on a target Edge IoT device.

Since we want to target a generic constrained and embedded platform with a few tens of bytes of RAM, our neural network should be as small as possible in term of weights (or parameters), but at the same time, it should have high classification performance (high F1-score), by considering the data-set described in Section 6.3. The architecture of the network we want to build should support the following requirements:

- combine data coming from SnO_2, ZnO-1, and ZnO-2 sensors;
- possibility to use previous sample of the sensors' readings (memory length, the value M introduced in Section 6.3.2);
- use of environmental variables such as relative humidity and temperature;
- employ a reduced number of hidden layers (1 or 2);
- employ a reduced number of neurons per layer;
- use only fully-connected layers;
- the activation function should be one of the following: *relu*, *logistics*, *tanh*, or *identity*;
- the network should be able to return one of the 7 concentration of CO (0, 2, 5, 10, 15, 20, and 25 ppm);
- learning rate equal to 0.001;
- regularization value equal to 0.0001.

The number of possible networks that satisfy these requirements is huge and it is not feasible to explore all possible combinations. Moreover, we

do not know, for instance, how many neurons we should use, at maximum, per layer. Due to this combinatorial explosion, we decide to resort to a meta-heuristic approach to efficiently explore the hyper-parameter space and discover solutions that satisfy our requirements. Formally, the problem that we want to solve can be described as follows

$$\underset{nn}{\text{minimize}}(-F1(nn), num\_params(nn)) \tag{6.2}$$

where nn is a tuple (described below) that identifies a valid architecture of a neural network trained and tested on the *TrSet* and *VSet* datasets, respectively. Above we said that we want to maximize the F1-score but it is equivalent to minimize the opposite of the F1-score. Regarding the $num\_params(nn)$ function, which computes the number of parameters inside a neural network, it can be expressed as follows

$$\begin{aligned} num\_params(nn) = (2 + length(S) \cdot M + 1) \cdot n_1 + \\ + (n_l + 1) \cdot n_o + \sum_{i=2}^{l} ((n_{i-1} + 1) \cdot n_i) \end{aligned} \tag{6.3}$$

where nn is a tuple composed of:

- S: the list of input sensors;
- M: the memory length presented in Section 6.3.2;
- l: the number of hidden layers;
- n_i: the number of neurons in the i-th hidden layer, with i from 1 to l;
- n_o: the number of neurons in the output layer (*i.e.,* 7).

This tuple is mapped on the chromosome (Figure 6.6) described in the next section. We want also to highlight that the +2 term refers to the current humidity and temperature input values, the +1 terms are relative to the hidden layers' biases, and if $l < 2$ the summation in the formula disappears.

We tackle this problem using a meta-heuristic approach that we explain in Section 6.4.1 and then we use a selection strategy as explained in Section 6.4.2.

6.4.1 Meta-heuristic approaches to hyper-parameters exploration

Meta-heuristic methods are a broad family of techniques and algorithms that may provide good solutions to an optimization problem. These procedures explore the solution space by looking for the best candidates. Usually, they are inspired by biological phenomena such as social behaviors (*e.g.,* swarms) or population (*e.g.,* evolution). Since our goal is to find a neural network that has a small number of weights and high detection capabilities, we need an algorithm able to optimize a multidimensional objective function. In the following, we present two different meta-heuristic approaches with the aim of providing a more generic framework. The first approach (Section 6.4.1) is based on a genetic algorithm (GA) that exploits some biologically inspired operators (*e.g.,* crossover, selection, mutation) to efficiently explore huge search spaces and identify a population of feasible solutions [161]. The latter approach (Section 6.4.1) relies on a particle swarm optimization (PSO) algorithm [162] that moves a swarm of particles (*i.e.,* solutions) in the search-space, according to some mathematical relations (update rules), by looking for the optimal solutions.

The genetic approach

Genetic approaches have been used for decades to solve optimization problems in many different domains. A good state-of-the-art algorithm, available from the soft computing literature, is the non-dominated sorting genetic algorithm II (NSGA-II) [163], a bio-inspired algorithm able to find optimized solutions against two or more objective functions, keeping them in a finite-sized archive of non-dominated solutions. We highlight that, in the following, we use the NSGA-II algorithm since it is one of the most adopted and well-known multi-objective genetic algorithms, while the comparison of different genetic approaches in tackling our specific optimization problem is out of the scope of this chapter. The interested reader is invited to refer to [161] for a recent comparison and detailed description of the most effective multi-objective genetic algorithms.
In order to identify the most suitable neural network architecture and sensors to use for our purposes, we model our problem as an integer optimization problem, encoding the above network architecture and requirements

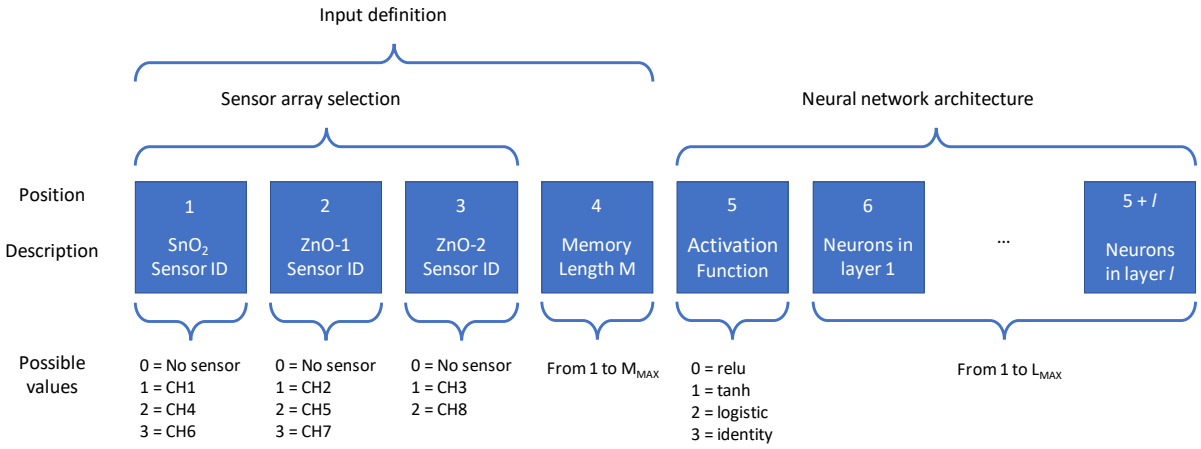


Figure 6.6: Structure of the chromosome. Each blue square is a gene (source: [133], p. 10).

in a chromosome that contains the hyper-parameters of the network. Each chromosome is composed of $5+l$ genes: the first 3 genes allow to select up to 3 different sensors of the sensor array: a SnO_2 sensor, a ZnO-1 sensor, and a ZnO-2 sensor. In this portion of the chromosome, a gene equal to 0 means that a sensor of that type is not selected, otherwise it selects one of the sensors. In the latter case, these genes may assume values of 1, 2, or 3 (except for the third gene, which cannot assume a value of 3, since we have only two sensors of type ZnO-2). In this way, it is possible to select 1, 2, or 3 sensors. If 0 sensors are selected, the solution is considered invalid and, therefore, it is immediately discarded. The fourth gene represents the number of readings (memory length M) used per sensor and it can assume values from 1 to M_{MAX}. The fifth gene represents the activation function and its mapping is the following: 0 for *relu*, 1 for *tanh*, 2 for *logistic*, and 3 for *identity*. The subsequent l genes define the number of neurons of the l hidden layers of the network. Every layer can incorporate from 1 neuron to L_{MAX} neurons. We performed different experiments, as we will explain in the next section, with $l = 1$ and $l = 2$ hidden layers. Figure 6.6 shows a representation of the chromosome structure. Clearly, each chromosome unequivocally identifies one, and only one, solution.

In order to efficiently explore the solution space with the NSGA-II algorithm, we need to configure some parameters of the algorithm. We identify the size of the solution archive as N_{pop} and the size of the off-spring pop-

Parameter	Value
Population Size N_{pop}	64
Off-Spring Population Size N_{os}	32
Number of Generations G	4000
Crossover Operator	Single-point
Crossover Probability P_c	1.0
Mutation Operator	Integer Polynomial
Mutation Probability P_m	$1/(5+l)$
Mutation Distribution Index η_m	20
PRNG Seed	42
Maximum memory length M_{MAX}	30
Maximum neurons per layer L_{MAX}	30

Table 6.3: NSGA-II set-up (source: [133], p. 9).

ulation as N_{os}. The off-spring population is the set of new solutions that will be computed in each generation. Then, we adopt the integer single-point crossover [164], applied with probability P_c, as a crossover operator. This operator takes two chromosomes, randomly splits them in a common point, and swaps the right-hand-sides of chromosomes. Furthermore, the integer polynomial mutation operator [165] has been selected as mutation operator, with probability P_m and distribution index η_m. This operator perturbs each gene of every chromosome of the off-spring solutions with probability P_m. It also prevents an early convergence of the algorithm if it gets stuck in a local optimum. Finally, the genetic algorithm requires a set of fitness functions, to be maximized or minimized, to decide which solution is preserved or dropped. Since we associate a neural network to each chromosome, we want to minimize the number of weights (or parameters) of the network and maximize the F1-score, or better minimize $-$F1-score, computed over the *VSet*. Each network is configured using the chromosome and trained using the *TrSet*. Finally, the NSGA-II algorithm is executed for a number of generations equals to G. Table 6.3 summarizes the configuration adopted in our experiments for the NSGA-II algorithm. We highlight that the configuration values for the crossover and the mutation operators were left to their default values within the jMetalPy library [166], while all the remaining parameters were set based on our past experience designing multi-objective optimization algorithms [167].

Finally, in order to speed up the convergence of the algorithm, we introduce two improvements during the generation of the off-spring population:

- *Kill-the-clone*: we check if some members of the off-spring population are clones of individuals already present in the solution archive: if yes, we set the fitness functions, the number of parameters and the F1-score to $+\inf$ and $-\inf$, respectively. These solutions will be automatically discarded during the sorting phase;

- *Reincarnation*: we keep a complete and separate list, known as the *Reincarnation List*, of all the solutions generated across all the generations. If an individual generated during the mating phase is present in the *Reincarnation List* and it is not a clone of a solution present in the population archive, we associate the already calculated fitness functions to the reincarnated solutions.

The genetic research of solutions has been implemented using the jMetalPy [166] framework, a Python implementation of the jMetal [168] framework, which is one of the reference tools for multi-objective optimization and meta-heuristic. Regarding the design and training of the neural network, we used the multi-layer perception (MLP) classifier available within the SciKit-Learn library [91], a state-of-the-art Python library for machine learning. We set all the pseudo-random number generators' (PRNGs) seeds to 42 (both for the MLP and the NSGA-II algorithm). Finally, for data pre-processing, we used standard Python libraries such as Pandas, NumPy, and SciPy. The comparison of different frameworks and/or implementations, developed in other programming languages, is out of the scope of this chapter.

The PSO-based approach

In this section, we introduce iSMPSO, a particle swarm optimization (PSO) alternative that can be used to tackle the same multi-objective integer problem (hyper-parameters optimization of a neural network) presented in the previous section. Specifically, in the following, we provide some performance comparisons in terms of hyper-volume [169] between the NSGA-II

and the iSMPSO algorithms. Our goal is to give the flavor of the opportunities offered by this multidisciplinary research, while to accurately benchmark the two algorithms against the proposed problem is out of the scope of this chapter.

iSMPSO is based on a PSO algorithm [170], which exploits the behavior of a swarm of particles (solutions) to optimize a problem. The original PSO algorithm [162] has been proposed multiple times in several different flavors, with support for single and multi-objective problems [171, 170] with real [171, 170], binary [172], and integer codifications [173]. Since our goal is to provide practitioners with some tools to be used in their settings, we selected the speed-constrained multi-objective PSO (SMPSO) algorithm, as it can be easily adapted to our needs. We kindly invite the reader to refer to [170] for a detailed description of the SMPSO algorithm. Natively, SMPSO supports polynomial mutation as turbulence operator, selects and stores the non-dominated solutions using the crowding distance metric [163], and stores them in an external archive. However, this algorithm was designed for real-valued problems, hence we first need to adapt it to support integer problems. Thus, we replace the particle position rule (Formula (6.4))

$$\overrightarrow{x}_i(t) = \overrightarrow{x}_i(t-1) + \overrightarrow{v}_i(t) \tag{6.4}$$

where $\overrightarrow{x}_i(t)$ and $\overrightarrow{v}_i(t)$ are the position and the velocity of the i-th particle at time t, with the position rule (Formula (6.5)) defined in [173].

$$\overrightarrow{x}_i(t) = ceil\left((\overrightarrow{x}_i(t-1) + \overrightarrow{v}_i(t)) \cdot 10^Y\right) \mod \overrightarrow{m}_i \tag{6.5}$$

In Formula (6.5), t is the time index, $\overrightarrow{v}_i$ is the same velocity vector of Formula (6.4), $\overrightarrow{x}_i$ is the position of the i-th particle, Y defines the number of digits of accuracy required by the application, we consider Y=0, thus the 10^Y term disappears. Finally, the mod operator computes the element-wise modulus of the argument with respect to $\overrightarrow{m}_i$, which is the vector of values of upper bounds of particles' position. Moreover, to fully support integer problems, we replace the real-valued polynomial mutation operator with the integer polynomial operator [165] with probability P_m and distribution index η_m. For convenience, we relabel this algorithm as iSMPSO, where i refers to the support for integer problems.

Parameter	Value
Swarm Size N_{sw}	64
Leaders Archive Size $N_{leaders}$	64
Number of Iterations G	600
Mutation Operator	Integer Polynomial
Mutation Probability P_m	$1/(5+l)$
Mutation Distribution Index η_m	20
PRNG Seed	42
Maximum memory length M_{MAX}	30
Maximum neurons per layer L_{MAX}	30

Table 6.4: PSO set-up (source: [133], p. 10).

