



Project Title	Artificial Intelligence in Secure PRIVacy-preserving computing coNTinuum
Project Acronym	AI-SPRINT
Project Number	101016577
Type of project	RIA - Research and Innovation action
Topics	ICT-40-2020 - Cloud Computing: towards a smart cloud computing continuum (RIA)
Starting date of Project	01 January 2021
Duration of the project	36 months
Website	www.ai-sprint-project.eu/

D1.1 - State of the Art Analysis

Work Package	WP1 Requirements & Architecture Definition
Task	T1.1 State of the Art Analysis
Lead author	Danilo Ardagna (POLIMI)
Contributors	Federica Filippini, Matteo Matteucci, Hamta Sedghani, Giacomo Verticale, Francesco Lattari (POLIMI), Eugenio Lomurno (POLIMI), Davide Cirillo, Daniele Lezzi, Francesc Lordan (BSC), Germán Moltó (UPV), André Martin (TUD), Patrick Thiem (C&H), Grzegorz Timoszuik (7BULLS), Michał Kłosiński (AF), Stephanie Parker (TRUST), Cristina Pepato (IDC)
Peer reviewers	Mahshid Mehrabi (TUD), Niccolò Zazzeri (TRUST)
Version	V1.0
Due Date	30/06/2021
Submission Date	30/06/2021

Dissemination Level

<input checked="" type="checkbox"/>	PU: Public
<input type="checkbox"/>	CO: Confidential, only for members of the consortium (including the Commission)
<input type="checkbox"/>	EU-RES. Classified Information: RESTREINT UE (Commission Decision 2005/444/EC)
<input type="checkbox"/>	EU-CON. Classified Information: CONFIDENTIEL UE (Commission Decision 2005/444/EC)
<input type="checkbox"/>	EU-SEC. Classified Information: SECRET UE (Commission Decision 2005/444/EC)



Versioning History

Revision	Date	Editors	Comments
0.1	6/5/2021	Danilo Ardagna	ToC definition
0.2	24/5/2021	Danilo Ardagna, German Moltó	Initial contributions by POLIMI and UPV
0.3	4/6/2020	Federica Filippini, Hamta Sedghani, Francesc Lordan, Cristina Pepato, Andrei Popa, Mahshid Mehrabi, Patrick Thiem, Davide Cirillo	Contributed to Sections 2.3, 3.4, 3.6, 6.1, 4.1, 4.5, 5.4
0.4	7/6/2020	Mahshid Mehrabi, André Martin, Francesco Lattari, Patrick Thiem	Contributed to Sections 6.1, 6.2, 3.1, 3.2, 3.4, 3.5, 6.3, 4.2
0.5	14/06/2020	Davide Cirillo, Danilo Ardagna	Section 3.6 completed. Review of Sections 2-3.4
0.6	16/06/2020	Andrei Popa	Contributed to 2.4, 3.4
0.7	17/06/2020	Christophe Baron, Danilo Ardagna	Section 3.7 completed, full review of current content
0.8	18/06/2021	Danilo Ardagna, Hamta Sedghani, Matteo Matteucci, Francesco Lattari, Riccardo Bertoglio, Giulio Fontana	Added contributions to Sections 4.4, 4.5 Revision of Sections 3.2, Section 3.7
0.85	20/06/2021	Matteo Matteucci, Danilo Ardagna	Revision of Sections 2.1.1, 3.5, 4.3, 4.2 Contributed to Executive Summary, Section 1, 3.9, 4.6, 5.6
0.9	21/06/2021	Matteo Matteucci, Eugenio Lomurno	Review of Section 4.3 Contributed to Section 6.5
0.95	28/06/2021	Stephanie Parker	Added Section 2.2
1.0	30/06/2021	Federica Filippini, Danilo Ardagna	Implemented Reviewers' comments

Glossary of terms

Item	Description
AF	Atrial Fibrillation
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
AutoML	Automated Machine Learning
CMP	Cloud Management Platform
CNN	Convolutional Neural Network
COMPSSs	COMP Superscalar
CPU	Central Processing Unit
CT	Computed Tomography
DDNN	Distributed Deep Neural Network
DL	Deep Learning
DNAS	Differentiable Neural Architecture Search
DNN	Deep Neural Network
DP	Differential Privacy
DSP	Digital Signal Processing
EAS	Edge application server
ECG	Electrocardiogram
EES	Edge Enabler Server
EI	Edge Intelligence
EMO	Evolutionary Multi-Objective Optimization
FaaS	Function as a Service
FATE	Federated AI Technology Enabler
FML	Federated Machine Learning
FTL	Federated Transfer Learning
GA	Genetic Algorithm
GAN	Generative Adversarial Network
GDPR	General Data Protection Regulation
GPIO	General-Purpose Input/Output
GPS	Global Positioning System
GPU	Graphics Processing Unit
GR	Group Report
HR	Heart Rate
IaaS	Infrastructure as a Service
IDE	Integrated Development Environment
IoT	Internet of Things
IIoT	Industrial Internet of Things
ILP	Integer Linear Programming
ISA	Instruction Set Architecture
ISG	Industry Specification Group
ISP	Image Signal Processor
K-NN	K-Nearest Neighbor

LSTM	Long Short-Term Memory
LUT	Lookup Table
MANO	Management and Network Orchestration
MEC	Mobile Edge Computing
MILP	Mixed Integer Linear Programming
ML	Machine Learning
MLP	Multilayer Perceptron
MPC	Multi-Party Computation
MRI	Magnetic Resonance Imaging
NAS	Network Architecture Search
NESAS	Network Equipment Security Assurance Scheme, jointly defined by 3GPP and GSMA
NFV	Network Function Virtualization
NFV SEC	ETSI NFV Security WG
NN	Neural Network
NPU	Neural Processing Unit
NSGA	Non-Dominated Sorting Genetic Algorithm
OpenFL	Open Federated Learning
PaaS	Platform as a Service
PCA	Principal Component Analysis
PPG	Photoplethysmography
QoS	Quality of Service
RNN	Recurrent Neural Network
SAI	Securing Artificial Intelligence
SGD	Stochastic Gradient Descent
SoC	System on a Chip
SoM	System on Module
SVM	Support Vector Machine
TC	Technical Committee
TCG	Trusted Computing Group
TEE	Trusted Execution environments
TinyML	Tiny Machine Learning
TOPI	TVM Operator Inventory
TPM	Trusted Platform Model
TPU	Tensor Processing Unit
TR	Technical Report
TS	Technical specification
VAE	Variational Autoencoder
VNF	Virtual Network Function
VPU	Visual Processing Unit
3GPP	3rd Generation Partnership Project
5GC	5G Core

Keywords

Cloud Computing; Cloud Infrastructure; Artificial Intelligence; Edge Computing; Computing Continuum; Software Design & Development; Cloud Trust & Security; Privacy;

Disclaimer

This document contains confidential information in the form of the AI-SPRINT project findings, work and products and its use is strictly regulated by the AI-SPRINT Consortium Agreement and by Contract no. 101016577.

Neither the AI-SPRINT Consortium nor any of its officers, employees or agents shall be responsible, liable in negligence, or otherwise however in respect of any inaccuracy or omission herein.

The contents of this document are the sole responsibility of the AI-SPRINT consortium and can in no way be taken to reflect the views of the European Commission and the REA.

Executive Summary

The aim of the AI-SPRINT “Artificial intelligence in Secure PRIVacy-preserving computing coNTinuum” project is to develop a platform composed of design and runtime management tools to seamlessly design, partition and operate Artificial Intelligence (AI) applications among the current plethora of cloud-based solutions and AI-based sensor devices (i.e., devices with intelligence and data processing capabilities), providing resource efficiency, performance, data privacy, and security guarantees.

The aim of this deliverable is to review the state-of-the-art in techniques used in the development and operation of AI applications in computing continua and the related technologies. The deliverable starts with an overview of AI applications and edge computing market trends. It examines the standardisation and open-source landscape, where many of the emerging standards are mainly focused on the network layer, also highlighting opportunities for open-source community involvement to expand technology availability. Then, the deliverable provides a background on AI applications design, also considering some advanced design trends (e.g., Network Architecture Search, Federated Learning, Deep Neural Networks partitioning) introducing the software development frameworks and hardware solutions which allow to run such applications in a computing continuum.

We then report an overview of the state-of-the-art of the solutions offered by the main cloud market players and the European providers. We also extensively discuss existing solutions for applications deployment, monitoring, runtime management, and scheduling considering the emerging Function as a Service paradigm.

In the last part of the deliverable, we report an overview of the performance modelling solutions, security, and privacy problems for AI applications in edge environments. It is found that some areas, such as component placement and design space exploration with privacy preservation and performance guarantees, are fairly under-developed, since solutions tailored for AI applications are lacking, and thus offer an opportunity for innovation.

Table of Contents

1.	Introduction	11
1.1	Scope of the document	11
1.2	Target audience	11
1.3	Structure of Document	11
2.	AI Applications in Computing Continua	13
2.1	Definition and terminology	13
2.1.1	AI Basic Concepts	13
2.1.2	OpenFog reference Architecture	18
2.2	Edge Computing Standardisation: State of Play	20
2.2.1	3GPP and Edge Computing	20
2.2.2	ETSI MEC	22
2.2.3	Edge Computing Security	26
2.2.4	Securing Artificial Intelligence	31
2.3	Importance of AI@EDGE for the Business	32
3.	AI technologies	34
3.1	Overview	34
3.2	Mainstream Frameworks for AI Development in Computing Continua and Edge Devices	36
3.2.1	Tiny Machine Learning	36
3.2.2	General Purpose Deep Learning Frameworks	40
3.2.3	Deep Learning Frameworks for Embedded Devices	43
3.2.4	Vendor-specific Deep Learning Frameworks	49
3.3	PyCOMPSs	54
3.4	Cloud-based solutions	54
3.4.1	Amazon Web Services	55
3.4.2	Microsoft Azure	58
3.4.3	Google Cloud Platform	59
3.4.4	IBM	60
3.4.5	Oracle	61
3.4.6	CloudSigma	62
3.4.7	1&1 Ionos	62
3.4.8	Scaleway	63
3.4.9	OVHcloud	63
3.4.10	Exoscale	64
3.4.11	Fuga	64
3.4.12	Linode	65
3.4.13	Vultr	65
3.4.14	Open Telekom Cloud	65
3.4.15	FaaS	67
3.5	Technologies and Solutions for Edge Intelligence	68
3.5.1	General Purpose Arm SoCs	68
3.5.2	Hardware Accelerated SoCs & SoMs	76
3.5.3	Hardware Accelerated Microcontrollers	88

3.6	Smartwatches and bands in stroke care	91
3.7	Cameras for Agriculture 4.0	93
3.7.1	Camera sensors for vine plant diseases and pests detection	94
3.7.2	Diseases detection on vine leaves	95
3.7.3	Grape detection	96
3.7.4	Vine canopy volume estimation	97
3.8	Drones Image Acquisition Technologies for Inspection and Maintenance	97
3.8.1	Drone components	97
3.8.2	Flight Controllers and Flight stacks	98
3.8.3	Companion Computers	98
3.9	Gaps filled by AI-SPRINT	99
4.	AI Application Advanced Design	100
4.1	DL models partitioning	100
4.2	Network Architecture Search	102
4.2.1	NAS dimensions	103
4.2.2	One-shot models	106
4.2.3	HardCoRe-NAS: Hard Constrained differentiable NAS	108
4.2.4	CARS: Continuous Evolution for Efficient NAS	110
4.2.5	Pareto-Frontier-aware Neural Architecture Generation for Diverse Budgets	112
4.2.6	Weak NAS predictors	113
4.2.7	FBnetV2: Differentiable NAS for Spatial and Channel Dimensions	115
4.3	Federated Learning and Privacy Preservation	117
4.3.1	Reference architecture	118
4.3.2	Data view	119
4.3.3	Machine Learning View	121
4.3.4	Implementations	123
4.4	AI Applications Performance Modelling	125
4.5	Edge Applications Components Placement and Design Exploration	128
4.6	Gaps filled by AI-SPRINT	130
5.	Runtime Frameworks for Computing Continua management	131
5.1	Virtualization, Containers and Containers Orchestrators	131
5.2	Deployment Technologies and Solutions	132
5.3	Monitoring	133
5.4	Runtime Management	136
5.5	Accelerated Clusters for AI	139
5.6	Gaps filled by AI-SPRINT	142
6.	Security and Privacy for Edge AI applications	144
6.1	Trusted Execution Environment	144
6.2	Patch management	145
6.3	Secure Boot	146

6.4	Network Security and Attacks in Edge Systems	149
6.5	Privacy issues in Edge systems for AI applications	151
6.6	Gaps filled by AI-SPRINT	153
7.	Conclusions	154
	References	155

List of Figures

Figure 2.1	- Example of image classification with Machine Learning	14
Figure 2.2	- Unsupervised learning used to project features in a new space	15
Figure 2.3	- Example of image classification with learned feature projection	15
Figure 2.4	- Example of image classification with learned feature representation and classifier	15
Figure 2.5	- Example of hierarchical feature representation	15
Figure 2.6	- Example of CNN architecture, specifically, the famous VGG16 architecture [Simonyan2014]	16
Figure 2.7	- ML model lifecycle	17
Figure 2.8	- AI-SPRINT Reference Architecture	19
Figure 2.9	- Edge landscape at the 5G network layer	25
Figure 2.10	- Positioning of EdgeGallery	26
Figure 2.11	- European Edge Market Growth. Source: IDC Edge Spending Guide, V2 2020 (January 2021)	33
Figure 3.1	- Percentage of information handled by micro-CNNs [Gousev2020]	38
Figure 3.2	- TinyML: holistic hardware–system–software co-design	39
Figure 3.3	- Workflow to deploy machine intelligence on edge devices with TensorFlow Lite	43
Figure 3.4	- TensorFlow Micro architecture	45
Figure 3.5	- PyTorch Mobile: workflow for mobile deployment	47
Figure 3.6	- TVM optimizing compiler framework workflow	49
Figure 3.7	- Intel OpenVINO workflow	49
Figure 3.8	- NVIDIA TensorRT accelerates every inference platform	51
Figure 3.9	- NVIDIA TensorRT workflow	51
Figure 3.10	- X-CUBE-AI core engine	53
Figure 3.11	- Green Grass Architecture	56
Figure 3.12	- Linpack benchmark of Raspberry Pi models	75
Figure 3.13	- UnixBench benchmark comparing Raspberry Pi and ODROID models	76
Figure 3.14	- NVIDIA® Jetson™ Board Support Package (BSP) architecture	78
Figure 3.15	- Internal architecture of NVDLA core	79
Figure 3.16	- RK3399 - High End MPU	80
Figure 3.17	- RK1808 - Mid Range MPU	81
Figure 3.18	- RV1109 - Entry Level MPU (new)	81
Figure 3.19	- NPU block diagram	82
Figure 3.20	- RZ/V2M Technical specifications	83
Figure 3.21	- RZ/V2M ONNX model import pipeline	83
Figure 3.22	- Intel® Movidius™ Neural Compute SDK tools	84
Figure 3.23	- Basic workflow to create a model for the Edge TPU	87
Figure 3.24	- Overview of the microcontroller families that support CubeAI	89
Figure 3.25	- Workflow to import models from DL frameworks and ONNX	89
Figure 3.26	- Espressif's ESP32-S3 Wi-Fi + Bluetooth LE SoC	91
Figure 3.27	- Esca detection by proximal sensing	96
Figure 4.1	- Overview of the DDNN architecture [Teerapittayanon2017]	101

Figure 4.2 - Graph model of JointDNN approach [Eshratifar2021]	102
Figure 4.3 - Overview of the workflow used by neural architecture methods [Elsken2019-2]	103
Figure 4.4 - Cells Architecture	104
Figure 4.5 - Simple example of a one-shot model	106
Figure 4.6 - One-shot models trained with constant dropout rates	107
Figure 4.7 - HardCoRe-NAS search space [Nayman2021]	109
Figure 4.8 - Recap of NAG algorithm	113
Figure 4.9 - t-SNE visualization of NAS-bench-201 on CIFAR100. Accuracies are encoded by color [Wu2021]	114
Figure 4.10 - Channel Search Challenges	116
Figure 4.11 - Spatial Search Challenges	117
Figure 4.12 - Federated Learning Framework	118
Figure 4.13 - Roles of Federated ML users	119
Figure 4.14 - Horizontal Federated Machine Learning [Yang2019]	120
Figure 4.15 - Vertical Federated Machine Learning [Yang2019]	120
Figure 4.16 - Federated Transfer Learning [Yang2019]	121
Figure 4.17 - Hierarchical framework of Federated Machine Learning	121
Figure 4.18 - Common concerns of Federated Machine Learning	122
Figure 5.1 - InfluxDB architecture	135
Figure 5.2 - Telegraph Processing	136
Figure 5.3 - Typical architecture of remote GPU virtualization frameworks	141
Figure 6.1 - Summary of threats, threat level and phase or location of occurrence [Bashun2013]	148
Figure 6.2 - Schema of a GAN-based Collaborative Learning attack [Hitaj2017]	152

List of Tables

Table 2.1 - Study and Work items in SA (Source: 3GPP)	21
Table 2.2 - Current 3GPP Specifications on Edge Computing (Source: 3GPP)	21
Table 2.3 - Standards for synergised architecture (Source: ETSI and 3GPP MRP Webinar Series)	24
Table 2.4 - Standards relevant for edge computing security (Source: ETSI)	30
Table 2.5 - Security Support Evolutions (Source: ETSI)	31
Table 3.1 - Positioning of TinyML with respect to cloud and edge AI	37
Table 3.2 - Cloud Provider comparison according to AI services and edge system support capabilities	66
Table 3.3 - Raspberry Pi 4 B	69
Table 3.4 - Raspberry Pi Zero W	70
Table 3.5 - ODROID-HC4	71
Table 3.6 - ODROID-H2+	72
Table 3.7 - ODROID-N2+	73
Table 3.8 - ODROID-XU4	74
Table 3.9 - Jetson family: technical specifications	77
Table 3.10 - Comparison of Rockchip SoCs	80
Table 3.11 - Dev Board	85
Table 3.12 - USB Accelerator	86
Table 3.13 - Dev Board Mini	87
Table 3.14 - Common smartwatches or bands on the market (adapted from [6])	93
Table 3.15 - Smartstriker	94
Table 3.16 - Bilberry	94
Table 3.17 - See&Spray	95
Table 3.18 - SmartSprayer	95

1. Introduction

1.1 Scope of the document

Artificial Intelligence (AI) and edge computing have recently emerged as major trends in the ICT industry. However, the heterogeneity of the technologies and software development solutions in use are evolving quickly and are still a challenge for researchers and practitioners. The aim of the AI-SPRINT “Artificial intelligence in Secure PRIVacy-preserving computing coNTinuum” project is to develop a platform composed of design and runtime management tools to seamlessly design, partition and operate AI applications among the current plethora of cloud-based solutions and AI-based sensor devices (i.e., devices with intelligence and data processing capabilities), providing resource efficiency, performance, data privacy, and security guarantees.

The aim of this deliverable is to review the state-of-the-art in techniques used in the development and operation of AI applications in computing continua and the related technologies. The deliverable starts with an overview of AI applications and edge computing market trends and standardisation work for the edge-cloud ecosystem, the enabling role of AI, as well as open-source initiatives and security challenges for both for edge computing and AI. Then, the deliverable provides a background on AI applications design, considering also some advanced design trends (e.g., Network Architecture Search, Federated Learning, Deep Neural Networks partitioning) introducing the software development frameworks and hardware solutions which allow to run such applications in a computing continuum. The goal of this deliverable is therefore to support this investigation with an analysis on the scientific and technical state-of-the-art of the solutions offered by the main cloud market players and the European providers. We also extensively discuss existing solutions for enterprise applications deployment, monitoring, runtime management, and scheduling considering the emerging Function as a Service paradigm. Furthermore, given the general objective of AI-SPRINT to support non-functional requirements of AI applications (e.g., application execution time, AI models accuracy, security, and privacy preservation), we also survey performance modelling solutions, security, and privacy technologies for AI applications in edge environments. It is found that some areas, such as component placement and design space exploration with privacy preservation and performance guarantees, are fairly under-developed, since solutions tailored for AI applications are lacking, and thus offer an opportunity for innovation.

1.2 Target audience

The *State of The Art Analysis* deliverable is intended for internal use to provide all partners of the consortium a snapshot of the current technologies and research literature contributions, and a shared terminology, although it is publicly available. The target audience is the AI-SPRINT technical team including all partners involved in the delivery of work packages 2,3 and 4 but it also serves as reference for the developers of the three use cases of the project.

1.3 Structure of Document

This document is organised in seven chapters which cover the following topics:

- Chapter 2 provides basic concepts, introducing definitions related to AI applications development and edge systems, also providing an overview of the standardization initiatives and market trends.
- Chapter 3 describes the state-of-the-art of the mainstream frameworks adopted for the development of AI application in computing continua and the solutions currently offered by the cloud big players

and by the main European cloud providers. Furthermore, the chapter overviews the hardware of intelligent sensors currently offered in the market, including the ones specific for the development of the AI-SPRINT use cases.

- Chapter 4 continues the overview by revising advanced solutions for the design of AI applications that concern deep learning models partitioning, network architecture search, federated learning, AI application performance modelling, and component placement in computing continua.
- Chapter 5 analyses runtime management frameworks and the related technologies and provides an overview of virtualization and cloud/container orchestration technologies, deployment and monitoring, and scheduling solutions for clusters adopting GPU accelerators.
- Chapter 6 is devoted to security and privacy issues for AI applications.
- Chapter 7, finally, concludes this deliverable and overviews the AI-SPRINT development path.

Out of the present investigation, our main achievement is the identification of the most relevant baselines to the AI-SPRINT work and of open problems in the area touched by the project. The reader is invited to consult deliverable *D1.2 - Requirement analysis* for a detailed requirement analysis that used the output of this survey to define the AI-SPRINT goals.

2. AI Applications in Computing Continua

Chapter 2 provides an overview on some basic concepts, introducing definitions related to AI applications development and edge systems, and also providing an overview of the standardization initiatives, which are mostly focused on mobile networks but with a growing expansion of edge-related initiative, as well as market trends.

In particular, some definitions on AI applications and the computing continua are provided in Section 2.1. Section 2.2 introduces the main standardization initiatives in edge computing, while Section 3.3 outlines the importance of deploying AI applications at the edge.

2.1 Definition and terminology

2.1.1 AI Basic Concepts

Artificial Intelligence (AI) is a branch of computer science aimed at developing *intelligent* systems, i.e., providing machines with the problem-solving and decision-making capabilities typical of a human mind. Nowadays, computers are still far from reaching a human-like intelligence, but AI is increasingly integrated in our devices, providing support in our daily tasks. What makes a system intelligent is the ability of perceiving the world, e.g., through sensors, and *learning* from the perceived data.

Machine Learning (ML) is a subfield of AI in charge of the research and the development of learning algorithms, through which systems can learn from examples and experience. Reporting the definition given in the book by T. Mitchell (1997) [Mitchell1997], “a computer program is said to learn from experience E with respect to some class of tasks T and a performance measure P , if its performance at task T , as measured by P , improves with experience E ”. Experience, which comes in the form of a huge amount of data commonly referred to as *training set*, is thus exploited by learning algorithms to build mathematical models to accomplish specific tasks. Depending on the kind of information contained in the observations, it is possible to identify different types of learning, commonly classified into the following paradigms:

- **Supervised learning:** each observation in the training set is a tuple $\langle \mathbf{x}, t \rangle$, where \mathbf{x} is the features vector, i.e., the input of the model, and t is the associated target or label, which is the desired expected output coming from some unknown distribution f , i.e., $t = f(\mathbf{x})$. In a supervised setting the system finds a good approximation of the mapping function f , by learning structures in the data. It is now clear that the name of this paradigm comes from the need of supervision, the targets, during the learning process. Collecting targets is one of the stages in the learning pipeline which requires the most effort and it usually requires intensive manual labelling of the collected examples. Supervised learning tasks are typically categorized, depending on the type of the available targets, into the following problems:

- *Classification:* in this kind of task the targets t are discrete. Indeed, the goal is to learn the function that assigns each input to one of a finite number of classes K , i.e.:

$$f(\mathbf{x}): \mathbb{R}^n \rightarrow \{1, \dots, K\}$$

This is the case, among others, of image classification, e.g., handwritten digits recognition, or text classification. Depending on the number of classes, the classification problem can be *binary* or *multi-class*, i.e., $K = 2$ or $K > 2$, respectively.

- *Regression:* in this case the targets t are continuous variables, i.e.:

$$f(\mathbf{x}): \mathbb{R}^n \rightarrow \mathbb{R}$$

Examples of regression problems are the prediction of the temperature, or of the selling price of a specific product.

Several algorithms have been successfully applied to solve supervised learning tasks, like Logistic Regression, K-Nearest Neighbours (K-NN), Support vector machine (SVM), Decision Tree, and the game-changer Artificial Neural Networks (ANNs) [Mitchell1997, Bishop2006, Goodfellow2016].

- **Unsupervised learning:** in an unsupervised setting the targets associated to the input features are not available during learning. In this case, the goal is to learn the correlation in the input data rather than the input-output mapping function as in the supervised case. Algorithms following this learning paradigm are typically employed to divide input data into groups, task known as clustering (e.g., K-means or DBSCAN), or to determine the distribution of data within the input space, i.e., density estimation [Bishop2006], or to project the data into a lower-dimensional space (e.g., Principal Component Analysis - PCA [Goodfellow2016]).
- **Reinforcement learning:** in this setting there is a continuous interaction between the system and the environment, and the goal is to find suitable actions based on the observations. Differently from the supervised case, no examples of desired output are provided during learning but, instead, they are discovered through a process of trials and errors. The system performs an action on the environment and receives an observation plus a reward. The goal is to find an optimal policy that maximizes the cumulative reward [Sutton1998].

The main focus of AI-SPRINT is on Supervised Learning and thus, in the following of this report most of the examples and literature will be based on this paradigm.

The performance of a machine learning algorithm strictly depends on the representation of the input data, or more in general, on the representation of the information contained in the observations defined in the form of features. Classical Machine Learning provides algorithms to learn patterns detectable by looking at these features in order to perform some tasks, but it does not deal with the representation of such features which is named Feature Engineering. Figure 2.1 depicts the scheme of a classification task from a classic ML perspective. While the classification algorithm is learned, the features given to the algorithm are manually designed. The performance of the classifier heavily depends on the ability of designing the right set of features.



Figure 2.1 - Example of image classification with Machine Learning. Only the classifier is learned. Features are manually designed

It goes without saying that the quality of the learned classifier heavily depends on the ability of designing the right set of features for the given tasks. However, handcrafting features can be very time-consuming and for many tasks it is difficult to know which features should be extracted from data. A possible solution comes from a learning paradigm which sits between the unsupervised and supervised learning ones, which can be defined as semi-supervised learning. In this setting, unlabelled data is exploited by some unsupervised algorithm to learn a new representation of the input features (see Figure 2.2), which is thus projected to a new space where similar examples have similar representations. Then, labelled data is used to learn the final classifier (regressor) in a supervised fashion. The above classification example now becomes the one depicted in Figure 2.3.

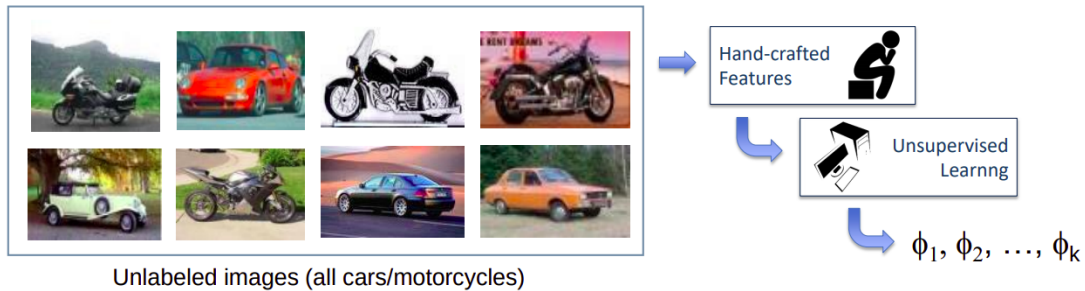


Figure 2.2 - Unsupervised learning used to project features in a new space

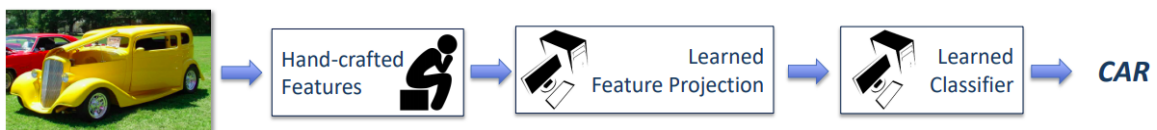


Figure 2.3 - Example of image classification with learned feature projection

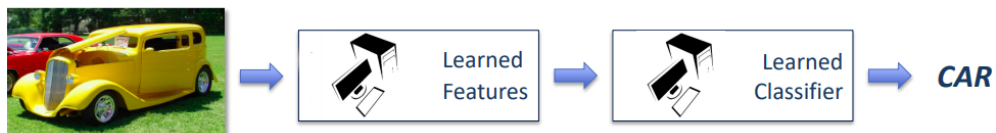


Figure 2.4 - Example of image classification with learned feature representation and classifier

Beside the improvements gained by transforming features in a new representation, human features designing is still needed as a pre-processing step on data. Deep Learning (DL) emerges in this context as a new field of ML to provide a way to automatically learn a feature representation directly from data, without relying on hand-crafted features (see Figure 2.4). DL can be seen as the attempt of putting together ML and representation learning, by jointly learning good general features and the final predictor. Representations are learned at different levels of abstractions by stacking multiple Neural Network (NN) layers, building complex representations on top of simpler ones. Thus, the idea of DL is to learn structured feature transforms which provide a hierarchical representation of input features (see Figure 2.5).

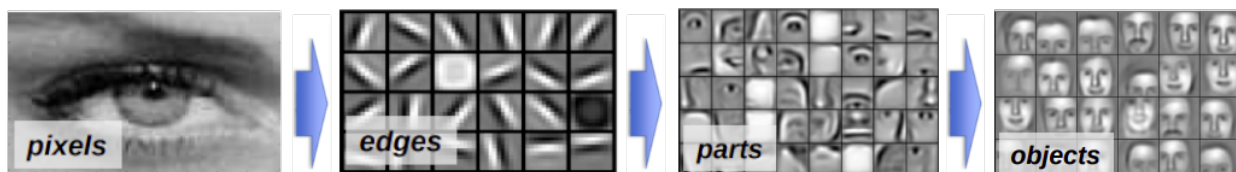


Figure 2.5 - Example of hierarchical feature representation

Deep *feedforward neural networks*, or *Multi-Layer Perceptrons (MLPs)*, are the baseline deep learning models defining a function $f(\mathbf{x}; \boldsymbol{\vartheta})$ in which the information flows from input, through the intermediate layers, and finally to the output, where $\boldsymbol{\vartheta}$ are the learnable parameters. *Convolutional Neural Networks¹ (CNNs)* [Goodfellow2016] are feedforward networks specialized in processing data with a grid-like topology, such as image data, which can be thought of as a 2D grid of pixels. What characterizes a CNN is the use of convolutional layers, in which shared parameters are learned to detect patterns in the data. The typical architecture of a CNN is depicted in Figure 2.6, which it is possible to notice the scheme depicted above in Figure 2.4. Indeed,

¹ <https://cs231n.github.io/convolutional-networks/>

there is a *feature extraction* network, composed of alternated convolution, activation, and *pooling* layers, followed by a *fully-connected* network, where every neuron of each layer is connected to every neuron of adjacent one, for the final prediction. The output produced by a convolutional layer is typically called *feature map*, while the learned parameters are the *kernels*, or *filters*. CNNs have been successfully employed in many computer vision tasks like, but not limited to image recognition, image segmentation, and object detection.

Other types of DL architecture are the *Recurrent Neural Networks (RNNs)*, which are an extension of the feedforward neural networks, enhanced with feedback connections to process sequential data, thus introducing the concept of *memory* during learning. Among the recurrent architecture are *Long Short-Term Memory (LSTM)* [Hochreiter1997] networks, which are capable of modelling very long-term dependencies, which is instead a limitation of the standard RNNs. This kind of architecture has become the first choice for processing sequential data in natural language related tasks as language modelling and machine translation, but also for other tasks like video analysis or time series anomaly detection.

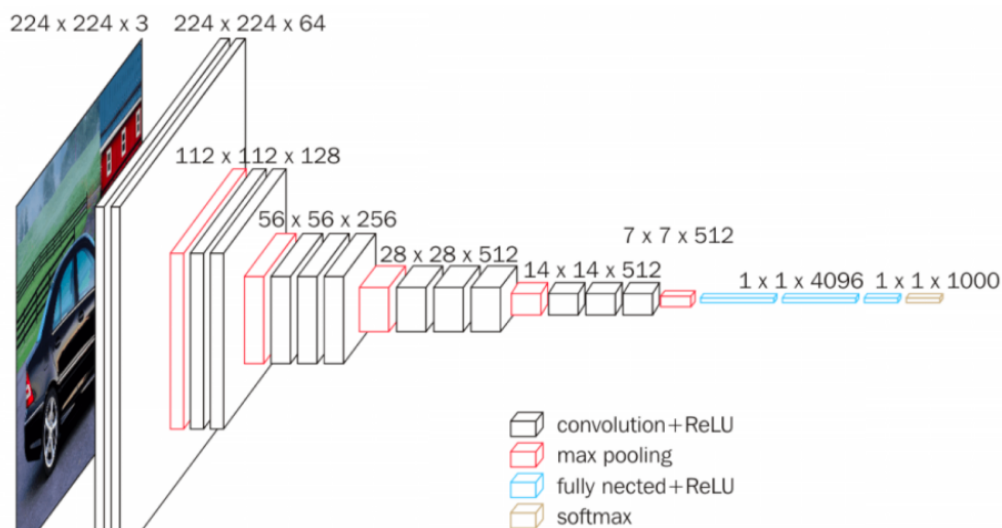


Figure 2.6 - Example of CNN architecture², specifically, the famous VGG16 architecture [Simonyan2014]

DL and, more in general, ML models, are learned during the so-called *training* phase, in which a huge amount of data collected into the so-called *training set* is used to find the mapping function between given input and output, in the case of supervised learning, or a new representation of data, in the case of unsupervised learning. The goal is to find the values of model *parameters* which minimize a certain error E , commonly known as *loss function* L . This is done by using specific *optimization* algorithms which minimizes a real function. Training in ANNs is usually performed by using iterative, *gradient-based* optimization algorithms, in which the model parameters are updated following the direction given by the gradient of the loss function given a *batch*, i.e., a sub-group, of examples from input data. The specific gradient-based algorithms are usually improvements of the well-known *stochastic gradient descent (SGD)*. Parameters of the optimization algorithm, and the algorithm itself, are part of the so-called training *hyperparameters*. In order to compute the gradients, the *backpropagation* algorithm is employed, which allows the information given by the loss to flow backwards through the network [Rumelhart1986].

² Figure source: <https://neurohive.io/en/popular-networks/vgg16/>

A model is learned using the training set. However, the real goal of training is to learn a model which is able to *generalize* on unseen data. When this does not happen, i.e., the learned model perfectly fits on training data, but it has very poor performance on unseen samples, the model is said to *overfit*. Overfitting is one of the main challenges encountered during training. One of the solutions to control overfitting, is to put aside some samples of the training data, which will be used as a *test set* for a very final evaluation of the model. If the amount of available data is large enough, it is possible to extract an additional non-overlapping *validation set*, which will be used to control overfitting during learning, eventually stopping the training when the model starts to overfit (*early stopping*). There are other techniques to avoid overfitting, to which we usually refer to as *regularization* methods [Goodfellow2016].

Once the model has been trained, it can be used to make predictions given new data. This process is often called *inference* and it is typically faster than training, since no optimization is required (for instance, no gradient computation) and only a forward pass through the model is needed.

Model training is only a single phase of the overall process through which a ML application is developed. A complete model lifecycle is depicted in Figure 2.7. It is a cyclical process which begins with the collection of raw data into the dataset used for model training (data ingestion). Then, data is prepared for training, i.e., it is cleaned (it can contain missing values, duplicates, invalid or noisy data) and pre-processed by applying specific transformations (data preparation). Thus, model training can be performed. In this phase the ML model is selected and learned by using the training set. Once the model has been trained, it is validated on the test set, to determine the performance of the model based on the project requirements (model validation). Finally, the model is integrated into real applications (model deployment). Typically, during the deployment process it is ensured fast inference by using specialized hardware and model optimizations (see Sections 3 and 4). What closes the cycle is the possibility of re-training the model in the case new data is available or the model becomes to have performance degradation through time. We refer to this behaviour as *model drift*, which can be divided into two main categories: *concept drift*, i.e., the statistical properties of the target variable change, and *data drift*, i.e., the statistical properties of data change (e.g., seasonal patterns).

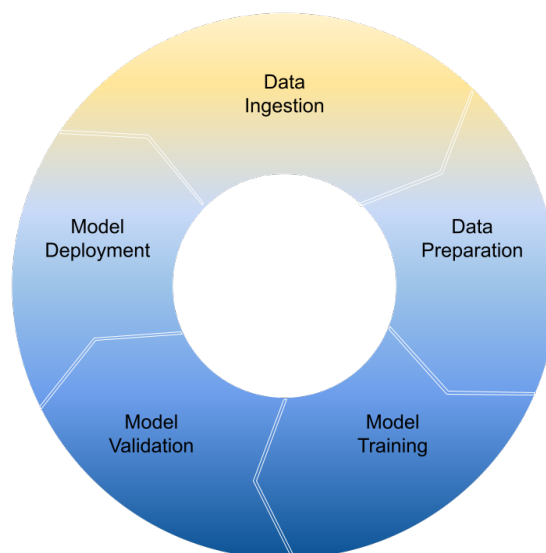


Figure 2.7 - ML model lifecycle

Most ML and DL algorithms depend on many hyperparameters, which have an impact on the performance of the learned model. Examples of hyperparameters are the choice of the correct optimization algorithm and

the related parameters, like the *learning rate*, *momentum*, or *batch size*, but also the ones related to the model itself, for instance, the k in the k -nearest neighbour algorithm, the number of convolutional layers, the number of neurons in a fully-connected layer, the activation function, and others. Choosing the right set of hyperparameters, also called *hyperparameter tuning*, is a challenging issue. The first approach is to choose the hyperparameters manually, a process that requires a good understanding of the considered task and good ability to analyse the results. However, it can result in a very expensive procedure, in terms of time and human effort, which can be increasingly challenging as the space of the possible choices for the hyperparameters grows. Manual hyperparameters tuning can be a good choice when we start from a good initialization, exploiting, for instance, the knowledge discovered by others having already experienced the use of the considered architecture in the specific task.

The process of selecting the right hyperparameters can be seen as an optimization algorithm, in which we are trying to select those hyperparameters that optimize an objective function, typically the validation error. Thus, specific optimization algorithms can be employed. When the number of hyperparameters is limited, one of the common algorithms is *grid search*. In this algorithm, the possible values for each hyperparameter must be declared and then, a model is trained for every joint specification of hyperparameters. The experiment that performs better on the validation set, as measured by the selected metric, is the one corresponding to the best hyperparameters. The main problem with such an algorithm is that the computational cost grows exponentially with the number of parameters.

In this context are gaining ground automated ML (AutoML) tools, which other than automating the main stages of a ML process like the data preparation or feature engineering, allows also to optimize the hyperparameters of the learning algorithms. Also, Network Architecture Search (NAS) techniques are part of this scene, allowing to automatically design model architectures (see Section 4.2).

2.1.2 OpenFog reference Architecture

The applications we will consider in AI-SPRINT are enterprise applications, mostly written in Python, that make intensive use of AI technologies and are based on multiple components running across a computing continuum. Specifically, components can be allocated on the cloud or on edge servers and AI-enabled sensors (i.e., sensors with some computing capabilities able to perform at least AI model inference), following the OpenFog Reference Architecture (RA) [OpenFog2017], adopted as an IEEE standard since 2018 [IEEEstandard2018]. The selection of AI-enabled edge devices is motivated by considerations detailed in the following. In enterprise systems, “edge” is the vast space or intermediary between endpoints and the core (defined as the epicenter of computing and storage capabilities of a company) [Morales2020]. It may include far and remote locations, as well as the shop floor of a manufacturing company, a retail store, a remote office, or a branch office, up to a colocation datacentre that is used as secondary or auxiliary resource beyond the central company datacentre. Every use case has a different way of leveraging a technology and requires a detailed business case. However, there are four main common aspects that show the impacts, and the benefits of enabling AI at the edge:

- **Latency:** performing AI inferencing at the edge enables low-latency use cases, with the possibility to take decisions on the gathered data without the need for information to travel in a round trip to a far datacentre or cloud location. According to the type of Edge, latency can vary from real time (zero-latency) to single digit milliseconds to the order of tens of milliseconds.
- **Connectivity:** the ability to continuously perform tasks and processes in case of intermittent connectivity to the cloud or the core location.
- **Bandwidth cost:** reducing data travelling to the core/cloud. Especially in the case of live HD images or video signals, the ability to perform AI at the edge may decide the viability of the business case.

- Privacy: several use cases (consider, e.g., those related to the processing of images coming from security cameras) need to process sensitive data, whose deployment in the cloud violates the prescribed security and privacy constraints. Performing the initial phases of AI inference (or training in case of federated learning, see also Section 6.5) at the edge, and offloading to the cloud as the last layers of networks (thus sending only pre-processed data, from which it is not possible to recover any sensitive information) supports the development of privacy-aware applications.

Based on the upon observations, according to the OpenFog RA, AI-SPRINT will consider a computing continuum structured according to different layers (or tiers), whose number may change according to the characteristics of the problem we are tackling. In particular, the requirements that have the strongest impact on determining the number of layers are related to: (i) the amount of work that should per performed on each layer, (ii) the number of available sensors and the computing capabilities of resources at each layer, (iii) the requirements about latency, reliability, and availability of the different nodes.

The AI-SPRINT reference architecture is reported in Figure 2.8. It includes four main layers, which can be further subdivided according to the use case characteristics. The initial layer of the fog computing paradigm hierarchy, denoted as Layer 0, includes IoT and AI-enabled edge sensors. These devices are mainly devoted to data collection; they can be used for data processing, provided that they have enough computation capabilities, especially when latency reduction is crucial for the applications (e.g., when operations should be performed at millisecond or sub-millisecond granularity). Layer 1 is characterized by edge servers, PCs and/or Raspberry Pi devices. Some devices (e.g., smartphones) will be considered at one of these layers according to the specific application scenario. For the sake of simplicity, if not differently specified, we normally identify both these layers with the term *edge*. The last layers of the fog computing paradigm hierarchy include the cloud. This can be further divided according to the type of resources that are exploited to deploy the application components: the assumed cloud deployment environments will be FaaS and IaaS, either public or private. Different cloud providers will be considered to avoid lock-in and to minimize costs, with components deployed at runtime, possibly, on multiple clouds.

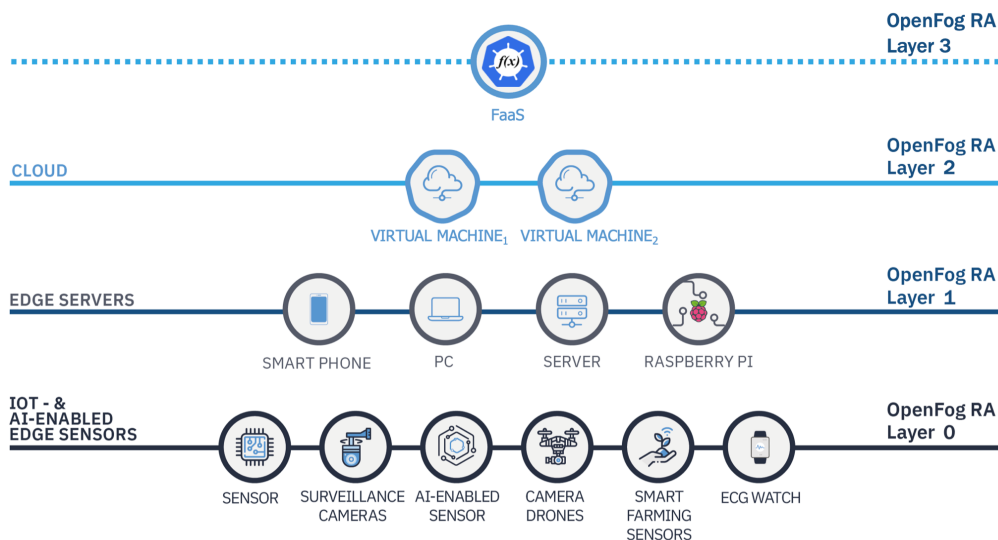


Figure 2.8 - AI-SPRINT Reference Architecture

AI-SPRINT will consider multiple edge computing paradigms: micro-clouds operated by the telecommunications operator (a.k.a. Mobile Edge Computing, MEC, as specified in the 5G standards), public cloudlets located at business premises, and private cloudlets located at the end-user premises. Edge devices will be based on general purpose processor and Arm architectures (see also Section 3.5), possibly accelerated

via GPUs or TPUs to perform real-time inference when needed; the AI application will be deployed via Docker containers (see Section 3.4) which can handle accelerators also on low-end architectures such as Arm processors.

2.2 Edge Computing Standardisation: State of Play

2.2.1 3GPP and Edge Computing

5G will help technologies like edge computing and artificial intelligence to harmonise and enable them to work together much more efficiently. AI and edge can both serve as building blocks but in diverse ways. From the network layer perspective, AI can further optimise edge computing applications. At the application layer, edge can be a building block for AI, e.g., by offloading limited capabilities from the device to the network.

Edge computing environments are a complex multi-supplier, multi-stakeholder ecosystem of devices, people, and things. This complexity makes coordination and cooperation across the diverse stakeholder group, with growing consensus on the need to work together to educate consumers and businesses and support the coordination of on-going standardisation³.

The ICT Standardisation 2021 Rolling Plan⁴ does not mention the on-going standardisation work on edge computing in 3GPP, where it is a basic functionality of the 5G core network (5GC). Standardisation work on edge started in Release 15, it is foundational in Release 17 (expected late 2021/early 2022) and is expected to feature in future releases and as we gradually move beyond 5G. Edge computing affects all parts of the 5GC with work taking place in the following working groups of SA (Service and System Aspects)⁵.

- SA1 (Services)⁶: Requirements for 5GC, including edge computing.
- SA2 (Architecture)⁷: Enhancements for edge computing in 5GC.
- SA3 (Security)⁸: Security for edge computing in 5GC.
- SA4 (Multimedia Codecs, systems and services)⁹: Streaming architecture extensions for edge computing in 5GC.
- SA5 (Management, orchestration and charging)¹⁰: Management and charging for edge computing.
- SA6 (Application enablement and critical communication applications for vertical markets)¹¹: Architecture for edge applications.

³ See for example, the webinar takeaways of Edge Computing - Industry Vertical Viewpoints, <http://global5g.org/online-tool-standards-tracker/webinar-takeaways-edge-computing-industry-vertical-viewpoints>, and 3GPP Standards on Edge Computing, <http://global5g.org/online-tool-standards-tracker/webinar-takeaways-3gpp-standards-edge-computing>, April 2021.

⁴ https://ec.europa.eu/growth/content/2021-rolling-plan-ict-standardisation-released_en.

⁵ https://global5g.org/sites/default/files/2021-04-22-VerticalWS-EdgeComputing_GeorgMayer.pdf.

<https://www.3gpp.org/specifications-groups/sa-plenary/sa1-services/home>.

⁶ <https://www.3gpp.org/specifications-groups/sa-plenary/sa1-services/home>.

⁷ <https://www.3gpp.org/specifications-groups/sa-plenary/sa2-architecture/home>.

⁸ <https://www.3gpp.org/specifications-groups/sa-plenary/sa3-security/home>.

⁹ <https://www.3gpp.org/specifications-groups/sa-plenary/sa3-security/home>.

¹⁰ <https://www.3gpp.org/specifications-groups/sa-plenary/sa5-telecom-management/home>.

¹¹ <https://www.3gpp.org/specifications-groups/sa-plenary/sa6-mission-critical-applications/home>.

SA2 and SA6 are the primary two groups in 3GPP for edge computing. The table below summarises key areas of standardisation on edge in three SA groups from the April 2021 webinar, 5G Vertical User Workshop – Edge Computing. Insights from the webinar help fill gaps in the 2021 Rolling Plan¹².

Table 2.1 summarises the main study and work items in 3GPP SA2, SA5 and SA6 for Release 17, along with their level of advancement.

3GPP WG	Study /work item	Release	Progress (in April 2021)
SA2	Study on enhancement of support for Edge Computing in 5GC (FS_enh_EC; SP-200093)	17	100%
SA2	Enhancement of support for Edge Computing in 5G Core network (eEdge_5GC; SP-201107)	17	60%
SA6	Application layer architecture, and deployment scenarios (FS_EDGEAPP, EDGEAPP)	17	100%
	<ul style="list-style-type: none"> • Study on Application Architecture for enabling Edge Applications • Architecture for enabling Edge Applications 		95%
SA5	SA5: Management aspects on Edge Computing (FS_Edge_Mgt)		

Table 2.1 - Study and Work items in SA (Source: 3GPP)

Table 2.2 summarises the relevant specifications for edge computing.

Relevant specifications for edge computing	
TS 23.222 ¹³ (Rel 15)	Common API Framework for 3GPP Northbound APIs
TS 23.501 ¹⁴ (Rel-15)	System architecture for the 5G System (5GS)
TS 23.502 ¹⁵ (Rel-15)	Procedures for the 5G System (5GS)
TS 23.503 ¹⁶ (Rel-15)	Policy and charging control framework for the 5G System (5GS); Stage 2
TR 23.748 ¹⁷ (Rel-17)	Study on enhancement of support for Edge Computing in 5G Core network (5GC)
TS 23.548 ¹⁸ (Rel-17)	5G System Enhancements for Edge Computing; Stage 2
TS 23.558 ¹⁹ (Rel-17)	Architecture for enabling Edge Applications (EA)

Table 2.2 - Current 3GPP Specifications on Edge Computing (Source: 3GPP)

¹²https://global5g.org/sites/default/files/VerticalWebinar_SA2_update_Edge_Computing_PuneetJain.pdf; <https://www.youtube.com/watch?v=k3jvI5qeJZ4&t=117s>.

¹³ <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3337>.

¹⁴ <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3144>.

¹⁵ <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3145>.

¹⁶ <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3334>.

¹⁷ <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3622>.

¹⁸ <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3856>.

¹⁹ <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3723>.

In SA6, the edge enabler layer (3GPP EDGEAPP) gives support for application clients to locate, connect and switch to the most suitable application server on the edge²⁰. It enables application servers on the edge to use the underlying 3GPP network to provide the best possible services. EDGEAPP allows the integration of APIs enabling application developers to focus better on application features. The architecture allows flexible deployment models and business relationships, e.g., multiple edge computing service providers with advanced application features for application context relocation or service continuity, EES capability exposure/APIs, service-based architecture.

2.2.2 ETSI MEC

According to the 2021 ICT Standardisation Rolling Plan (March 2021) - (Chapter on cloud and edge computing), ETSI MEC provides IT and cloud computing capabilities within the access segments of network infrastructure in close proximity to users. It is developing a set of standardised APIs to enable MEC services. The access network gives application developers and content providers a service environment with ultra-low latency and high bandwidth with direct access to real-time network information that can be used by applications and services to offer context-related services.

Two sources help to update and expand on the Rolling Plan:

- ETSI White Paper #36 – Harmonising standards for edge computing – A synergised architecture leveraging ETSI ISG MEC and 3GPP specifications, July 2020²¹.
- Edge Computing: Industry Vertical Viewpoints, webinar April 2021²².

Architectures, Standards and Open Source

Standards for deploying cloud at the edge are required when there is a need for solutions involving multiple stakeholders, realising scales of economy, avoiding lock-in, and enabling multi-vendor solutions using best of breed components from each provider.

Examples of where standards are particularly required for edge deployments with mobile network operator (MNO) infrastructure include: common infrastructure capabilities, smart application placement, discovery of and optimal (re)-routing, service continuity, cloud applications designed to enhance the user experience by leveraging network services, edge federation across multiple MNOs.

As different standardisation activities cover various aspects and complement each other to a large extent, the ETSI white paper #36 proposes an architecture that supports diverse market-driven use cases and related requirements. It leverages existing standards to form a “synergised mobile edge cloud architecture” with common practices for developers, thereby creating a single application software module running on edge environments. The scope of the relevant standards groups (ETSI ISG MEC, 3GPP, GSMA and 5GAA) in relation to the architecture are summarised in Table 2.3.

²⁰

https://global5g.org/sites/default/files/5G%20Vertical%20User%20Webinar%20Series_3GPP%20SA6%20EDGEAPP_SA6%20Chair_SureshChitturi.pdf.

²¹ https://www.etsi.org/images/files/ETSIWhitePapers/ETSI_wp36_Harmonizing-standards-for-edge-computing.pdf.

²² <http://global5g.org/online-tool-standards-tracker/webinar-takeaways-edge-computing-industry-vertical-viewpoints>; <https://www.youtube.com/watch?v=ul55CDqAz7M&t=21s>.

Standards Body/Group/Association	Scope and Purpose
ETSI ISG MEC²³	<p>Open and standardised IT service environment, allowing 3rd-party (edge-unaware and edge aware) applications to be hosted at the edge of the mobile network, capable of exposing network and context information. It has specified:</p> <ul style="list-style-type: none"> ● Common and extensible application enablement framework for delivering services. ● Specific service-related APIs for information exposure and programmability. ● Management, orchestration and mobility-related APIs, facilitating the running of applications at the right location at the right time and ensuring service continuity. <p>ETSI MEC is studying MEC federations to enable the shared usage of MEC services and applications for multi-operator, multi-network and multi-vendor environments.</p> <p>Beyond the white paper, the April 2021 webinar highlighted that:</p> <ul style="list-style-type: none"> ● The fundamental MEC specifications are ready. ETSI MEC is recognised in the industry as the leading SDO for application enablement and edge computing. ● MEC is well positioned to satisfy the needs of an Operator Platform and help monetising the operator’s network using MEC APIs. ● MEC is flexible and extensible. ● The number of edge initiatives in the industry has considerably grown and they should leverage MEC to ensure common practices and tools for the developers. This will help to ensure adoption and accelerate time-to-market²⁴.
3GPP SA6	<ul style="list-style-type: none"> ● Defines architecture (EDGEAPP) to enable edge applications through the specification of an enabling layer easing communication between application clients deployed at the edge. ● EDGEAPP offers benefits for edge-aware applications through direct interaction with the device hosted Edge Enabler Client. The architecture also enables the CAPIF (Common API Framework²⁵) as a standardised means for providing and accessing APIs in the edge cloud.
3GPP SA2	<ul style="list-style-type: none"> ● Defines the architecture for mobile core networks including 5G. ● Defines how user traffic is routed to the appropriate application servers in the edge clouds. ● Provides means for application to provision traffic steering rules.

²³ <https://www.etsi.org/technologies/multi-access-edge-computing>.

²⁴

<https://global5g.org/sites/default/files/MEC%20for%203GPP%20MRP%20Webinar%20on%20Edge%20Computing%20for%20Industry%20Verticals.pdf>.

²⁵ <https://www.3gpp.org/common-api-framework-capif>.

3GPP SA5	<ul style="list-style-type: none"> • Specifies underlying life-cycle management of application servers in the edge cloud and charging aspects for edge services.
GSMA	<ul style="list-style-type: none"> • Focuses on setting requirements and implementing agreements, using applicable standards. • Specifying requirements and end-to-end high-level architecture for a unified Operator Platform to help operators make their assets and capabilities consistently available to developers and enterprises across networks and national boundaries. • Fosters involvement of both standards bodies and open-source communities. <ul style="list-style-type: none"> • Phase 1: Federating multiple operators’ edge computing infrastructure to give application providers access to global edge cloud to run distributed, low latency applications via a common set of APIs.
5GAA (5G Automotive Association)²⁶	<ul style="list-style-type: none"> • Defines requirements and implementation recommendations for cellular V2X applications using multi-access computing as per ETSI MEC specifications. <p>The 5GAA MEC4AUTO WG aims to demonstrate the use of multi-access edge computing (MEC) technology for automotive services, e.g., when two distinct automotive vendors can truly test at least three use cases involving two distinct MNOs using network infrastructure from two distinct vendors²⁷.</p>

Table 2.3 - Standards for synergised architecture (Source: ETSI and 3GPP MRP Webinar Series)

Figure 2.9 visualises the state of play for the edge industry application ecosystem, spanning the fusion of network and compute capabilities and standardised compute capabilities.

²⁶ <https://5gaa.org/>

²⁷ https://global5g.org/sites/default/files/5GAA_MEC4AUTO_Presentation_at_3GPP_MRP_Webinar_2021_04_20.pdf.

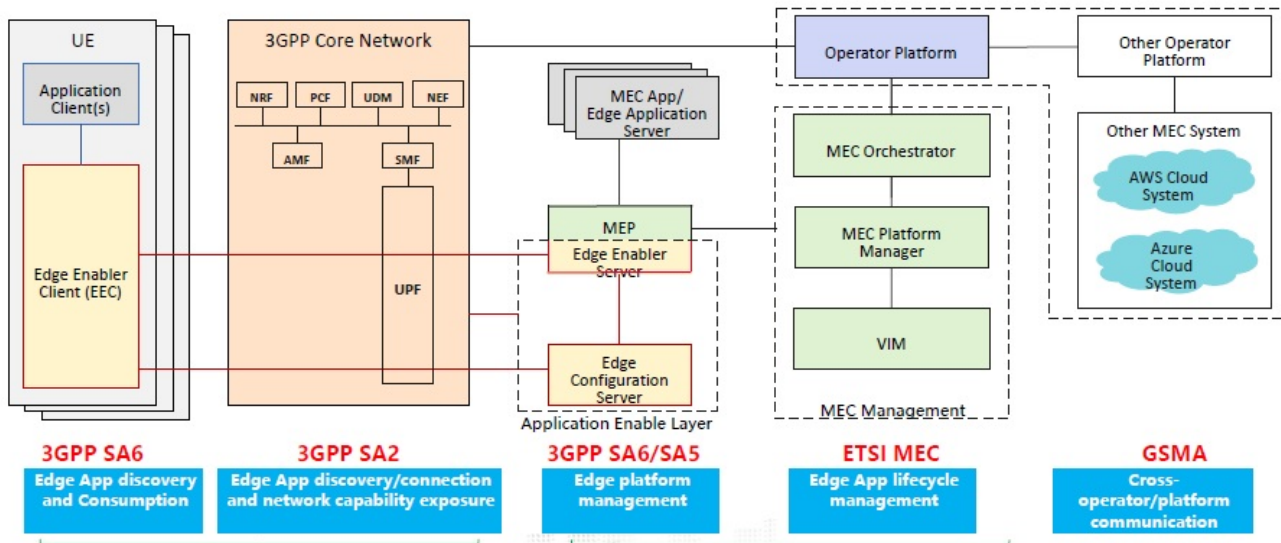


Figure 2.9 - Edge landscape at the 5G network layer

Open source plays a key role in shaping and accelerating deployments of edge clouds. Diverse open-source projects pertaining to the cloud may also apply to edge clouds, e.g., when offering high throughput, low latency, high availability, horizontal scalability. Other open-source initiatives are emerging to specifically support on-going standards for edge-cloud deployments in conjunction with mobile networks. Value propositions are expected to come from supporting developers and speeding up the time to deployment. Vertical specific open-source components can also enhance edge cloud deployments by offering microservices that can be used by application developers through APIs. Therefore, the involvement of open-source communities has the potential for additional developments supporting on-going standards, alignment, and technology enablement. It may also help update the landscape and pinpoint news areas for standardisation and position open-source initiatives within it.

One such open-source initiative is EdgeGallery, an initiative based in China that focuses on the network of edge computing and IIoT applications. The following summary stems from a webinar in April 2021 on edge computing and industry vertical viewpoints²⁸. Its aims at including, creating a unified platform for the ecosystem for verticals and operators; operator-led edge computing architecture and capability de-facto standards (e.g. 3GPP) and lowering the threshold for enterprise application deployment and building a B2B ecosystem. Figure 2.10 reflects its positioning statement.

²⁸ www.edgegallery.org; <https://www.youtube.com/watch?v=CovSM57JUyc>.

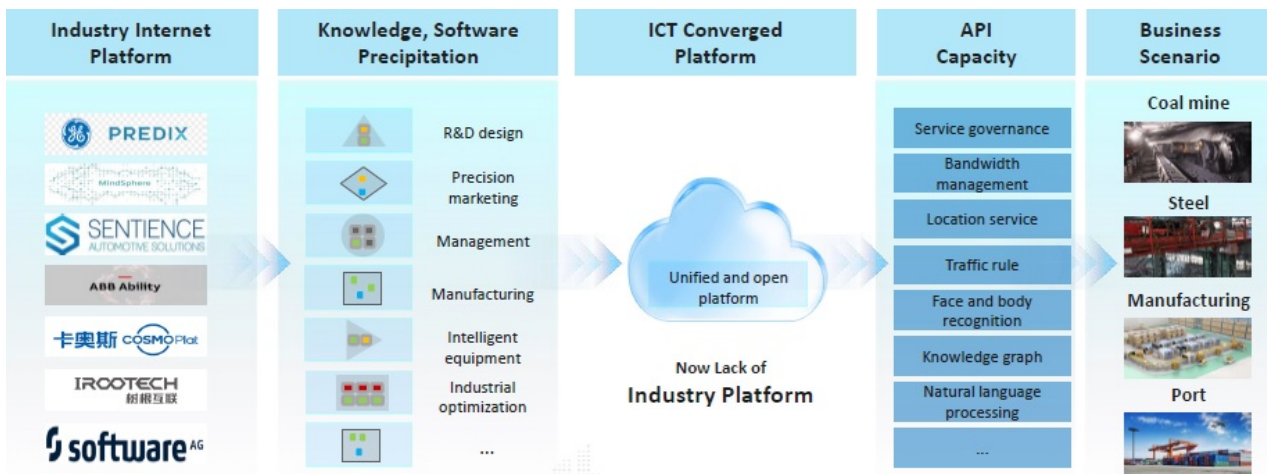


Figure 2.10 - Positioning of EdgeGallery

2.2.3 Edge Computing Security

In May 2021, ETSI published a white paper entitled, *ETSI MEC Security: Status of Standards Support and Future Evolutions*²⁹, with a focus on security-related use cases and requirements, highlighting inadequate industry approaches to cloud security. Tackling issues like security, privacy and trust need to be prioritised along with the challenges brought by edge cloud federations and (far) edge devices, e.g., IoT environment, with end-to-end (E2E) approaches by adopting standards relevant to edge computing systems. In this context, end-to-end security in edge environments has implications on all the elements coming from stakeholders in the system.