The iSMPSO algorithm is able to handle and optimize the integer problem presented above (Formula (6.2)) that aims at finding the best neural network architectures that minimize the number of network parameters and maximize the F1-score. A solution is identified with a particle as depicted by Figure 6.6 and we define N_{sw} as the size of the swarm of particles considered during every iteration of the optimization algorithm. We invite the reader to refer to the previous section for a detailed description of the optimization problem. We run the iSMPSO algorithm for G iterations by setting the PRNG seed to 42. At the end of the algorithm execution, the Pareto front is stored in the external leader archive (with maximum size $N_{leaders}$) and we call N_{pareto} the number of solutions stored. Finally, Table 6.4 summarizes the configuration adopted for our experiments with the PSO algorithm.

Even in this case, we highlight that the parameters of the mutation operator are the default values in the jMetalPy library [166]. The other parameters have been chosen to have a good exploration of the search space.

Finally, due to the likelihood to generate cloned or reincarnated solutions, we adopt the *Kill-the-clone* and *Reincarnation* strategies (Section 6.4.1) to speed-up the convergence of our experiments.

This approach and the iSMPSO algorithm have been implemented using the same Python libraries (*i.e.,* jMetalPy, SciKit-Learn, SciPy, NumPy, Pandas) used to implement the genetic method presented above.

6.4.2 K-Means-based selection strategy

After G generations or iterations, we get a set of N solutions that have different hyper-parameters, performance, sizes, and ability to generalize future data points. If we run the genetic approach, N is equal to $N_{pop} = 64$, if we consider the full population, or N_{pareto}. If we consider the solutions that belong to the Pareto front. If we run the PSO-based approach, N is the number of solutions that belong to the leader archive and it is equal to N_{pareto}.

These N solutions are the best solutions found up to this point. However, we still do not know which network we should implement on our Edge IoT device. We need a selection strategy that identifies the most suitable network that satisfies our requirements: we have to minimize the number of parameters (low resource utilization) and maximize the F1-score (generalize subsequent data). First of all, we need to estimate the ability of each network to correctly detect the CO concentration of unseen sampling campaigns. This step is also important to understand if the networks are over-fitting the training data or they can generalize over unseen data. Thus, we compute the F1-score by inferring the *TSet* over each network previously trained using the *TrSet*. We get a new cloud of points in the solution space identified by the number of parameters of the network and the F1-score computed over the *TSet*. This step updates only the F1-score associated with each solution and not the number of parameters, which are fixed with the solution. The number of parameters depends on the network architecture. A simple selection strategy might be the selection of a solution from this cloud, but, which solution do we select? The one with the highest F1-score, the one with the lowest number of parameters, the one on the knee or another one? In the following, we describe a selection technique based on the K-means algorithm that has the same or better performance than choose the solution that minimizes the distance between the ideal (and impossible) solution with F1-score equal to 1.0 and 0 parameters.

Given the entire set of solutions, first of all, we need to scale the fitness functions to bring the values near the $[0.0, 1.0]$ interval. The number of parameters has a big excursion and it may assume values above 1000, thus

Algorithm 2 Sorting function that shifts solutions by the centroid and scaled with respect to the standard deviation (source: [133], p. 11).

```
1: function SortWRTScaledStd(solution_list, μ_i, σ_i)
2:     for j ← 0 to length(solution_list_cluster_i) do
3:         scaled_values{0} ← (solution_list{j}.obj{0} − μ_i{0})/σ_i{0}   ▷ X: number of parameters
4:         scaled_values{1} ← (solution_list{j}.obj{1} − μ_i{1})/σ_i{1}   ▷ Y: F1-score
5:         d{j} ← euclidean_distance(scaled_values, (0, 0))
6:         if not (scaled_values{0} < 0 and scaled_values{1} > 0) then
7:             d{j} ← −d{j} ▷ Change the sign if solution is not in the second quadrant
8:     sorted_list ← sort_list_wrt_list(solution_list, d)
9:     return sorted_list
```

we scale the number of parameters by 1000. On the other side, the F1-score is already defined in the desired interval. Making the number of parameters "comparable" with the F1-score allows us to easily compare solutions. Now, we apply the K-means algorithm with K clusters and we obtain K sets of solutions. The algorithm returns centers of mass of clusters μ_i, known as centroids, that might not be solutions, and we compute also the standard deviation σ_i of each cluster, using only solutions that belong to the considered cluster. From each cluster, we need to identify a solution that should minimize the number of parameters and maximize the F1-score of the cluster. First, considering one cluster at a time, we shift all the solutions associated with the cluster and the centroid by μ_i to have the centroid in the origin, then we scale the shifted fitness values by the standard deviation σ_i of the cluster. Now, we compute the Euclidean distance d between each shifted and scaled fitness value with the origin (0, 0). If a solution is not in the second quadrant of the space identified by the scaled number of parameters (X-axis) and scaled F1-score (Y-axis), where the scaled centroid is the origin, we change the sign of d. We sort the solutions from the biggest to the smallest value of d. Geometrically, we select the solution that is the farthest from the centroid in the second quadrant, if there is not any solution in such quadrant, we select the closest solution to the centroid. Algorithm 2 presents the pseudo-code of the sorting function and we select the first solution of the sorted list as the best solution for the current cluster. Applying this function on all sets of

Algorithm 3 Sorting function that shifts and scales solutions by the centroid (source: [133], p. 11).

```
1: function SortWRTScaledCentroid(solution_list, μ_K)
2:     for j ← 0 to length(solution_list) do
3:         scaled_values{0} ← solution_list{j}.obj{0}/μ_K{0}) − 1
4:         scaled_values{1} ← solution_list{j}.obj{1}/μ_K{1}) − 1
5:         d{j} ← euclidean_distance(scaled_values, (0, 0))
6:         if not (scaled_values{0} < 0 and scaled_values{1} > 0) then
7:             d{j} ← −d{j} ▷ Change the sign if solution is not in the second quadrant
8:     sorted_list ← sort_list_wrt_list(solution_list, d)
9:     return sorted_list
```

solutions and choosing the first element, we get the $solution\_list_K$ that is a list with K possible solutions. Now, we compute the centroid μ_K, and we apply Algorithm 3. This function is similar to the one described in Algorithm 2, except for how it scales the fitness values of solutions: here we subtract and divide $solution\_list_K$ by μ_K. We get a sorted version of the $solution\_list_K$ list: the best solution is the first element of the list. Up to this point, we have a solution that is the best if we apply the K-means algorithm with K clusters. However, how many clusters do we have to identify? Or in other words, what is a good value of K? In order to surpass this issue, we compute the best solutions for every integer value of K from 1 to the number of solutions N, *i.e.,* to $N = N_{pop} = 64$ if we consider the full population of the genetic approach or to $N = N_{pareto}$ if we are considering the Pareto front. It is expected that Algorithm 3 returns a limited set of unique best solutions since a solution might be the best solution for different values of K. We count how many times a solution is present in the list, then we build the list of unique solutions $solution\_list_U$. We also store in the $.min\_K$ attribute the minimum value of K such that the best cluster associated with K has only one element (*i.e.,* the best solution). We remove all the solutions that have been counted only 1 time. Solutions that are elected only once as best solutions are less prone to be good solutions compared to solutions that were elected more times. Now, we compute the centroid μ_U and the standard deviation σ_U using the list of unique solutions ($solution\_list_U$). We apply again Algorithm 2 and we get a solution. We get the sorted list of unique solutions($sorted\_solution\_list_U$),

then we check if the first element of the list lays in the second quadrant. If yes, we declare that this solution is the best. Otherwise, we select the solution associated with the smallest value of the *.min_K* attribute. Now, we have the best solution that we want to implement on our Edge IoT device and we infer the *ESet* to check the performance of this solution over a "dry" set (RH 18%).

This selection strategy has been implemented in Python3 using Pandas, NumPy, SciKit-Learn [91](for the K-Means algorithm), Scipy, and Matplotlib libraries.

6.5 Analysis of results

The meta-heuristic approaches presented in Section 6.4 allow us to efficiently explore the space of hyperparameters to find the best neural network. We can use a brute-force strategy, *e.g.,* computing all the possible combinations, but this exploration may take months or, even, years to be completed. Before starting to discuss which is the best neural network, we want to preliminary compare the genetic and the PSO-based (Section 6.4.1) search algorithms to understand if one method outperforms the other. To tackle this objective, the literature reports many different ways to compare meta-heuristic algorithms such as the hyper-volume (HV) [169] or the inverted generational distance(IGD) [174]. We may use one of these or other metrics, but, for simplicity, we decide to use the HV metric since it requires only a reference point (we select (0; 700)) and not the true Pareto front, which is unknown in real-world problems like our problem. In the following, we will depict the outcomes of the comparison by running experiments using the data-sets prepared in Section 6.3 (*5321-split* and *6221-split*) with neural networks with $l = 1$ or $l = 2$ hidden layers. Neural networks are trained using the *TrSet* and validated using the *VSet*. The computing environment is based on a Ubuntu 18.04.2 machine powered by an Intel i7-8665U CPU (quad-core CPU with Intel Hyper-Threading technology, 8 virtual cores) and 16 GB of RAM. Using the code optimizations presented at the end of Section 6.4.1, the time required to execute $G = 4000$ generations (genetic approach) or $G = 600$ iterations (PSO-based

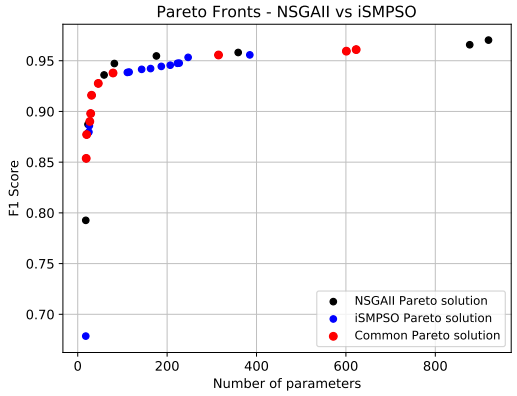
Table 6.5: Hyper-volumes of the approximated Pareto fronts identified during different experiments (source: [133], p. 12)

Experiment	NSGA-II @ 4000 generations	iSMPSO @ 600 iterations
Data-set: *5321-split* Hidden layers: 1	649.491	647.137
Data-set: *5321-split* Hidden layers: 2	655.617	651.576
Data-set: *6221-split* Hidden layers: 1	674.142	674.019
Data-set: *6221-split* Hidden layers: 2	672.793	672.75

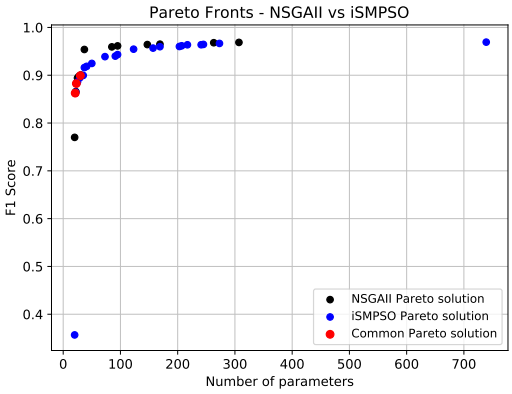
approach) is around 24 h if we simultaneously train models using 7 virtual cores. Figure 6.7 compares the Pareto Front approximations identified by the two meta-heuristic algorithms (NSGA-II and SMPSO). By computing the hyper-volume (w.r.t. the reference point (0; 700)), we directly observe from Table 6.5 that NSGA-II returns a slightly wider front (*i.e.,* higher HV) than the PSO-based approach. As we explained above, our objective is to create a framework that allows us to discover a model architecture without consuming too much (research) effort. In line with this further requirement, we opted for the NSGA-II algorithm because it fits well our needs (integer problems, multi-objectives, etc.). If we do a metaphor, we literally used the meta-heuristic algorithms to find the best solutions as we use a screwdriver against a specific screw head. Furthermore, we are not interested in Pareto fronts with very long tails, rather we privilege prominent knees since it is the starting point of successive analysis in the pipeline, *i.e.,* selection of the "best solution" using a particular decision strategy.

In the following sections, we will present the research of the best neural network using only the output of the genetic approach. At this point, combining the output of the genetic approach with a selection strategy allows us to identify the structure of a tiny fully-connected neural network that can be implemented on a constrained smart gas sensor. We consider 4 different selection approaches:

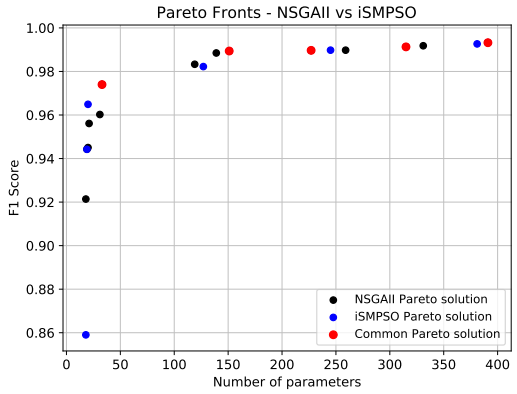
- **S1 - Minimum Parameters**: We select the network that minimizes the number of parameters independently from the *TSet*;
- **S2 - Best F1**: We select the network that maximizes the F1-score over the *TSet*;



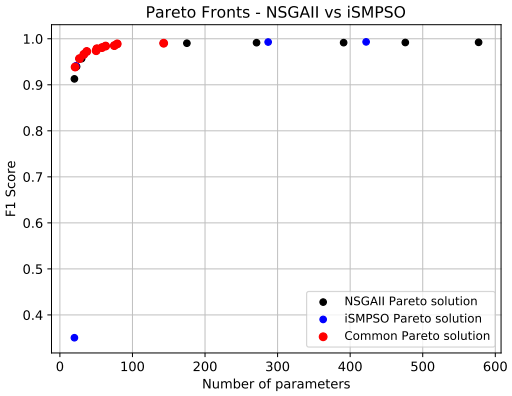
(a) Dataset: 5321-split Layers: 1



(b) Dataset: 5321-split Layers: 2



(c) Dataset: 6221-split Layers: 1



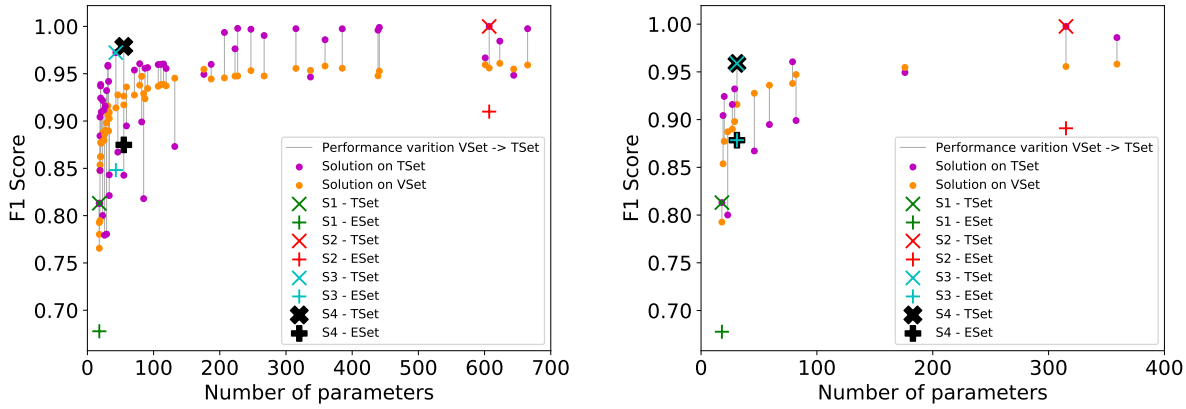
(d) Dataset: 6221-split Layers: 2

Figure 6.7: Comparisons between the approximated Pareto fronts returned by the NSGA-II and iSMPSO algorithms. Blue dots represent solutions belonging to the Pareto front returned by iSMPSO, while black dots are the solutions of the front identified by NSGA-II. Finally, red dots are the solutions present on both fronts (source: [133], p. 13).