The white paper gives an overview of ETSI MEC standards and support for security with references to other relevant standards in the domain, e.g., ETSI TC CYBER, ETSI ISG NFV, 3GPP SA3, as well as cybersecurity regulation potentially applicable to edge computing, concluding with perspectives on future evolutions and standards directions. Standards for the infrastructure virtualisation and management include critical building blocks for MEC system security design.

IoT use cases may differ substantially both within and across industry segments. Sectorial requirements and national requirements for privacy and security can bring further complications in terms of overall performance and compliance. E2E security in MEC systems is therefore very important. Moreover, MEC platforms can handle application traffic and various application elements may reside not only in MEC hosts but also on User Equipment (UE) and end-devices, where the security capabilities of the latter come into the E2E picture of security. The deployment of IoT devices in a MEC environment may also involve additional support that an IoT device may require, e.g., due to security constraints, power limitations and compute or communication capabilities.

Because of the highly distributed nature of MEC clouds, Trusted Computing concepts are highly recommended. The Trusted Computing Group (TCG)³⁰ defines the widely accepted TP standard and related standards. In MEC environments, TCG is important for physical platform security.

The benefits of edge computing will strengthen the promise of 5G and expand the prospects for several new and enhanced use cases, including virtual and augmented reality, IoT, Industrial IoT, autonomous driving, real-time multiplayer gaming, split computing – once the large-scale infrastructure and pervasive Wide Area

²⁹ https://www.etsi.org/images/files/ETSIWhitePapers/ETSI_WP_46-_MEC_security.pdf

³⁰ <https://trustedcomputinggroup.org/>

Networks are in place³¹. However, in the 3GPP 5G system, MEC use cases represent many of the higher risk threat scenarios. 3GPP specifications and standardisation work come into play through SA3 (security), which assesses security issues once the core work has been completed.

Finally, the evolution of edge computing will go hand in hand with related infrastructural technologies, cloud, and networking. Hence security challenges will be associated with challenges in these technologies with an additional “edge twist” in terms of being a highly distributed cloud reliant on network functionality.

The SDO outputs to date spotlighted in the white paper are summarised in Table 2.4.

SDO WG/TC Output	Purpose
ETSI ISG NFV³²	Describe and specify virtualisation requirements, NFV architecture framework, functional components, and their interfaces, as well as protocols and APIs for these interfaces
NFV Security WG (NFV SEC)³³	Analysis of specifications and reports on security requirements and solutions for NFVI (NFV infrastructure)
ETSI MEC³⁴	MEC architecture design variant allowing for different deployment options of MEC systems.
ETSI ISG NFV³⁵	ETSI GS NRV-SEC 024: provide a “whole system” security monitoring and management framework.
ETSI NFV security WG^{36 37}	ETSI GR NFV-SEC 011 and ETSI GS NFV-SEC 010: security and protocols necessary to securely support lawful interception in a virtualisation environment which include LI architecture and retained data protection.
ETSI ISG NFV^{38 39}	ETSI GS NFV-SOL 013 and ETSI GS NFV-SEC022: security for NFVI and NFV MANO API. It is essential to ensure APIs do not allow an attacker a single point of entry into all layers of the MEC environments.

³¹ https://www.3gpp.org/news-events/2152-edge_sa6.

³² <https://www.etsi.org/technologies/nfv>.

³³ https://docbox.etsi.org/isg/nfv/open/Publications_pdf/Specs-Reports/NFV-SEC%20022v2.6.1%20-%20GS%20-%20API%20Access%20Token%20Spec.pdf.

³⁴ <https://docbox.etsi.org/ISG/NFV/Open/Drafts>.

³⁵ ETSI GS NFV-SEC 024 V2.8.1 (2020-06): “Network Functions Virtualisation (NFV) Phase 2); Security; Access Token Specification for API access”, Link: https://www.etsi.org/deliver/etsi_gs/NFV-SEC/001_099/022/02.08.01_60/gs_NFV-SEC022v020801p.pdf.

³⁶ ETSI GR NFV-SEC 011 V1.1.1 (2018-04): “Network Functions Virtualisation (NFV); Security; Report on NFV LI Architecture”, Link: https://www.etsi.org/deliver/etsi_gr/NFV-SEC/001_099/011/01.01.01_60/gr_nfv-sec011v010101p.pdf.

³⁷ ETSI GS NFV-SEC 010 V1.1.1 (2016-04): “Network Functions Virtualisation (NFV); NFV Security; Report on Retained Data problem statement and requirements”, Link: https://www.etsi.org/deliver/etsi_gs/nfv-sec/001_099/010/01.01.01_60/gs_nfv-sec010v010101p.pdf

³⁸ ETSI GS NFV-SOL 013 V3.4.1 (2021-01): “Network Functions Virtualisation (NFV) Release 3; Protocols and Data Models; Specification of common aspects for RESTful NFV MANO APIs”, Link: https://www.etsi.org/deliver/etsi_gs/NFV-SOL/001_099/013/03.04.01_60/gs_NFV-SOL013v030401p.pdf.

³⁹ ETSI GS NFV-SEC 022 V2.8.1 (2020-06): “Network Functions Virtualisation (NFV) Release 2; Security; Access Token Specification for API Access”, Link: https://www.etsi.org/deliver/etsi_gs/NFV-SEC/001_099/022/02.08.01_60/gs_NFV-SEC022v020801p.pdf.

NFV security WG^{40 41 42}	ETSI GS NFV-SEC 023: container security. ETSI GS NFV-SEC 026: isolation and trust domain. ETSI GR NFV-SEC 027: NFVI security assurance. The aim of this work is to make the NFVI secure enough for wide range deployment scenarios and use cases.
ETSI TC CYBER^{43 44}	EN 303 645: consumer IoT requirements for security and data protection, specifying security and data protection provisions for consumer IoT devices that connect to network infrastructures and interact with associated services. Draft TS 103 701 defines a conformance assessment methodology while work item CYBER.0062 offers guidance on vulnerability disclosures.
ETSI TC CYBER⁴⁵	Draft ETSI TS 103 486: secure and manageable identity management scheme applicable to IoT devices in a MEC environment. Requirements of cryptographic methods that establish trust in Authority-Attribute trees, with data structures representing identity information that can be encoded in a suitable ontology (e.g., SAREF[15]).
ETSI TC CYBER⁴⁶	Draft ETSI TS 103 742: basic good practices for the cybersecurity of communication networks such as one that the deployment of IoT applications in a MEC environment may involve, including policy aspects and their lifecycle.
TCG⁴⁷	Baseline TPM specification (TCG1) is an international ISO/IEC standard (ISO/IEC 11889).
3GPP⁴⁸	Draft TS 23.548 (5G system Enhancements for Edge Computing). Release 17. Support of EAS discovery and edge relocation in various connectivity models, network information provisioning to local applications with low latency.
3GPP SA3⁴⁹	TR33.839: security enhancements on the support for edge computing in the 5G core network (5GC) as part of Release 17. This TR covers a key issue from a security perspective based on the procedure to protect the EAS discovery

⁴⁰ Draft ETSI GS NFV-SEC 023 V0.0.4 (2020-07): " Network Functions Virtualisation (NFV); Security; Container Security Specification - Release 4", Link: <https://docbox.etsi.org/ISG/NFV/Open/Drafts->

⁴¹ Draft ETSI GS NFV-SEC 026 V0.0.3 (2021-02): "Network Functions Virtualisation (NFV) Release 4; Security; Isolation and trust domain specification - Release 4 ", Link: <https://docbox.etsi.org/ISG/NFV/Open/Drafts>.

⁴² Draft ETSI GR NFV-SEC 027 V0.0.1 (2021-04): "Network Functions Virtualisation (NFV) Release 4; Security; Report on security assurance of NFVI - Release 4", Link: <https://docbox.etsi.org/ISG/NFV/Open/Drafts>.

⁴³ Draft ETSI EN 303 645 V2.1.0 (2020-04): "Cyber Security for Consumer Internet of Things: Baseline Requirements", Link: https://www.etsi.org/deliver/etsi_en/303600_303699/303645/02.01.00_30/en_303645v020100v.pdf.

⁴⁴ Draft ETSI TS 103 701 V0.0.7 (2021-03): "Cyber Security for Consumer Internet of Things: Conformance Assessment of Baseline Requirements", Link: https://docbox.etsi.org/CYBER/CYBER/Open/Latest_Drafts/CYBER-0050v007-TS103701-Cybersecurity-assessment-for-consumer-IoT-product.pdf.

⁴⁵ Draft ETSI TS 103 486: "CYBER; Identity Management and Discovery for IoT", DTS/CYBER-0014' Work Item: https://portal.etsi.org/webapp/WorkProgram/Report_WorkItem.asp?WKI_ID=47653.

⁴⁶ Draft ETSI TS 103 742: "CYBER; Baseline Cybersecurity for a Communications Network", DTS/CYBER-0055' Work Item: https://portal.etsi.org/webapp/WorkProgram/Report_WorkItem.asp?WKI_ID=58919.

⁴⁷ ISO/IEC 11889 (2015): "Information technology – Trusted platform module library – Part1: Architecture", <https://www.iso.org/standard/66510.html>.

⁴⁸ 3GPP TS 23.548: "Study on application architecture for enabling Edge Applications", https://www.3gpp.org/ftp/Specs/archive/23_series/23.548/.

⁴⁹ 3GPP TR 33.839: "Study on security aspects of enhancement of support for edge computing in 5G Core (5GC)", https://www.3gpp.org/ftp/specs/archive/33_series/33.839.

	message, securely expose the network information to local applications and authorises the UE EAS service access during edge data network relocation with seamless change.
3GPP SA6⁵⁰	TS 23.558 (EDGEAPP architecture): Security of the transport for edge interfaces should include aspects like confidentiality, integrity and relay protected to prevent any attacker from eavesdropping, manipulation and/or replay. Authentication and authorisation to access EES capability is key to avoiding unauthorised information and attackers from flooding the EES to launch a Denial-of-Service Attack. The re-use of the CAPIF security model for authentication and authorisation has been proposed.
3GPP⁵¹	TS 33.501: Proposed re-use of SBI based security for message protection to ensure confidentiality, integrity and replay protection. To enable DNS security, DNS over (D)TLS can be used for secure discovery of edge services. Release 15 (Non-standalone). Like TR 33.848, this report explores additional threats and mitigations necessary to design, test and deploy functions in a virtual environment.
3GPP SA3⁵²	TR 33.848: Study on security impacts of virtualisation. Security impacts of virtualisation. SA3 has so far identified 28 key security issues applicable to MEC. Some of these can be addressed through existing standards and security best practices but others require the development of new security solutions or deployment mitigations.
3GPP SA3⁵³	TR33.818: Security Assurance Methodology (SECAM), leading to the introduction of a set of security assurance specifications (SCAS) for 3GPP virtualised network products. This TR provides security assurance mechanisms in 3GPP virtualised environment, such as threats related to the integration of ETSI VNF concepts and interfaces in the 3GPP virtualised system. Release 17.
NESAS, jointly defined by 3GPP and GSMA⁵⁴	NESAS-2.0. The Network Equipment Security Assurance Scheme (NESAS) is an industry-wide framework to facilitate improvements in security levels across the mobile industry. It provides a security baseline to evidence that network equipment satisfies a list of security requirements and has been developed based on vendor development and product lifecycle processes that provide security assurance. The scheme should be used globally as a common baseline, on top of which individual operators or national IT security agencies may want to put additional security requirements.

⁵⁰ 3GPP TS 23.548: "Study on application architecture for enabling Edge Applications", https://www.3gpp.org/ftp/Specs/archive/23_series/23.548/.

⁵¹ 3GPP TS 33.501: "Security architecture and procedures for 5G System", https://www.3gpp.org/ftp/Specs/archive/33_series/33.501/.

⁵² <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3574>.

⁵³ <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3543>.

⁵⁴ Network Equipment Security Assurance Scheme – Overview; Version 2.0", 05 February 2021, Link: <https://www.gsma.com/security/wp-content/uploads/2021/02/FS.13-NESAS-Overview-v2.0.pdf>.

<p>ISO/IEC⁵⁵</p>	<p>ISO 15408-1 (2009; reviewed in 2015) - Security assurance: Common Criteria technology is a global reference. It defines a framework for specifying the functional and security requirements of a computer system operating in a given environment and subject to certain assurance levels.</p> <p>Tailored schemes for specific constraints of 5G ecosystems, e.g., GMSA NESAS could also play a role in meeting regulatory requirements for 5G infrastructure components.</p>
------------------------------------	---

Table 2.4 - Standards relevant for edge computing security (Source: ETSI)

Future evolutions of security support

The ETSI white paper highlights the following five evolutions that will bring new security challenges, with several base security enablers that will require future standardisation to support the evolution of edge computing:

- More radical approach to cloud-native like serverless proposals.
- Consolidation of hybrid clouds and NFV through the seamless integration of acceleration mechanisms both in computing and networking: in-networking computing.
- The integration of acceleration mechanisms both in computing and network forwarding.
- The pervasiveness of AI and its strong reliance on dependable data flows.
- The evolution of networks beyond 5G.

Table 2.5 summarises the main standardisation requirements expected from these evolutions.

Evolution Focus	High-level standardisation requirements
Confidential computing	Protect data while being processed and stored through hardware-based isolation for the processing of payloads. Currently based on vendor specific designs and threat models. It requires a standardised approach to hardware-based acceleration.
Network topology attestation	Ability to verify that a given network flow, at any given plane, will pass through the specific functions, optionally preserving a given order. A promising way forward for completing attestation mechanisms is inline operation and management along with programmable packet forwarding devices.
AI pervasiveness	Address challenges in terms of privacy, especially when using highly centralised data stores. New techniques enable collaborative AI/ML without centralised data. This allows accurate AI/ML models to be trained while retaining the privacy and locality of private and sensitive data. Evident applicability of edge computing. Need for common, secure, and standardised data and knowledge representation models.
Disintermediation mechanisms	This advent of disintermediation mechanisms provided by Distributed Ledger Technologies presents possibilities for distributed security

⁵⁵ <https://www.iso.org/standard/50341.html>

	solutions, including the incorporation of reputation mechanisms and dynamic trust assessment. DLT has the potential to protect the integrity of AI data via immutable records and distributed trust between stakeholders.
Cryptographic models and protocols	These methods and models are rooted at the computational complexity of solving certain mathematical problems and are threatened by the advent of quantum computers. Ongoing standardisation activities should be incorporated into future edge computing practice. NIST is leading standardisation efforts towards interoperability of secure cryptographic systems with existing protocols and networks ⁵⁶ . ETSI ISG QKD (Quantum Key Distribution) is working on interfaces and operational procedures for integrating this technology with network operations and services ⁵⁷ .

Table 2.5 - Security Support Evolutions (Source: ETSI)

2.2.4 Securing Artificial Intelligence

Artificial intelligence has been driven by the rapid progress of deep learning and its wide applications, such as image classification, object detection, speech recognition and language translation. The ETSI Group Report on *Securing Artificial Intelligence (SAI) Mitigation Strategy Report, ETSI GR SAI 005 V1.1.1*⁵⁸, published in March 2021, focuses on deep learning, and explores the existing mitigating countermeasure attacks.

The GR summarises existing and potential approaches against poisoning attacks, backdoor attacks, evasion attacks, model stealing attacks and data extraction attacks for AI-based systems. Existing mitigations can become less effective over time due to the rapid evolution of attack technology for AI-based systems though mitigation approaches and their rationales remain. Most of the approaches analysed come from academic settings with assumptions that need to be considered when applied in practice.

It outlines available methods for securing AI-based systems by mitigating known or potential security threats identified in the recent ENISA threat landscape publication⁵⁹ and ETSI GR SAI 004 Problem Statement Report⁶⁰. It also discusses security capabilities, challenges, and limitations when adopting mitigation for AI-based systems in certain potential use cases. As such, the GR serves as a securing AI technical survey reference for planning, designing, developing, deploying, operating, and maintaining AI-based systems, setting a baseline for a mutual understanding of relevant AI cyber security threats and mitigations that will be key for widespread deployment and acceptance of AI systems and applications. In future, more research work needs to be done on automatic verification and validation, explainability and transparency, and novel security techniques to counter emerging AI threats.

The GR describes the workflow of machine learning models, where the model life cycle includes both development and deployment stages. Based on this, it summarises existing and potential mitigation approaches against training attacks, that is, mitigations to protect the ML model from poisoning and backdoor attacks, and against inference attacks, including those from evasion, model stealing, and data extraction.

⁵⁶ <https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization>. .

⁵⁷ <https://www.etsi.org/technologies/quantum-key-distribution>.

⁵⁸ https://www.etsi.org/deliver/etsi_gr/SAI/001_099/005/01.01.01_60/gr_SAI005v010101p.pdf.

⁵⁹ <https://www.enisa.europa.eu/topics/threat-risk-management/threats-and-trends>.

⁶⁰ https://www.etsi.org/deliver/etsi_gr/SAI/001_099/004/01.01.01_60/gr_SAI004v010101p.pdf.

Mitigation approaches are firstly summarised as model enhancement and model-agnostic, and then grouped by their rationales.

2.3 Importance of AI@EDGE for the Business

The availability of huge datasets and technology advances in Big Data, the Internet of Things (IoT) and fast connectivity are showing the potential impact of AI new systems and services, digital assistants, robots, and drones on companies. AI is today a general-purpose technology, with the ability to leverage data transforming it into actionable insights, extended and packaged into different types of applications across all industries, such as voice and image recognition, machine translation, control of assisted driving and autonomous vehicle navigation. The EC's White Paper on artificial intelligence [EuropeanCommission2020], published for public consultation in February 2020, defines AI as one of the most important applications of the data economy. The EC considers AI as an instrument to harness the value of data in order to build European sustainable economic growth and societal wellbeing.

Since AI relies on an extensive base of data, it is usually associated with environments where computational and storage capabilities are at their best, being within a core datacentre of a company, or in the cloud. In fact, AI is becoming a critical platform play for all cloud service providers in Europe. As many organizations are choosing to move increasing volumes of data into the cloud as part of their digital transformation, cloud represents a key enabler for AI and is increasingly becoming the choice for organizations building AI because of its flexibility and scalability when it comes to running processes as well as its capabilities with respect to organizing and managing enterprise data.

Still, according to IDC's Multi-cloud survey 2020, on-premises IT is the preferred way to deliver AI workload for 51% of the European respondents, followed by Hybrid Cloud (28%) and Public Cloud (15%). To understand the impact of AI at the edge, though, a distinction between different AI workloads should be made. Broadly speaking, this can be divided into the training of an AI model, and the inference process. Training requires heavy computing and storage capabilities, having the datacentre or the cloud as the most convenient environment to perform the task. On the other hand, when the model is trained, inference does not require the same performance, opening new possibilities.

Edge is gaining increasing attention as the approach of moving infrastructure and workloads closer to where data is created and consumed is now promising for different use cases across multiple industries, especially in combination with AI and IoT. According to the 2018 European AI survey, already 40% of the respondents were deploying AI inferencing at edge locations. With the advancements seen in the silicon industry and the ability to run increasingly efficient AI models on small footprint devices, edge will become the preferred environment where AI models will be applied [Roberti2020].

According to IDC's Edge Spending Guide, V2 2020, the AI component of the European Enterprise Edge spending will become increasingly important, driving 10% of the market by 2024, and passing the \$4B mark (see Figure 2.9). For comparison, the figure for the US shows a bigger market, almost two times the size of the European one. Still, estimated growth rates are very similar, with 4-years compound annual growth rates aligned right below 30%, highlighting how the interest in AI at the edge is skyrocketing across different regions.

Looking at the European scenario, beyond professional services, that will be interested in the build out of the edge value chain, to provide AI solutions at the edge to customers in other industries, Retail, Manufacturing and Transportation companies will be the verticals driving the growth. Further, it is important to note that AI at the edge will play a role in enabling key use cases both to increase customer experience and to ensure better operation reducing costs, like in the case of automated preventive maintenance, or automated threat

intelligence and prevention systems, applicable across different verticals, or in industry-specific use cases like diagnosis and treatment systems for Healthcare [IDC Edge Spending Guide, V2 2020 (January 2021)].

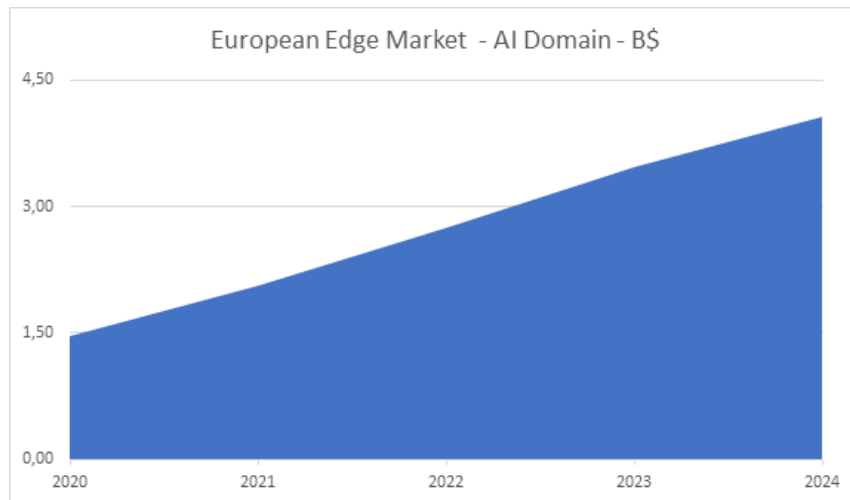


Figure 2.11 - European Edge Market Growth. Source: IDC Edge Spending Guide, V2 2020 (January 2021)

3. AI technologies

Chapter 3 describes the state-of-the-art of the traditional frameworks adopted in the computing continua for the development of AI application. Moreover, it presents the solutions currently offered by the cloud big players and by the main European cloud providers. The chapter overviews the hardware of intelligent sensors currently offered in the market, including those that will be adopted to develop the AI-SPRINT use cases.

In particular, an initial overview on edge intelligence is presented in Section 3.1, while Section 3.2 focuses on the main frameworks for the design and development of AI applications. Section 3.3 describes PyCOMPSs, a framework for parallel, distributed applications. Section 3.4 gives an overview of cloud-based solutions related to AI, and Section 3.5 provides a snapshot on the available technologies for edge intelligence. Then, application-specific technologies are presented. In particular, Section 3.6 describes wearable sensors for stroke care, Section 3.7 focuses on cameras for Agriculture 4.0, and Section 3.8 provides insights on drone technologies for inspection and maintenance. Finally, Section 3.9 provides a description of the AI frameworks, boards, and accelerators for the considered case studies.

3.1 Overview

It is recognized that the three most important elements for an intelligent application are: data, model, and computation. Correspondingly, the complete deployment of most intelligent applications (unsupervised learning-based application is not included, which however is not a target in AI-SPRINT) includes three components: data collection and management, training, and inference. Computation remains a hidden component that is essential for the other three components. Combined with an edge environment, these three obvious components turn into edge cache (data collection and storage at the edge), edge training (training at the edge), and edge inference (inference at the edge) respectively. Computation at the edge traditionally is done via offloading. Hence, the hidden component turns into the edge offloading (computation at the edge).

Edge Caching

In edge intelligence, edge caching refers to a distributed data system's proximity to end-users. It collects and stores the data generated by edge devices and surrounding environments and the data received from the Internet to support intelligent applications for users at the edge. Edge devices, such as monitoring devices and sensors, record the environmental information. Such data are collected at reasonable places and used for processing and analysis by intelligent algorithms to provide services for end-users (for example, the video captured by cameras could be cached on vehicles for aided driving [Xu2017]).

Edge Training

Edge training refers to a distributed learning procedure that learns the optimal values of all the weights, the bias, or the hidden patterns based on the training set cached at the edge. Different from traditional centralized training procedures on powerful servers or computing clusters, edge training predominantly occurs on edge servers or edge devices, which are usually not as powerful as centralized servers or computing clusters. The model/algorithm is trained either on a unique device (solo training [Lane2016]) or by the collaboration of edge devices (collaborative training [McMahan2017, IEEEGuide2021]) with training sets cached at the edge. The availability of the acceleration module speeds up the training, while the optimization module rectifies

problems in training: for example, updating the frequency, cost, privacy, and security issues. The uncertainty estimates module controls the uncertainty in training. Once trained, the computed parameters can be persisted to storage (file), program memory (RAM), and deployed as an Application Programming Interface (API) — the model can be executed in resource-constrained devices in an offline fashion. The deployed models are monitored for drift and retrained as necessary.

Edge Inference

In order to employ trained models for inference in an edge environment some critical issues have to be faced. In particular, we must consider:

- how to make models applicable for their deployment on edge devices or servers;
- how to accelerate edge inference to provide real-time responses.

For the first problem, researchers mainly focus on two research directions: designing new models/algorithms that consider fewer requirements on the hardware naturally suitable for edge environments, and compressing existing models to reduce unnecessary operation during inference. In this last direction researchers focus on compressing existing models to obtain thinner and more compact models which are more computation- and energy-efficient with negligible or even no loss on accuracy. There are five commonly used approaches to model compression:

- *Low-rank approximation*: the main idea is to use the multiplication of low-rank convolutional kernels to replace kernels of high dimension. This is based on the fact that a matrix could be decomposed into the multiplication of multiple matrices of smaller size [Jaderberg2014];
- *Knowledge distillation*: is based on transfer learning [Yosinski2014], which trains a NN of smaller size with the distilled knowledge from a larger model. The large and complex model is called teacher model, while the compact model is referred as student model, which takes the benefit of transferring knowledge from the teacher network;
- *Compact layer design*: in Deep NNs, if weights end up to be close to zero, the computation is wasted. A fundamental way to solve this problem is to design compact layers in NNs, which could effectively reduce the consumption of resources, e. g., memories and computation power. Some approaches consist of introducing sparsity and replacing fully connected layers with global average pooling [Szegedy2015];
- *Network pruning*: the main idea is to delete unimportant parameters, since not all parameters are important in highly precise Deep NNs [LeCun1989, Hassibi1992]. Consequently, connections with less weights are removed, which converts a dense network into a sparse one;
- *Parameter quantization*: a very Deep NN involving many layers with millions of parameters, consumes a large amount of storage and slows down the training procedure. However, highly precise parameters in NNs are not always necessary in achieving high performance, especially when these highly precise parameters are redundant [Denil2013]. For example, some approaches consist in the conversion of weight parameters of intermediate layers in 8-bit fixed point integers [Vanhoucke2011]. Results show that the total required memory shrinks 3-4 times and that the quantised model could achieve a 10 times speedup over an optimised baseline and a 4 times speedup over an aggressively optimised float point baseline without affecting the accuracy.

The next sections provide a complete analysis on the technologies currently used for the development of AI applications in computing continua which are the target deployment models for AI-SPRINT.

3.2 Mainstream Frameworks for AI Development in Computing Continua and Edge Devices

Together with the advances in ML algorithms and in device technology, we are witnessing an increase of the number of specific tools and frameworks, with the goal of making faster and easier the implementation and integration for these algorithms. This section is provided to give a comprehensive overview of the main frameworks on the scene. We start by introducing the concept of Tiny Machine Learning (TinyML), which is targeting limited capacity devices that we can encounter in computing continua. Then, we provide a survey of the frameworks specialized in the development and deployment of ML solutions. Popular general-purpose multi-platform frameworks, i.e., TensorFlow, Keras, PyTorch, and Caffe, are first presented, followed by the ones designed to deploy AI models on generic embedded devices, i.e., TensorFlow Lite, TensorFlow Lite Micro, PyTorch Mobile, and Apache TVM. Finally, some specific-vendor frameworks are described, i.e., NVIDIA TensorRT, Intel OpenVINO Toolkit, and STM32Cube.AI.

3.2.1 Tiny Machine Learning

Today everything in the physical world would be translated preferably into the corresponding digital version. More specifically, this process consists of adjusting a transducer that translates information generated by some sensor towards some digital component. Most of the time this mechanism is accomplished by micro-controllers. On the other side, the endpoint handling digital information, responsible for assigning some meaning to sensor data may be under the form of Edge AI or even Cloud AI. However, some fundamental issues arise in existing intelligent scenarios. In particular, AI processes can be highly inefficient in terms of energy consumption, they are computation-intensive, and they present strict requirements on resources, e.g., Central Processing Unit (CPU), Graphics Processing Unit (GPU), memory, and network, which make it impossible to be available anytime and anywhere for end users.

Furthermore, the relatively vast amount of data exchanged from sensor transducers to the processing unit in FOG architectures introduces several privacy issues as highlighted by the European Union in the General Data Protection Regulation (GDPR) related to securing the private information of users. If mobile users upload their data to the cloud for a specific intelligent application, they will take the risk of privacy leakage: for example, the personal data might be extracted by malicious hackers or companies for illegal purposes [Xu2020]. Nowadays, users' personal, extremely sensitive data like photos and voice recordings are kept indefinitely by the companies that collect them. Users can neither delete them nor restrict the purposes for which it is utilized. Furthermore, centrally kept data are subject to legal subpoenas and extra-judicial surveillance. In most cases, the best way to protect privacy is to confine such data to the device that has acquired them so that sensor data never leaves it.

Moreover, in cloud AI applications, there is a giant volume of data generated and collected by billions of mobile users and IoT devices, distributed at the network edge. Uploading such a volume of data to the cloud consumes significant bandwidth resources, which would also result in unacceptable latency for users. Lastly, the nature of the data exchange, whether it is wired or wireless, can be a source of unreliability, too.

It is exactly in this context that TinyML emerges. It comes into the boundary between the physical and the digital world enabling machine intelligence right next to the physical world, or even on both sides. Conversely to the cloud-based intelligent applications (as also initially discussed in Section 2.1.2), TinyML provides numerous benefits:

- *Energy efficiency*: the devices on which it is deployed consume relatively few energy (order of mWs). It makes battery-operated devices suitable for intelligence inference;
- *Privacy*: the information exchanged by TinyML devices is mostly "meta-data" guaranteeing privacy by design and the overall amount of data is processed locally;

- *Improved latency:* the latency is improved achieving a better designing level and even offline execution is allowed. It makes TinyML suitable for real-time decision-making, like car automation or alongside workers at factory assembly lines;
- *Reliability:* there are no connectivity issues due to the totally no-exchange of digital information — think about Apple Siri (until the current iOS 14 release) and Microsoft’s Cortana that are based on cloud computing and they would not function if the network is unavailable;
- *Reduced costs:* from one side consuming relatively few energy has a direct impact on the energy costs compared to more powerful solutions. On the other hand, TinyML is cheaper than cloud-based intelligent solutions because it requires less storage and replaces the processing costs.

TinyML represents a burgeoning field at the intersection of embedded systems and ML. The world has over 250 billion microcontrollers [ICInsight2020] with strong growth projected over coming years. As such, a new range of embedded applications is emerging for Neural Network (NN) models. Because these models are extremely small (a few hundred KBs), running on microcontrollers or DSP-based embedded subsystems, they can operate continuously with minimal impact on device battery life [David2021]. Table 3.1 highlights the differences in terms of hardware machine intelligence is commonly deployed. It is analogously remarkable the difference with the design of a TinyML deployment and with adopted hardware platforms as well.

Technology	Implemented Design	Hardware
Cloud AI	Deep NN on the cloud	Tensor Processing Unit (TPU), Field Programmable Gate Array (FPGA), GPU, CPU
Edge AI	Optimized algorithms and CNN-light	System on Chip (SoC), additionally with Neural Processing Unit (NPU) accelerators
TinyML	CNN-micro	Microprocessor Control Unit (MCU) with no hardware accelerators

Table 3.1 - Positioning of TinyML with respect to cloud and edge AI

The NN model difference between TinyML, and cloud of course, but solely considering the smaller edge, is astonishing and this difference leads to a hardware choice that inherits all the benefits listed so far. Moreover, it is of pairwise importance to consider the data sources the TinyML applications would gather and from where. From Figure 3.1, it is clear that most of the information handled by such micro-CNNs are coming from the space around the device. Capturing data starting from CMOS cameras, IR cameras, microphones, or optical sensors, TinyML can be everywhere around us.

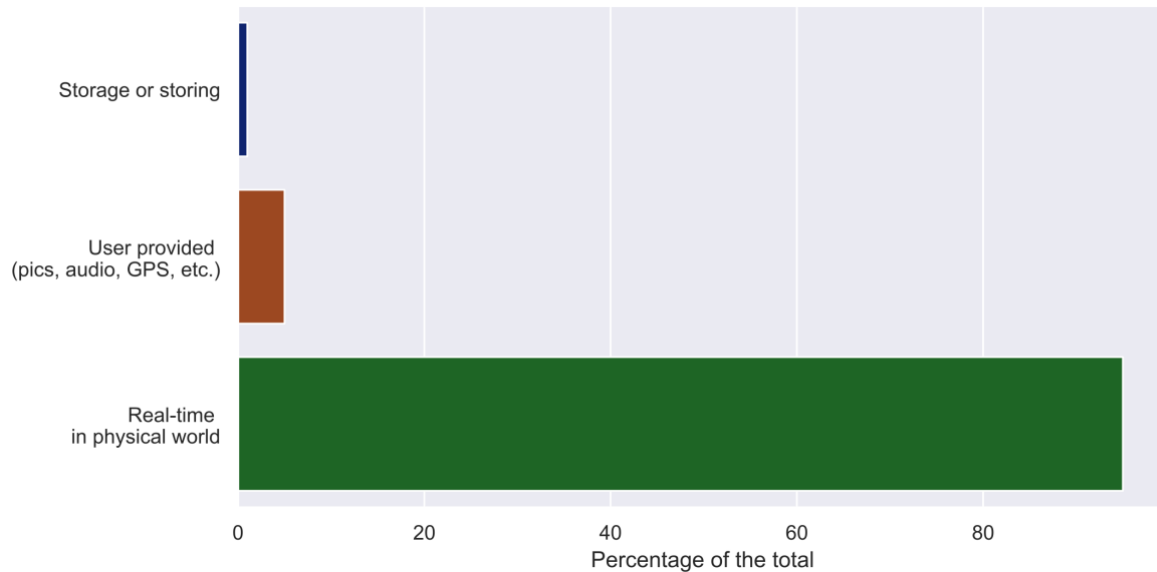


Figure 3.1 - Percentage of information handled by micro-CNNs [Gousev2020]

We have a diverse ecosystem allowing TinyML to grow faster [David2021]. On one side, the hardware production allows to have available on the market more efficient platforms; on the other side, the software infrastructure becomes more mature, and we have available more efficient algorithms. Moreover, many corporate and venture capitalists invest in such technologies and the number of start-ups increases too. However, this is also limiting TinyML because hardware vendors have related but separate needs. TinyML frameworks are often tied to specific devices, and it is hard to perform a proper choice as they are optimized based on hardware, software, and the specific vertically integrated solution.

Some specific technical challenges make developing a ML framework for embedded systems extremely complex:

- **Missing Features:** Embedded platforms are defined by their tight limitations. Therefore, many advances from the past few decades that have made software development faster and easier are unavailable to these platforms because the resource trade-offs are overly expensive. Examples include dynamic memory management, virtual memory, an operating system, a standard instruction set, a file system, floating-point hardware, and other tools that seem fundamental to modern programmers. Though some platforms provide a subset of these features, a framework targeting widespread adoption in this market must avoid relying on them.
- **Fragmented Market and Ecosystem:** Many embedded system uses only require fixed software developed alongside the hardware, usually by an affiliated team. The lack of applications capable of running on the platform is therefore much less important than it is for general-purpose computing. Moreover, backward instruction-set-architecture (ISA) compatibility with older software matters less than in mainstream systems because everything that runs on an embedded system is probably compiled from the source code anyway. As follows, embedded hardware can aggressively diversify to meet power requirements, whereas even the latest x86 processor can still run instructions that are nearly three decades old. These differences indicate the pressure to converge on one or two dominant platforms or ISAs is relatively weaker in the embedded space, leading to fragmentation. Many ISAs support thriving ecosystems, and the benefits they bring to particular applications outweigh developers' cost of switching. Companies indeed allow developers to include their ISA extensions. Matching the broad variety of embedded architectures are the numerous toolchains and integrated

development environments (IDEs) that support them. Many of these systems are exclusively available through a commercial license with the hardware manufacturer, and in cases where a customer has requested specialized instructions, they may be inaccessible to everyone. These arrangements have no open-source ecosystem, leading to device fragmentation that prevents the development team from producing software that runs on many separate embedded platforms.

- Resource Constraints:* People who build embedded devices do so because a general-purpose computing platform exceeds their design limits. The most dominant drivers are: (i) cost, with a microcontroller typically selling for less than a few dollars [ICInsight2020]; (ii) power consumption, as embedded devices may require just a few milliwatts of power, whereas mobile and desktop CPUs require watts; and (iii) form factor, since capable microcontrollers can be smaller than a grain of rice. To accommodate their needs, hardware designers trade-off capabilities. A general characteristic of an embedded system is its low memory capacity. At one end of the spectrum, a big embedded system has a few megabytes of flash ROM and at most a megabyte of SRAM. At the other end, a small embedded system has just a few hundred kilobytes or fewer, often split between ROM and RAM. These constraints mean both working memory and permanent storage are significantly smaller than most software designed for general-purpose platforms would assume. To be specific, the size of the compiled code in storage requires minimization. Most software designed for general-purpose platforms contains code that often goes uncalled on a given device. Choosing the code path at run-time is a better use of engineering resources than shipping more highly custom executables. Such run-time flexibility is complex to justify when code size represents a concern, and the potential uses are much less. As a result, developers must break through a library’s abstraction if they want to carry out modifications to suit their target hardware.

TinyML follows the guidelines introduced in Section 3.1 to develop intelligent applications on edge devices as well as IoT embracing architectures, techniques, tools, and approaches capable of performing on-device analytics for different sensing modalities (like vision, audio, motion) with an energy cost below 1mW, targeting predominately battery-operated devices. The most correct definition of a TinyML implementation is that of a holistic hardware–system–software co-design (see Figure 3.2) that is joint with extreme optimization and innovation in all these three areas. However, as discussed in the previous section related to the development constraints, there is no unique framework to follow for building and deploying a TinyML application, but everything depends on the application tasks and the characteristics of the underlying hardware device and its supported platform(s).

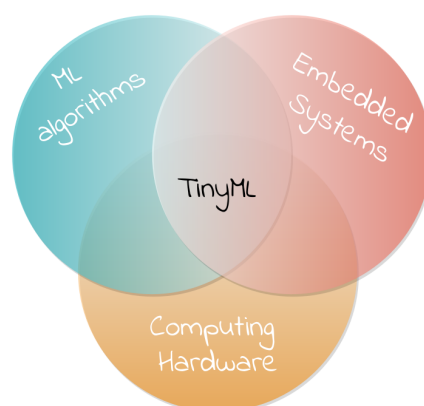


Figure 3.2 - TinyML: holistic hardware–system–software co-design

In the remaining part of this section, we present frameworks that aim to deploy AI models on a generic embedded device and, as a consequence, those which can be defined as all-purpose multi-platform frameworks. Subsequently, we will go deepening some vendor-specific frameworks.

3.2.2 General Purpose Deep Learning Frameworks

TensorFlow

TensorFlow⁶¹ is a free and open-source software library for machine learning created and maintained by the Google Brain team. Since its first release in late 2015, it has become one of the most popular deep learning frameworks in the world. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers and developers easily build and deploy ML powered applications. Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, and Tensor Processing Units - TPUs⁶²) with few code changes, and from desktops to clusters of servers to mobile and edge devices. Furthermore, it provides support for distributed training, allowing portions of the network graph to be computed on different processes and servers, and for scaling computation on multiple GPUs, on one machine, on multiple machines in a network, or on Cloud TPUs.

TensorFlow computations are expressed as stateful dataflow graphs. The name TensorFlow derives from the operations that such neural networks perform on multidimensional data arrays, which are referred to as *tensors*. In a dataflow graph, nodes represent tensors or operations, and the edges represent input and output of computations. Using dataflow graphs allows exploiting parallelism, distributed execution, and portability. The computation of the entire graph, or parts of it, is performed within a TensorFlow *session*, which encapsulates the environment in which operations are executed and tensors are evaluated. In 2019, Google announced TensorFlow 2.0, whose main updates are Keras integration, described in the following sub-section, and *eager execution*, for immediate iteration and intuitive debugging. Eager execution is an imperative programming environment that evaluates operations immediately, without building graphs: operations return concrete values instead of constructing a computational graph to run later. This makes it easy to get started with TensorFlow and debug models.

Keras

Keras⁶³ is a high-level API specifically developed to enable fast experimentation. It provides an abstract interface of other deep learning frameworks, typically referred to as *backends*. In the last years, Keras updates mainly concentrated on the integration with Tensorflow 2.0, becoming itself a part of the framework as the *tf.keras* package.

Thanks to Keras abstractions, building ML models becomes very intuitive. Indeed, it provides basic building blocks that can be combined to form the final model. The core data structures are *layers*, like Conv2D and Dense (fully-connected) layers, and *models*, through which multiple layers are combined, e.g., the *Sequential* model, which consists of a linear stack of layers. Keras functional API allows to build graphs of layers or to

⁶¹ <https://www.tensorflow.org/>

⁶² <https://cloud.google.com/tpu/docs/tpus>

⁶³ <https://keras.io/>

write custom models from scratch via subclassing of the *Layer* class, which encapsulates both a state, i.e., the layer weights, and a transformation from inputs to outputs, i.e., a *call* (forward pass).

The main advantages of using Keras are:

- Better user experience though a very flexible API
- Availability of some deep learning models with their pre-trained weights
- Built-in support for multi-GPU and distributed training

PyTorch

PyTorch⁶⁴ is an open-source machine learning library primarily developed by Facebook's AI Research lab based on the Torch library, an open-source library based on the Lua programming language released in 2002. It is mainly used for applications such as computer vision and natural language processing. Thanks to its user-friendly interface it gained popularity in the last years becoming one of the preferred platforms for deep learning research.

PyTorch defines its own data structure, i.e., the *torch.Tensor*, which is a multi-dimensional matrix containing elements of a single data type, and provides operations for tensor computing. PyTorch tensor computing is similar to Numpy⁶⁵ computing on arrays, but the former can be accelerated via GPU. One of the main characteristics of PyTorch is the **dynamic computation of graphs** and one of the fastest implementations of *reverse-mode auto-differentiation* through the PyTorch Autograd package, which basically records the operations performed and then uses them to compute gradients. Dynamic graphs allow easier debugging, arbitrary changes to a model without rebuilding overhead, and variable-length inputs and outputs. Furthermore, PyTorch builds its functions as Python classes, thus it can be integrated with Python functions and packages. This, together with its *imperative programming style*, makes developing in PyTorch intuitive and easy to debug.

There are a couple of ways in which PyTorch differentiate from TensorFlow:

- **Ease of use:** compared to Tensorflow, PyTorch offers more freedom to customize layers, thanks also to its tight integration with the Python language. However, the release of its 2.0 version, with the Keras integration, made Tensorflow a good candidate in terms of ease of use too.
- **Static vs. dynamic graphs:** one of the historical differences between the two frameworks was the generation of computational graphs. TensorFlow works on static graphs, i.e., the computation graph of the model must be first defined and then executed, while PyTorch offers the advantages of using dynamic graphs as described before, without the need of using special sessions. However, with the introduction of the eager execution in TensorFlow 2.0 this gap has been reduced, and we can have a dynamic computation similar to the PyTorch one also in TensorFlow.
- **Data parallelism:** it is the other historical difference between the two frameworks. Using *torch.nn.DataParallel* it is possible to wrap any model which is replicated on each device and the computation is parallelized over the batch dimension. Multi-node data parallelism is obtained instead through the *torch.nn.DistributedDataParallel* module, which replicates the model across the defined machines. Implementing data parallelism in Tensorflow 1.X was very difficult and a lot of effort was required to obtain the same PyTorch behaviour. With the introduction of Tensorflow 2.0 things changed and thanks to the *tf.distributed.Strategy* API we can obtain data parallelism with very few lines of code.

⁶⁴ <https://pytorch.org/>

⁶⁵ <https://numpy.org/>

- **Visualization:** when it comes to visualization of the training process, TensorFlow held the lead for a while. TensorFlow uses its TensorBoard⁶⁶ visualization toolkit, which is very complete and provides all the visualization tools needed to debug and understand running experiments. At the beginning PyTorch developers used Visdom⁶⁷, but the features provided by Visdom are very minimalistic and limited. Subsequently, PyTorch integrated TensorBoard visualization through the `torch.utils.tensorboard` utilities, acquiring the same visualization capabilities provided by TensorFlow.

Caffe

Caffe⁶⁸ (Convolutional Architecture for Fast Feature Embedding) is a deep learning framework, originally developed at University of California, Berkeley. Caffe development relies on compiling from C++ source code, and it offers bindings for Python and MATLAB. Its core objectives are strongly oriented toward image segmentation and classification, a specialization that makes it a distinct offering in the run of open-source Python-centered deep learning libraries. It has an expressive architecture that is suitable for the development and modelling of DL models, and which allows for transparent and easy switching between GPU and CPU operations. It is fast and even the project's long-standing claim of being able to process 60 million images in one day (equating to 1 millisecond per image for inference, or 4 milliseconds per image for learning) have been exceeded with recent updates to the library. One of Caffe's other biggest strengths comes in the form of access to a large number of pre-trained models from the Model Zoo deep net repository⁶⁹. It may lack TensorFlow's high-level APIs and generalization, but the extra work needed to implement layers makes for a lean and well-optimized release code. This is fit for mobile and cloud environments without the overhead of complex framework architectures.

In 2017, Facebook Research released Caffe2 to open source. This new version offers quantized computation, additional hardware support, new capacity for high-volume distributed training, and better support for Recurrent Neural Networks (RNNs) than the original. It also brings its own implementation of Model Zoo. In May 2018, Caffe2 was integrated into PyTorch. Caffe is well-suited for edge deployment, less so for abstract research. Caffe2 is heavily used within the Facebook machine learning community.

ONNX

Open Neural Network Exchange (ONNX⁷⁰) is an open-source artificial intelligence ecosystem of technology companies and research organizations that establish open standards for representing machine learning algorithms and software tools to promote innovation and collaboration in the AI sector. ONNX is the first step in enabling different tools to work together, by allowing them to share models. What DL frameworks have in common is that they provide an interface to build computation graphs and runtimes that process these graphs. Thus, graphs serve as intermediate representation (IR). The ONNX format represents a common IR with the goal of establishing a powerful ecosystem among frameworks. ONNX specification consists of the following components:

1. A definition of an extensible computation graph model
2. Definitions of standard data types

⁶⁶ <https://www.tensorflow.org/tensorboard>

⁶⁷ <https://ai.facebook.com/tools/visdom/>

⁶⁸ <https://caffe.berkeleyvision.org/>

⁶⁹ https://caffe.berkeleyvision.org/model_zoo.html

⁷⁰ <https://onnx.ai/>

3. Definitions of built-in operators

Points 1 and 2 form the ONNX IR and built-in operators are divided into a set of primitive operators and functions.

The top-level ONNX construct is a *Model*, a structure that associates metadata with a graph containing all the executable elements. The metadata is used to determine whether an implementation is able to execute the model. Each model must explicitly name the operator sets, which define the available operators and their version. Each operator used within a graph must be declared by one of the operator sets imported by the model. Computation graphs are structured as a list of nodes (with no cycles), each one representing a call to an operator having zero or more inputs and one or more outputs. Each graph must define the names, types and shapes of its input and outputs. Names of nodes, inputs, output, initializers, and attributes are provided in namespaces.

There are two official ONNX variants, i.e., *ONNX* and *ONNX-ML*, which differ for the supported types and operators. ONNX definition recognizes only tensors as input and output types, while ONNX-ML also recognizes sequences and maps.

3.2.3 Deep Learning Frameworks for Embedded Devices

TensorFlow Lite

TensorFlow Lite⁷¹ is a set of tools to help developers run TensorFlow models on mobile, embedded, and IoT devices. It enables on-device ML inference with low latency and limited binary size. TensorFlow Lite consists of two core components:

- *Converter*: it converts TensorFlow models into an efficient form for use by the interpreter and can introduce optimizations to improve binary size and performance.
- *Interpreter*: it runs specially optimized models on many different hardware types including mobile phones, embedded Linux devices, and microcontrollers.

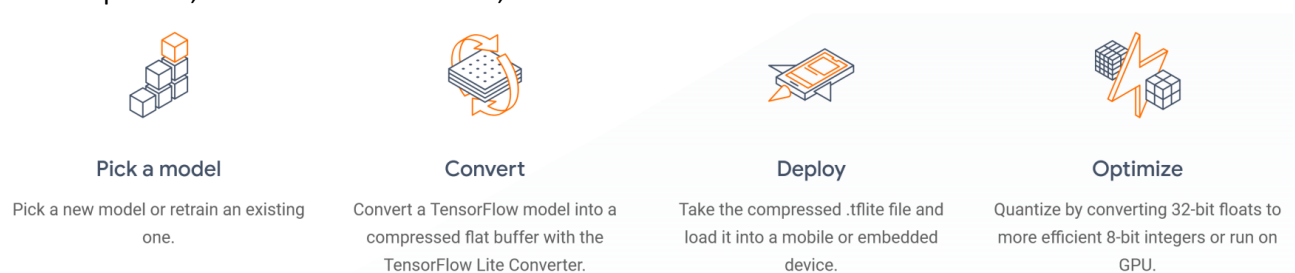


Figure 3.3 - Workflow to deploy machine intelligence on edge devices with TensorFlow Lite⁹

The step-by-step workflow to deploy machine intelligence on edge devices with TensorFlow Lite (as depicted in the Figure 3.3) is the following.

- *Model Selection*: A TensorFlow model is a data structure that contains the logic and knowledge of a ML network trained to solve a particular problem. There are many ways to obtain a TensorFlow model, from using pre-trained models to training your own. To adopt a model with TensorFlow Lite, there is

⁷¹ <https://www.tensorflow.org/lite>

a conversion step which translates a full TensorFlow model into the TensorFlow Lite format. It is impossible to design or train a model using TensorFlow Lite. Even now another approach is considered: transfer learning [Yosinski2014].

- *Model Conversion:* TensorFlow Lite is designed to execute models efficiently on mobile and other embedded devices with limited computation and memory resources. Some of this efficiency comes from utilizing a format for storing models: FlatBuffer, a file format designed to be space-efficient. TensorFlow models must be converted into FlatBuffer format to be handled by TensorFlow Lite.
- *Run Inference with the Model:* Running data through a model to obtain predictions requires a model, an interpreter, and input data. The TensorFlow Lite interpreter comprises a library that takes a model file, executes the operations it defines on input data and provides access to the output. The interpreter works across multiple platforms and provides a simple API for running TensorFlow Lite models from Java, Swift, Objective-C, C++, and Python. As follows, it is easy to use from both major mobile platforms (iOS and Android) but also embedded Linux remains a prominent platform for deploying ML with TensorFlow Lite. Additionally, the interpreter can be configured with delegates to make use of hardware acceleration on different devices. The GPU Delegate allows the interpreter to run proper operations on the device GPU.
- *Model Optimization:* TensorFlow Lite provides optimization tools that typically reduce the size of the model, the time it takes to run, or both. This can come at the cost of a reduction in terms of accuracy, but the reduction is often slight enough it is worthwhile. Moreover, optimized models may require slightly more complex training, conversion, or integration. ML optimization represents an evolving field, and TensorFlow Lite Model Optimization Toolkit is continually growing as new techniques have been developed. One of the most useful optimizations is quantization. By default, the weights and biases in a model are stored as 32-bit floating-point numbers so that high-precision calculations can occur during training. In TensorFlow Lite, quantization allows reducing the precision of these numbers so that they fit into 8-bit integers, a four-times reduction in size. To a greater extent, since it is effortless for a CPU to perform math with integers than with floats, a quantized model will run faster.

Finally, with a focus on the deployment of the converted and optimized model on microcontrollers, the last step is to convert such a model into a C source file that can be included in a binary file and loaded directly on the memory of a resource-constrained device. This is explicitly done because most of the microcontrollers do not have a filesystem and even if they did, the extra code required to load a model from the disk would be wasteful given their limited space at disposal.

TensorFlow Lite Micro

TensorFlow Lite Micro is an open-source ML inference framework for running DL models on embedded systems, based on the work of David et al. [David2021]. It inherits TensorFlow Lite model conversion and optimization infrastructure while addressing the efficiency demands imposed by embedded-system resource constraints and fragmentation challenges that make cross-platform interoperability nearly impossible. The framework employs a one-of-a-kind interpreter-based approach that allows for flexibility, portability and easy adaption to new applications or features, while addressing these unique challenges. The deployment of a TensorFlow Lite Micro application is divided into several steps:

1. The first step is to load into memory a live NN model object. Through the client API, the application developer creates an “operator resolver” object. The API OpResolver controls which operators link to the final binary, reducing file size.

2. The second step is to provide a contiguous memory “arena” that stores intermediate results and other variables required by the interpreter. This is required because it is assumed that dynamic memory allocation is not available.
3. The third step is to create an interpreter instance and pass it the model, the operator resolver, and the arena as arguments. An alternative to an interpreter-based inference engine is to use C or C++ to generate native code from a model during export. This can improve performance at the expense of portability because the code must be recompiled for each target. During the initiation phase, the interpreter allocates all necessary memory from the arena. Any subsequent allocations are avoided to prevent heap fragmentation causing errors in long-running applications. OpResolver, which is provided by the application, maps the serialized model’s operator types to the implementation functions.
4. The fourth step is execution. The application retrieves and populates pointers to memory regions that represent the model inputs (often derived from sensors or other user-supplied data). When the inputs are ready, the application calls the interpreter to run the model calculations. This procedure entails iterating through the topologically sorted operations, locating the inputs and outputs using offsets calculated during memory planning (see Figure 3.4), and calling the evaluation function for each operation.
5. Finally, the interpreter returns control to the application after evaluating all operations. Invocation is a straightforward blocking call. Because most Microprocessor Control Units (MCUs) are single-threaded and use interrupts for urgent tasks, this is acceptable. However, an application can still perform one from a thread, and platform-specific operators can continue to distribute their work across processors. Once the invocation is complete, the application can query the interpreter to locate the arrays containing the model-calculation outputs and then use those outputs.

Threading and multitasking support are not included in the framework because such features would need less portable code and operating system dependencies. However, multi-tenancy is supported. The framework can run multiple models, as long as they do not need to run at the same time.

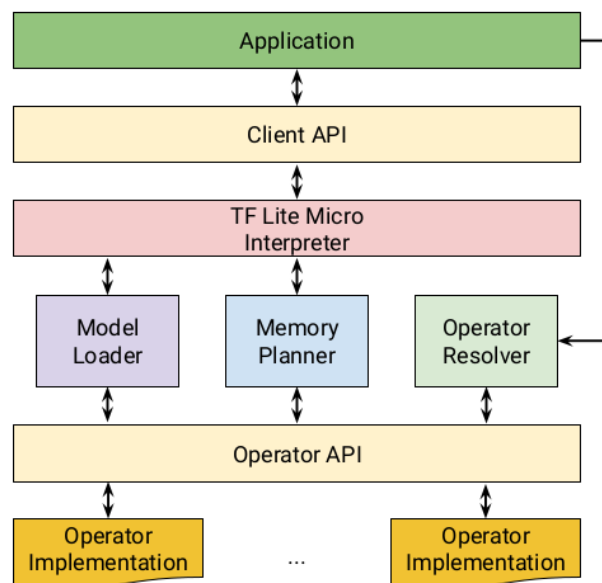


Figure 3.4 - TensorFlow Micro architecture

Despite the fact that TensorFlow Lite Micro was designed to address the unique challenges of microcontroller development, the standard TensorFlow Lite framework may be easier to integrate when working with more powerful devices (e.g., an embedded Linux device). Indeed, as stated in the technical documentation⁷², the following limitations should be considered:

- Support for a limited subset of TensorFlow operations, which limits the architectures that is possible to run. The list of supported operations can be found in the file `all_ops_resolver.cc` in the Tensorflow repository⁷³
- Support for a limited set of devices
- Low-level C++ API requiring manual memory management
- On device training is not supported.

PyTorch Mobile

PyTorch Mobile⁷⁴ is a new Facebook framework, which stays entirely in the PyTorch ecosystem, for deploying models on edge devices. PyTorch Mobile provides an end-to-end workflow which simplifies the production for mobile devices. At the moment it is in beta stage, but it will be available soon as a stable release. Among the main advantages is the capability of directly converting a PyTorch model to a mobile-ready format without the need of using other tools or frameworks. Furthermore, its initial release supports several quantization techniques to shrink model sizes, i.e., post-training quantization, dynamic quantization, and quantization-aware training. In Figure 3.5 a typical workflow for mobile deployment is reported.

⁷² <https://www.tensorflow.org/lite/microcontrollers>

⁷³ <https://github.com/tensorflow/tensorflow>

⁷⁴ <https://pytorch.org/mobile/home/>

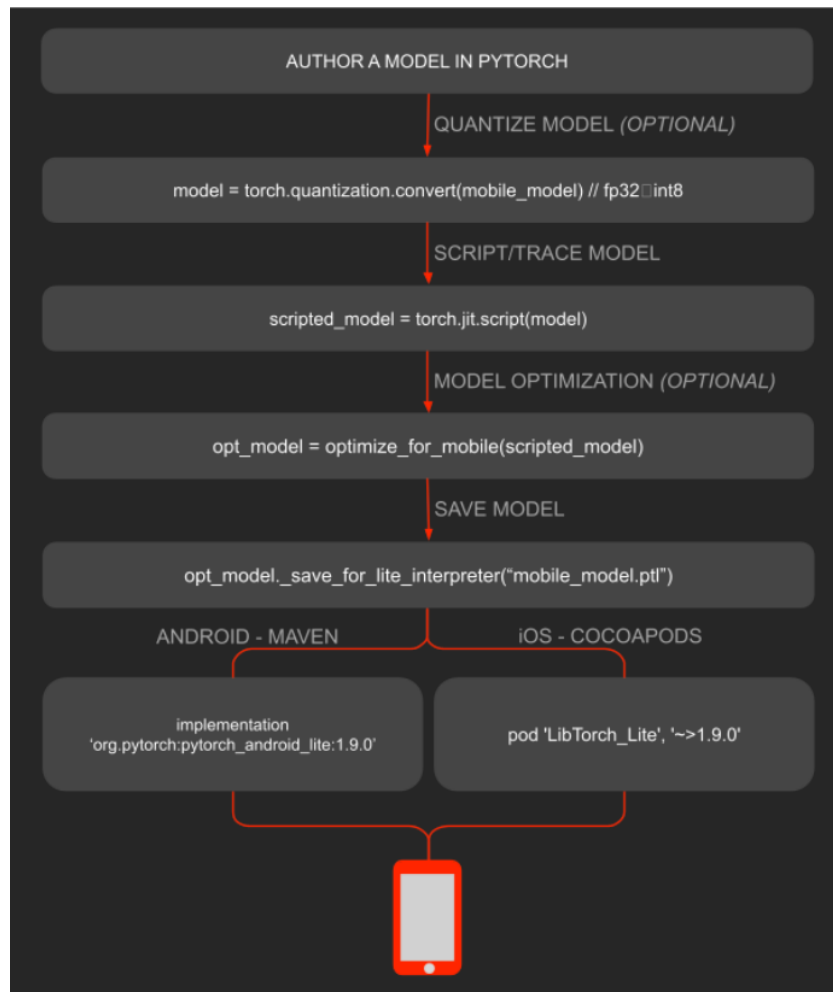


Figure 3.5 - PyTorch Mobile: workflow for mobile deployment (from the project website¹²)

The current experimental build is limited to CPU, which limits the inference speed. Recently, the following features have launched in prototype: GPU support on iOS, GPU support on Android, DSP and NPU support on Android.

Apache TVM

Apache TVM⁷⁵ is an open-source ML compiler framework for CPUs, GPUs, and ML accelerators. The Apache TVM Project’s vision is to host a diverse community of experts and practitioners in ML, compilers, and systems architecture to create an open-source framework that is accessible, extensible, and automated for optimizing current and emerging ML models for any hardware platform. The main features of TVM are:

- Compilation of DL models into minimum deployable modules
- Infrastructure to automatic generate and optimize models on more backend with better performance

⁷⁵ Figure source: <https://tvm.apache.org/docs/>

In this section, we describe the process required to deploy an application in TVM, which is schematized in Figure 3.6:

- *Model import:* TVM can import models from other frameworks, such as ONNX, Tensorflow, or PyTorch, in the importer layer. Model conversion in Relay, TVM's high level model language, represents a model that has been imported into TVM. Relay is a NN functional language and intermediate representation. It is compatible with traditional data flow-style representations and functional-style scoping (let-binding) which makes it a fully featured differentiable language. Moreover, it allows the user to mix the two programming styles. Relay performs several high-level optimizations on the model before running the Relay Fusion Pass. TVM includes a Tensor Operator Inventory with predefined templates of common computations to aid in the conversion process.
- *Model lowering in tensor expression:* Lowering is the transformation of a higher-level representation into a lower-level representation. Relay Fusion Pass reduces the model from the higher-level Relay representation to a smaller set of sub-graphs, with each node representing a task. A task is a collection of computation templates expressed in Tensor Expression, where the template parameters control how the computation is executed on hardware. A schedule is the specific ordering of computation defined by parameters to the Tensor Expression template.
- *Tuning:* Tuning is the process of searching the Tensor Expression parameter space for an optimized schedule for the target hardware. There are several optimization options available, each of which necessitates varying degrees of user interaction. Among the optimization options are:
 - *AutoTVM:* A search template is specified by the user for the schedule of a Tensor Expression task or Tensor Expression sub-graph. To produce an optimized configuration, AutoTVM directs the search of the parameter space defined by the template. As part of the TVM Operator Inventory, AutoTVM requires users to define manually templates for each operator.
 - *Ansor/Auto-Schedule:* Ansor can automatically search an optimization space with much less intervention and guidance from the end user by using a TVM Operator Inventory (TOPI) of operations. Tensor Expression templates are used by Ansor to guide the search.
- *Choosing the optimal configuration for the model:* Following tuning, an optimal schedule for each task is selected. Regardless of whether it is AutoTVM or AutoSchedule, schedule records in JSON format are generated and used by this step to build an optimized model.
- *Lowering to a hardware specific compiler:* Following the selection of an optimized configuration based on the tuning step, the model is reduced to the representation expected by the target compiler for the hardware platform. This is the final phase of code generation, to produce an optimized model that can be deployed in production. TVM supports a variety of compiler backends, including:
 - *LLVM:* targeting arbitrary microprocessor architecture including standard x86 and Arm processors, AMDGPU and NVPTX code generation, and any other LLVM supported platform.
 - *Specialized compilers:* such as NVCC, NVIDIA's compiler.
 - *Embedded and specialized targets:* which are implemented through TVM's Bring Your Own Codegen framework.