- **S3 - Minimum Distance**: We select the network that minimizes the Euclidean distance between the network and the ideal (and impossible) network with F1-score equal to 1.0 and 0 parameters. In order to make the number of parameters "comparable" with the F1-score, we scale this metric by 1000;

- **S4 - K-means based strategy**: We select the network using the criterion presented in Section 6.4.2.

It is worth recalling that the selection of the network is done by inferring the best neural networks with the *TSet* that is composed of two sampling



(a) Full population.

(b) Population on Pareto front.

Figure 6.8: Solutions identified by the genetic research using the *5321-split* and $l = 1$. Crosses highlight solutions selected using different criteria and their performance (source: [133], p. 13).

campaigns (July 24th and 25th 2018), as presented in Section 6.3.2. During these campaigns, RH was kept at 54%, which is a possible value in our houses. In this way, we test if the model selected is working well in "real" conditions. Moreover, selected models are inferred with data sampled in a subsequent campaign (July 27th, 2018) when RH was 18%. This evaluation has the aim to verify if the model can work also with a different value of humidity (dry condition).

6.5.1 Data-set with *5321-split* and $l = 1$

The goal of this experiment is to identify the best neural network with only one hidden layer ($l = 1$) using the *5321-split* of our data-set. The data-set split is presented in Section 6.3.2.

Starting from Figure 6.8a, we may adopt different selection strategies to identify the best network. Using the strategy **S1**, the best network has 18 parameters, an F1-score of 0.813 and 0.678 over the *TSet* and *ESet*, respectively. Strategy **S3** returns a network with 43 weights and it reaches an F1-score of 0.972 over the *TSet* and 0.848 over the *ESet*. Using Section 6.4.2 strategy (**S4**), the best solution has 55 parameters and achieves an F1-score of 0.979 and 0.875 over the *TSet* and the *ESet*, respectively. Finally, if we select the best solution using the strategy **S2**, the best so-

Table 6.6: Solutions selected using the *5321-split* and $l = 1$ (source: [133], p. 14).

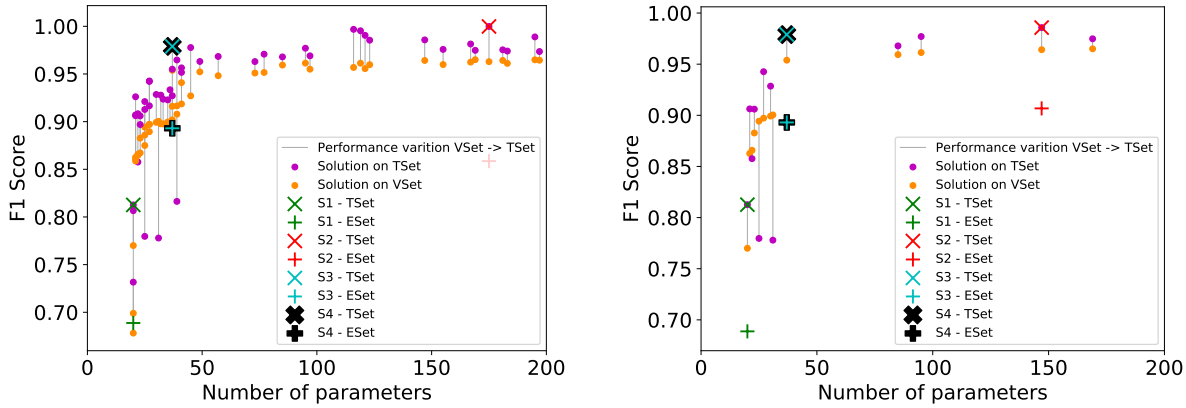
Population	Selection Criterion	Sensors Used	M	Activation Function	Hidden Neurons	Weigths	F1-score (TSet)	F1-score (ESet)
Full	S1	CH2	1	relu	1	18	0.813	0.678
	S2	CH2,CH8	20	relu	12	607	0.99995	0.910
	S3	CH2,CH8	1	logistic	1	43	0.972	0.848
	S4	CH2,CH8	3	logistic	3	55	0.979	0.875
Pareto	S1	CH2	1	relu	1	18	0.813	0.678
	S2	CH2,CH8	17	tanh	7	315	0.997	0.891
	S3	CH2,CH8	1	identity	2	31	0.959	0.878
	S4	CH2,CH8	1	identity	2	31	0.959	0.878

lution reaches an F1-score of 0.99995 over the *TSet* and 0.910 over the *ESet* with a network of 607 parameters. This strategy performs better than ours over the F1-score, however, this solution requires 11 times more weights than the solution identified by our strategy (55 weights) with an increment of 0.02 and 0.035 of the F1-score over the *TSet* and *ESet*, respectively. Now, considering the population that lays over the Pareto front (Figure 6.8b), the network selected with the best F1-score (**S2**) achieves a value of 0.997 over the *TSet* with a network of 315 parameters, 10 times more than **S3** and **S4** strategies, which identify the same network composed of 31 parameters and it achieves an F1-score of 0.959 and 0.878 over the *TSet* and *ESet*, respectively. Finally, **S1** selects the same network as in the full population analysis. More details about the structures of selected networks are present in Table 6.6.

6.5.2 Data-set with *5321-split* and $l = 2$

The goal of this second experiment is similar to the previous one, however, this time we try to find the best networks with $l = 2$ hidden layers. Starting from the full population that is depicted in Figure 6.9a, strategy **S1** selects a network with 20 weights that achieves an F1-score of 0.813 when inferring the *TSet* and 0.689 when inferring the *ESet*. **S3** and **S4** identify as the best solution the same network composed of 37 parameters able to reach an F1-score of 0.979 and 0.893 over the two sets. The network with the best F1-score (**S2**) has 175 parameters and it hits an F1-score of 0.9998 (*TSet*) and 0.859 (*ESet*).

Now, we consider the population over the Pareto front (Figure 6.9b). In this case, selection strategies **S1**, **S3**, and **S4** identify the same solutions elected as best solutions in the full population analysis. However, strategy **S2** selects a network with 147 weights that achieves an F1-score of 0.986



(a) Full population.

(b) Population on Pareto front.

Figure 6.9: Solutions identified by the genetic research using the *5321-split* and $l = 2$. Crosses highlight solutions selected using different criteria and their performance (source: [133], p. 14).

Table 6.7: Solutions selected using the *5321-split* and $l = 2$ (source: [133], p. 14).

Population	Selection Criterion	Sensors Used	M	Activation Function	Hidden Neurons	Weigths	F1-score (TSet)	F1-score (ESet)
Full	S1	CH2	1	identity	1, 1	20	0.813	0.689
	S2	CH2,CH8	1	relu	5, 11	175	0.9998	0.859
	S3	CH2,CH8	1	tanh	2, 2	37	0.979	0.893
	S4	CH2,CH8	1	tanh	2, 2	37	0.979	0.893
Pareto	S1	CH2	1	identity	1, 1	20	0.813	0.689
	S2	CH2,CH8	11	tanh	2, 9	147	0.986	0.907
	S3	CH2,CH8	1	tanh	2, 2	37	0.979	0.893
	S4	CH2,CH8	1	tanh	2, 2	37	0.979	0.893

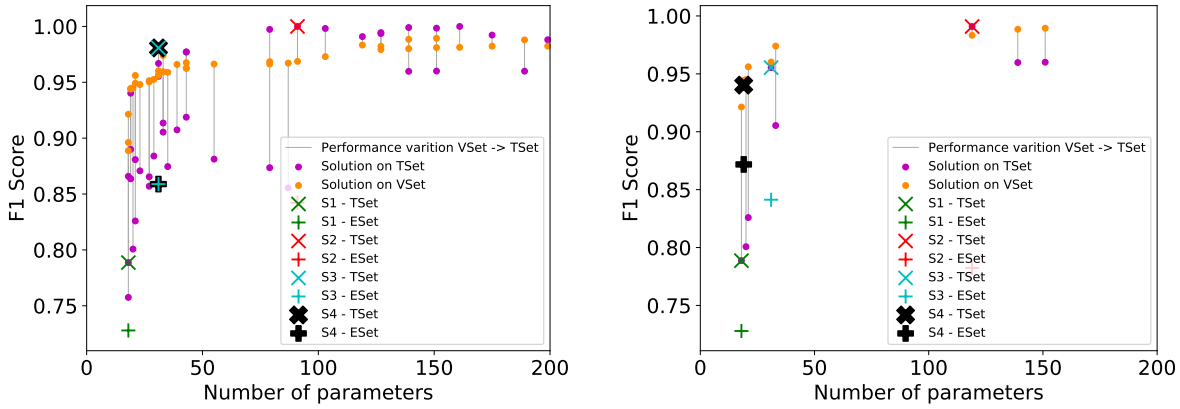
over the *TSet* and 0.907 over the *ESet*. This solution is almost 4 times bigger than the network selected by **S4** with the F1-scores higher only by 0.007 over the *TSet* and 0.014 over the *ESet*.

Finally, we highlight that when we use the *5321-split*, the most selected sensors are CH2 (ZnO-1) and CH8 (ZnO-2). More details about the structures of networks are present in Table 6.7.

6.5.3 Data-set with *6221-split* and $l = 1$

In this experiment, we look for the best network with $l = 1$ hidden layer and using the *6221-split* presented in Section 6.3.2.

Starting from the full population, Figure 6.10a strategies achieve values of the F1-score of 0.789 (**S1**, 18 parameters), 0.99995 (**S2**, 91 parameters), 0.981 (**S3**, 31 parameters), and 0.981 (**S4**, 31 parameters) over the *TSet*. Similarly, if we feed the networks with *ESet*, we get the following F1-scores:



(a) Full population.

(b) Population on Pareto front.

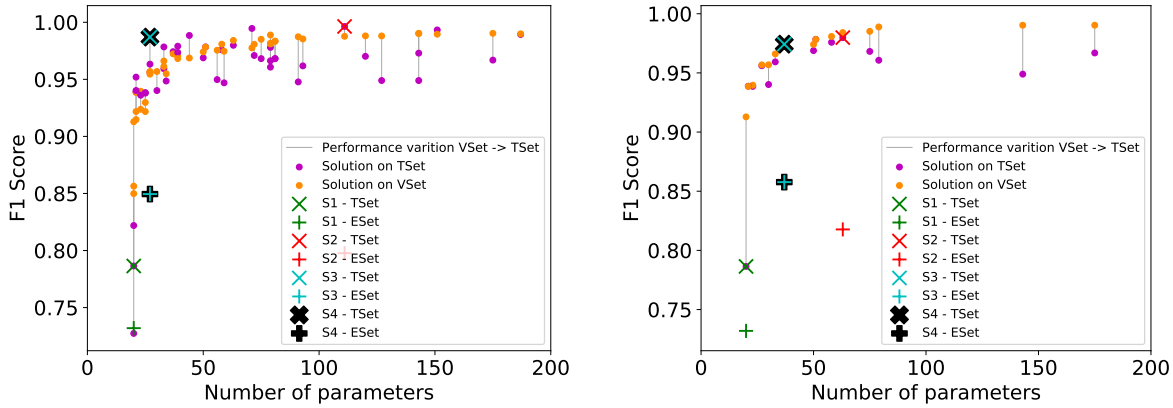
Figure 6.10: Solutions identified by the genetic research using the *6221-split* and $l = 1$. Crosses highlight solutions selected using different criteria and their performance (source: [133], p. 15).

Table 6.8: Solutions selected using the *6221-split* and $l = 1$ (source: [133], p. 15).

Population	Selection Criterion	Sensors Used	M	Activation Function	Hidden Neurons	Weigths	F1-score (TSet)	F1-score (ESet)
Full	S1	CH2	1	identity	1	18	0.789	0.728
	S2	CH2,CH3	1	logistic	7	91	0.99995	0.842
	S3	CH2,CH3	1	identity	2	31	0.981	0.859
	S4	CH2,CH3	1	identity	2	31	0.981	0.859
Pareto	S1	CH2	1	identity	1	18	0.789	0.728
	S2	CH2,CH3	3	relu	7	119	0.991	0.782
	S3	CH2,CH3	1	relu	2	31	0.955	0.841
	S4	CH2,CH3	1	identity	1	19	0.940	0.872

0.728 (**S1**), 0.842 (**S2**), 0.859 (**S3**), and 0.859 (**S4**). Moving to the Pareto front (Figure 6.10b), strategy **S1** is the only approach that elects the same solution identified in the full-population analysis. Instead, **S2** picks as best network a model with 119 parameters with an F1-score of 0.991 over the *TSet* and 0.782 over the *ESet*. Moreover, we can note that this solution is over-fitted over the data related to RH 54% since there is a drop of the F1-score computed over the two sets. Strategy **S3** selects a network with 31 parameters that achieves values of F1-score of 0.955 and 0.841 over the *TSet* and *ESet*, respectively. Finally, our strategy (**S4**) achieves the best F1-score over the *ESet* (0.872) with respect to the other strategies and a value of 0.940 over the *TSet*.