The compiler-specific generated code can then be reduced to machine code at the end of this process.

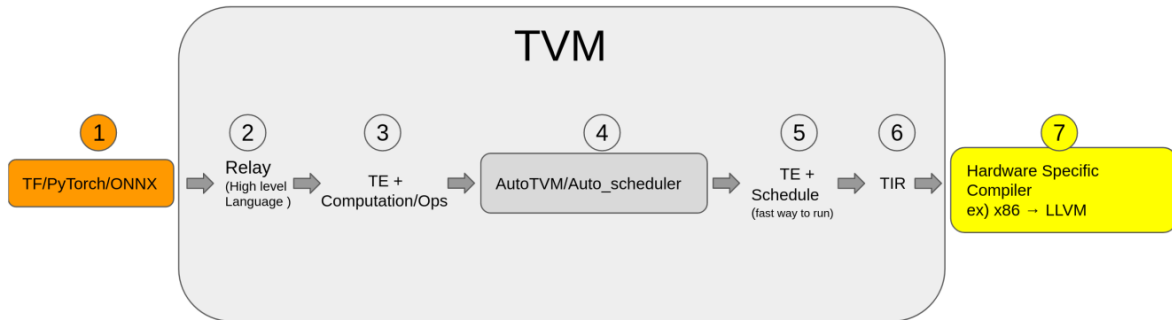


Figure 3.6 - TVM optimizing compiler framework workflow

3.2.4 Vendor-specific Deep Learning Frameworks

Intel OpenVINO

OpenVINO⁷⁶ by Intel is available as a comprehensive toolkit for rapidly developing applications and solutions that solves a variety of tasks. Based on the latest generations of NNs, including CNNs models, recurrent and attention-based networks, the toolkit extends computer vision and non-vision workloads across Intel hardware, maximizing performance. It accelerates applications with high-performance, AI, and DL inference deployed from the edge to the cloud.

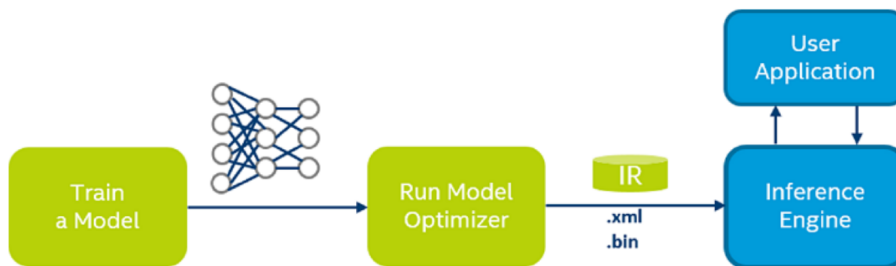


Figure 3.7 - Intel OpenVINO workflow

The OpenVINO workflow consists of four steps and reflects some similarity with the TensorFlow Liteworkflow, described previously (see Figure 3.7).

- **Model preparation:** analogously with what has been discussed with TensorFlow Lite, there are many ways to prepare and train a DL model. In addition to the model preparations already discussed in the previous section, OpenVINO allows to download a pre-trained model from the Open Model Zoo, which includes DL solutions to a variety of vision problems, including object recognition, face recognition, pose estimation, text detection, and action recognition, at a range of measured complexities.
- **Model conversion and optimization:** one of the core components of OpenVINO is the Model Optimizer: a cross-platform command-line tool that converts a trained NN from its source framework to an open-source, nGraph-compatible Intermediate Representation (IR) for use in inference

⁷⁶ <https://docs.openvinotoolkit.org/latest/index.html>

operations. The Model Optimizer imports models trained in popular frameworks like Caffe, TensorFlow, MXNet, Kaldi, and ONNX and performs a few optimizations to remove excess layers and group operations when possible into simpler, faster graphs. The Intermediate Representation is a pair of files describing the model:

- .xml: describes the network topology
- .bin: contains the weights and biases binary data

If a NN model contains layers that are not in the list of known layers for supported frameworks, both the conversion and optimization process adjust them through the use of Custom Layers allowing the end-user to plug in its implementation for existing or completely new operations.

- *Running and tuning inference:* The other core component of OpenVINO is the Inference Engine, which manages the loading and compiling of the optimized NN model, runs inference operations on input data, and outputs the results. Inference Engine can execute synchronously or asynchronously, and its plugin architecture manages the appropriate compilations for execution on multiple Intel devices, including both workhorse CPUs and specialized graphics and video processing platforms. OpenVINO Tuning Utilities can be used with the Inference Engine to trial and test inference on a model. The benchmark utility utilizes an input model to run iterative tests for throughput or latency measures, and the Cross-Check Utility compares the performance of differently configured inferences. There is, in addition, a Post-Training Optimization Tool designed to accelerate the inference of DL models by employing specific methods without model retraining or fine-tuning, like post-training quantization. To apply post-training algorithms the tool requires the model to be converted into the OpenVINO IR and a calibration dataset representing a use case scenario.
- *Packaging and deployment:* Once the model is optimized for inference, OpenVINO outputs the relative run-time package for a variety of devices which span from the Intel CPUs to Intel Processor Graphics to the Intel Neural Compute Stick 2 and Vision Accelerator Design with Intel Movidius Visual Processing Units (VPUs), see also Section 3.5.5. Besides this, a deployment manager is available for Python CLI to accommodate other specialized needs.

NVIDIA TensorRT

The NVIDIA TensorRT⁷⁷ is a C++ library that facilitates high-performance inference on NVIDIA GPUs. TensorRT can optimize and deploy applications to the data centre, as well as embedded and automotive environments (Figure 3.8). It is designed to work in a complementary fashion with training frameworks such as TensorFlow, that has integrated TensorRT so that it can be used to accelerate inference within the framework.

⁷⁷ <https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html>

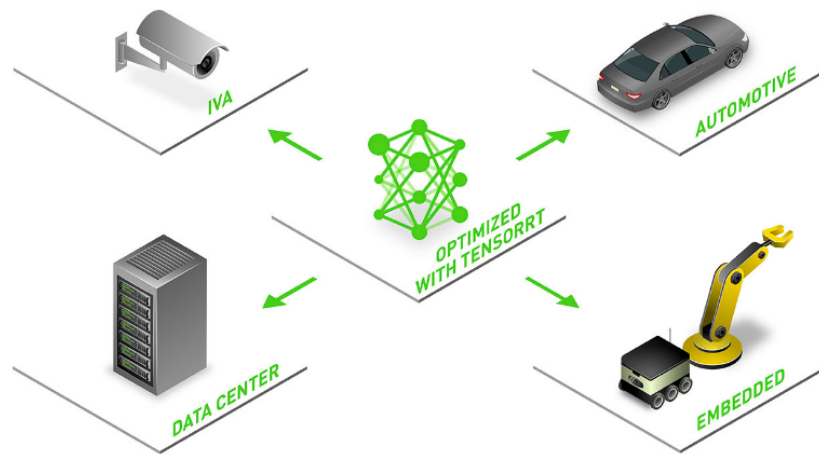


Figure 3.8 - NVIDIA TensorRT accelerates every inference platform⁷⁸

Alternatively, TensorRT can be used as a library within a user application: users can run custom layers through TensorRT using the Plugin interface. The GraphSurgeon utility provides the ability to map TensorFlow nodes to custom layers in TensorRT, thus enabling inference for many TensorFlow networks with TensorRT. It includes parsers for importing existing models from Caffe, ONNX, or TensorFlow, and C++ and Python APIs for building models programmatically.

TensorRT focuses specifically on running an already-trained network quickly and efficiently on a GPU for the purpose of generating a result. It optimizes the network by combining layers and optimizing kernel selection for improved latency, throughput, power efficiency, and memory consumption. If the application specifies, it will additionally optimize the network to run in lower precision, further increasing performance and reducing memory requirements.

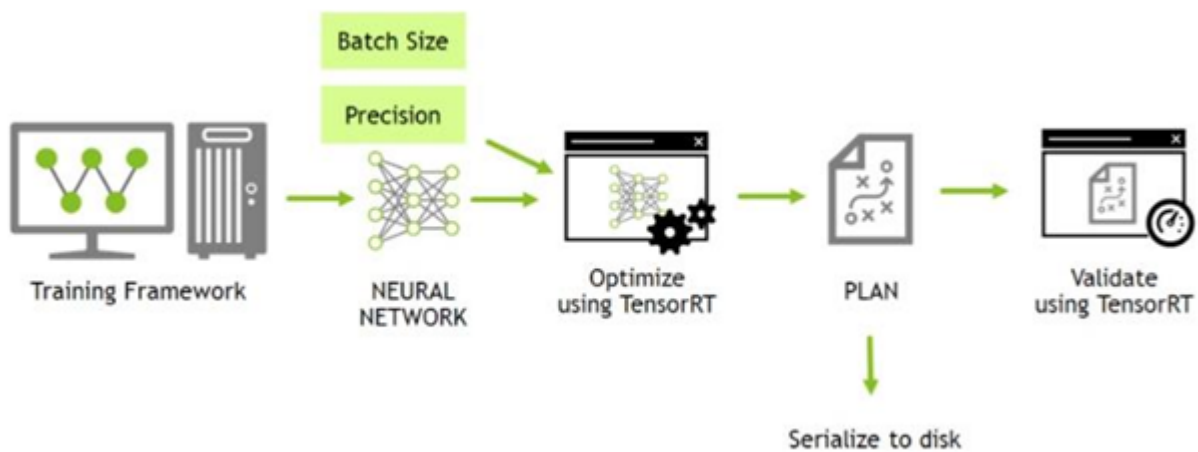


Figure 3.9 - NVIDIA TensorRT workflow⁷⁹

⁷⁸ <https://developer.nvidia.com/tensorrt>

⁷⁹ Figure source: <https://docs.nvidia.com/deeplearning/tensorrt/quick-start-guide/index.html>

The NVIDIA TensorRT Documentation defines the workflow (depicted in Figure 3.9) for developing and deploying a DL model split in three steps:

- *Training:* Like the previous two platforms in development and deployment, the training phase does not involve the adoption of TensorRT. So, during the training phase, the data scientists and developers will start with a statement of the problem and then they will design the structure of the network and train the model. During training, they will monitor the learning process which may provide feedback which will cause them to revise the loss function, acquire or augment the training data. At the end of this process, they will validate the model performance and save the trained model.
- *Developing a deployment solution:* TensorRT documentation treats this part with multiple sub-steps, described in the following:
 - As a first step, in this case, it is highly suggested to understand what priority is willing to be given to the wanted implementation since there are a diverse number of systems that might incorporate the trained model.
 - Once the architecture of the inference solution is defined, an inference engine from the saved network using TensorRT can be implemented. There are a number of ways to do this depending on the training framework used and the network architecture. Generally, this means to take the saved NN and parse it from its saved format into TensorRT using the ONNX parser, Caffe parser, or UFF parser.
 - After the network is parsed different optimization options can be adopted — batch size, workspace size, mixed precision, and bounds on dynamic shapes. These options are chosen and specified as part of the TensorRT build step where an optimized inference engine is built, based on the chosen network.
 - Once created an inference engine using TensorRT, the next step is the validation that reproduces the results of the model as measured during the training process. FP32 or FP16 choices should match the results quite closely but, selecting 8-bit floating point integers, there may be a small gap between the accuracy achieved during training and the inference accuracy.
 - The last phase consists in writing out the inference engine in a serialized format. This is also called a plan file.
- *Solution deployment:* When a plan file containing an optimized inference model is available, the next step consists of deploying that file into a production environment. How to create and deploy the plan file will depend on the specific environment. The TensorRT library will be linked to the deployment application which will call into the library when it wants an inference result. To initialize the inference engine, the application will first deserialize the model from the plan file into an inference engine. TensorRT is usually used asynchronously, therefore, when the input data arrives, the program calls an enqueue function with the input buffer and the buffer in which TensorRT should put the result. In an embedded scenario, the deployment process involves the following steps:
 - Export the trained network to a format such as UFF or ONNX which can be imported into TensorRT
 - Write a program that uses the TensorRT C++ API to import, optimize, and serialize the trained network to a plan file
 - Build and run *make_plan* on the host system to validate the trained model before its deployment to the target system
 - Copy the trained network to the target system. Re-build and re-run the *make_plan* program on the target system to generate a plan file

STM32Cube.AI

STMicroelectronics allows to deploy pre-trained NNs thanks to STM32Cube.AI⁸⁰ which is an extension pack of the widely used STM32CubeMX configuration and code generation tool enabling AI on STM32 Arm Cortex-M-based microcontrollers. The deployment workflow on such devices includes the following steps:

- *Train the NN model:* as for TensorRT, training is performed using off-the-shelf DL frameworks. Among STM32Cube.AI features, the native support for several DL frameworks such as Keras and TensorFlow Lite, and for all the frameworks which can export models to the ONNX format, like PyTorch.
- *Convert NN into optimized code for the STM32 MCU:* the next step is to embed the pre-trained NN into an MCU (optimized code minimizing complexity and memory requirements). The STM32Cube.AI extends STM32CubeMX by providing an automatic NN library generator, the X-CUBE-AI⁸¹, an STM32Cube Expansion Package (see Figure 3.10), that converts pre-trained NNs into a library automatically integrated in the final project. It provides an automatic and advanced NN mapping tool to generate and deploy an optimized and robust C-model implementation of a pre-trained NN (DL model) for the embedded systems with limited and constrained hardware resources. In addition, it provides specific MCU and board filtering to guide users to select the right devices, which fit the requirements, e.g., RAM or Flash memory size, given the users models.
- *Deployment:* the final prototype of the application is obtained through the integrated software packages, i.e., function packs, which include low-level drivers, middleware libraries and sample applications assembled into a single software package. Developers can easily start from these examples and make modifications to fit their specific application.

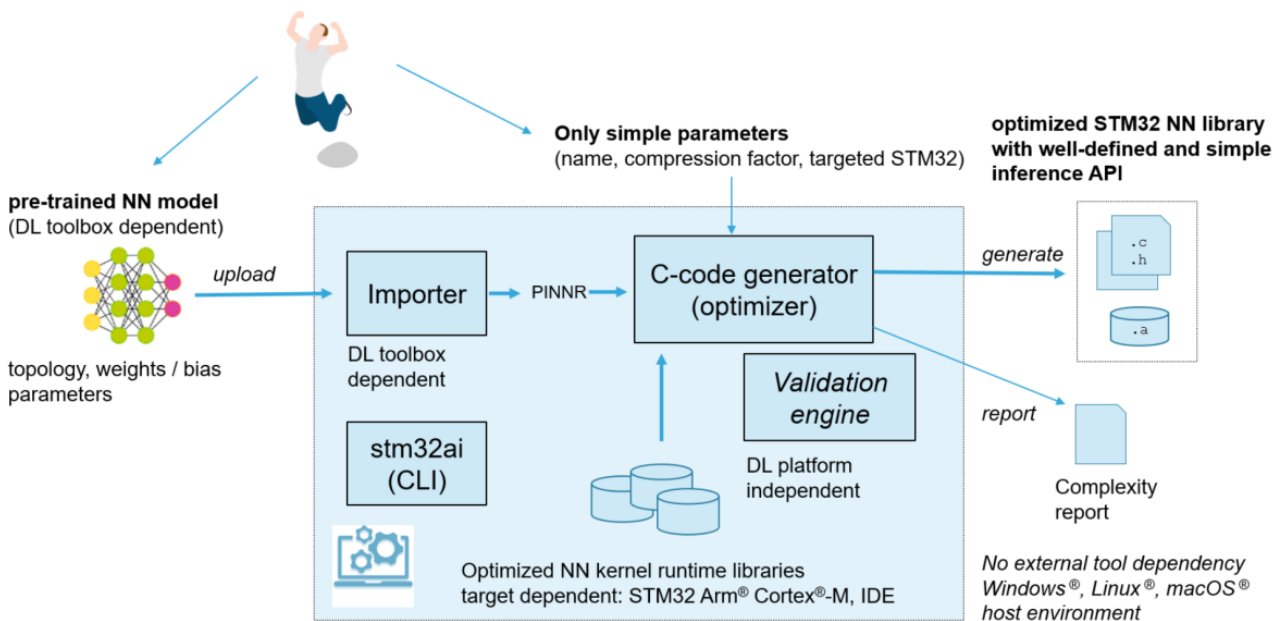


Figure 3.10 - X-CUBE-AI core engine

⁸⁰ https://www.st.com/content/st_com/en/ecosystems/stm32-ann.html

⁸¹ <https://www.st.com/en/embedded-software/x-cube-ai.html>

3.3 PyCOMPSs

Developing applications able to efficiently use distributed infrastructure has some implicit concerns: decomposing the problem into computing units (tasks), map those tasks onto the available compute resource and decide the optimal moment to start the task execution (scheduling), enabling data-sharing mechanisms, handle both the hardware and software specificities of the underlying nodes (portability and interoperability). Edge computing not only has not solved those issues, but it has increased the complexity by adding the handling of network disruptions and handovers.

COMP Superscalar (COMPSs) [Lordan2014] is a framework that aims to ease the development of parallel, distributed applications. The core of the framework is its programming model with which developers write their code in a sequential, infrastructure-unaware manner releasing them from all the above-mentioned concerns. At execution time, a runtime system decomposes the application into tasks and orchestrates their executions on a distributed platform guaranteeing the sequential consistency of the code.

COMPSs' main characteristic is that developers code applications using regular programming languages as if the code was to be run on a single-core computer. For the runtime system to detect the inherent tasks, application developers need to select, using annotations or decorators depending on the language, a set of methods whose invocations become tasks to be executed asynchronously and, potentially, in an out-of-order manner.

To guarantee the sequential consistency of the code, the runtime system monitors the data values accessed by each task -- the arguments, callee object and results of the method invocation -- to find dependencies among them. For the runtime to better-exploit the application parallelism, developers need to describe how the method operates (reads, generates or updates) on each data value by indicating its directionality (IN, OUT, INOUT, respectively). Such data values can be either files, primitive types, objects or even streams.

PyCOMPSs is the Python approach to the COMPSs programming model. Besides the regular task dependency detection and the execution orchestration, it is enabled with novel features such as integration with Numba for just-in-time task compilation or support for data types such as streams, persistent objects, or collections. Moreover, the development environment has some additional tools such as integration with Jupyter Notebook or the PyCOMPSs player, a command line tool building on containers for application testing purposes.

PyCOMPSs integrates dislib [Alvarez2019], a distributed computing library highly focused on machine learning that delegates on PyCOMPSs the distribution of the data processing. Inspired by scikit-learn, dislib provides various supervised and unsupervised learning algorithms through an easy-to-use API and achieves high performance relying on Numpy's mathematical implementations. The cornerstone of dislib is the dsArray: a built-in 2-dimensional array that allows parallel operation and that is used as data input or intermediate data structure by most of the algorithms implemented within the library. Additionally, PyCOMPSs, supports also the European Distributed Deep Learning library (EDDL), a library developed within the context of the DeepHealth European project [Flich2020]. EDDL API is centered around the concepts of Tensor - a class encapsulating all element-wise and linear algebra operations on a matrix - and Neural Network. EDDL allows the parallel and distributed execution of the training of AI models. For doing so, it delegates on PyCOMPSs the distribution of the execution of its algorithms.

3.4 Cloud-based solutions

This section is intended to give an overview of the currently available commercial cloud-based solutions and services related to AI solutions, edge computing, IoT, and FaaS. The goal is to provide a snapshot of what

services are currently available in the market, to position the AI-SPRINT project that aims to define a novel framework for developing and operating AI applications. Therefore, the analysis also includes the main cloud big players, but it is focused on European cloud providers and cloud providers operating in Europe.

The first part of the section provides an overview of the current market-specific situation of a selection of relevant cloud providers with focus on specific components and features related to AI solutions, IoT and edge device management, as well as security aspects. In the second part, the use of Functions as a Service (FaaS) will also be discussed.

Not all providers sell IoT and Edge services. In terms of edge capabilities, also not all providers offer similar features. Some of the key features which would be needed by the AI-SPRINT use cases are:

- Possibility to manage centrally the code which must be deployed on edge devices
- Possibility to manage edge devices using a device shadow (or digital twin) in the cloud (administrators update the device shadow, the synchronization between device shadow and actual device happens automatically when the edge device is connected)
- Possibility to change from the cloud configuration edge device settings
- Possibility to transfer easily data (e.g., files, telemetry) from edge and cloud
- Possibility to encrypt data transferred between edge devices and cloud
- Availability of SDKs which allows to control IoT devices on the edge

According to the current status of the cloud market, cloud providers edge services do not offer any specific AI or ML capabilities, nor performance guarantees in terms of application components execution time. However, assuming that the edge device is powerful enough, inference can be run on the edge device in best-effort. All cloud providers offer proprietary solutions, interoperability among providers is not supported.

3.4.1 Amazon Web Services

AI related Solutions

AWS provides many AI-related and ML-related services⁸². Some examples are:

- Amazon SageMaker, a service that enables the creation, training, and provisioning of machine learning models at scale
- Amazon Augmented AI, a simple implementation of human verification of ML predictions
- Amazon Elastic Inference, accelerating deep learning inference
- Amazon Forecast, increase forecast accuracy using machine learning
- Amazon Healthlake, making sense of health data
- Amazon Rekognition, analyzing images and videos

and much more. Besides the mentioned individual solutions that tackle several specific problems, AWS also offers a service called Amazon SageMaker Autopilot, which completely automates the creation and training of classification or regression machine learning models. In addition, the well-known frameworks such as PyTorch, TensorFlow or Apache MXNet are also provided to be worked with. With the AWS Neuron SDK, Developers can use their own models and integrate them in frameworks like Tensorflow, PyTorch or MXNet.

⁸² <https://aws.amazon.com/machine-learning/ai-services/>

Furthermore, AWS provides the ability to use scalable high-performance infrastructure for compute-intensive machine learning using GPU as a service, harnessing the power of GPUs like NVIDIA A100. Besides that, it is also possible to make use of Amazon EC2 Inf1 instances which offer fast inferencing at low cost, by using ML-specialized AWS Inferentia-Chips. Furthermore, AWS announced to offer AMD GPU instances anytime soon.

Edge Solutions

AWS IoT is a cloud service which allows “things” to publish data to the AWS Cloud. The “things” authenticate with the cloud using a certificate and communicate using HTTP or MQTT. On the cloud side, messages published by the “things” can be routed to other AWS services, i.e., they can be saved, can be used to trigger functions, can be added to a queue, or can be streamed to other services. AWS IoT assumes all things send data in JSON format, the routing of messaging is performed by specifying a set of rules based on the content of the message (e.g., “if device_type is xy and temperature is above 45 degrees, then execute action abc”).

AWS IoT allows managing edge devices using device “shadows”, i.e., device settings can be defined in the cloud, and, as soon as the device is on-line, the shadow and the actual device are synchronized.

Edge support of the IoT device is implemented using AWS IoT Greengrass. The AWS Greengrass core (see Figure 3.11) is a software component which can be installed on supported devices; the software component enables a subset of services to be run directly on the edge (e.g., Lambda function). An edge device manages all the “things” connected to it, but the edge device interacts with the cloud, and it is managed from the cloud. The edge device does not need cloud connection to run.

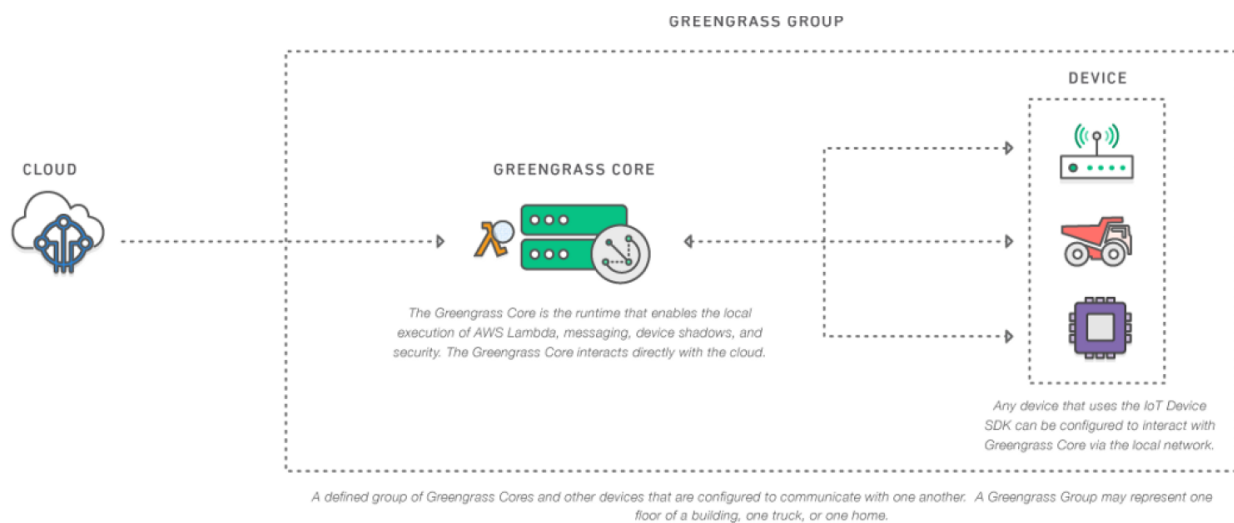


Figure 3.11 - Green Grass Architecture

AWS also runs a service called AWS Outpost, which is basically a server rack with several servers on which AWS services can be run on-premises but managed on the cloud. For Amazon customers, the AWS Outpost server rack is a black box completely managed by AWS, it just needs Internet connection and power supply. As a summary, AWS IoT Greengrass Core:

- Provides local services (compute, messaging, state, security), and communicates locally with devices that run the AWS IoT Device SDK.
- Runs in 64-bit CPU-based devices (x86 or Arm) that run a general-purpose OS such as Linux.

While AWS IoT Greengrass offers the following features:

- An AWS Lambda runtime that allows to execute serverless instructions if needed. The AWS Lambda could also run code which allows AI inference on the cloud.
- A shadow implementation, i.e., each “thing” is associated with a JSON file (the shadow) where all the parameters/variables are set and can be modified with Lambda functions from the core or from the cloud.
- A message manager, which allows “things” to send messages when the core is temporarily not available (e.g., restart or update).
- Possibility to manage edge resources from the cloud by grouping them, a group is composed of the things and the Greengrass Core.
- A discovery service, it is a service mainly used by the things to get certificates to connect to a Greengrass Core.
- An over-the-air update agent that allows updating one or more Greengrass Cores on a network at the same time or on predefined schedules. The devices with the Greengrass core will need to have the WiFi activated for this feature to work.
- Local resources access on the Greengrass Core device if it is needed, the resources can be anything.
- A machine learning inference, meaning that the training is done on the cloud servers, but the model is on the Greengrass device and can perform in real time what it was trained to do.
- Connectors help to implement reusable business logic, interact with cloud and local services (including AWS and third-party services), ingest and process device data, enable device-to-device calls using MQTT topic subscriptions and user-defined Lambda functions. This module works as a package that can be deployed on a core without the hassle of learning new APIs or protocols.

Application development is supported by the AWS IoT Device SDK which:

- Allows devices to interact locally with AWS IoT Greengrass Cores.
- Runs in almost any device that supports C++, Node.js, Java, or Python 2.7, 3.7, and 3.8.

Security Aspects

All data flowing across the AWS Global Network connecting the AWS datacentres and regions is automatically encrypted at the physical layer before it leaves the AWS secured sites. There are other layers of encryption, such as for all cross-region VPC peering traffic and customer or service-to-service TLS connections. AWS provides tools to encrypt data in transit and at rest to ensure that only authorized users can access it.

AWS Greengrass protects user data through the secure authentication and authorization of devices, through secure connectivity in the local network, between local devices and the cloud. Device security credentials function in a group until they are revoked, even if connectivity to the cloud is disrupted, so that the devices can continue to securely communicate locally. Each edge device can be secured using a certificate, which allows authentication and encryption of data in transit. AWS Greengrass does not encrypt data at rest, it relies on the security provided by the underlying Linux system. Any application running on AWS Greengrass can encrypt its data, also using AWS services which store secrets and passwords (AWS Greengrass supports local encryption of passwords or secrets retrieved from the cloud).

In terms of official security measurements, AWS is certified with ISO/IEC 27001 (Information Security), ISO/IEC 27017 (Cloud Security) and ISO/IEC 27018 (Privacy Protection). Additionally, AWS provides a stack of security services which can be combined with other services.

3.4.2 Microsoft Azure

AI related Solutions

Besides AWS, Microsoft Azure also offers a wide variety of AI-related services⁸³ like:

- Azure Cognitive Services, a family of AI-services and cognitive APIs for the development of intelligent apps
- Azure Synapse Analytics, a service made for Big Data analysis
- Azure Machine Learning, end-to-end scalable platform with experimentation and model management
- Azure Computer Vision, out of the box image analysis
- Azure Health Bot, build and deploy AI-powered healthcare services
- Azure Auto ML, to create and train ML models automatically

and many other services for detecting anomalies, wrangling data, or deploying ML pipelines. There is also an option to get a free demo account to test most of the said services for a month with an included budget of 170\$. All services are managed and distributed through the unified Azure platform. Azure provides access to powerful GPUs like NVIDIA K80, P100, P40, M60 and V100 with high compute capabilities for AI related problems.

Edge Solutions

Azure offers a wide set of IoT services: IoT Hub, IoT Central, Time Series Insights, Azure Digital Twins for connectivity and analysis and IoT Edge and Azure Sphere.

The IoT Hub and IoT Central services are similar to AWS IoT. They allow things to publish data to the cloud; messages sent by the cloud are routed to other Azure services based on their content. The messages can be routed to a blob storage, an Azure function, an application running on a Kubernetes cluster etc.

A Digital Twin is the Azure equivalent of the AWS Shadow; it allows reading data and writing configuration on things which might be offline; the synchronization between things and their digital twin is managed by Azure and happens automatically when the things are on-line.

Data sent from things to the Cloud can be indexed, queried, and visualized using the service Time Series Insights. The IoT Edge service allows deploying containers on Edge devices. As in the case of AWS, once a software component is installed on an edge device, the device can be managed centrally from the cloud and a subset of Azure services can be run on the edge devices (e.g., blob storage). IoT Edge also allows the deployment of containers on the cloud; these containers can interact with things attached to the edge device using HTTP or MQTT.

Azure Sphere OS is a custom Linux-based microcontroller operating system created by Microsoft to run on an Azure Sphere-certified chip and to connect to the Azure Sphere Security Service. The Azure Sphere OS provides a platform for IoT application development, including both high-level applications and real-time capable applications.

⁸³ <https://azure.microsoft.com/de-de/services/machine-learning/>

Security Aspects

The security model of IoT Edge is similar to Greengrass. The edge device authenticates and encrypts data in transit by using a certificate. Data stored using the Edge IoT services (e.g., a file stored on an Azure Blob) is not encrypted by default, Edge IoT relies on the underlying Linux system to encrypt data and manage permissions at file level. Azure Sphere devices do have enhanced security, they can also be managed by an Azure IoT Edge device.

IoT Edge supports several standardized device management patterns, such as Reboot, Factory Reset, Configuration Change, Firmware Upgrade, Reporting progress and status. The device management capabilities are part of the IoT Hub service, they are not specific to the Edge IoT devices, but they also apply to them.

In terms of official security measurements, Azure is certified with ISO/IEC 27001 (Information Security), ISO/IEC 27017 (Cloud Security), ISO/IEC 27018 (Privacy Protection) and ISO/IEC 27701 (Privacy Information Management System). Additionally, Azure provides a stack of security services which can be combined with other services.

3.4.3 Google Cloud Platform

AI related Solutions

The solutions and services for Machine Learning and AI technologies of Google are similar to those of Amazon and Microsoft. There are several services which one can use to develop, train, test or deploy AI related solutions and services. The main ones are:

- AutoML, to create and train custom ML models automatically
- Vision AI, pretrained models to identify or detect text, emotions, images and more
- Video AI, to detect and classify videos
- Cloud Natural Language, classification for unstructured text
- Text-to-speech, speech synthesis in multiple languages and voices
- Speech-to-Text, speech recognition and transcription
- Deep Learning Containers, preconfigured and optimized containers for deep learning environments

and more. On an infrastructure level, AI optimized, and cost-effective hardware is available to support training and inference of AI models. Google provides access to powerful GPUs like NVIDIA K80, P100, P4, T4, V100, and A100 or TPU (Tensor Processing Unit) with even higher compute capabilities for AI related problems. One TPU v2 outperforms 8 NVIDIA v100 GPUs with up to 27x lower training times and at lower cost, according to Google. As a last remark, GPUs are available as virtual instances, therefore not at bare metal.

Edge Solutions

The IoT offering of Google Cloud is similar to those of AWS and Azure. Google Cloud IoT service can be used to manage things, collect data from them, update their configuration and visualize collected data. The service uses a pub/sub communication architecture, enabling it to subscribe to specific data sent by things. The service allows central configuration of the things (similar to Digital Tween or Shadow Device used by Azure and AWS, respectively).

Edge support is similar to the one offered by Azure: a runtime environment is installed on the Edge device, which can then be managed from the Cloud. Edge devices can run containers which can be deployed centrally.

Security Aspects

Device life-cycle management is managed using a device registry in the cloud. Google offers a set of tools to provision and maintain edge devices, similar to AWS and Azure. Edge device authentication is managed using certificates. Data is encrypted on transit; local encryption relies on the capabilities of the edge devices' underlying OS.

In terms of official security measurements, Google Cloud Platform is certified with ISO/IEC 27001 (Information Security), ISO/IEC 27017 (Cloud Security) and ISO/IEC 27018 (Privacy Protection). Additionally, Google provides a stack of security services which can be combined with other services.

3.4.4 IBM

AI related Solutions

Similar to Amazon, Google and Microsoft IBM is a full stack cloud provider. Therefore, in terms of AI services, IBM is also offering a bunch of AI related services. Namely there are services like:

- IBM Watson Assistant, provide virtual assistants on any website
- IBM Watson Language Translator, API for translation with domain-specific models
- IBM Watson Natural Language Classifier, Visual tool and API for text classification
- IBM Watson Speech-to-Text and Text-to-Speech
- IBM Watson Machine Learning, Infrastructure for running AI models at scale
- IBM Streaming Analytics, Dashboard for real-time analysis of data streams

and more. As IBM offers those services, it needs to be stated that the given services are not as diverse as those from Amazon, Google and Microsoft. It lacks especially in AutoML and visual AI capabilities. From an infrastructural view, IBM offers vGPU instances as well as bare metal GPU instances like NVIDIA V100, P100 or T4.

Edge Solutions

IBM offers some Edge/IoT services such as:

- IBM Edge Application Manager, an autonomous management platform for edge computing
- IBM Watson IoT Platform, SaaS for device management, monitoring and data storage.

No information is available on edge SDKs or centralized edge specific code management.

Security Aspects

IBM provides a fairly big stack of security services. for example:

- IBM Cloud Data Shield, Runtime encryption for data-in-use protection on Kubernetes clusters

- IBM Cloud Certificate Manager, Certificate management for cloud resources
- IBM Cloud Hardware Security Module, Tamper-resistant hardware to store and process cryptographic keys
- IBM Cloud Security and Compliance Center, SaaS to define and audit the compliance posture of your cloud
- IBM Cloud Security Advisor, Dashboard for security management, analysis and remediation

and more. In terms of official security measurements, IBM is certified with ISO/IEC 27001 (Information Security), ISO/IEC 27017 (Cloud Security) and ISO/IEC 27018 (Privacy Protection).

3.4.5 Oracle

AI related Solutions

Oracle is another big cloud provider, which recently improved its AI service and tool portfolio. Among others, Oracle offers the following solutions:

- Oracle AutoML, to create and train custom ML models automatically
- Oracle Data Analysis Platform, featuring ready to use cloud infrastructure, analysis and visualization for data science with dozens of tools/frameworks
- Oracle Integration and Migration, to efficiently prepare and ingest data

and more additional helper tools and services. Currently Oracle offers GPU as a Service options like NVIDIA V100 and P100. Oracle plans to offer additional GPU acceleration options in their cloud environments, for example with the NVIDIA A100. However, currently there is no GPU acceleration available for Oracle AutoML or other high-level services.

Edge Solutions

Oracle offers a PaaS solution named IoT Cloud to manage Edge/IoT devices centrally. It supports features like gateway management, direct connections, the management of software on the devices, the monitoring of IoT device metrics and the setting of the security features. Oracle uses REST and Oauth for connectivity.

Security Aspects

In terms of security features, Oracle offers many services and tools. There are for example services to secure the clouds infrastructure like:

- autonomous Linux, eliminate complexity and human error to reduce cost, increase security and availability, automatic updates and security improvements
- key management
- root of trust
- isolated network virtualization

whereas there are also services to ensure cloud application security like:

- Identity Cloud Services
- Risk Management Cloud, automate advanced security and transaction monitoring to strengthen financial controls, ensure separation of duties (SoD), stop fraud, and streamline audit workflows
- CASB (cloud access security broker), gain visibility and detect threats on the entire cloud stack for workloads and applications, for example, via real-time threat intelligence feeds and use predictive analytics to manage threats.

In terms of official security measurements, Oracle is certified with ISO/IEC 27001 (Information Security), ISO/IEC 27017 (Cloud Security), ISO/IEC 27018 (Privacy Protection).

3.4.6 CloudSigma

AI related Solutions

According to our analysis, at current state, the German cloud provider CloudSigma does not provide any AI related services, apps or platforms. Furthermore, there are not any offerings to use GPUs or any other AI acceleration hardware.

Edge Solutions

There are no special offerings or services which provide edge solution support or deeper management characteristics. However, it is possible to connect and send data from several IoT devices to the CloudSigma platform via REST.

Security Aspects

For the REST functionalities, CloudSigma uses HTTP Basic authentication. No more special security measures could be identified. However, in terms of official security measurements CloudSigma is certified with ISO/IEC 27001 (Information Security), ISO/IEC 27017 (Cloud Security) and ISO/IEC 27018 (Privacy Protection).

3.4.7 1&1 Ionos

AI related Solutions

There are no known AI services available. Also, no GPU acceleration capabilities are offered by the German cloud provider 1&1 Ionos.

Edge Solutions

As far as the analysis we performed shows, there are no Edge/IoT related services.

Security Aspects

In terms of official security measurements, 1&1 Ionos is certified with ISO/IEC 27001 (Information Security).

3.4.8 Scaleway

AI related Solutions

This French cloud provider offers some AI related services such as:

- Machine Learning Images, ready-to-go Data Science environment with pre-installed Anaconda, Scikit, Nvidia Rapids, and Pytorch
- AI Inference, which allows anyone with a trained model to deploy it in a Scaleway environment without DevOps skills.

Scaleway also provides GPU acceleration at the IaaS level, but only the NVIDIA Tesla P100 is available.

Edge Solutions

Scaleway provides the IoT Hub service in which it is possible to manage centrally all connected IoT devices and to make use of features like:

- IoT Routes, a gateway to other cloud services or third-party cloud providers with object storages, databases or REST routes
- IoT Networks, use protocols like MQTT, REST, Sigfox and LoRa
- Metrics, device or hub level usage overview (message counter and activity)
- Virtual Devices, generate virtual representations of real devices and use them with all IoT Hub services
- Identification and Security, manage identification and security of devices with three modes, Mutual-authentication TLS, Server-authentication TLS or Plain (no security)

Security Aspects

Scaleway is providing various security measurements such as TLS via MQTT on IoT level or complete data encryption of cloud servers. In terms of official security measurements, Scaleway is certified with ISO/IEC 27001 (Information Security).

3.4.9 OVHcloud

AI related Solutions

This French cloud provider offers some AI related services, for example:

- OVHcloud AI Training, out-of-the-box training environment with Kubernetes orchestration
- OVHcloud ML Serving, ML service deployment API
- NVIDIA NGC Platform, containerized NVIDIA GPUs.

Furthermore, OVHcloud provides some GPU as a Service capabilities with up to 4 NVIDIA Tesla V100 GPUs. However, there are no more GPU options to choose from.

Edge Solutions

OVHcloud provides an integrated platform to manage IoT devices and edge nodes. The connection to these nodes is based on ssh, but very limited documentation is reported on the provider web site.

Security Aspects

In terms of official security measurements, OVHcloud is certified with ISO/IEC 27001 (Information Security), ISO/IEC 27017 (Cloud Security), ISO/IEC 27018 (Privacy Protection) and ISO/IEC 27701 (Privacy Information Management System). Furthermore, OVHcloud also provides health data related compliance to ensure that health data is securely processed and stored.

3.4.10 Exoscale

AI related Solutions

This Swiss cloud provider does not provide any AI related services. However, it offers GPU acceleration with up to 4 NVIDIA V100 virtual on demand GPU instances.

Edge Solutions

There are rudimentary edge related services like centralized edge device management, load balancing, private networks, or elastic IPs.

Security Aspects

No specific services could be identified. In terms of official security measurements, Exoscale is certified with ISO/IEC 27001 (Information Security), ISO/IEC 27017 (Cloud Security), ISO/IEC 27018 (Privacy Protection). Additionally, Exoscale also is certified with the CSA Star.

3.4.11 Fuga

AI related Solutions

Fuga is a cloud provider located in the Netherlands. However, for the time being, no AI services or GPU acceleration capabilities are provided.

Edge Solutions

No edge related services are offered by the Dutch company.

Security Aspects

No specific services could be identified. In terms of official security measurements, Exoscale is certified with ISO/IEC 27001 (Information Security).

3.4.12 Linode

AI related Solutions

No AI related services are offered by the US American cloud provider. In terms of GPU acceleration, there is a small offering with NVIDIA RTX Quadro 6000 series GPUs. Note that, compared to other European cloud providers which offer P100, V100 or A100 GPUs, the RTX Quadro 6000 is less performant.

Edge Solutions

No edge related services are offered.

Security Aspects

No specific services could be identified. In terms of official security measurements, Linode is certified with ISO/IEC 27001 (Information Security).

3.4.13 Vultr

AI related Solutions

No AI services or GPU acceleration capabilities are offered by this US American cloud provider.

Edge Solutions

No edge related services could be identified.

Security Aspects

No specific services could be identified. There are also no known security related certifications achieved by this company.

3.4.14 Open Telekom Cloud

AI related Solutions

This German cloud provider offers only the following AI related service:

- ModelArts-Beta, a platform to design, develop and train machine learning models

In terms of general analytics, Data Ingestion Service and Data Warehouse Service are offered. The provider supports GPU acceleration at IaaS level with Nvidia T4.

Edge Solutions

In terms of edge management, a service called Intelligent Edge Fabric (IEF) is offered. The company website states that the service will be available soon (<https://open-telekom-cloud.com/en/products-services/intelligent-edge-fabric>). However, according to the official documentation, feature services like Edge-node management, Edge-application management and automatic application scheduling will be provided.

Security Aspects

Some basic services like Firewall-as-a-Service, a Key management service and a web application firewall are provided. In terms of official security measurements, Open Telekom Cloud is certified with ISO/IEC 27001 (Information Security), ISO/IEC 27017 (Cloud Security), ISO/IEC 27018 (Privacy Protection).

Table 3.2 summarizes the core aspects of this analysis at a glance, whereas red fields mean no provision at all, yellow means partial provision and green means that a big stack, a lot of services or various options are available.

	AI related services	GPU as a Service	Edge Solutions	Security Aspects
AWS	Green	Green	Green	Green
Azure	Green	Green	Green	Green
Google	Green	Green	Green	Green
Cloudsigma	Red	Red	Yellow	Yellow
IBM	Green	Green	Yellow	Green
1&1 Ionos	Red	Red	Red	Yellow
Scaleway	Yellow	Yellow	Green	Yellow
OVHcloud	Yellow	Yellow	Yellow	Yellow
Oracle	Green	Green	Green	Green
Exoscale	Yellow	Yellow	Yellow	Yellow
Fuga	Red	Red	Red	Yellow
Linode	Red	Yellow	Red	Yellow
Vultr	Red	Red	Red	Red
Telekom	Yellow	Yellow	Red	Yellow

Table 3.2 - Cloud Provider comparison according to AI services and edge system support capabilities

In conclusion, the big players in cloud computing like AWS, Microsoft Azure, Google, Oracle and IBM are offering various services options for AI related and/or edge focused development with good GPU acceleration capabilities by simultaneously providing a solid security. Smaller cloud providers like Scaleway, OVHcloud, Exoscale and Open Telekom Cloud are also solid providers, but with some limited support to AI services and

edge systems. Finally, some providers such as 1&1 Ionos, Cloudsigma, Fuga, Linode, and Vultr do not offer enough features and services, at least in the areas discussed, and cannot be considered data science or AI-driven cloud providers.

3.4.15 FaaS

Cloud computing has become in the last decade the main option for virtualized computing. It has achieved increasing both hardware utilization and the ability to execute disparate computing workloads with complex requirements on shared computing infrastructures. Initial service delivery models, such as Infrastructure as a Service (IaaS), are exemplified by public Cloud services such as Amazon EC2 and on-premises Cloud Management Platforms (CMPs) such as OpenStack. These were later extended to accommodate additional models such as Platform as a Service (PaaS) and, more recently, Functions as a Service (FaaS) in the quest for rising the level of abstraction for application developers at the expense of relying on the infrastructure provider for automated elasticity, efficient virtual infrastructure provisioning and improved resource allocation.

Pioneer FaaS services, exemplified by public Cloud services such as AWS Lambda [AWSLambda] and Azure Functions [AzureFunctions], provide event-driven execution of functions coded in some programming languages supported, offering automated resource allocation, and highly elastic capabilities that superseded the ones found in traditional IaaS offerings. For example, a Lambda function can support up to 3000 concurrent executions. This could only be achieved by using lightweight virtualization technologies, as is the case of Firecracker [FireCracker] which allows deploying micro Virtual Machines (microVMs) in non-virtualized environments in less than a second.

Initially, AWS Lambda functions only supported certain programming languages. This required a cumbersome migration procedure to execute scientific applications on said service, involving additional programming. Still, scientific applications can benefit from the high elasticity provided by serverless computing and the scale-to-zero capability that incurs in no cost unless the functions are being executed. The combination of both features allows to deploy elastic applications that are latently available in the Cloud at a zero cost and that can rapidly and automatically provision the required computing resources in order to cope with sudden workload spikes.

The increased popularity of serverless computing together with the increased maturity of Container Orchestration Platforms paved the way for the surge of multiple open-source developments mimicking the functionality offered by public serverless offerings. This is the case of OpenFaaS, Apache OpenWhisk, Kubeless, KNative and Fission, to name just a few. These frameworks provide the ability to create user-defined functions typically packaged as Linux containers that can be invoked on-demand and the execution takes place coordinated by an orchestration platform such as Kubernetes. However, mimicking the functionality offered by public serverless offerings leaves out support for complex scientific applications for data processing, with large dependencies, strict execution requirements and that may eventually need accelerated hardware such as GPUs.

Among the cloud providers reviewed in the previous section, FaaS services are provided by the big players (AWS, Azure, IBM, and Oracle), while in Europe only Scaleway and OVHcloud provide some FaaS functionalities. In particular, Scaleway serverless platform allows users to deploy their FaaS and Containerized Applications (CaaS) in a managed infrastructure, while OVHcloud provides a managed Kubernetes system based on OpenFaaS (an open-source framework for building Serverless functions with Docker and Kubernetes).

3.5 Technologies and Solutions for Edge Intelligence

In the last years, we are witnessing advances in both edge computing and AI technology. While the latter demonstrated its capability of exploiting big data to reach state-of-the-art performance in various fields, the former allowed to push services, intended as both computation and data storage, to the proximity of the end users, i.e., the edge nodes of the networks. The synergistic interaction between AI and edge computing was an unavoidable step that resulted in a new paradigm defined as edge intelligence (EI) [Li2018][Wang2019][Xu2020], intended as a set of connected systems and devices for, as discussed in Section 2.1.2, data collection, caching, processing, and analysis. The key aspect of this new interdisciplinary is that the analysis of the data is performed locally, i.e., to the proximity to where it is collected, with the aim of speeding up the processing and to ensure the privacy and security of data. In this context, AI can play an important role, by customizing Machine Learning (ML) and Deep Learning (DL) models to take advantage of edge big data, thus enhancing the quality of the analysis. However, existing ML/DL models are computationally intensive and require specific resources, e.g., CPU, GPU, memory, and network. In the following, we provide an up-to-date overview of the embedded systems capable of hosting ML/DL models and of supporting common learning frameworks and tools such as Tensorflow, Tensorflow Lite, PyTorch, Caffe and ONNX (see Section 3.2).

3.5.1 General Purpose Arm SoCs

A system on a chip (SoC) is an integrated circuit (also known as a "chip") that integrates all or most components of a computer. These components almost always include a central processing unit (CPU), memory, input/output ports and secondary storage, often alongside other components such as radio modems and a graphics processing unit (GPU) – all on a single substrate or microchip. We analyse the most widely adopted solutions in the following tables. In particular, the boards are compared specifying the included processors, accelerators, integrated memory, and communication interfaces. Furthermore, the supported OS, Containers (e.g., Docker and BalenaOS), ML frameworks, and tools described in Section 3.2 are reported.

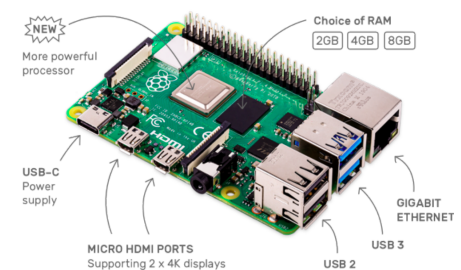
<p>Raspberry Pi 4 B</p>	
<p>General description</p>	<p>The Raspberry Pi 4 Model B is the latest version in the Raspberry Pi mini-computer series, and it is a very competitive platform for machine learning inferencing at the edge.</p>
<p>SoC:</p>	<ul style="list-style-type: none"> ● Broadcom BCM2711, Quad core Cortex-A72 (Arm v8) 64-bit @ 1.5GHz ● VideoCore VI @ 500 MHz
<p>Accelerator</p>	<p>-</p>
<p>Memory</p>	<p>2GB, 4GB or 8GB LPDDR4-3200 SDRAM</p>
<p>Connectivity</p>	<p>2.4 GHz and 5.0 GHz 802.11ac Wireless, Bluetooth 5.0, BLE, Gigabit Ethernet</p>
<p>OS</p>	<p>Linux</p>
<p>Container</p>	<p>Docker, BalenaOS</p>
<p>Supported Frameworks and Tools</p>	<ul style="list-style-type: none"> ● TensorFlow, TensorFlow Lite, PyTorch, Caffe ● ONNX

Table 3.3 - Raspberry Pi 4 B⁸⁴

⁸⁴ <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/>

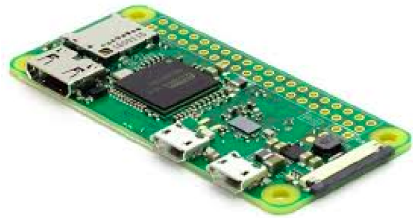
Raspberry Pi Zero W	
General description	This is a low-cost version of the Raspberry Pi, which extends the Pi Zero family and comes with added wireless LAN and Bluetooth connectivity.
SoC:	<ul style="list-style-type: none"> • Broadcom BCM2835, 1GHz, single-core • VideoCore IV @ 250 MHz
Accelerator	-
Memory	512MB
Connectivity	802.11 b/g/n Wireless, Bluetooth 4.1, BLE
OS	Linux
Container	Docker, BalenaOS
Supported Frameworks and Tools	<ul style="list-style-type: none"> • TensorFlow, TensorFlow Lite, PyTorch, Caffe • ONNX

Table 3.4 - Raspberry Pi Zero W⁸⁵

⁸⁵ <https://www.raspberrypi.org/products/raspberry-pi-zero-w/>

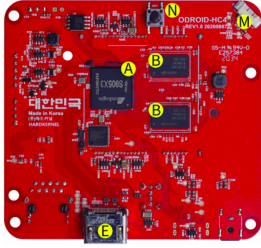
ODROID-HC4	
General Description	<p>The ODROID devices are an alternative to the Raspberry Pi. Moreover, they usually mount a dissipator, which maintains the SoC cooler, thus, they behave better on heavy load. On top of that, they do not have high power consumption, usually between 6W and 7W.</p> <p>The ODROID-HC4 board is based on the same ARM CPU as the ODROID-C4.</p>
SoC:	<ul style="list-style-type: none"> ● Amlogic S905X3, Quad-Core Cortex-A55 1.800GHz ● Mali-G31 MP2 GPU with 4 x 650Mhz
Accelerator	-
Memory	DDR4 4GB
Connectivity	GbE LAN (10/100/1000 Mbps)
OS	Linux
Container	Docker, BalenaOS
Supported Frameworks and Tools	<ul style="list-style-type: none"> ● TensorFlow ● ONNX

Table 3.5 - ODROID-HC4⁸⁶

⁸⁶ <https://wiki.odroid.com/odroid-hc4/odroid-hc4>

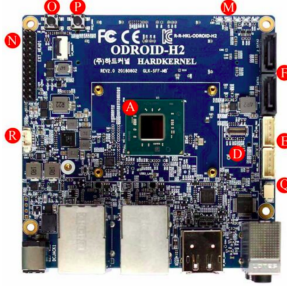
ODROID-H2+	
General description	Powerful x86 64bit single board computer with large memory capacity and advanced IO ports.
SoC:	<ul style="list-style-type: none"> ● Intel Quad-core J4115, 2.3Ghz ● Intel UHD Graphics 600 (Gen9.5) 700Mhz
Accelerator	-
Memory	MAX 32GB (Not included)
Connectivity	2.5Gbit Ethernet
OS	Linux / Android
Container	Docker, BalenaOS
Supported Frameworks and Tools	<ul style="list-style-type: none"> ● TensorFlow ● ONNX

Table 3.6 - ODROID-H2+⁸⁷

⁸⁷ <https://wiki.odroid.com/odroid-h2/start>


ODROID-N2+	
General description	The ODROID-N2+ is a speed upgrade to the original ODROID-N2 that is more powerful, stable, and faster performing than the N1.
SoC:	<ul style="list-style-type: none"> ● Amlogic S922X Processor Quad-core Cortex-A73 (up to 2.4Ghz) and Dual-core Cortex-A53 (up to 2Ghz) ● Mali-G52 GPU with 6 x Execution Engines (800Mhz)
Accelerator	-
Memory	4GB / 2GB DDR4
Connectivity	Gigabit Ethernet
OS	Linux / Android
Container	Docker, BalenaOS
Supported Frameworks and Tools	<ul style="list-style-type: none"> ● TensorFlow ● ONNX

Table 3.7 - ODROID-N2+⁸⁸

⁸⁸ <https://wiki.odroid.com/odroid-n2/odroid-n2>

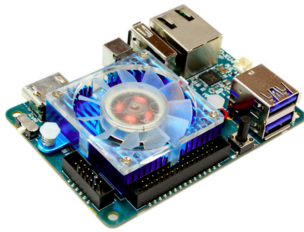
ODROID-XU4	
General description	ODROID-XU4 is a new generation of computing device with more powerful, energy-efficient hardware and a smaller form factor.
SoC:	<ul style="list-style-type: none"> ● Samsung Exynos5422 Cortex™-A15 2Ghz and Cortex™-A7 Octa core ● Mali-T628 MP6
Accelerator	-
Memory	2GB
Connectivity	Gigabit Ethernet
OS	Linux / Android
Container	Docker, BalenaOS
Supported Frameworks and Tools	<ul style="list-style-type: none"> ● TensorFlow ● ONNX

Table 3.8 - ODROID-XU4⁸⁹

In the following, we provide some insights on the performance of the SoCs introduced above, i.e., Raspberry and ODROID families, as reported on reference benchmarks. In particular, the Linpack⁹⁰ benchmarks provide a measure of the floating-point computing power of a system. The measured performance consists of the number of 64-bit floating-point operations, generally additions and multiplications, a computer can perform per second, also known as FLOPS. In particular, Figure 3.12 provides a comparison among Raspberry Pi models. It is possible to observe how the Raspberry Pi 4 dominates the benchmark with its single-precision (SP), double-precision (DP), and single-precision accelerated with NEON (SP NEON), an advanced single instruction multiple data architecture extension, scores, resulting faster than the Raspberry Pi Zero W but also than the Raspberry Pi 3 Model B+ and A+.

⁸⁹ <https://wiki.odroid.com/odroid-xu4/odroid-xu4>

⁹⁰ <http://www.netlib.org/benchmark/hpl/>

Linpack Benchmark

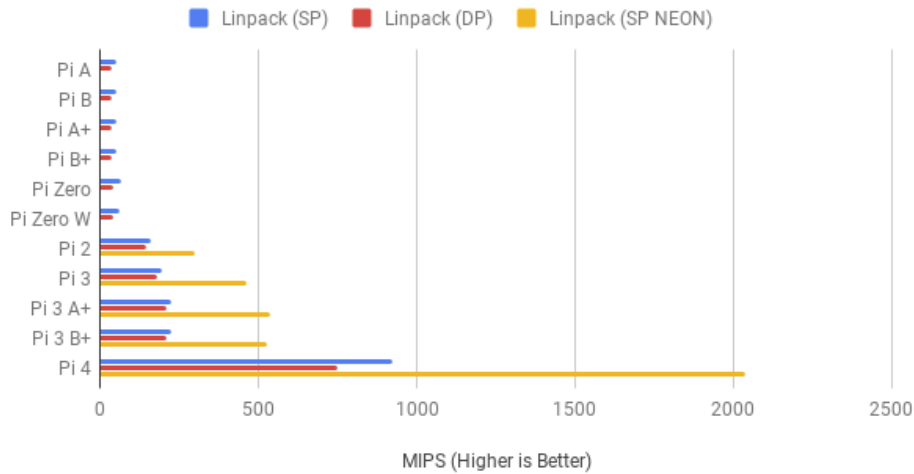


Figure 3.12 - Linpack benchmark of Raspberry Pi models⁹¹

Other benchmarks usually considered to compare EI devices as complete systems are the UnixBench⁹² benchmarks which provide a basic indicator of the performance of a Unix-like operating system. UnixBench is used to measure performance when running single or multiple tasks. Rather than performing a CPU, RAM or disk benchmarking, it provides an analysis of the overall system performance, which depends not only on the hardware, but also on the operating system and libraries. This is done by targeting tests at specific areas. In particular, we observe the following tests:

- *Drystone*: it is used to measure and compare the performance of computers. The test focuses on string handling, as there are no floating point operations. It is heavily influenced by hardware and software design, compiler and linker options, code optimization, cache memory, wait states, and integer data types.
- *Whetstone*: this test measures the speed and efficiency of floating-point operations. This test contains several modules that are meant to represent a mix of operations typically performed in scientific applications. A wide variety of C functions including sin, cos, sqrt, exp, and log are used as well as integer and floating-point math operations, array accesses, conditional branches, and procedure calls. This test measure both integer and floating-point arithmetic

Figure 3.13 compares different ODROID generations and the Raspberry Pi board. We can observe that all the ODROID boards are faster than the Raspberry Pi 4.

⁹¹ <https://medium.com/@ghalfacree/benchmarking-the-raspberry-pi-4-73e5afbcd54b>

⁹² <https://github.com/kdlucas/byte-unixbench>

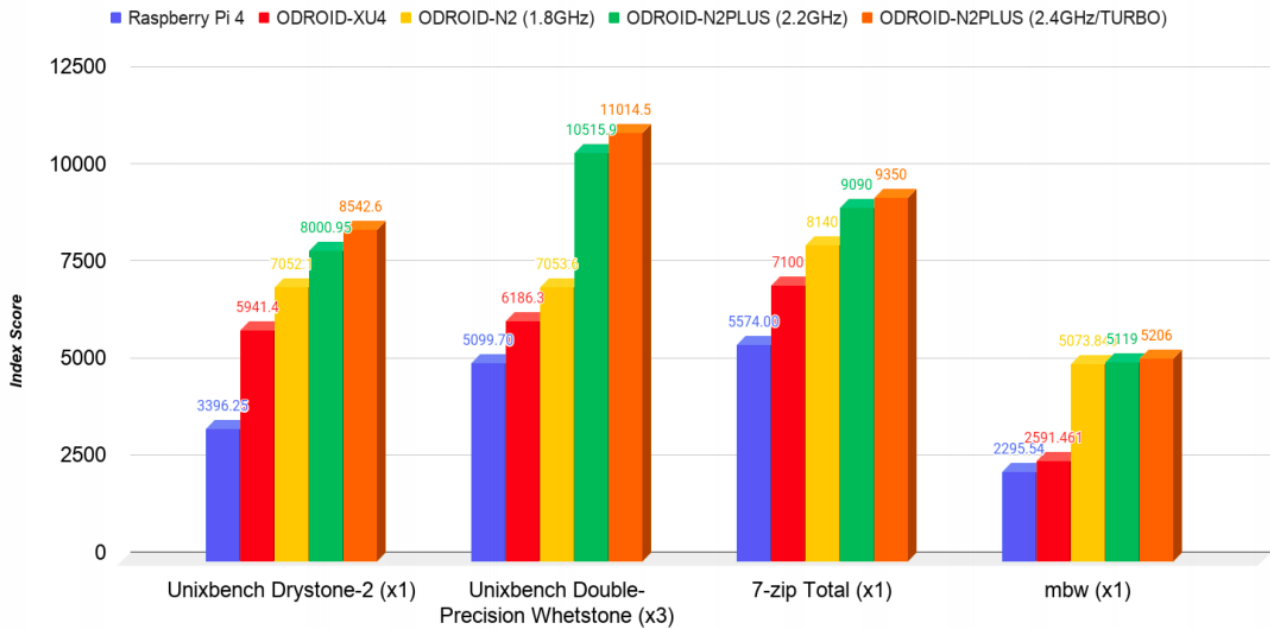


Figure 3.13 - UnixBench benchmark comparing Raspberry Pi and ODROID models⁹³

3.5.2 Hardware Accelerated SoCs & SoMs

In the following, we introduce the SoCs and system on module (SoM) that have Hardware Acceleration for deep learning. A SoM is a board-level circuit that integrates a system function in a single module. A typical application is in the area of embedded systems. Unlike a single-board computer, a SOM serves a special function like a SoC. The device integrated in the SOM typically requires a high level of interconnection for reasons such as speed, timing, bus-width etc., in a highly integrated module. There are benefits in building a SoM, as for SoC; one notable result is to reduce the cost of the baseboard or the main printed circuit board.