Finally, Table 6.8 reports additional information about the structure of the best networks.



(a) Full population.

(b) Population on Pareto front.

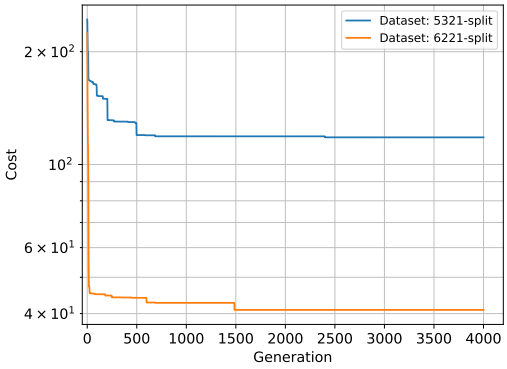
Figure 6.11: Solutions identified by the genetic research using the *6221-split* and $l = 2$. Crosses highlight solutions selected using different criteria and their performance (source: [133], p. 15).

6.5.4 Data-set with *6221-split* and $l = 2$

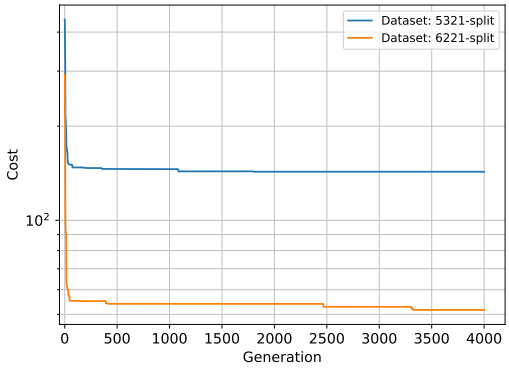
This experiment uses the same data-set split as the previous one, however, here we try to identify the best network that has $l = 2$ hidden layers. Figure 6.11a depicts the full population solutions. **S1** and **S2** select networks that reach an F1-score of 0.786 and 0.996, respectively, over the *TSet* with networks made of 20 and 111 weights. Inferring the *ESet*, the **S1** best solution achieves 0.732 and the **S2** network has an F1-score of 0.798. We notice a big drop in the performance of the latter solution. Then, **S3** performs well as **S4** since they identify a solution with 27 parameters, and an F1-score of 0.987 over the *TSet* and 0.849 over the *ESet*. Considering the Pareto front (Figure 6.11b), **S1** performs as in the full-population analysis. **S2** returns a network of 63 parameters, instead, **S3** and **S4** select a network with 37 weights. The former network performs well on the *TSet* (0.980), however, we notice a drop of the F1-score when we infer the *ESet* (0.818). This network might be more able to analyze RH 54% data. The latter network reaches an F1-score of 0.974 (*TSet*) and *0.858* (*ESet*). Finally, when we use the *6221-split* of our data-set, the best pair of sensors are CH2 (ZnO-1) and CH3 (ZnO-2). More details about the selected networks are available in Table 6.9.

Table 6.9: Solutions selected using the *6221-split* and $l = 2$ (source: [133], p. 15).

Population	Selection Criterion	Sensors Used	M	Activation Function	Hidden Neurons	Weights	F1-score (TSet)	F1-score (ESet)
Full	S1	CH2	1	identity	1, 1	20	0.786	0.732
	S2	CH2,CH3	2	relu	8, 3	111	0.996	0.798
	S3	CH2,CH3	1	relu	2, 1	27	0.987	0.849
	S4	CH2,CH3	1	relu	2, 1	27	0.987	0.849
Pareto	S1	CH2	1	identity	1, 1	20	0.786	0.732
	S2	CH2,CH3	1	tanh	4, 3	63	0.980	0.818
	S3	CH2,CH3	1	tanh	2, 2	37	0.974	0.858
	S4	CH2,CH3	1	tanh	2, 2	37	0.974	0.858



(a) Convergence of NSGA-II algorithm with neural network with $l = 1$ hidden layers.



(b) Convergence of NSGA-II algorithm with neural network with $l = 2$ hidden layers.

Figure 6.12: Convergence plots of the NSGA-II algorithm using different datasets and number of hidden layers l (source: [133], p. 16).

6.5.5 Selection of the best solution

As we presented in the experiments above, every time we use a different data-set split, the number of hidden layers, or selection strategy, we find a different best solution.

This is also highlighted by the convergence plots shown in Figure 6.12 of our experiments since we get a different approximation of the Pareto Front. In our settings, we have to deal with a multi-objective optimization problem, and for the sake of visualization, the cost function depicted in Figure 6.12 is defined as

$$cost(pf) = HV(ip, ref) - HV(pf, ref) \tag{6.6}$$

where HV is the hyper-volume as defined in [169]. More in detail, ref is the reference point used to compute the hyper-volume and it is equal to (0.0; $max\_parameters$), where this last value is the maximum number of

weights that a neural network can have in our experiments. It is computed by feeding Formula (6.3) with the maximum number of sensors (*i.e.,* 3), the number of hidden layers l, the maximum value of the memory length M, and the maximum number of neurons (as defined in Table 6.3). Instead, ip is (1.0, 0.0) and it is related to an ideal network with 0 weights and an F1-score equal to 1.0. This is the same ideal network used to define the **S3** strategy. This function (Formula (6.6)) is computed over the approximation of the Pareto front (pf) identified at the end of every generation.

Once the algorithm has converged, we need a decision-maker able to select the best network. We decide to use the strategy **S4** presented in Section 6.4.2 that uses an approach based on the K-means algorithm. We compared four different strategies to deliver a simple baseline, however, the benchmarking of selection strategies is out of the scope of this chapter. The reader may adopt a different selection strategy, based on the problem at hand.

6.6 Remarks

The widespread of Edge Computing and IoT devices are pushing the development of novel applications where intelligence is not running anymore only in the cloud but also at the edge of the network and on the sensing devices.
Previous chapters covered how to design Edge IoT applications by spreading the intelligence and processing at the edge of the network. We moved the intelligence on commodity devices, *i.e.,* gateways, to support smaller and dumb devices.
This chapter, instead, covers the design of the AI model, *i.e.,* a neural network, that has to be deployed on embedded IoT devices to realize an Edge Intelligence IoT application. We presented a bio-inspired AI-based framework to design the entire pipeline from the sensors to the assessment of the AI model that should be executed on a target device, considering also system constraints, *i.e.,* RAM, CPU, and so on.

We applied this approach to a set of real applications, known as chemometric applications, with the objective to detect the right concentrations of gaseous compounds from the air. To deliver our framework and validate it in real settings, we designed the entire research pipeline from the design and production of the sensors (given the availability of a clean-room at our premises, the MNF facility at Bruno Kessler Foundation, Italy) to the performance assessment of the (sub-)optimal AI model, passing through the data collection in a controlled environment, pre-processing, neural network hyper-parameter selection and training. We called our approach *Meta-NChemo*, since it combines meta-heuristic techniques with chemometrics and neural networks. Even if the approach has been specifically tailored for chemometric analysis, it can be easily adapted and adopted in many different domains to build Edge Intelligence IoT applications.

At this point, we have the tools to build an end-to-end Edge Intelligence IoT application, from the design of the processing that should be executed by devices, to the design of different AI models that have to be executed at the edge of the network. The next chapter, which concludes this thesis, presents some final remarks about the research presented in this and the previous chapters, tries to highlights some future developments, and enforces the need for Edge Computing and Edge Intelligence in future IoT applications.

Chapter 7

Conclusions

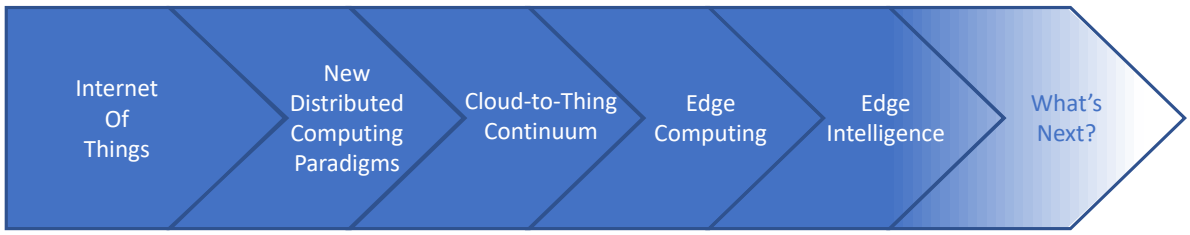


Figure 7.1: This is the very last chapter of this thesis. After a recap of the previous chapters and how this thesis answers the main research questions, this chapter presents some future directions and how IoT applications that exploit Edge Intelligence may become a trending topic in many different fields.

The terms Internet of Things, IoT Platforms, and Edge Computing have strongly pushed the academic and industrial research communities to identify, design, build, and deliver everyday better IoT applications that have deeply changed the world we are living in today. Indeed, even if these technologies are now ubiquitous, IoT started to be widely investigated in 2011 when Gartner inserted the term Internet of Things on its Hype Cycle of Emerging Technologies [4]. IoT reached the very top of the curve in 2014 [5] and in 2015 [175] new IoT-related technologies became popular like IoT platforms and Connected Homes. In 2017, smart devices like smart bulbs, smart lockers, connected speakers empowered with vocal assistants (*e.g.,* Amazon Alexa) became mainstream [12]. In the same months, Edge Computing has become a hot topic thanks, also, to big Internet players that have rolled-out new Edge Computing platforms like AWS Greengrass (we talked about it in Chapter 2). However, at that time, most of the effort was invested to bring cloud-scale computations at the edge of

the network also by exploiting the concept of Fog Computing to migrate and orchestrate functionalities among the different tiers of the architecture (Figure 2.2). Then, given the huge developments around AI and the design of Edge AI hardware accelerators, Edge AI appeared on the rising slope of the Gartner's Hype Cycle for Emerging Technologies in 2018 [176] and it is still, in 2020, in the top part of the curve [177].
This thesis, started in November 2017, contributes to the paradigm shift from traditional cloud-based IoT applications, toward Edge IoT applications, to Edge Intelligence IoT applications. Every chapter proposes a design approach or the ingredients for IoT applications to support this shift starting from research hypes and novelties in the field. As result, this thesis poses different milestones and provides guidelines and recommendations to practitioners that will create the next generation of Intelligent IoT applications at the edge of the network.
More in detail, Chapter 2 presented and discussed the broad and complex Edge IoT platform landscape. We observed and reviewed different platforms considering different players: open-source communities, industrial consortia, commercial products, and standard initiatives. This chapter represents the foundations of the entire thesis since it gives the initial ingredients to design an IoT application at the edge of the network.
Subsequently, Chapter 3 discussed an approach to design and implement an IoT application that exploits the so-called *Cloud-to-Thing continuum*. The application implements IRESE, an anomaly detection method based on unsupervised machine learning techniques, to detect rare events from an audio stream. Here the processing and the AI algorithm were moved from the cloud to a powerful-enough device at the edge, *i.e.,* an IoT gateway, connected to a microphone for audio sampling.
Then, Chapter 4 presented how to push the processing even more toward data sources, *i.e.,* on sensing devices, enabling novel Edge IoT applications that reduce bandwidth usage, latency, and increase privacy (by design). Here we presented a framework to fine-tune the parameters of processing techniques, how to remotely reconfigure devices based on the application needs (COTA, Configuration Over The Air) and system constraints.
Chapter 5 described and evaluated the main Edge AI Accelerators available on the market. We identified the steps and transformations required

to convert a cloud-scale deep learning model for a target edge device. Then, we conducted a systematic analysis of Edge AI Accelerators' performance by executing different personal-scale AI models in different execution configurations. Benchmarks highlighted different important aspects that have to be kept in mind when practitioners design their own applications, such as the impact of communication interfaces (*e.g.,* USB 2.0 vs USB 3.0) and device heating.
Finally, Chapter 6 presented a bio-inspired AI-based methodology to identify the neural network architecture for Edge Intelligence IoT applications by combining the design of the processing pipeline with the system constraints imposed by edge devices. Models trained with our method, after a few transformations (*i.e.,* the ones presented in Chapter 5), can be easily executed on edge devices. We validated our framework, called MetaNChemo, on a real set of sensing applications, known as chemometric applications. Experiments conducted over our framework returned tiny neural networks, with $20 \sim 50$ weights, with high-detection performance ($F1score \geq 0.95$).

7.1 What's next?

In the next few years, we will experience a broad development and adoption of Edge Intelligence IoT applications in many different scenarios, thanks also to the development of novel AI methods. This opens many new exciting research opportunities from system design, intelligence algorithm design, application design perspectives. From the system design perspective, the challenges cover the possibility to design intelligent systems that can efficiently execute complex AI models on hardware devices. Models should have a reduced energy footprint (devices are usually battery-powered), low latency, enhanced privacy (by design), low bandwidth utilization, and very high accuracy without the need of a remote entity. Moreover, if devices have enough computing power, they may locally fine-tune (*i.e.,* continuous learning) their models to better react to external stimuli. For instance, a greater opportunity is offered by neuromorphic computing chips, which

mimic neuro-biological architectures present in the nervous system (*i.e.,* neurons). Edge AI accelerators offer another interesting opportunity. As presented in Chapter 5, a lot of effort has to be pushed to design an efficient system that removes almost all the hardware bottlenecks, which may introduce undesired delays and subsequent performance drops. At the same time, they need to be more energy-friendly given that such devices should be embedded in battery-powered devices and do not overheat, to avoid any possible injury. We envision that a new generation of Edge AI accelerators will be available soon as co-processors inside MCUs and CPUs units without the need for any external device. An example is represented by the A11/A12/A13/A14 Bionic System-on-Chips (SoCs), from Apple, that can execute trillions of operations per second. They contain a special processor, called Neural Engine, that makes machine learning instruction really fast.

Besides the research on systems designed that exploit Edge Intelligence, a lot of effort has to be pushed on the design of AI models that can fully exploit the power of the edge of the network, *i.e.,* the target computing device. As our results showed, if the architecture of a neural network is optimized for a specific hardware platform, it delivers better performance. However, the space of all the possible model architectures is huge and a wide exploration is unfeasible. A great research opportunity is offered by meta-heuristic techniques that are able to explore such solution space by combining system constraints with performance objectives. We envision that in the future we will have tools that will ask us only the sensors or the data on which we want to build the system and, autonomously, the tool will return the best target platform, the system performance characterization, and the model that better suits to the problem. Other research opportunities are offered by the distributed deployments of Edge Intelligence applications. Given that every smart device of a given application will embed an AI model, the devices them-self will specialize (or fine-tune) their model to a specific environment/objective. This will create a plethora of different models, similar among them, that will express and address different aspects of the same application. Edge Intelligence, in combination with Federated Learning approaches, may create new design approaches for future intelligent applications.

Finally, Edge Intelligence will enable new scenarios and application domains. Just to cite a few:

- Space and satellites are currently one of the main hypes. Thanks to the huge investments of public agencies and private companies, satellites are becoming crucial tools in our lives (*e.g.,* navigation, Earth observation, telecommunications, micro-gravity experiments, etc.), however, the data volume generated is becoming huge. It has been estimated that the Copernicus Sentinels fleet (Sentinels 1A, 1B, 2A, 2B, 3A, and 3B) produces, on average, $\sim 20TB$ of data every day [178]. Data volumes will increase and it will become unfeasible to move such an amount of data to the Earth, thus AI processing will be executed directly on satellite and only the processed data will be streamed. The first attempt of this approach is under test by the European Space Agency (ESA) that launched a CubeSat, known as ϕ-sat-1 (PhiSat-1) [179], that embeds some AI model to detect clouds from images. Only images that satisfy some requirements (*i.e.,* cloud coverage lower than a threshold) will return on Earth, with a consequent reduction of transferred data. Other missions are under development [180].

- Smart and precision agriculture is another really hot and active domain. Here, devices are typically deployed in very sparse environments where electric and connectivity infrastructures are not usually available or not powerful enough (*e.g.,* a LoRAWAN network). Thus, Edge Intelligence applications may enable many new applications. For instance, a smart device powered with a camera can run some AI models to understand the current growth status of fruits, detect possible diseases, damages, and so on. The device, thus, will only send a few data extracted from the pictures instead of the entire image.

- Digital industry is one of the trends we are experiencing today, but, in the next years, industries will become even more digital. The possibility to place sensors next to running equipment, with the opportunities offered by Edge Intelligence, will strongly push the development of Condition Monitoring (CM) and Predictive Maintenance (PM) applications where the equipment itself will be able to understand, in a

time-tight manner, its own status and take fast actions (*e.g.,* switch off, slow-down, etc.). This intelligence, which will be deployed on the equipment, or really close to it, will be specialized for the monitored equipment to better understand its behavior and increase accuracy. Federated learning approaches may be used to improve the overall efficacy by combining the intelligence available from wide deployments and scenarios.

These are just a few possible domains where Edge Intelligence will play a fundamental role. Many other domains and opportunities will appear in the next years generating a market value of billions of dollars.

What's next?

Bibliography

[1] Chetan Sharma Consulting, "Correcting the IoT History," 2016. [Online]. Available: http://www.chetansharma.com/correcting-the-iot-history/

[2] K. Ashton, "That "Internet of Things" thing," *RFID Journal*, 2009. [Online]. Available: https://www.rfidjournal.com/that-internet-of-things-thing-3

[3] IERC, "Internet of Things," 2014. [Online]. Available: http://www.internet-of-things-research.eu/about_iot.htm

[4] J. Fenn and H. LeHong, "Hype Cycle for Emerging Technologies, 2011," Gartner Inc., Tech. Rep. G00215650, Jul. 2011. [Online]. Available: https://www.gartner.com/en/documents/1754719/hype-cycle-for-emerging-technologies-2011

[5] H. LeHong, J. Fenn, and R. Leeb-du Toit, "Hype Cycle for Emerging Technologies, 2014," Gartner Inc., Tech. Rep. G00264126, Jul. 2014. [Online]. Available: https://www.gartner.com/en/documents/2809728/hype-cycle-for-emerging-technologies-2014

[6] D. Evans, "The internet of things: How the next evolution of the internet is changing everything," *Cisco Internet Business Solutions Group (IBSG)*, vol. 1, pp. 1–11, 2011.

[7] A. R. Biswas and R. Giaffreda, "IoT and cloud convergence: Opportunities and challenges," in *2014 IEEE World Forum on Internet of Things (WF-IoT)*, Mar. 2014, pp. 375–376.

[8] Deloitte, "Global artificial intelligence industry whitepaper," Deloitte, Tech. Rep., 2019. [Online]. Available: https://www2.

deloitte.com/content/dam/Deloitte/cn/Documents/technology-media-telecommunications/deloitte-cn-tmt-ai-report-en-190927.pdf

[9] M. Antonini and M. Vecchio, "IoT-Panic: The Cloud Just Disappeared!" *IEEE IoT Newsletter*, Jan. 2021. [Online]. Available: https://iot.ieee.org/newsletter/january-2021/iot-panic-the-cloud-just-disappeared

[10] M. Satyanarayanan, "The Emergence of Edge Computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan. 2017.

[11] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, Oct. 2009.

[12] M. Walker, "Hype Cycle for Emerging Technologies, 2017," Gartner Inc., Tech. Rep. G00314560, Jul. 2017. [Online]. Available: https://www.gartner.com/en/documents/3768572/hype-cycle-for-emerging-technologies-2017

[13] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, May 2016.

[14] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[15] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing - MCC '12*. New York, New York, USA: ACM Press, 2012, p. 13.

[16] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep Learning for IoT Big Data and Streaming Analytics: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2923–2960, 2018.

[17] M. Antonini, M. Vecchio, and F. Antonelli, "Fog Computing Architectures: A Reference for Practitioners," *IEEE Internet of Things Magazine*, vol. 2, no. 3, pp. 19–25, Sep. 2019.

[18] IEEE, "IEEE 1934-2018 - IEEE standard for adoption of OpenFog reference architecture for fog computing," 2018. [Online]. Available: https://standards.ieee.org/standard/1934-2018.html

[19] M. A. Nadeem and M. A. Saeed, "Fog computing: An emerging paradigm," in *2016 sixth international conference on innovative computing technology (INTECH)*. IEEE, Aug. 2016, pp. 83–86.

[20] M. Taneja and A. Davy, "Resource aware placement of data analytics platform in fog computing," *Procedia Computer Science*, vol. 97, pp. 153–156, 2016.

[21] H. R. Arkian, A. Diyanat, and A. Pourkhalili, "MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications," *Journal of Network and Computer Applications*, vol. 82, pp. 152–165, Mar. 2017.

[22] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, "Fog computing: Principles, architectures, and applications," *CoRR*, vol. abs/1601.0, Jan. 2016. [Online]. Available: http://arxiv.org/abs/1601.02752

[23] M. Aazam and E.-N. Huh, "Fog computing micro datacenter based dynamic resource estimation and pricing model for IoT," in *2015 IEEE 29th international conference on advanced information networking and applications*. IEEE, Mar. 2015, pp. 687–694.

[24] R. K. Naha, S. Garg, D. Georgakopoulos, P. P. Jayaraman, L. Gao, Y. Xiang, and R. Ranjan, "Fog computing: Survey of trends, architectures, requirements, and research directions," *IEEE Access*, vol. 6, pp. 47 980–48 009, 2018.

[25] A. Sinaeepourfard, J. Krogstie, S. A. Petersen, and D. Ahlers, "F2c2C-DM: A fog-to-cloudlet-to-cloud data management architecture in smart city," in *2019 IEEE 5th world forum on internet of things (WF-IoT)*. IEEE, Apr. 2019, pp. 590–595.

[26] The OpenFog Consortium, "OpenFog reference architecture technical paper," OpenFog Consortium, Tech. Rep., 2017.

[Online]. Available: http://site.ieee.org/denver-com/files/2017/06/OpenFog_Reference_Architecture_2_09_17-FINAL-1.pdf

[27] Industrial Internet Consortium, "Industrial Internet Consortium & OpenFog FAQ | Industrial Internet Consortium." [Online]. Available: https://www.iiconsortium.org/IIC-OF-faq.htm

[28] The Linux Foundation, "The Linux Foundation Launches New LF Edge to Establish a Unified Open Source Framework for the Edge," Jan. 2019. [Online]. Available: https://www.linuxfoundation.org/en/press-release/the-linux-foundation-launches-new-lf-edge-to-establish-a-unified-open-source-framework-for-the-edge/

[29] ——, "State of the Edge Report 2018," The Linux Foundation, State of the Edge Reports, Tech. Rep., 2018. [Online]. Available: https://www.lfedge.org/wp-content/uploads/2020/07/StateoftheEdge20181.pdf

[30] ——, "Project Stages: Definitions and Expectations - LF Edge - LF Edge Confluence," 2020. [Online]. Available: https://wiki.lfedge.org/display/LE/Project+Stages%3A+Definitions+and+Expectations

[31] F. Lardinois, "Baidu Cloud launches its open-source edge computing platform," Jan. 2019. [Online]. Available: https://social.techcrunch.com/2019/01/09/baidu-cloud-launches-its-open-source-edge-computing-platform/

[32] AIOTI WG03 - loT Standardisation, "AIOTI High Level Architecture (HLA)," The Alliance for the Internet of Things Innovation, Tech. Rep. Release 4.0, Jun. 2018. [Online]. Available: https://aioti.eu/wp-content/uploads/2018/06/AIOTI-HLA-R4.0.7.1-Final.pdf

[33] ITU-T Study Group 13, "Overview of the Internet of Things," Internation Telecommunication Union, Tech. Rep. Recommendation ITU-T Y.2060, Jun. 2012. [Online]. Available: https://www.itu.int/rec/T-REC-Y.2060-201206-I

[34] ETSI, "oneM2M - Functional Architecture," European Telecommunications Standards Institute, Tech. Rep. ETSI TS 118 101 V2.10.0, Oct. 2016. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/118100_118199/118101/02.10.00_60/ts_118101v021000p.pdf

[35] K. Yamamoto and C. Mladin, "Study on Edge and Fog Computing in oneM2M systems," oneM2M consortium, Tech. Rep. oneM2M-TR-0052-V-0.13.1, Sep. 2020. [Online]. Available: https://member.onem2m.org/Application/documentapp/downloadLatestRevision/?docId=26390

[36] The Linux Foundation, "Why EdgeX." [Online]. Available: https://www.edgexfoundry.org/why_edgex/why-edgex/

[37] C. Pautasso, O. Zimmermann, M. Amundsen, J. Lewis, and N. Josuttis, "Microservices in practice, part 1: Reality check and service design," *IEEE Software*, vol. 34, no. 1, pp. 91–98, 2017.

[38] The Linux Foundation, "EdgeX foundry architecture web page," The Linux Foundation, Tech. Rep., 2021. [Online]. Available: https://www.edgexfoundry.org/software/platform/

[39] Amazon Web Services, "AWS greengrass developer guide," 2018. [Online]. Available: https://docs.aws.amazon.com/greengrass/

[40] Microsoft, "Microsoft azure IoT edge," 2018. [Online]. Available: https://azure.microsoft.com/en-us/services/iot-edge/

[41] M. Antonini, M. Vecchio, F. Antonelli, P. Ducange, and C. Perera, "Smart Audio Sensors in the Internet of Things Edge for Anomaly Detection," *IEEE Access*, vol. 6, pp. 67 594–67 610, 2018.

[42] A. I. Rana, G. Estrada, M. Sole, and V. Muntes, "Anomaly detection guidelines for data streams in big data," in *2016 3rd international conference on soft computing & machine intelligence (ISCMI)*. IEEE, Nov. 2016, pp. 94–98.

[43] D. M. Hawkins, *Identification of outliers*. Dordrecht: Springer Netherlands, 1980.

[44] J. R. Beniger, V. Barnett, and T. Lewis, "Outliers in statistical data." *Contemporary Sociology*, vol. 9, no. 4, p. 560, Jul. 1980.

[45] D. S. Moore, G. P. McCabe, and B. A. Craig, *Introduction to the practice of statistics*. W.H. Freeman, 2009.

[46] M. Vecchio, P. Azzoni, A. Menychtas, I. Maglogiannis, and A. Felfernig, "A Fully Open-Source Approach to Intelligent Edge Computing: AGILE's Lesson," *Sensors*, vol. 21, no. 4, p. 1309, Feb. 2021.

[47] Z. H. Janjua, M. Vecchio, M. Antonini, and F. Antonelli, "IRESE: An intelligent rare-event detection system using unsupervised learning on the IoT edge," *Engineering Applications of Artificial Intelligence*, vol. 84, pp. 41–50, Sep. 2019.

[48] C. C. Aggarwal, P. S. Yu, J. Han, and J. Wang, "A framework for clustering evolving data streams," in *Proceedings 2003 VLDB conference*. Elsevier, 2003, pp. 81–92.

[49] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: an efficient data clustering method for very large databases," *Proceedings of the 1996 ACM SIGMOD international conference on Management of data - SIGMOD '96*, vol. 25, no. 2, pp. 103–114, 1996.

[50] F. Murtagh and P. Legendre, "Ward's hierarchical agglomerative clustering method: Which algorithms implement ward's criterion?" *Journal of Classification*, vol. 31, no. 3, pp. 274–295, Oct. 2014.

[51] D. Wang, D. S. Yeung, and E. C. C. Tsang, "Structured One-Class Classification," *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, vol. 36, no. 6, pp. 1283–1295, Dec. 2006.