NVIDIA - Jetson SoMs

NVIDIA® Jetson™ is the world’s leading platform for AI on the edge. Its high-performance, low-power computing for DL and computer vision makes it the ideal platform for compute-intensive projects. The Jetson platform includes a variety of Jetson modules together with the NVIDIA JetPack™ SDK. Each Jetson module is a computing system packaged as a plug-in unit (System on Module). NVIDIA offers a variety of Jetson modules with different capabilities, which are summarized in Table 3.9, while Figure 3.14 shows the NVIDIA® Jetson™ Board Support Package (BSP) architecture.

⁹³ https://www.odroid.co.uk/index.php?route=product/product&path=246_239&product_id=868

	Jetson Nano	Jetson TX2	Jetson XAVIER NX	Jetson AGX XAVIER
GPU	128 Core Maxwell - 0.5 TFLOPs (FP16)	256 Core Pascal - 1.3 TFLOPs (FP16)	384 Core Volta - 21 TOPs (INT8)	512 Core Volta - 32 TOPs (INT8)
CPU	4 Core Arm A57 (1.43GHz)	6 Core Denver (1.4GHz) and 4 Core A57 (2GHz)	6 Core Carmel Arm 64bit (4x1.4GHz, 2x1.9GHz)	8 Core Carmel Arm 64bit (2.265GHz)
RAM	4GB 64bit LPDDR4 25.6 GB/s	8GB 128bit LPDDR4 58 GB/s	8GB 128bit LPDDR4 51.2 GB/s	16GB 256bit LPDDR4 136.5 GB/s
Storage	16GB eMMC	32GB eMMC	16GB eMMC	32GB eMMC
Camera	12 lanes (3x4 or 6x2) MIPI CSI-2	12 lanes (3x4 or 6x2) MIPI CSI-2	12 lanes (3x4 or 6x2) MIPI CSI-2	16 lanes (4x4 or 8x2) MIPI CSI-2
Ethernet	1 GigE	1 GigE	1 GigE	1 GigE
WiFi/BT	External	Onboard	External	External
Supported Frameworks	TensorFlow, TF-Lite(*), PyTorch, Caffe2, Keras, MXNet, ONNX	TensorFlow, TF-Lite(*), PyTorch, Caffe2, Keras, MXNet, ONNX	TensorFlow, TF-Lite(*), PyTorch, Caffe2, Keras, MXNet, ONNX	TensorFlow, TF-Lite(*), PyTorch, Caffe2, Keras, MXNet, ONNX
Container	Docker (via nvidia-docker2) and BalenaOS	Docker (via nvidia-docker2) and BalenaOS	Docker (via nvidia-docker2) and BalenaOS	Docker (via nvidia-docker2) and BalenaOS

Table 3.9 - Jetson family: technical specifications⁹⁴

(*) TF-Lite is not encouraged by NVIDIA, so there is little support for it. However, it has been successfully used on Jetson Nano and Jetson XAVIER NX.

⁹⁴ <https://developer.nvidia.com/embedded/jetson-modules#family>

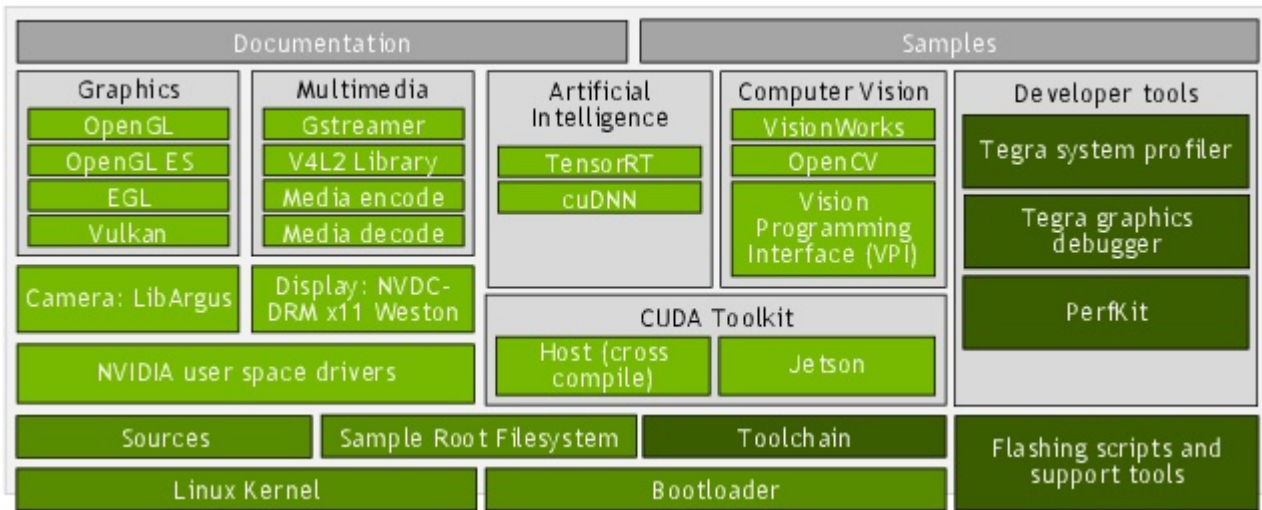


Figure 3.14 - NVIDIA® Jetson™ Board Support Package (BSP) architecture⁹⁵

In particular, we are interested in the Artificial Intelligence modules:

- *NVIDIA TensorRT*: provides a high-performance neural network inference engine for production deployment of deep learning applications, as described in Section 3.2.4.
- *cuDNN*: NVIDIA CUDA Deep Neural Network library.

Both NVIDIA® Jetson Xavier™ NX and Jetson AGX Xavier contain the dual NVIDIA Deep Learning Accelerator (NVDLA⁹⁶) engines. The NVDLA is an open architecture that promotes standardization in designing DL inference accelerators. Most of the computation on CNNs uses the same mathematical operations and can be grouped on five basic layer types: convolution, activation, pooling, normalization, and fully-connected. These operations have extremely predictable memory access patterns and thus can be highly accelerated with application-specific hardware that exploits those patterns. Figure 3.15 NVLDA internal architecture.

⁹⁵ Figure source: <https://docs.nvidia.com/jetson/archives/l4t-archived/l4t-3231/index.html#page/Tegra%20Linux%20Driver%20Package%20Development%20Guide/overview.html>

⁹⁶ <http://nvdla.org/>

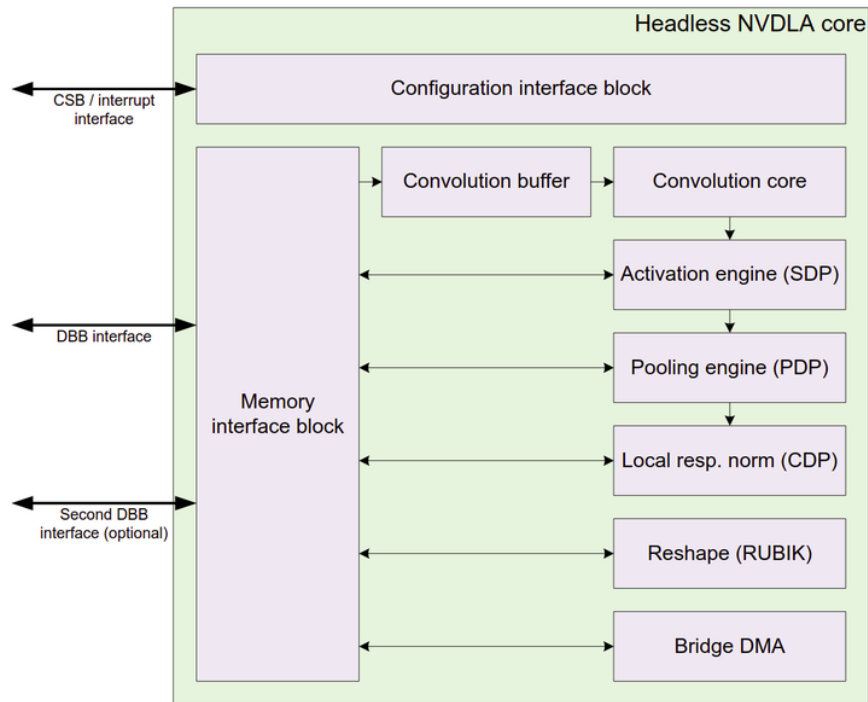


Figure 3.15 - Internal architecture of NVDLA core⁹⁷

Rockchip SoCs

Rockchip is a Chinese fabless semiconductor company based in Fuzhou. It produces SoC products for tablets & PCs, streaming media TV boxes, AI audio & vision, and IoT hardware. Finally, they released some chips with hardware acceleration for neural networks. In the following table, the most interesting SoCs are reported from the EI point of view. Their hardware acceleration unit is called Neural Processing Unit (NPU) and it packs from 1.2 TOPs up to 3.0 TOPs (8 bit), which is enough for most embedded applications. Just to have a comparison, the Jetson Nano packs 0.5 TFLOPs (16 bit), therefore, we should expect similar (or better) performance. Table 3.10 provides a comparison of the main Rockchip SoCs.

	RV1109	RV1126	RK1808	RK3399
CPU	Dual-Core CortexA7	Quad-Core CortexA7	Dual-Core CortexA35	Dual-Core CortexA72 + Quad-Core CortexA53
GPU	-	-	-	Arm Mali-T864
NPU	1.2 TOPs	2.0 TOPs	3.0 TOPs	3.0 TOPs
RAM	External 32bit DDR4/LPDDR4	External 32bit DDR4/LPDDR4	External 32bit DDR4/LPDDR4	External 64bit DDR4/LPDDR4

⁹⁷ Figure source: <http://nvdla.org/primer.html>

Camera	1x MIPI-CSI 2lanes	1x MIPI-CSI 2lanes	1x MIPI-CSI 2lanes	2x MIPI-CSI 4lanes
Ethernet	100 Mbit/s	100 Mbit/s	100 Mbit/s	100 Mbit/s
WiFi/BT	External	Onboard	External	External
OS	Linux	Linux	Linux	Linux, Android
Supported Frameworks	Caffe, TensorFlow, TF-Lite, ONNX, PyTorch, Keras, Darknet	Caffe, TensorFlow, TF-Lite, ONNX, PyTorch, Keras, Darknet	Caffe, TensorFlow, TF-Lite, ONNX, PyTorch, Keras, Darknet	Caffe, TensorFlow, TF-Lite, ONNX, PyTorch, Keras, Darknet
Container	Docker	Docker	Docker	Docker

Table 3.10 - Comparison of Rockchip SoCs

BalenaOS (see Section 5.1) might also become supported. At the moment, it is supported only by RV3328. Particularly interesting are three products, which can be classified as High End MPU, Mid-Range MPU, and Entry Level MPU. All of them pack both a General Purpose CPU and a NPU core. The technical specifications are summarized in Figures 3.16, 3.17, and 3.18.

- CPU**
 - Big-Little architecture: Dual-core Cortex-A72+Quad-core Cortex-A53, 64-bit CPU
 - ARM Mali-T860MP4 GPU, OpenGL ES1.1/2.0/3.0/3.1/3.2, Vulkan 1.0, OpenCL 1.2, DX11
- NPU**
 - 3.0Tops, 1920 INT8 MACs/192 INT16 MACs/64 FP16 MACs , MAX to 800MHZ
- Memory**
 - Dual channel DDR3-1866/DDR3L-1866/LPDDR3-1866/LPDDR4-1866
 - Support eMMC 5.1 with HS400, SDIO 3.0 with HS200
- Display**
 - Dual display engine up to 4096x2160 and 2560x1600
 - Dual channel MIPI-DSI TX, 4 lanes per channel
 - eDP 1.3 with PSR, 4 lanes up to 10.8Gbps
 - HDMI 2.0a with HDCP 1.4/2.2, up to 4K 60Hz
 - DisplayPort 1.2 with 4 lanes, up to 4K 60Hz
 - HDR10/HLG display with conversion between Rec.2020 and Rec.709
- Multi-Media**
 - 4K VP9 and 4K 10-bit H.265/H.264 video decoder, up to 60fps
 - 1080P other video decoders (VC-1, MPEG-1/2/4, VP8)
 - 1080P video encoders for H.264 and VP8
 - Security Video Path, OP-TEE, support Widevine Level1, PlayReady
 - Video post processor: de-interlace, de-noise, enhancement for edge/detail/color
- Camera**
 - Dual 13M ISP and dual MIPI CSI-2
- External interface**
 - Built-in dual Type-C with USB 3.0 and DisplayPort Alternate mode
 - PCIe v2.1 (4 full-duplex lanes), up to 2.5Gbps/lane
 - Embedded RGMII interface three channels I2S, SPDIF output
- PMU**
 - RK808-D / RK818-3
- Availability**
 - MP Now
 - FCBGA828

Figure 3.16 - RK3399 - High End MPU⁹⁸

⁹⁸ Figure source: http://opensource.rock-chips.com/wiki_RK3399

- CPU**
 - Dual core ARM Cortex-A35, 1.6GHz
- NPU**
 - 3.0 Tops for INT8 or 300 GOPs for INT16 or 100 GFLOPs for FP16
- Memory**
 - 32bit DDR3-1600/DDR3L-1600/ LPDDR2-1066 /LPDDR3-1600 / DDR4-2133
 - Support eMMC 4.51, Serial Nor Flash booting
- Display**
 - 4-lane, MIPI DSI, up to 1080P60
 - 18-bit parallel RGB panel, up to 1280*800
- Multi-Media**
 - 1080P@60fps H.264 Video decoder
 - 1080P@30fps H.264 Video encoder
- Camera**
 - 2M ISP, MIPI CSI-2 up to 4 data lane
- External interface**
 - RGMII interface
 - PCI-e 2.1, dual link; mux with USB 3.0
 - USB 2.0 OTG and USB 2.0 host
 - Dual SDIO 3.0 interface for Wi-Fi and SD card
 - 8ch I2S with TDM/PCM, 2ch I2S, VAD function
 - I2C/UART/SPI interface
- PMU**
 - RK809-2
- Availability**
 - MP Now
 - FCCSP420LD 14X14, 0.5mm pitch

Figure 3.17 - RK1808 - Mid Range MPU⁹⁹

- CPU**
 - Dual core ARM Cortex-A7, 1.5GHz
 - RISC-V MCU 400MHz
- NPU**
 - 1.2Tops, support INT8/ INT16
- Memory**
 - 32bit DDR3/DDR3L/LPDDR3/ DDR4/LPDDR4
 - Support eMMC 5.1, SPI Flash, Nand Flash
 - Support fast booting
- Display**
 - MIPI-DSI/RGB interface
 - 1080P @ 60 FPS
- 2D Graphics Engine**
 - Support rotation , x-mirror , y-mirror
 - Support alpha blending
 - Support scale down/up
- Multi-Media**
 - 5M ISP 2.0 with 3F HDR(Line-based/Frame-based/DCG)
- Camera**
 - Support 2*MIPI CSI /LVDS/sub LVDS
 - DVP interface with BT.656/BT.1120
 - 2688 x 1520@30 fps+1280 x 720@30 fps
 - 3072 x 1728@30 fps+1280 x 720@30 fps
 - 2688 x 1944@30 fps+1280 x 720@30fps
- External interface**
 - RGMII interface with TSO network acceleration
 - USB 2.0 OTG and USB 2.0 host
 - Dual SDIO 3.0 interface for Wi-Fi and SD card
 - 8ch I2S with TDM/PDM, 2ch I2S
- PMU**
 - RK809-2
- Availability**
 - SMIC 14nm, FCCSP 0.65

Figure 3.18 - RV1109 - Entry Level MPU (new)¹⁰⁰

⁹⁹ Figure source: https://www.rock-chips.com/a/en/products/RK1808_Datasheet_V1.2_20190527.pdf source: http://opensource.rock-chips.com/images/4/43/Rockchip_RK1808_Datasheet_V1.2_20190527.pdf

¹⁰⁰ Figure source: https://www.rock-chips.com/a/en/products/RV11_Series/2020/0427/1074.html

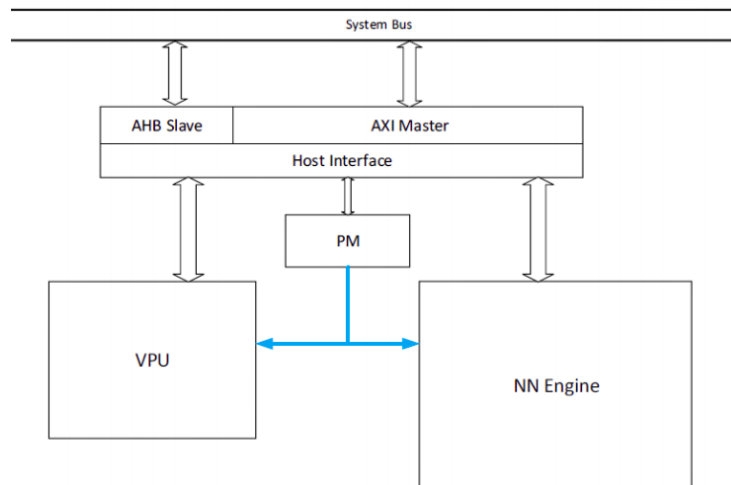


Figure 3.19 - NPU block diagram¹⁰¹

The Neural Processing Units (see Figure 3.19) are cores specifically designed to run standard CNNs. They have the following characteristics:

- **Powerful NPUs**
 - Embedded neural network hardware acceleration
 - Up to 3.0 TOPs@INT8 Inference (300 GOPs@INT16, 100 GFLOPs@FP16)
- **Heterogeneous computing**
 - NN kernel supports convolutional and fully-connected layers
 - NPU's internal VPU unit supports OpenCL / OpenVX API for custom layer support and reduces synchronization time
 - CPU/PPU for precision compute and for future network layers
- **Easy-to-use model conversion tool**
 - Support one-click conversion, to a mainstream architecture model¹⁰²
- **Typical applications:**
 - People counting / Face recognition / Gesture recognition
 - Products quality inspection

Renesas SoCs

Renesas Electronics Corporation is a Japanese semiconductor manufacturer headquartered in Tokyo, Japan. Its core business is microcontrollers, but they also released a SoC specifically designed for embedded applications in need of NN hardware acceleration.

¹⁰¹ Figure source: <https://wiki.radxa.com/RockpiN10/hardware/npu>

¹⁰² <https://github.com/rockchip-linux/rknn-toolkit>

- CPU and DDR Memory Interfaces
 - 2x Cortex-A53 up to 1.0GHz
 - 32-bit LPDDR4-3200
- Vision and Artificial Intelligence
 - AI Accelerator; DRP-AI at 1.0 TOPS/W class
 - Image Signal Processor (ISP) of multi-stream available
 - Camera Interface; 2x MIPI CSI
 - Face and Human Detection Engine
- Video and Graphics, Display
 - H.265/H.264 Multi Codec
Encoding; h.265 up to 2160p30, h.264 up to 1080p120
Decoding; h.265 up to 2160p30, h.264 up to 1080p120
 - 2D Graphics Engine; 200MPixels/s
 - Display; MIPI-DSI (4-lane), HDMI 1.4a
- High Speed Interfaces
 - 1x Gigabit Ethernet
 - 1x USB3.1 Gen1 Host/Peripheral
 - 1x PCIe Gen 2 (1-lane)
 - 2x SDIO 3.0
 - 1x NAND Flash Interface ONFI1.0
 - 1x eMMC 4.5.1
- Hardware Security Engine provided
- Package: FCBGA, 15x15mm 0.5mm pitch

Figure 3.20 - RZ/V2M Technical specifications¹⁰³

Their main product for EI is the RZ/V2M which main characteristics can be summarized as follows (see also Figure 3.20 for the main technical specifications):

- It has an AI dedicated hardware IP, DRP-AI, configured with a Dynamically Reconfigurable Processor (DRP) and AI-MAC, combining both a high-speed AI inference and low power consumption (1TOPS/W class power performance)
- In addition, the image signal processor (ISP) is highly robust, producing a stable image independent of the environment, allowing for a high AI recognition accuracy
- With these features, the RZ/V2M realizes low power consumption, which is a challenge for embedded devices, making heat dissipation measures easier. Since heat sinks and cooling fans are no longer needed, the equipment can be miniaturized, and the bill of material cost can be reduced
- It also features high-speed communication interfaces such as USB 3.1, PCI-Express, Gigabit Ethernet, and many CPU peripheral functions. Thus, it can also be used in a variety of applications
- Runs TinyYOLO at 33 fps
- Runs Linux and can be used with Docker
- Support ONNX to import ML models. The pipeline is schematized in Figure 3.21.



Figure 3.21 - RZ/V2M ONNX model import pipeline¹⁰⁴

Intel SoCs

Having bought Altera, an FPGA company, Intel solutions are mainly based on this kind of technology. However, since AI-SPRINT focuses on interchangeable technologies, we are not going to consider FPGAs as they require

¹⁰³ Figure source: <https://www.renesas.com/us/en/products/microcontrollers-microprocessors/rz-cortex-ampus/rzv2m-dual-cortex-a53-lpddr4x32bit-ai-accelerator-isp-4k-video-codec-4k-camera-input-fhd-display-output>

¹⁰⁴ Figure source: <https://www.renesas.com/us/en/document/bro/rz-family-microprocessors-brochure?language=en>

much more hardware specific work. Instead, we focus on the Intel SoCs which integrate their own NN hardware acceleration called VPU (Vision Processing Unit), namely Intel Movidius.

Intel Movidius is a custom design built to run inference of deep neural networks for images. It is powered by Intel Movidius Vision Processing Unit which is customized for Computer Vision. The main characteristics are the following:

- *Processor:* Intel Movidius Myriad X Vision Processing Unit (VPU)
- *Supported frameworks:* TensorFlow and Caffe
- *Compatible operating systems:* Ubuntu* 16.04.3 LTS (64 bit), CentOS* 7.4 (64 bit), and Windows® 10 (64 bit)
- *Container:* Docker

Application development is supported by the Intel Movidius Neural Compute SDK (NCSDK) which provides tools for profiling, tuning, and compiling a DNN model on compatible neural compute devices. The NCSDK includes a set of software tools to compile, profile, and validate DNNs as well as the Intel Movidius Neural Compute API (NCAPI) for application development in C/C++ or Python. The available tools are summarized in the following Figure 3.22.

Tool	Description
mvNCCompile	Converts a Caffe/TensorFlow* network and associated weights to an internal Intel® Movidius™ compiled format for use with the Intel® Movidius™ Neural Compute API.
mvNCProfile	Provides layer-by-layer statistics to evaluate the performance of Caffe/TensorFlow networks on your neural compute device.
mvNCCheck	Compares the inference results from running the network on your neural compute device vs. Caffe/TensorFlow for network compilation validation.

Figure 3.22 - Intel® Movidius™ Neural Compute SDK tools¹⁰⁵

Google Coral - SoM and ASIC

Coral is a platform of hardware and software components from Google that allows to build devices which can locally run AI models, providing hardware acceleration for NNs on the edge. Differently from the previous technologies, Coral devices are based on the Edge TPU co-processor (Tensor processing unit), a small application-specific integrated circuit (ASIC) from Google. The Edge TPU was specifically designed to power state-of-the-art neural networks at high speed, with a low power cost.

Google offers both Production SoMs and Prototyping boards. AI-SPRINT will adopt mainly the prototyping boards (development kits) as they are easier to work with. The prototyping boards are: Dev Board, USB Accelerator, and Dev Board Mini. The production boards are: Mini PCIe Accelerator, M.2 Accelerator A+E key, B+M key, and M.2 Accelerator with Dual Edge TPU and System-on-Module (SoM). The key specifications are reported in Tables 3.11-3.13.

¹⁰⁵ Figure source: https://movidius.github.io/ncsdk/tools/tools_overview.html

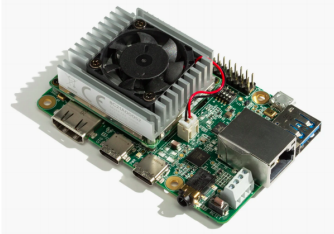
Dev Board	
General description	The Dev Board is a single-board designed to perform fast machine learning inferencing in a small form factor.
SoC:	<ul style="list-style-type: none"> ● NXP i.MX 8M SoC (quad Cortex-A53, Cortex-M4F) ● Integrated GC7000 Lite Graphics
Accelerator	Google Edge TPU coprocessor: 4 TOPS (int8); 2 TOPS per watt
Memory	1 GB LPDDR4 (option for 2 GB or 4 GB coming soon)
Connectivity	Gigabit Ethernet port, Wi-Fi 2x2 MIMO (802.11b/g/n/ac 2.4/5GHz) and Bluetooth 4.2
OS	Mendel Linux (derivative of Debian)
Container	Docker, BalenaOS
Supported Frameworks	TensorFlow Lite API (assisted by the PyCoral API)

Table 3.11 - Dev Board¹⁰⁶

¹⁰⁶ <https://coral.ai/products/dev-board/>


USB Accelerator	
General description	The Coral USB Accelerator enables high-speed machine learning inferencing on a wide range of systems, by adding an Edge TPU coprocessor.
SoC:	-
Accelerator	Google Edge TPU coprocessor: 4 TOPS (int8); 2 TOPS per watt
Memory	-
Connectivity	-
OS	Linux (On the host)
Container	Docker (On the host)
Other info	<ul style="list-style-type: none"> • Can only be used via USB. • Host computer must have the Edge TPU runtime and API library

Table 3.12 - USB Accelerator¹⁰⁷

¹⁰⁷ <https://coral.ai/products/accelerator/>


Dev Board Mini	
General description	The Coral Dev Board Mini is a single-board computer that provides fast machine learning (ML) inferencing in a small form factor.
SoC:	<ul style="list-style-type: none"> • MediaTek 8167s SoC (Quad-core Arm Cortex-A35) • IMG PowerVR GE8300 (integrated in SoC)
Accelerator	Google Edge TPU coprocessor: 4 TOPS (int8); 2 TOPS per watt
Memory	2 GB LPDDR3
Connectivity	Wi-Fi 5 (802.11a/b/g/n/ac); Bluetooth 5.0
OS	Mendel Linux (derivative of Debian)
Container	Docker, BalenaOS
Supported Frameworks	TensorFlow Lite API (assisted by the PyCoral API)

Table 3.13 - Dev Board Mini¹⁰⁸

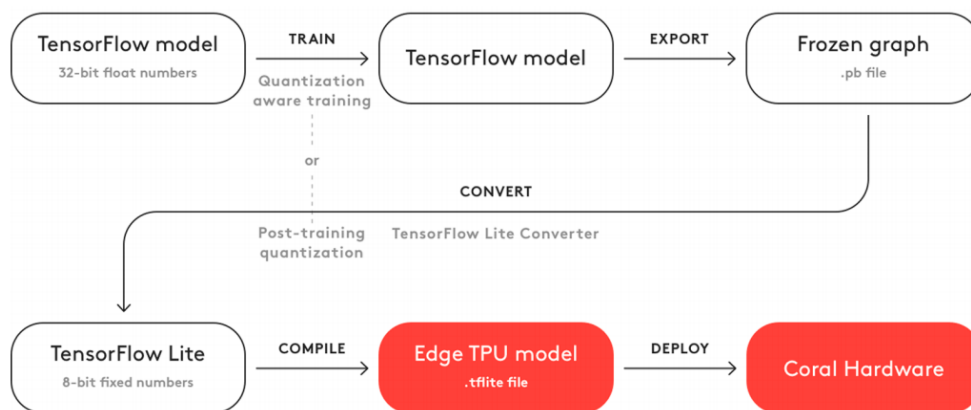


Figure 3.23 - Basic workflow to create a model for the Edge TPU¹⁰⁹

All the TPU devices support only Tensorflow Lite models that are fully 8-bit quantized and compiled specifically for the Edge TPU (see Figure 3.23). Furthermore, it is not possible to train a model directly with Tensorflow Lite. Instead, the model must be converted from a Tensorflow file (such as a .pb file) to a Tensorflow Lite file

¹⁰⁸ <https://coral.ai/products/dev-board-mini/>

¹⁰⁹ Figure source: <https://coral.ai/docs/edgetpu/models-intro/#compatibility-overview>

(a .tflite file), using the Tensorflow Lite converter. Finally, there are also some limitations on the supported operations, but the most used ones are all present (e.g., Conv2d, MaxPool2d, FullyConnected, Softmax, ReLU, etc).

Quectel - SoMs

Quectel is the world's leading supplier of cellular IoT modules. It basically produces Narrowband (NB) & LTE Modules and SoMs for IoT. It has recently released two new SoMs specifically designed for AI applications, Quectel SC66 and SC20. SC66 is equipped with a Neural Processing Engine based on the Qualcomm SDM660 chipset. It features 8x Kryo260 cores (custom design, 64-Bit) that are divided into two clusters: a fast cluster of four cores with up to 2.2GHz and a power saving efficiency cluster with up to 1.8GHz. In addition to the 8 CPU cores, the SC66 integrates an Adreno 512 GPU with an LPDDR4X memory controller (dual-channel 1866 MHz). AI Engine's hardware includes a Hexagon Vector Processor, Adreno GPU (Graphics Processing Unit, as workhorse of the AI) and Kryo CPU. Other characteristics include:

- Cellular (LTE), WiFi and Bluetooth. But the version with only WiFi & BT is available
- OS: Android 9.0
- Two memory solutions:
 - 32GB + 3GB
 - 64GB + 4GB
- Supports Tensorflow

If the SC66 is the High Tier module, the SC20 is the Entry Level Tier with the following characteristics:

- Qualcomm® MSM8909 (1.1Ghz Quadcore Processor)
- Two memory solutions:
 - 8GB + 1GB
 - 16GB +2GB
- Cellular (LTE), WiFi and Bluetooth connectivity. But the version with only WiFi & BT is available
- OS: Android 7.1 / Linux
- Supports Tensorflow Lite.

3.5.3 Hardware Accelerated Microcontrollers

A Microcontroller is a small computer on a single metal-oxide-semiconductor integrated circuit chip, characterized by low costs and limited computational power. They usually integrate many peripherals like ADC, Timers, Interrupts, PWMs, etc., and they lack a MMU (Memory Management Unit), which means that they cannot host a normal OS. In the following, we present the main Microcontrollers solutions that provide Hardware Acceleration for DL.

STMicroelectronics - CubeAI

ST's hardware acceleration works by using the microcontroller's internal Digital Signal Processing (DSP), which allows running NNs via the CubeAI library. Figure 3.24 shows the microcontroller families that support CubeAI.