[52] J. Santos, P. Leroux, T. Wauters, B. Volckaert, and F. De Turck, "Anomaly detection for Smart City applications over 5G low power wide area networks," in *NOMS 2018 - 2018 IEEE/IFIP network operations and management symposium*. IEEE, Apr. 2018, pp. 1–9.

[53] M. Goldstein and S. Uchida, "A Comparative Evaluation of Unsupervised Anomaly Detection Algorithms for Multivariate Data," *PLOS ONE*, vol. 11, no. 4, p. e0152173, Apr. 2016.

[54] S. M. Erfani, S. Rajasegarar, S. Karunasekera, and C. Leckie, "High-dimensional and large-scale anomaly detection using a linear one-class SVM with deep learning," *Pattern Recognition*, vol. 58, pp. 121–134, 2016.

[55] X. Miao, Y. Liu, H. Zhao, S. Member, C. Li, and S. Member, "Distributed online one-class support vector machine for anomaly detection over networks," *IEEE Transactions on Cybernetics*, vol. 49, no. 4, pp. 1475–1488, 2018.

[56] Z. He, X. Xu, J. Z. Huang, and S. Deng, "A Frequent Pattern Discovery Method for Outlier Detection," in *Advances in Web-Age Information Management*, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, Q. Li, G. Wang, and L. Feng, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, vol. 3129, pp. 726–732.

[57] M. D. Ruiz, M. J. M. Bautista, D. Sanchez, M. A. Vila, and M. Delgado, "Anomaly detection using fuzzy association rules," *International Journal of Electronic Security and Digital Forensics*, vol. 6, no. 1, p. 25, 2014.

[58] J. Kevric, S. Jukic, and A. Subasi, "An effective combining classifier approach using tree algorithms for network intrusion detection," *Neural Computing and Applications*, vol. 28, no. S1, pp. 1051–1058, Dec. 2017.

[59] A. R. Hilal, A. Sayedelahl, A. Tabibiazar, M. S. Kamel, and O. A. Basir, "A distributed sensor management for large-scale IoT indoor acoustic surveillance," *Future Generation Computer Systems*, vol. 86, pp. 1170–1184, 2018.

[60] L. Koc, T. A. Mazzuchi, and S. Sarkani, "A network intrusion detection system based on a Hidden Naïve Bayes multiclass classifier," *Expert Systems with Applications*, vol. 39, no. 18, pp. 13 492–13 500, Dec. 2012.

[61] A. S. A. Aziz, S. E.-O. Hanafi, and A. E. Hassanien, "Comparison of classification techniques applied for network intrusion detection and classification," *Journal of Applied Logic*, vol. 24, pp. 109–118, Nov. 2017.

[62] X. Zhu and A. B. Goldberg, "Introduction to Semi-Supervised Learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 3, no. 1, pp. 1–130, Jan. 2009.

[63] R. A. R. Ashfaq, X.-Z. Wang, J. Z. Huang, H. Abbas, and Y.-L. He, "Fuzziness based semi-supervised learning approach for intrusion detection system," *Information Sciences*, vol. 378, pp. 484–497, Feb. 2017.

[64] H. Song, Z. Jiang, A. Men, and B. Yang, "A Hybrid Semi-Supervised Anomaly Detection Model for High-Dimensional Data," *Computational Intelligence and Neuroscience*, vol. 2017, pp. 1–9, 2017.

[65] G. E. Hinton, "Reducing the Dimensionality of Data with Neural Networks," *Science*, vol. 313, no. 5786, pp. 504–507, Jul. 2006.

[66] A. Mesaros, T. Heittola, A. Diment, B. Elizalde, A. Shah, E. Vincent, B. Raj, and T. Virtanen, "DCASE 2017 challenge setup: Tasks, datasets and baseline system," in *Proceedings of the detection and classification of acoustic scenes and events 2017 workshop (DCASE2017)*, 2017, pp. 85–92.

[67] H. Lim, J. Park, and Y. Han, "Rare sound event detection using 1D convolutional recurrent neural networks," DCASE2017 Challenge, Tech. Rep., 2017.

[68] E. Cakir and T. Virtanen, "Convolutional recurrent neural networks for rare sound event detection," DCASE2017 Challenge, Tech. Rep., 2017.

[69] W. Kaiwu, Y. Liping, and Y. Bin, "Audio events detection and classification using extended R-FCN approach," DCASE2017 Challenge, Tech. Rep., 2017.

[70] F. Vesperini, D. Droghini, D. Ferretti, E. Principi, L. Gabrielli, S. Squartini, and F. Piazza, "A hierarchic multi-scaled approach for rare sound event detection," DCASE2017 Challenge, Tech. Rep., 2017.

[71] D. Oh and I. Yun, "Residual error based anomaly detection using auto-encoder in SMD machine sound," *Sensors*, vol. 18, no. 5, p. 1308, Apr. 2018.

[72] Y. Koizumi, S. Saito, H. Uematsu, Y. Kawachi, and N. Harada, "Unsupervised detection of anomalous sound based on deep learning and the Neyman–Pearson lemma," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 27, no. 1, pp. 212–224, Jan. 2019.

[73] S. K. Bose, B. Kar, M. Roy, P. K. Gopalakrishnan, and A. Basu, "ADEPOS: anomaly detection based power saving for predictive maintenance using edge computing," in *Proceedings of the 24th asia and south pacific design automation conference*. New York, NY, USA: ACM, Jan. 2019, pp. 597–602.

[74] F. Aurino, M. Folla, F. Gargiulo, V. Moscato, A. Picariello, and C. Sansone, "One-class SVM based approach for detecting anomalous audio events," in *2014 international conference on intelligent networking and collaborative systems*. IEEE, Sep. 2014, pp. 145–151.

[75] B. Elizalde, A. Shah, S. Dalmia, M. H. Lee, R. Badlani, A. Kumar, B. Raj, and I. Lane, "An approach for self-training audio event detectors using web data," in *2017 25th european signal processing conference (EUSIPCO)*. IEEE, Aug. 2017, pp. 1863–1867.

[76] J. Salamon, C. Jacoby, and J. P. Bello, "A Dataset and Taxonomy for Urban Sound Research," in *Proceedings of the 22nd ACM international conference on Multimedia*. Orlando Florida USA: ACM, Nov. 2014, pp. 1041–1044.

[77] J. Socoró, F. Alías, and R. Alsina-Pagès, "An anomalous noise events detector for dynamic road traffic noise mapping in real-life urban and suburban environments," *Sensors*, vol. 17, no. 10, p. 2323, Oct. 2017.

[78] R. M. Alsina-Pagès, J. Navarro, F. Alías, and M. Hervás, "Homesound: Real-time audio event detection based on high performance computing for behaviour and surveillance remote monitoring," *Sensors*, vol. 17, no. 4, 2017.

[79] X. Xia, R. Togneri, F. Sohel, and D. Huang, "Random forest classification based acoustic event detection utilizing contextual-information and bottleneck features," *Pattern Recognition*, vol. 81, pp. 1–13, Sep. 2018.

[80] D. O'Shaughnessy, "Linear predictive coding," *IEEE Potentials*, vol. 7, no. 1, pp. 29–32, Feb. 1988.

[81] J. D. Markel and A. H. Gray, *Linear Prediction of Speech*, ser. Communication and Cybernetics. Berlin, Heidelberg: Springer Berlin Heidelberg, 1976, vol. 12.

[82] P. Bansal, S. A. Imam, and R. Bharti, "Speaker recognition using MFCC, shifted MFCC with vector quantization and fuzzy," in *2015 International Conference on Soft Computing Techniques and Implementations (ICSCTI)*, Oct. 2015, pp. 41–44.

[83] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, no. 4, pp. 357–366, 1980.

[84] X. Valero and F. Alias, "Gammatone cepstral coefficients: Biologically inspired features for non-speech audio classification," *IEEE Transactions on Multimedia*, vol. 14, no. 6, pp. 1684–1689, Dec. 2012.

[85] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge: Cambridge University Press, 2014.

[86] Z. Ghahramani, "Unsupervised learning," in *ML summer schools 2003: Advanced lectures on machine learning*, O. Bousquet, U. von Luxburg, and G. Rätsch, Eds. Springer Berlin Heidelberg, 2004, pp. 72–112.

[87] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering data streams: theory and practice," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 3, pp. 515–528, May 2003.

[88] M. Carnein, D. Assenmacher, and H. Trautmann, "An empirical comparison of stream clustering algorithms," in *Proceedings of the computing frontiers conference*. New York, NY, USA: ACM, May 2017, pp. 361–366.

[89] L. Rokach and O. Maimon, "Clustering Methods," in *Data Mining and Knowledge Discovery Handbook*, O. Maimon and L. Rokach, Eds. New York: Springer-Verlag, 2005, pp. 321–352.

[90] B. McFee, M. McVicar, S. Balke, C. Thomé, C. Raffel, D. Lee, O. Nieto, E. Battenberg, D. Ellis, R. Yamamoto, J. Moore, R. Bittner, K. Choi, P. Friesch, F.-R. Stöter, V. Lostanlen, S. Kumar, S. Waloschek, Seth, R. Naktinis, D. Repetto, C. Hawthorne, C. Carr, W. Pimenta, P. Viktorin, P. Brossier, J. F. Santos, JackieWu, Erik, and A. Holovaty, "Librosa/Librosa: 0.6.1," May 2018. [Online]. Available: https://zenodo.org/record/1252297

[91] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, no. 85, pp. 2825–2830, 2011.

[92] "The TUT audio dataset." [Online]. Available: https://webpages.tuni.fi/arg/datasets

[93] Google, "The audio dataset by Google." [Online]. Available: https://research.google.com/audioset/

[94] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys*, vol. 41, no. 3, pp. 1–58, Jul. 2009.

[95] M. Vecchio, R. Giaffreda, and F. Marcelloni, "Adaptive Lossless Entropy Compressors for Tiny IoT Devices," *IEEE Transactions on Wireless Communications*, vol. 13, no. 2, pp. 1088–1100, Feb. 2014.

[96] R. Ul Islam, M. S. Hossain, and K. Andersson, "A novel anomaly detection algorithm for sensor data under uncertainty," *Soft Computing*, vol. 22, no. 5, pp. 1623–1639, Mar. 2018.

[97] S. Trilles, O. Belmonte, S. Schade, and J. Huerta, "A domain-independent methodology to analyze IoT data streams in real-time. A proof of concept implementation for anomaly detection from environmental data," *International Journal of Digital Earth*, vol. 10, no. 1, pp. 103–120, Jan. 2017.

[98] L. Lyu, J. Jin, S. Rajasegarar, X. He, and M. Palaniswami, "Fog-empowered anomaly detection in IoT using hyperellipsoidal clustering," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1174–1184, Oct. 2017.

[99] S. Rajasegarar, A. Gluhak, M. Ali Imran, M. Nati, M. Moshtaghi, C. Leckie, and M. Palaniswami, "Ellipsoidal neighbourhood outlier factor for distributed anomaly detection in resource constrained networks," *Pattern Recognition*, vol. 47, no. 9, pp. 2867–2879, Sep. 2014.

[100] P. Rathore, A. S. Rao, S. Rajasegarar, E. Vanz, J. Gubbi, and M. Palaniswami, "Real-Time Urban Microclimate Analysis Using Internet of Things," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 500–511, Apr. 2018.

[101] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, Feb. 2018.

[102] J. Ye, T. Kobayashi, and T. Higuchi, "Smart audio sensor on anomaly respiration detection using FLAC features," in *2012 IEEE Sensors*

Applications Symposium Proceedings. Brescia, Italy: IEEE, Feb. 2012, pp. 1–5.

[103] J. Thones, "Microservices," *IEEE Software*, vol. 32, no. 1, pp. 116–116, Jan. 2015.

[104] S. S. Stevens, J. Volkmann, and E. B. Newman, "A Scale for the Measurement of the Psychological Magnitude Pitch," *The Journal of the Acoustical Society of America*, vol. 8, no. 3, pp. 185–190, Jan. 1937.

[105] M. Slaney, "Auditory toolbox: a Matlab toolbox for auditory modelling work," Interval Research Corporation, Tech. Rep. 1998-010, 1998. [Online]. Available: https://engineering.purdue.edu/~malcolm/interval/1998-010/

[106] S. Young, G. Evermann, M. J. F. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. Odell, D. Ollason, D. Povey, A. Ragni, V. Valtchev, P. Woodland, and C. Zhang, *The HTK book*, 3rd ed. Cambridge University Press, 2015.

[107] ITU-T Study Group 12, "One-way transmission time," Internation Telecommunication Union, Tech. Rep. Recommendation G.114, May 2003. [Online]. Available: https://www.itu.int/rec/T-REC-G.114

[108] P. J. Rousseeuw and K. van Driessen, "A fast algorithm for the minimum covariance determinant estimator," *Technometrics*, vol. 41, no. 3, pp. 212–223, Aug. 1999.

[109] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation Forest," in *2008 Eighth IEEE International Conference on Data Mining*. Pisa, Italy: IEEE, Dec. 2008, pp. 413–422.

[110] ——, "Isolation-Based Anomaly Detection," *ACM Transactions on Knowledge Discovery from Data*, vol. 6, no. 1, pp. 1–39, Mar. 2012.

[111] N. Chinchor, "MUC-4 evaluation metrics," in *Proceedings of the 4th conference on Message understanding - MUC4 '92*. McLean, Virginia: Association for Computational Linguistics, 1992, p. 22.

[112] G. E. Poliner and D. P. W. Ellis, "A Discriminative Model for Polyphonic Piano Transcription," *EURASIP Journal on Advances in Signal Processing*, vol. 2007, no. 1, p. 048317, Dec. 2006.

[113] A. Mesaros, T. Heittola, and T. Virtanen, "Metrics for Polyphonic Sound Event Detection," *Applied Sciences*, vol. 6, no. 6, p. 162, May 2016.

[114] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, and F. Kawsar, "An Early Resource Characterization of Deep Learning on Wearables, Smartphones and Internet-of-Things Devices," in *Proceedings of the 2015 International Workshop on Internet of Things towards Applications*. Seoul South Korea: ACM, Nov. 2015, pp. 7–12.

[115] M. Almeida, S. Laskaridis, I. Leontiadis, S. I. Venieris, and N. D. Lane, "EmBench: Quantifying Performance Variations of Deep Neural Networks across Modern Commodity Devices," in *The 3rd International Workshop on Deep Learning for Mobile Systems and Applications - EMDL '19*. Seoul, Republic of Korea: ACM Press, 2019, pp. 1–6.

[116] M. Antonini, T. H. Vu, C. Min, A. Montanari, A. Mathur, and F. Kawsar, "Resource Characterisation of Personal-Scale Sensing Models on Edge Accelerators," in *Proceedings of the First International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things - AIChallengeIoT'19*. New York, NY, USA: ACM Press, 2019, pp. 49–55.

[117] Google, "TensorFlow models on the Edge TPU." [Online]. Available: https://coral.ai/docs/edgetpu/models-intro/

[118] P. Gysel, M. Motamedi, and S. Ghiasi, "Hardware-oriented approximation of convolutional neural networks," *CoRR*, vol. abs/1604.03168, 2016. [Online]. Available: http://arxiv.org/abs/1604.03168

[119] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Salt Lake City, UT: IEEE, Jun. 2018, pp. 2704–2713.

[120] Google, "Post-Training Quantization | TensorFlow Lite." [Online]. Available: https://www.tensorflow.org/lite/performance/post_training_quantization

[121] L. Peng, L. Chen, Z. Ye, and Y. Zhang, "AROMA: A Deep Multi-Task Learning Based Simple and Complex Human Activity Recognition Method Using Wearable Sensors," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 2, no. 2, pp. 1–16, Jul. 2018.

[122] S. Katayama, A. Mathur, T. Okoshi, J. Nakazawa, and F. Kawsar, "Situation-Aware Conversational Agent with Kinetic Earables (demo)," in *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*. Seoul Republic of Korea: ACM, Jun. 2019, pp. 657–658.

[123] A. Mathur, A. Isopoussu, F. Kawsar, N. Berthouze, and N. D. Lane, "Mic2Mic: using cycle-consistent generative adversarial networks to overcome microphone variability in speech systems," in *Proceedings of the 18th International Conference on Information Processing in Sensor Networks*. Montreal Quebec Canada: ACM, Apr. 2019, pp. 169–180.

[124] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size," *arXiv:1602.07360 [cs]*, Nov. 2016. [Online]. Available: http://arxiv.org/abs/1602.07360

[125] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[126] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," *arXiv preprint arXiv:1905.11946*, 2019.

[127] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Boston, MA, USA: IEEE, Jun. 2015, pp. 1–9.

[128] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE, Jul. 2017, pp. 2261–2269.

[129] S. Chernbumroong, S. Cang, A. Atkins, and H. Yu, "Elderly activities recognition and classification for applications in assisted living," *Expert Systems with Applications*, vol. 40, no. 5, pp. 1662–1674, Apr. 2013.

[130] R. Wang, F. Chen, Z. Chen, T. Li, G. Harari, S. Tignor, X. Zhou, D. Ben-Zeev, and A. T. Campbell, "StudentLife: assessing mental health, academic performance and behavioral trends of college students using smartphones," in *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. Seattle Washington: ACM, Sep. 2014, pp. 3–14.

[131] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, Apr. 2017.

[132] A. C1055-03, "Standard Guide for Heated System Surface Conditions That Produce Contact Burn Injuries," ASTM International West Conshohocken, ed. Philadelphia, PA, Tech. Rep. ASTM C1055-03, 2014.

[133] M. Antonini, A. Gaiardo, and M. Vecchio, "MetaNChemo: A metaheuristic neural-based framework for chemometric analysis," *Applied Soft Computing*, vol. 97, p. 106712, Dec. 2020.

[134] CDC, "Disease of the Week - CO Poisoning," Mar. 2021. [Online]. Available: http://www.cdc.gov/dotw/carbonmonoxide/index.html

[135] A. Bagolini, A. Gaiardo, M. Crivellari, E. Demenev, R. Bartali, A. Picciotto, M. Valt, F. Ficorella, V. Guidi, and P. Bellutti, "Development of MEMS MOS gas sensors with CMOS compatible PECVD inter-metal passivation," *Sensors and Actuators B: Chemical*, vol. 292, pp. 225–232, Aug. 2019.

[136] G. Zonta, M. Astolfi, D. Casotti, G. Cruciani, B. Fabbri, A. Gaiardo, S. Gherardi, V. Guidi, N. Landini, M. Valt, and C. Malagù, "Reproducibility tests with zinc oxide thick-film sensors," *Ceramics International*, vol. 46, no. 5, pp. 6847–6855, Apr. 2020.

[137] K. Mirabbaszadeh and M. Mehrabian, "Synthesis and properties of ZnO nanorods as ethanol gas sensors," *Physica Scripta*, vol. 85, no. 3, p. 035701, Mar. 2012.

[138] J. Del Ser, E. Osaba, D. Molina, X.-S. Yang, S. Salcedo-Sanz, D. Camacho, S. Das, P. N. Suganthan, C. A. Coello Coello, and F. Herrera, "Bio-inspired computation: Where we stand and what's next," *Swarm and Evolutionary Computation*, vol. 48, pp. 220–250, Aug. 2019.

[139] D. J. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms," in *Proceedings of the 11th international joint conference on artificial intelligence - volume 1*. Morgan Kaufmann Publishers Inc., 1989, pp. 762–767.

[140] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning," *arXiv:1712.06567 [cs]*, Apr. 2018, arXiv: 1712.06567. [Online]. Available: http://arxiv.org/abs/1712.06567

[141] S. Lander and Y. Shang, "EvoAE – A New Evolutionary Method for Training Autoencoders for Deep Learning Networks," in *2015 IEEE 39th Annual Computer Software and Applications Conference*. Taichung, Taiwan: IEEE, Jul. 2015, pp. 790–795.

[142] D. Whitley, T. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: optimizing connections and connectivity," *Parallel Computing*, vol. 14, no. 3, pp. 347–361, Aug. 1990.

[143] Xin Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, Sep. 1999.

[144] F. Gomez, J. Schmidhuber, and R. Miikkulainen, "Accelerated neural evolution through cooperatively coevolved synapses," *Journal of Machine Learning Research*, vol. 9, pp. 937–965, Jun. 2008.

[145] K. O. Stanley and R. Miikkulainen, "Evolving Neural Networks through Augmenting Topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, Jun. 2002.

[146] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat, "Evolving Deep Neural Networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Elsevier, 2019, pp. 293–312.

[147] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proceedings of the Genetic and Evolutionary Computation Conference*. Berlin Germany: ACM, Jul. 2017, pp. 497–504.

[148] N. Bacanin, T. Bezdan, E. Tuba, I. Strumberger, and M. Tuba, "Monarch Butterfly Optimization Based Convolutional Neural Network Design," *Mathematics*, vol. 8, no. 6, p. 936, Jun. 2020.

[149] G. H. de Rosa, J. P. Papa, and X.-S. Yang, "Handling dropout probability estimation in convolution neural networks using metaheuristics," *Soft Computing*, vol. 22, no. 18, pp. 6147–6156, Sep. 2018.

[150] S. Oreski and G. Oreski, "Genetic algorithm-based heuristic for feature selection in credit risk assessment," *Expert Systems with Applications*, vol. 41, no. 4, pp. 2052–2064, Mar. 2014.

[151] A. Rosales-Perez, S. Garcia, J. A. Gonzalez, C. A. Coello Coello, and F. Herrera, "An Evolutionary Multiobjective Model and Instance Selection for Support Vector Machines With Pareto-Based Ensembles," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 6, pp. 863–877, Dec. 2017.

[152] A. Unler, A. Murat, and R. B. Chinnam, "mr2PSO: A maximum relevance minimum redundancy feature selection method based on swarm intelligence for support vector machine classification," *Information Sciences*, vol. 181, no. 20, pp. 4625–4641, Oct. 2011.

[153] B. Xue, M. Zhang, and W. N. Browne, "Particle Swarm Optimization for Feature Selection in Classification: A Multi-Objective Approach," *IEEE Transactions on Cybernetics*, vol. 43, no. 6, pp. 1656–1671, Dec. 2013.

[154] S. Feng, F. Farha, Q. Li, Y. Wan, Y. Xu, T. Zhang, and H. Ning, "Review on Smart Gas Sensing Technology," *Sensors*, vol. 19, no. 17, p. 3760, Aug. 2019.

[155] A. Vergara, S. Vembu, T. Ayhan, M. A. Ryan, M. L. Homer, and R. Huerta, "Chemical gas sensor drift compensation using classifier ensembles," *Sensors and Actuators B: Chemical*, vol. 166-167, pp. 320–329, May 2012.

[156] K. Wang, W. Ye, X. Zhao, and X. Pan, "A Support Vector Machine-Based Genetic AlgorithmMethod for Gas Classification," in *2017 2nd International Conference on Frontiers of Sensors Technologies (ICFST)*. Shenzhen: IEEE, Apr. 2017, pp. 363–366.

[157] B. Szulczynski, K. Arminski, J. Namiesnik, and J. Gebicki, "Determination of Odour Interactions in Gaseous Mixtures Using Electronic Nose Methods with Artificial Neural Networks," *Sensors*, vol. 18, no. 2, p. 519, Feb. 2018.

[158] J. G. Casey, A. Collier-Oxandale, and M. Hannigan, "Performance of artificial neural networks and linear models to quantify 4 trace gas species in an oil and gas production region with low-cost sensors," *Sensors and Actuators B: Chemical*, vol. 283, pp. 504–514, Mar. 2019.

[159] N. Barsan and U. Weimar, "Understanding the fundamental principles of metal oxide based gas sensors; the example of CO sensing with SnO $_2$ sensors in the presence of humidity," *Journal of Physics: Condensed Matter*, vol. 15, no. 20, pp. R813–R839, May 2003.

[160] US Department of Labor, "Permissible Exposure Limits - Annotated Tables | Occupational Safety and Health Administration." [Online]. Available: https://www.osha.gov/annotated-pels

[161] M. T. M. Emmerich and A. H. Deutz, "A tutorial on multiobjective optimization: fundamentals and evolutionary methods," *Natural Computing*, vol. 17, no. 3, pp. 585–609, Sep. 2018.

[162] J. Kennedy, R. C. Eberhart, and Y. Shi, *Swarm intelligence*, ser. The Morgan Kaufmann series in evolutionary computation. San Francisco: Kaufmann, 2001.

[163] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[164] C. R. Reeves and J. E. Rowe, *Genetic Algorithms—Principles and Perspectives: A Guide to GA Theory*, ser. Operations Research/Computer Science Interfaces Series, R. Sharda and S. Voß, Eds. Boston, MA: Springer US, 2002, vol. 20.

[165] K. Deb and S. Tiwari, "Omni-optimizer: A generic evolutionary algorithm for single and multi-objective optimization," *European Journal of Operational Research*, vol. 185, no. 3, pp. 1062–1087, Mar. 2008.

[166] A. Benítez-Hidalgo, A. J. Nebro, J. García-Nieto, I. Oregi, and J. Del Ser, "jMetalPy: A Python framework for multi-objective optimization with metaheuristics," *Swarm and Evolutionary Computation*, vol. 51, p. 100598, Dec. 2019.

[167] F. Marcelloni and M. Vecchio, "Enabling energy-efficient and lossy-aware data compression in wireless sensor networks by multi-objective evolutionary optimization," *Information Sciences*, vol. 180, no. 10, pp. 1924–1941, May 2010.

[168] J. J. Durillo and A. J. Nebro, "jMetal: A Java framework for multi-objective optimization," *Advances in Engineering Software*, vol. 42, no. 10, pp. 760–771, Oct. 2011, tex.ids: durillo2011jmetal publisher: Elsevier.

[169] C. Fonseca, L. Paquete, and M. Lopez-Ibanez, "An Improved Dimension-Sweep Algorithm for the Hypervolume Indicator," in *2006 IEEE International Conference on Evolutionary Computation*. Vancouver, BC, Canada: IEEE, 2006, pp. 1157–1163.

[170] A. Nebro, J. Durillo, J. Garcia-Nieto, C. Coello Coello, F. Luna, and E. Alba, "SMPSO: A new PSO-based metaheuristic for multi-objective optimization," in *2009 IEEE Symposium on Computational Intelligence in Milti-Criteria Decision-Making*. Nashville, TN, USA: IEEE, Mar. 2009, pp. 66–73.

[171] J. J. Durillo, J. García-Nieto, A. J. Nebro, C. A. C. Coello, F. Luna, and E. Alba, "Multi-Objective Particle Swarm Optimizers: An Experimental Comparison," in *Evolutionary Multi-Criterion Optimization*, M. Ehrgott, C. M. Fonseca, X. Gandibleux, J.-K. Hao, and M. Sevaux, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, vol. 5467, pp. 495–509.

[172] Mojtaba Ahmadieh Khanesar, Mohammad Teshnehlab, and Mahdi Aliyari Shoorehdeli, "A novel binary particle swarm optimization," in *2007 Mediterranean Conference on Control & Automation*. Athens, Greece: IEEE, Jun. 2007, pp. 1–6.

[173] A. S. A. Beegom and M. S. Rajasree, "Integer-PSO: a discrete PSO algorithm for task scheduling in cloud computing systems," *Evolutionary Intelligence*, vol. 12, no. 2, pp. 227–239, Jun. 2019.

[174] C. A. Coello Coello and M. Reyes Sierra, "A Study of the Parallelization of a Coevolutionary Multi-objective Evolutionary Algorithm," in *MICAI 2004: Advances in Artificial Intelligence*, G. Goos, J. Hartmanis, J. van Leeuwen, R. Monroy, G. Arroyo-Figueroa, L. E. Sucar, and H. Sossa, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, vol. 2972, pp. 688–697.

[175] M. Walker and B. Burton, "Hype Cycle for Emerging Technologies, 2015," Gartner Inc., Tech. Rep. G00289755, Jul. 2015. [Online]. Available: https://www.gartner.com/en/documents/3100227/hype-cycle-for-emerging-technologies-2015

[176] M. Walker, "Hype Cycle for Emerging Technologies, 2018," Gartner Inc., Tech. Rep. G00340159, Jun. 2018. [Online]. Available: https://www.gartner.com/en/documents/3885468/hype-cycle-for-emerging-technologies-2018

[177] Gartner Inc., "5 Trends Drive the Gartner Hype Cycle for Emerging Technologies, 2020," 2020. [Online]. Available: //www.gartner.com/smarterwithgartner/5-trends-drive-the-gartner-hype-cycle-for-emerging-technologies-2020/

[178] T. Esch, S. Üreyen, J. Zeidler, A. Metz–Marconcini, A. Hirner, H. Asamer, M. Tum, M. Böttcher, S. Kuchar, V. Svaton, and M. Marconcini, "Exploiting big earth data from space – first experiences with the timescan processing chain," *Big Earth Data*, vol. 2, no. 1, pp. 36–55, Jan. 2018. [Online]. Available: https://doi.org/10.1080/20964471.2018.1433790

[179] European Space Agency, "ESA phi-sat," 2020. [Online]. Available: https://www.esa.int/Applications/Observing_the_Earth/Ph-sat

[180] ——, "Next artificial intelligence mission selected - ESA," Sep. 2020. [Online]. Available: https://www.esa.int/Applications/Observing_the_Earth/Ph-sat

[181] E. Comini, G. Faglia, and G. Sberveglieri, Eds., *Solid state gas sensing*. New York, N.Y: Springer, 2009.

[182] G. Neri, "First Fifty Years of Chemoresistive Gas Sensors," *Chemosensors*, vol. 3, no. 1, pp. 1–20, Jan. 2015.

[183] A. Dey, "Semiconductor metal oxide gas sensors: A review," *Materials Science and Engineering: B*, vol. 229, pp. 206–217, Mar. 2018.

[184] M. Valt, B. Fabbri, A. Gaiardo, S. Gherardi, D. Casotti, G. Cruciani, G. Pepponi, L. Vanzetti, E. Iacob, C. Malagù, P. Bellutti, and V. Guidi, "Aza-crown-ether functionalized graphene oxide for gas sensing and cation trapping applications," *Materials Research Express*, vol. 6, no. 7, p. 075603, Apr. 2019.

[185] A. Gaiardo, B. Fabbri, V. Guidi, P. Bellutti, A. Giberti, S. Gherardi, L. Vanzetti, C. Malagù, and G. Zonta, "Metal Sulfides as Sensing Materials for Chemoresistive Gas Sensors," *Sensors*, vol. 16, no. 3, p. 296, Feb. 2016.

[186] T. Han, A. Nag, S. Chandra Mukhopadhyay, and Y. Xu, "Carbon nanotubes and its gas-sensing applications: A review," *Sensors and Actuators A: Physical*, vol. 291, pp. 107–143, Jun. 2019.

[187] Y.-F. Sun, S.-B. Liu, F.-L. Meng, J.-Y. Liu, Z. Jin, L.-T. Kong, and J.-H. Liu, "Metal Oxide Nanostructures and Their Gas Sensing Properties: A Review," *Sensors*, vol. 12, no. 3, pp. 2610–2631, Feb. 2012.

[188] H. Ji, W. Zeng, and Y. Li, "Gas sensing mechanisms of metal oxide semiconductors: a focus review," *Nanoscale*, vol. 11, no. 47, pp. 22 664–22 684, 2019.

[189] N. Barsan and U. Weimar, "Conduction model of metal oxide gas sensors," *Journal of electroceramics*, vol. 7, no. 3, pp. 143–167, 2001.

[190] G. Korotcenkov, *Handbook of Gas Sensor Materials: Properties, Advantages and Shortcomings for Applications Volume 1: Conventional Approaches*, ser. Integrated Analytical Systems. New York, NY: Springer New York, 2013.

[191] R. Baron and J. Saffell, "Amperometric Gas Sensors as a Low Cost Emerging Technology Platform for Air Quality Monitoring Applications: A Review," *ACS Sensors*, vol. 2, no. 11, pp. 1553–1566, Nov. 2017.

[192] B. Fabbri, M. Valt, C. Parretta, S. Gherardi, A. Gaiardo, C. Malagù, F. Mantovani, V. Strati, and V. Guidi, "Correlation of gaseous emissions to water stress in tomato and maize crops: From field to laboratory and back," *Sensors and Actuators B: Chemical*, vol. 303, p. 127227, Jan. 2020.

[193] G. Zonta, G. Anania, M. Astolfi, C. Feo, A. Gaiardo, S. Gherardi, A. Giberti, V. Guidi, N. Landini, C. Palmonari, A. de Togni, and C. Malagù, "Chemoresistive sensors for colorectal cancer preventive screening through fecal odor: Double-blind approach," *Sensors and Actuators B: Chemical*, vol. 301, p. 127062, Dec. 2019.

[194] M. T. Carter, J. R. Stetter, M. W. Findlay, B. J. Meulendyk, V. Patel, and D. Peaslee, "Amperometric Gas Sensors: From Classical Industrial Health and Safety to Environmental Awareness and Public Health," *ECS Transactions*, vol. 75, no. 16, pp. 91–98, Sep. 2016.

[195] H. Benhebal, M. Chaib, T. Salmon, J. Geens, A. Leonard, S. D. Lambert, M. Crine, and B. Heinrichs, "Photocatalytic degradation of phenol and benzoic acid using zinc oxide powders prepared by the sol–gel process," *Alexandria Engineering Journal*, vol. 52, no. 3, pp. 517–523, Sep. 2013.

[196] A. Gaiardo, B. Fabbri, A. Giberti, M. Valt, S. Gherardi, V. Guidi, C. Malagù, P. Bellutti, G. Pepponi, D. Casotti, G. Cruciani, G. Zonta, N. Landini, M. Barozzi, S. Morandi, L. Vanzetti, R. Canteri, M. Della Ciana, A. Migliori, and E. Demenev, "Tunable formation of nanostructured SiC/SiOC core-shell for selective detection of SO2," *Sensors and Actuators B: Chemical*, vol. 305, p. 127485, Feb. 2020.

Websites accessed: May 28, 2021.

Appendix A

Chemoresistive sensors

In the last years, research around solid-state gas sensors has received a strong boost thanks to the development of innovative devices for the monitoring of gaseous molecules. Solid-state gas sensors are divided into four broad categories: optical sensors, quartz microbalances, electrochemical, and chemoresistive gas sensors [181]. Each type of gas sensor shows advantages and disadvantages; however, it is undeniable that the chemoresistive gas sensors are the most investigated because of their great versatility [182, 183]. A great effort has been devoted to the investigation of the sensing properties of several nanostructured semiconductors, such as Metal OXides (MOX), metal sulfides, polymers, graphene, and carbon nanotubes, to identifying the optimal sensing material [182, 184, 185, 186]. MOX sensors have been studied since the early 1970s and they are still the most widely used materials in the field of chemoresistive gas sensors, due to the high tunability of their physical and chemical properties and functionality [182, 183]. Furthermore, MOX can be synthesized in the form of low-dimensional 0D, 1D, 2D, and 3D nanostructures, which showed promising gas-sensing performances including fast response, high sensitivity, and very low detection limit [187].

Several studies have been carried out on the gas sensing mechanism modeling, which is typically based on the change in the thickness of the

Part of this appendix appears in the following publication that I co-authored:
M. Antonini, A. Gaiardo, and M. Vecchio "MetaNChemo: A meta-heuristic neural-based framework for chemometric analysis." *Applied Soft Computing*, vol. 97, p. 106712, Dec. 2020. Copyright Elsevier (2020). DOI: 10.1016/j.asoc.2020.106712.

MOX nanostructure depletion layer [182, 188, 189]. The depletion layer thickness is strongly dependent on the charge and amount of oxygen adsorbed on the nanostructure surface. On the one hand, the oxygen charge relies on the working temperature of the sensor [190]. On the other hand, the interaction of the sensing material with reducing or oxidizing gases significantly influences the amount of oxygen adsorbed on the MOX surface [182]. For n-type MOX semiconductors, such as SnO_2 and ZnO, the charge carriers are the electrons, thus an increase of the oxygen concentration adsorbed on the surface involves an increase of the semiconductor electrical resistivity, whereas a decrease of the adsorbed oxygens results in an increase of the free electrons in the semiconductor, involving a decrease in the MOX electrical resistivity [182]. The sensing material is usually deposited on a substrate, which plays the role of microheater, mechanical support and contains interdigitated electrodes necessary to measure the electrical resistance of the nanostructured semiconductor [135].

MOX chemoresistive gas sensors are particularly low cost, small, stable, highly sensitive and showed the advantage of higher throughput and amenability for large-scale integration. Nevertheless, they still show some shortcomings that limit their widespread use, including lack of selectivity and a constant drift of the sensor signal over time [190]. Indeed, Chapter 6 has also the duty to show some research opportunities offered by the combination of chemoresistive gas sensors with AI techniques, as also supported by [154]. Chemoresistive sensors have been successfully applied in indoor and outdoor air quality monitoring [191], precision agriculture [192], analysis and diagnosis of clinical disease with non-invasive methods [193], and safety in the workplace [194].

A.1 Chemoresistive sensors production at FBK

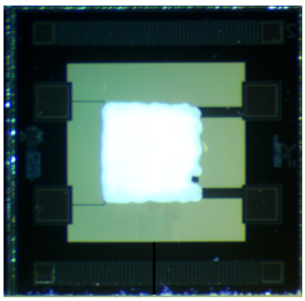
The chemoresistive gas sensors used in Chapter 6 were entirely produced in the laboratories of the Micro-Nano Facilities (MNF)[1] group of the Bruno Kessler Foundation (FBK, Trento, Italy). The gas sensor development can be divided into two main parts: the microfabrication process of silicon

[1] `https://mnf.fbk.eu/`.

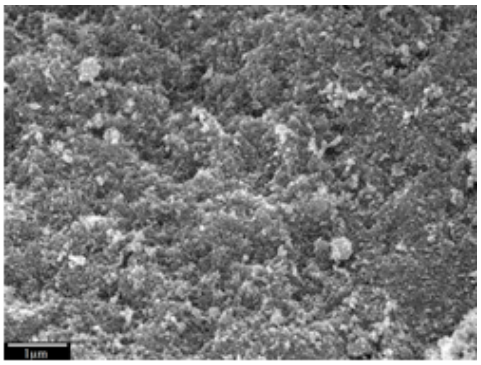
substrates and the synthesis of MOX semiconductors.

In the first step of the microfabrication process, a triple layer of $SiO_2/Si_3N_4/SiO_2$ (ONO) was deposited over a double-side polished silicon wafer using thermal oxidation and low-pressure chemical vapor deposition techniques. The thickness of the three layers was chosen to obtain a stack with zero mechanical stress. Afterward, a deposition of Ti (10 nm) and Pt (120 nm) layers was carried out by electron beam evaporation over the ONO stack. The Pt/Ti double layer was defined through a photolithographic process to obtain the heater. In the subsequent step, a SiO_2 insulating layer was deposited by using plasma-enhanced chemical vapor deposition, to electrically insulate the heater and the interdigitated electrodes. Then, Ti/Pt interdigitated electrodes were deposited and patterned by repeating the same process block used for the heater. Thermal treatment at 650 °C in N_2 was performed after each layer deposition, to thermal stabilize the microheaters at the MOX paste firing temperature. Finally, the membrane was released by chemically etching the silicon wafer from the backside. In a wafer, there were about 1200 devices, divided lastly by using a dicing saw. A scheme of the final microheater cross-section is reported in Figure A.1a. The microfabrication process is explained in detail in [135]. The MOX synthesized in Chapter 6 were SnO2 nanograins (SnO_2), ZnO nanograins (ZnO-1), and ZnO nanorods (ZnO-2). All of them were prepared through sol–gel synthesis. For SnO_2 and ZnO-1, the starting metallorganic reagents (Sn(II) ethylexanoate and Zn(II) acetate) were dissolved in 2-propanol. Then, water was added dropwise to the two solutions to obtain the formation of the SnO_2 and ZnO gels. The two syntheses are reported in detail in [135, 136]. The ZnO-2 powder was produced by dissolving Zn(II) acetate in a refluxed ethanol solution at 60 °C. Afterwards, acid oxalic was added to the solution to allow the growth of ZnO nanorods [195]. All the three gel were left precipitate in the solution overnight. The produced powders, separated from solutions by the filtration method, were dried at 100 °C and then calcined at 650 °C for 2 hours in an oven. The powders were then mixed with a suitable amount of alpha-terpineol and ethylcellulose to obtain printable pastes [196]. Afterward, the sensing pastes were deposited onto silicon microheaters through screen printing deposition (Figure A.1b) [135]. After deposition, the films

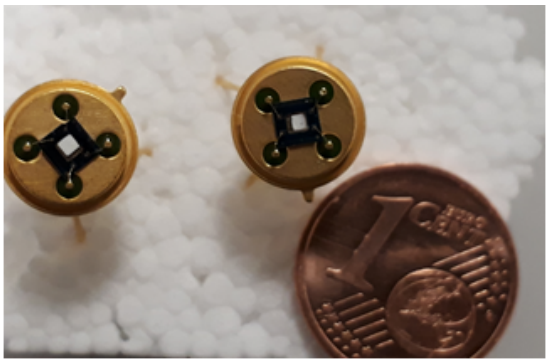
(a) Schema of a microheater cross-section.



(b) A microheater screen printed with SnO_2.



(c) A scanning electron microscope image of the SnO_2 thick film.



(d) Two devices bonded on T039 open packages.

Figure A.1: Schema and pictures of the sensors realized in the MNF facility at FBK (source: [133], p. 4).

were calcined for 2 hours at 650 °C to thermal stabilize them and to evaporate the solvents (Figure A.1c). The final devices were bonded on standard metallic open packages (TO39) by employing a gold ball bonding system (Figure A.1d) [135].