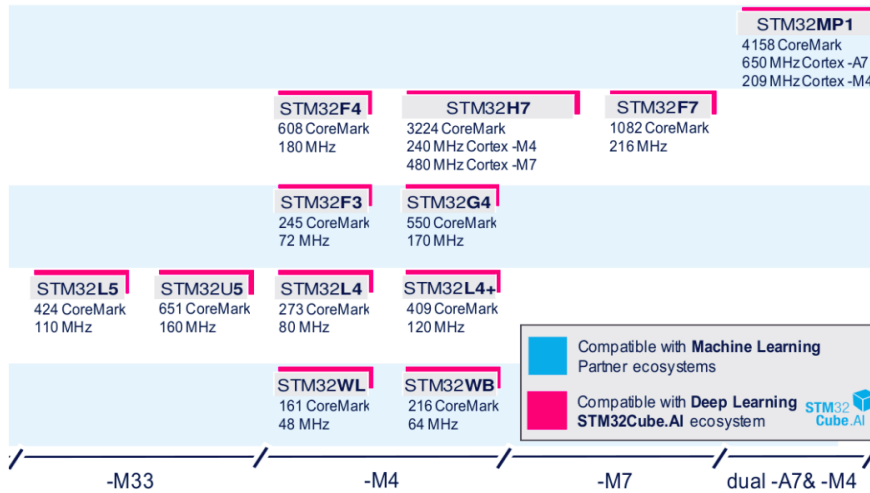


Figure 3.24 - Overview of the microcontroller families that support CubeAI¹¹⁰

The CubeAI library supports Keras, Tensorflow Lite, ONNX and PyTorch frameworks (see Figure 3.25). By using the STM32 Cube AI framework we can import the model and convert it into optimized code which will run on the MCU.

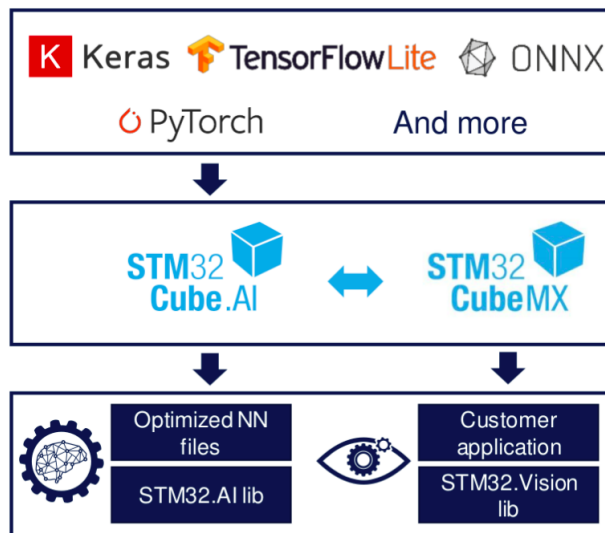


Figure 3.25 - Workflow to import models from DL frameworks and ONNX¹¹¹

However, a microcontroller cannot support a normal OS such as Linux and therefore no container can be employed. The only OS that can be installed is μ Linux, which is a variation of the Linux kernel that targets microcontrollers without an MMU. The only exception is the STM32MP1, as it features both an STM32F4

¹¹⁰ Figure source: https://www.st.com/content/ccc/resource/sales_and_marketing/presentation/product_presentation/group0/69/82/bf/ae/5a/8b/40/91/STM32CubeAI_press_pres/files/STM32CubeAI_press_pres.pdf/jcr:content/translations/en.STM32CubeAI_press_pres.pdf

¹¹¹ Figure source: https://www.st.com/content/ccc/resource/sales_and_marketing/presentation/product_presentation/group0/69/82/bf/ae/5a/8b/40/91/STM32CubeAI_press_pres/files/STM32CubeAI_press_pres.pdf/jcr:content/translations/en.STM32CubeAI_press_pres.pdf

microcontroller and a SoC (650 MHz Cortex -A7, 209 MHz Cortex -M4). Therefore, it supports Linux and containers such as Docker.

Maxim Integrated - MAX78000

MAX78000 is an Ultra-Low-Power Arm Cortex-M4 Processor from Maxim Integrated, which is used for AI applications. The MAX78000 is built to enable NNs and combines energy-efficient AI processing with ultra-low-power microcontrollers. The hardware-based convolutional neural network (CNN) accelerator enables battery-powered applications to execute AI inferences while spending only microjoules of energy. The core technical specifications are as follows:

- Dual Core Ultra-Low-Power Microcontroller
 - Arm Cortex-M4 Processor with FPU Up to 100MHz
 - 512KB Flash and 128KB SRAM
 - Optimized Performance with 16KB InstructionCache
 - 32-Bit RISC-V Coprocessor up to 60MHz
- Neural Network Accelerator
 - Highly Optimized for Deep Convolutional Neural Networks
 - 442k 8bit Weight Capacity with 1,2,4,8-bit Weights
 - Programmable Input Image Size up to 1024 x 1024pixels
 - Programmable Network Depth up to 64 Layers
 - Programmable per Layer Network Channel Widths up to 1024 Channels
 - 1- and 2-Dimensional Convolution Processing
 - Flexibility to Support Other Network Types, Including MLP and Recurrent Neural Networks
- Power Management Maximizes Operating Time for Battery Applications
 - 2.0V to 3.6V SIMO Supply Voltage Range
 - Dynamic Voltage Scaling Minimizes Active CorePower Consumption
 - 22.2µA/MHz While Loop Execution at 3.0V from Cache (CM4 only)

MAX78000 is the most power-efficient embedded device by Maxim Integrated. Multiple high-speed and low-power communications interfaces are supported, including the Inter-IC Sound (I²S) and a parallel camera interface (PCIF). MAX78000 supports Tensorflow and PyTorch, but it works similarly to CubeAI, i.e., the model must be quantized and transformed into embedded code. No support for OS like Linux is provided.

Espressif - ESP32-S3

The ESP32-S3¹¹² (see Figure 3.26) is the latest WiFi/BT Module (basically a microcontroller with connectivity) from Espressif. ESP32-S3 is a dual-core Xtensa LX7 MCU, capable of running at 240 MHz. Apart from its 512 KB of internal SRAM, it also comes with integrated 2.4 GHz, 802.11 b/g/n Wi-Fi and Bluetooth 5 (LE) connectivity that provides long-range support. It has 44 programmable general-purpose input/output (GPIOs) and supports a rich set of peripherals. It also supports larger, high-speed octal SPI flash, and PSRAM with configurable data and instruction cache. Being a microcontroller, it runs only a real-time operating system,

¹¹² <https://www.espressif.com/en/products/socs/esp32-s3>

and this is the case of FreeRTOS (a real-time operating system kernel for embedded devices that has been ported to 35 microcontroller platforms and distributed under the MIT license).

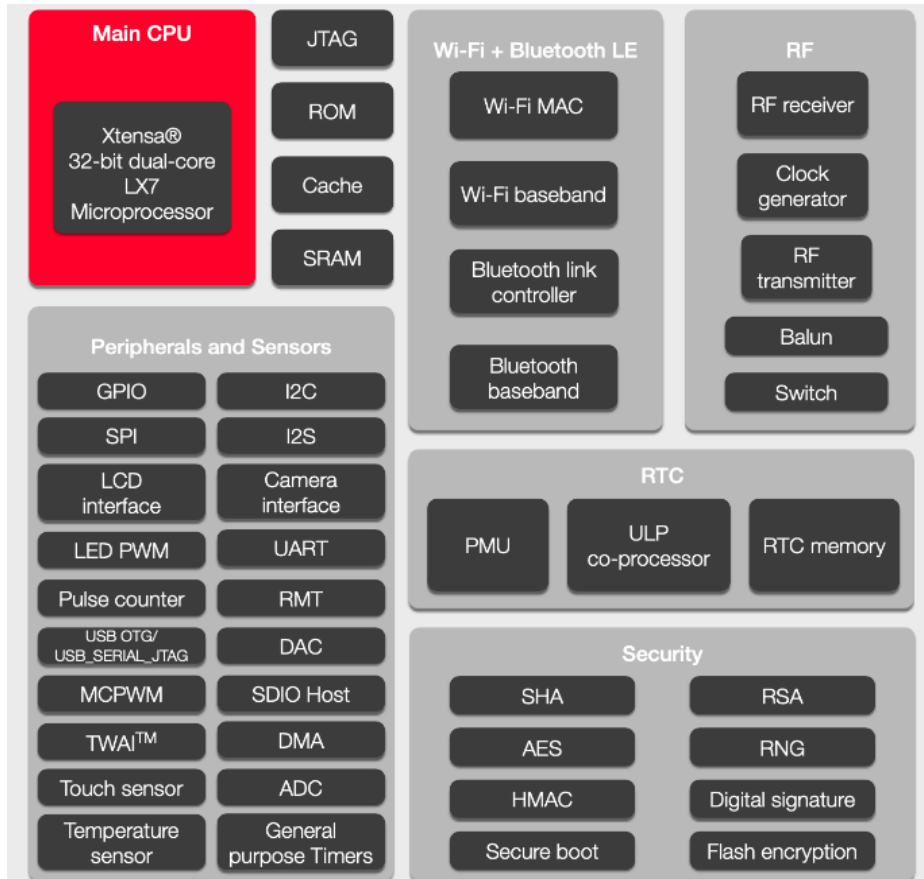


Figure 3.26 - Espressif's ESP32-S3 Wi-Fi + Bluetooth LE SoC¹¹³

3.6 Smartwatches and bands in stroke care

Stroke is the second leading cause of death and third leading cause of disability in adults worldwide [GBD 2016]. It is a common disease, with increasing incidence in the aging population, that affects one in four people over their lifetime. Stroke is defined as a neurological deficit due to an acute focal injury in the central nervous system by a vascular cause. Its presentations can be ischaemic, resulting from a reduced blood flow generally due to an arterial occlusion, or haemorrhagic, resulting from a rupture of cerebral arteries [Campbell2020]. Clinical examination and brain imaging, such as computed tomography (CT) scan or magnetic resonance imaging (MRI), are crucial to differentiate the stroke subtypes. 90% of strokes are attributable to modifiable risk factors [ODonnell2010], such as high blood pressure, smoking, diabetes, physical inactivity, and, specifically for ischaemic stroke, atrial fibrillation (AF) [Chugh2014; Tu2010], which is an arrhythmia of the atrial chambers of the heart. Knowing the timing of stroke is critical for effective medical intervention. However, the sudden onset of stroke can be masked if it is unwitnessed, under-recognized or not communicated. For this reason, new strategies for preventing and monitoring stroke or its recurrence have

¹¹³ Figure source: https://www.espressif.com/en/news/ESP32_S3

been developed, such as the use of smart wearable devices for cardiovascular care, which represents one of the most promising approaches [Bayoumy2021].

Wearable devices are connected electronic devices that can be worn as accessories or embedded into clothing. They include medical earbuds, patches, chest straps, smartwatches or bands, smart rings, and others. Thanks to the ability of performing long-term continuous monitoring, wearable devices, especially smartwatches or bands (Table 3.14), enable virtual or remote cardiovascular care with critical benefits for stroke and the reduction of hospitalization expenses [Kuehn2016]. The market size for wearable devices is expected to rise to \$70 billion by the year 2025 [Ajami2015]. Remote monitoring for cardiovascular care by wearable devices generally consists of three components: 1) sensors collecting data on physiological parameters; 2) a network or communication interface transferring this data to a remote terminal, such as a smartphone, and 3) a remote cloud analytics platform.

Common wearable sensors for cardiovascular care include activity sensors (e.g., triaxial accelerometer, barometer, Global Positioning System or GPS), biometric sensors (e.g., photoplethysmography or PPG, electrocardiography or ECG, oscillometry), invasive and non-invasive biochemical sensors (e.g., monitoring of blood glucose and electrolytes, sweat and hydration status), and biomechanical sensors (e.g., ballistocardiography, seismocardiography, dielectric sensors). Most of the parameters measured by those sensors have been traditionally recorded only during clinic visits with limitations for a detailed, continuous, and objective assessment in real-life environments.

Accelerometers are largely used to monitor physical activity. Among the most common types of accelerometers, the differential capacitive accelerometer is the most employed in wearable devices due to low power consumption, fast response to motion and superior accuracy [Yang2010].

Heart rate (HR) and heart rhythm measurements are generally performed by using ECG and PPG. Some smartwatches or bands can record a single-lead ECG, where a contralateral finger can be placed on a negative electrode on the side of the device while the back of the watch serves as a positive electrode. Single-lead ECGs can diagnose arrhythmias such as AF, especially when coupled with PPG. By recording the variable intensity of photons continuously reflected through the skin, PPG measures changes in the microvascular blood volume, which indicates pulse waves that can be recorded as a tachogram [Kamialisic2018]. To preserve battery life, most wearables activate PPG during physical activity or intermittently during rest or sleep. PPG accuracy can be affected by skin colour, moisture, tattoos, and variable contact with the skin [Dagher2020].

Benchmark studies comparing the performance in HR accuracy of PPG sensors across different wearable devices are rather scarce. One study compared the Apple Watch 3 and the Fitbit Charge 2 showing similar accuracies across 24 hours and during various activities [Nelson2019]. In particular, the Apple Watch 3 exhibits a mean agreement of 95% with the golden standard ECG and the Fitbit Charge 2 a mean agreement of 91%. Another study compared wrist-worn sensors (Fitbit Blaze, Apple Watch, Garmin Forerunner 235, TomTom Spark Cardio) and one chest strap (Polar H7), showing that PPG HR readings from wrist-worn sensors are less accurate than the chest monitor [Etiwy2019]. Studies that specifically focus on algorithms for AF detection via PPG sensors report an overall sensitivity of 90-96% and specificity of 85-99% [Krivoshei2017; McManus2013; Brasier2019; Dörr2019]. Obtaining an adequate PPG signal at the wrist represents the main technical limitation for these devices. Nevertheless, while one study found that it was not possible to analyse the PPG signal in 21.8% of the records for poor signal quality [Dörr2019], another study on a most recent device reported only 3% of unclassifiable records [OscaAsensi2020].

Finally, recent studies focus on the effectiveness of AF diagnosis through wearable devices, such as the mSToPS study [Steinhubl2018] that showed a higher rate of new AF diagnosis via ECG monitoring, or the Apple Heart study [PerezApple2019], the largest to date, that showed the possibility of AF diagnosis in individuals

without a known history of the disease. In this regard, the ongoing HEARTLINE trial [HEARTLINE2020] is evaluating the improved clinical outcomes of PPG-enabled devices for AF detection.

Company	Device name	Parameter measurement
Adidas	miCoach Fit Smart	HR, PA
Apple	Apple Watch	HR, PA, falls, sleep, ECG
Biobeat	BB-613WP	HR, PA, cuff-less BP
Fitbit	Flex, One, Charge	HR, PA, sleep
Garmin	Vivoactive, Vivofit, Forerunner	HR, PA, sleep
Huawei	Huawei Watch GT, Huawei Band	HR, PA, SPO2
Karacus	DIONE, TRITON	HR, PA
Omron	HeartGuide	HR, PA, sleep, cuff BP
Samsung	GearFit 2	HR, PA, sleep
SmartCardia	INYU	HR, PA, ECG
TomTom	TomTom Spark	HR, PA
Withings	Steel HR, Move, Move ECG, Pulse HR	HR, PA, sleep, ECG. SPO2
Google	Wear OS on different hardware manufacturers	HR, PA, sleep
Arrhythmia algorithm S.L.	RITHMI	HR, ECG, SPO2
AliveCor	KardiaBand	HR, ECG
Microsoft	Microsoft Band	HR, PA, sleep
Nike	FuelBand	PA
Under Armour + HTC	UA Band	HR, PA, sleep
Xiaomi	Mi Band	HR, PA
Fitbug	Fitbug ORB	PA, sleep

Table 3.14 - Common smartwatches or bands on the market (adapted from [6]). BP, blood pressure; ECG, electrocardiogram; HR, heart rate; PA, physical activity; SPO2, arterial oxygen saturation

3.7 Cameras for Agriculture 4.0

The specific agricultural field that AI-SPRINT tackles is viticulture. Due to its specific features, viticulture is especially suitable for the application of Agriculture 4.0 approaches and techniques, and particularly of the AI-SPRINT approach. The “Viticulture 4.0” use case of AI-SPRINT aims at optimizing wine-making practices by considering the spatial and temporal variability of information collected from vine plants. Therefore, capturing relevant information is crucial to produce meaningful analyses. In viticulture, weather, soil, botanical and economic data are currently considered: AI-SPRINT intends to complement these with sensor data streams collected by the agricultural machines.

Botanical information can be collected at several scales, from satellite to microscopic. Unlike large-scale cultivation (like maize, bean, tomatoes, etc.), where information can be massively collected through remote sensing (e.g., using satellites or drones), viticulture requires considering the vertical dimension of the plant. Thus, in the context of AI-SPRINT, we will focus on proximity detection. In the following sections, we review the relevant technologies available for this purpose.

3.7.1 Camera sensors for vine plant diseases and pests detection

Generally used in the broad agricultural sector, camera sensors can also be integrated into systems to detect vine diseases and directly act on them by spraying or cutting. Tables 3.15-3.18 provide a review of the available devices of this kind and show how the application of these sensors to viticulture is still in its initial phase.


<p>Smartstriker by Carbon Bee</p>	
<p>General description</p>	<ul style="list-style-type: none"> • This camera integrates a GPS receiver with three different imaging sensors, i.e., RGB sensor, IR sensor, and Hyperspectral sensor • Primarily used in general agriculture, Carbon Bee started to deploy this camera in viticulture, too

Table 3.15 - Smartstriker

<p>Bilberry by Bilberry SAS</p>	
<p>General description</p>	<ul style="list-style-type: none"> • This camera integrates an RGB sensor and a GPS receiver • Only used in general agriculture

Table 3.16 - Bilberry

<p>See&Spray by Blue River Technology</p>	
<p>General description</p>	<ul style="list-style-type: none"> ● This camera integrates an RGB sensor and a GPS receiver ● Only used in general agriculture

Table 3.17 - See&Spray


<p>SmartSprayer by Bosch/Xarvio/Amazone</p>	
<p>General description</p>	<ul style="list-style-type: none"> ● This camera integrates a GPS receiver, an RGB sensor and a red light source ● Only used in general agriculture

Table 3.18 - SmartSprayer

3.7.2 Diseases detection on vine leaves

Disease detection can be performed through RGB, IR, and Hyperspectral imaging sensors. Drones were the first platform used for this application and can carry one or more sensors, but their limited payload is a strong constraint in many cases. Images are taken in the visible spectrum, but also certain specific wavelength ranges in the near-infrared (NIR, around 800 nm) and the Red Edge (about 735 nm) are detected. This information is used to calculate specific multispectral indices, such as the NDVI (*Normalized Difference Vegetation Index*), which provide information on the health state of the crop, in particular vigor or stress. When multispectral indices highlight an anomalous situation, the winegrower must personally visit the vine to diagnose the issue: for instance, to decide if it is due to water stress or to diseases.

RGB cameras are currently being considered to detect Esca and Flavescences Dorée diseases (see Figure 3.27). For fungal diseases such as late blight and powdery mildew, multispectral cameras are preferred. Carbon Bee and RGX System seem to take the lead for onboard solutions, with a sensor embedding three cameras and AI. Still, those sensors are devised for the detection of single diseases.

Moving from curative to preventive by analyzing images is not easy because similar symptoms are associated with different plant diseases. Plant diseases lead to confusion also because plant stress is inherently poly-symptomatic.



Figure 3.27 - Esca detection by proximal sensing

At regular intervals of time, several images taken on the same plot should make it possible to better detect these diseases by considering their temporal evolution or even to follow the expansion of infections and identify the most vulnerable areas to better target treatments. Remote sensing makes it possible to grid plot fields with different resolutions. On a wheat field, the images provided give an idea, for example, of the extension of wheat rust to a degree of accuracy of 95%. The detection carried out by a drone at very low altitude or by a camera onboard a tractor pushes the investigation to the scale of the spot on the sheet. It allows to have an image with a very high resolution and target the treatments at the scale of the vine strain, for example.

Unconventional sensors would make it possible to detect diseases before the onset of symptoms. Still, this technology is expensive and remains very complex to transpose from the laboratory to the plot. However, this would significantly reduce the costs of treatments since they would be applied very early. AI applied to images taken in the field can locate the places where the disease is present. However, the resolution of the images (whether taken in proximity or from remote sensing) must be sufficient to distinguish the different vine elements to treat, for example, only the vine leaves (for more details, see [Samson2021]).

3.7.3 Grape detection

The interest towards detection of grapes is due to the fact that estimating their volume leads to yield estimation, which is of extreme interest to the winegrower. Grape detection research is oriented on early grape volume analysis, while the grapes are still visible and not yet hidden by the leaves. An RGB and RGB-D camera combined with a LiDAR or other 3D sensor are usually needed. The objective is to count and measure the volume of grapes to produce a yield estimation [Santos2020], [Kurtser2020].

3.7.4 Vine canopy volume estimation

Estimation of the vine canopy is important to determine plant state and can be used as input data for other processes. AI-SPRINT, in particular, considers the use case where this information is used to control the spraying of phytosanitary products.

It is possible to estimate the volume of the vine canopy by LiDAR and other 3D industrial sensors [Cherariet2020]. GREGOIRE has validated the tridimensional reconstruction of the vine canopy with a 3D Infrared device mounted on a harvesting machine^{114,115}.

3.8 Drones Image Acquisition Technologies for Inspection and Maintenance

Unmanned aerial vehicles (commonly called drones or abbreviated as UAV) became recently a tool of choice for visual inspection of industrial infrastructure. There are numerous reasons for that. The major ones are:

- Ease of transport: drones are fairly compact and can be transported easily even to remote locations
- Flight time: advanced drone technology allows for longer flight times making the inspection process cost and time efficient
- Image quality: images or video footage taken by drones are of much better quality than shots taken from the ground using telephoto lenses and tripods (common alternative to drone-based acquisition)
- Safety: one of alternatives to drone operations is climbing onto infrastructure to inspect damages in detail which is a time consuming and dangerous process.

Modern drones offer not only longer flight times but also substantial lift capabilities. More lift allows more equipment on-board. At the same time, development of energy-efficient, miniaturized electronics is progressing as well. Combining these two technologies together, enable the building of flying multi-sensor devices with significant on-board data processing and communication capabilities.

3.8.1 Drone components

This section will cover general architecture of the multicopter, which is the UAV type most commonly used for industrial inspection, with special attention to components relevant for AI-SPRINT.

A multicopter can be split into the following elements:

- Frame: the base drone is built on
- Motors: responsible for rotating the propellers
- Propellers: they are usually of two types and depending on motor rotation direction they create air pressure difference which is the source of lift
- ESC (electronic speed controller): controls the amount of power provided to each motor and communicates with the flight controller
- Batteries and cables: energy source and distribution

¹¹⁴ www.pleinchamp.com/actualite/machinisme~un-guidage-automatique-sans-gps-pour-les-automoteurs-gregoire

¹¹⁵ www.ifm40.fr/nos-solutions/robotique-vehicules-autonomes/machine-a-vendanger-autonome-la-vision-3d-au-service-de-lagriculture/

- Radio transmitter: responsible for communication with the drone (at least 4 channels are required)
- Flight controller: is basically a SoC with some internal sensors (accelerometer, gyroscope, magnetometer, barometer etc.). It is the central component of the drone; gathers data from external (like GPS, obstacle avoidance sensors) and internal sensors, communicates with the remote controller and ESC. The flight controller runs the flight control software. There are multiple flight controller stacks available on the market running both open source and proprietary, closed software.
- Sensors: additional sensors are used either for real-time flight control or data collection, some of them include:
 - Cameras (usually equipped with gimbal for stabilization and angle control)
 - Distance sensors (also used as obstacle avoidance); sensors can leverage various measurement techniques like: lidar, sonar, ToF (infrared time-of-flight), stereoscopic camera (with depth estimation)
 - Airspeed sensors
 - GPS (also with real-time kinematic positioning)
 - Optical flow sensor (camera and lidar based speed estimation)
 - 3D LiDAR, 3D laser mapping
- Companion computer: additional computers used for additional computation, for example real-time image analysis; practically any SoC with proper interfaces can be used for that purpose. Detailed analysis of suitable devices, including GPU equipped devices is available in Section 3.5.
- Communication peripherals: various communication components like: GSM modules, WIFI, bluetooth etc.

3.8.2 Flight Controllers and Flight stacks

Flight controllers with their software are key components responsible for UAVs ability to fly. Software running on the controller is especially important. It affects the drone flight characteristics, how to pilot the drone or the level of autonomy. The most widely used open-source platforms available today are:

- PX4: probably the best-known open-source autopilot software (not only for drones) written in C/C++; using open-hardware Pixhawk as reference platform¹¹⁶
- ArduPilot: extensively used flight control stack written in C/C++¹¹⁷
- Cleanflight: C based open-source project based on 8-bit MultiWii¹¹⁸
- Betaflight: based on Cleanflight and Baseflight¹¹⁹
- INAV: Cleanflight fork focusing on GPS related features¹²⁰
- Paparazzi UAV: both open-software and open-hardware project focusing on autonomous flight written in C and Python¹²¹

3.8.3 Companion Computers

From an AI-SPRINT standpoint, UAV companion computers are the key element. Even though companion computers are not responsible for core drone functions, they can provide key data for drone operations by analyzing real time data from sensors and providing feedback to flight control or the drone operator. There is

¹¹⁶ <https://px4.io/>

¹¹⁷ <https://ardupilot.org/>

¹¹⁸ <http://cleanflight.com/>

¹¹⁹ <https://betaflight.com/>

¹²⁰ <https://github.com/inavFlight/inav/wiki>

¹²¹ https://wiki.paparazziuav.org/wiki/Main_Page

a wide range of devices available for that purpose. Some are capable of running even computer vision models based on deep neural networks in real time by leveraging GPU chips. Exhaustive list of such SoCs is presented in Section 3.5. Flight controllers and companion computers communicate over MAVLink protocol, an efficient, reliable, low overhead protocol and well suited for low-bandwidth links and resource-constrained systems.

3.9 Gaps filled by AI-SPRINT

In AI-SPRINT, an application is an enterprise application, mostly written in Python, that makes intensive use of AI technologies and is based on multiple components running across a computing continuum. Scikit-learn and Python machine learning libraries (e.g., xgboost, statsmodels, etc.), PyTorch, Keras, and TensorFlow will be the main frameworks and libraries that will be considered initially. Edge and IoT devices will be targeted via the container abstraction either via docker, when supported, or via the BalenaOS while optimization for the target hardware will be performed via vendor specific frameworks.

The AI-SPRINT framework will be more flexible than existing programming solutions. AI-SPRINT will address most of the challenges related to the composition of AI applications and their deployment and execution on the edge with transparent offloading to cloud platforms depending on load and constraints requirements. Specific highlights will be put on security and QoS in the definition of the framework and design abstractions will be defined accordingly. The PyCOMPSs programming model will be used as base technology. Applications will be designed transparently with respect to the execution platform and their execution will be delegated to the runtime. Moreover, AI application developers will be able to develop code that can seamlessly run, without changes, on traditional batch systems or on clouds using a FaaS model avoiding cloud vendor lock-in.

The next section will introduce the state-of-the-art of advanced features for AI applications design and development that will be addressed by AI-SPRINT and will constitute the main innovations of the framework.

4. AI Application Advanced Design

Chapter 4 reviews the advanced solutions developed for the design of AI applications. These involve DL models partitioning, network architecture search, federated learning, AI application performance modelling, and component placement in computing continua. In particular, the description of DL models partitioning, and the corresponding strategies are presented in Section 4.1. Section 4.2 introduces network architecture search, while Section 4.3 presents the federated learning paradigm, with a particular emphasis on its adoption for privacy preservation. Section 4.4 describes how the performance of AI applications can be modelled. Finally, components placement strategies in computing continua are presented in Section 4.5.

4.1 DL models partitioning

Deep Neural Networks (DNNs) are one of the most popular topics in machine learning in these years. However, DNNs usually need powerful resources for performing their huge computations. The work in [Yadwadkar2019] investigates the DNN inference phase and discusses the challenges that inference serving systems encounter these days, analysing the gaps between what users expect and the existing inference serving solutions. The identified challenges include the selection of a model-variant (which means diverse requirements lead to different models), the existence and need of heterogeneous hardware architectures, the varying load and query patterns and the start-up latency of the DNN frameworks. Authors also define two concepts: *managed* and *model-less inference serving system*. A managed inference serving system is an inference serving system that automates resource provisioning for models to serve users' inference queries satisfying their requirements. A model-less system is an interface to the managed inference serving system where users are able to focus on querying an inference for their tasks without needing to think of models, and the trade-offs offered by model-variants. According to the challenges mentioned above, they define and make a case for managed and model-less inference serving systems and indicate the open research directions to realize this vision.

Splitting the DNN to several parts and executing in a distributed way on edge-cloud devices, is a commonly adopted approach to overcome DNN computation requirements in computing continua. Accordingly, there are many works to partition DNN such as [Teerapittayanon2017] which proposes a distributed deep neural network (DDNN) over distributed computing hierarchies. DDNN maps a trained DNN onto heterogeneous physical devices distributed locally, at the edge, and in the cloud. The vertical lines in Figure 4.1 represent the DNN pipeline which connects the horizontal bars (NN layers). The authors consider different settings for DDNNs. The cloud-based DDNN is the standard DNN running in the cloud (Figure 4.1a). The next deployment considers a cloud-based DDNN by introducing end devices and a local exit point that may classify samples before the cloud (Figure 4.1b), possibly adding multiple end devices and performing a local aggregation together with classification (Figure 4.1c). Finally, Figure 4.1d and Figure 4.1e extend the scenarios reported in Figure 4.1b and Figure 4.1c by adding edge layers between the cloud and end devices, while Figure 4.1f shows how the edge can also rely on multiple distributed end devices.

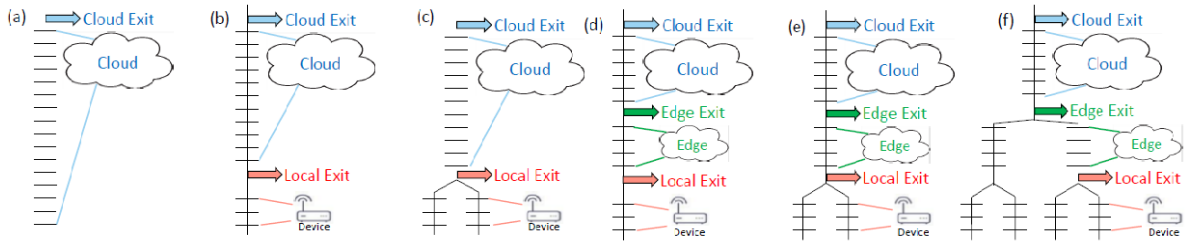


Figure 4.1 - Overview of the DDNN architecture [Teerapittayanon2017]

Similarly, a lightweight scheduler has been proposed in [Kang2017] to automatically partition DNN computation between mobile devices and datacentres at the granularity of neural network layers. Authors profile the mobile device and the server to generate performance prediction models for the spectrum of DNN layer types. This only needs to be done once for a given set of mobile and server platforms. They select the best DNN partition points by utilizing the layer performance prediction models which are used to estimate the latency of executing layers on mobile and cloud. The work also estimates the power of executing layers on the mobile device and calculates the wireless data transfer latency based on the latest wireless network bandwidth. The best partition point is identified according to the target. If the target is latency, the point minimising the total execution and communication time is selected. If the target is energy, the point that minimizes the total energy consumption is identified.

On the other hand, there are some other methods like early exiting and pruning to reduce the execution time and resources. Two examples of such methods are proposed in [Li2020] and [Shiand2019], respectively. In [Li2020], authors focused on inference phase combined with DNN right sizing that further reduces computation latency via early exiting inference at an appropriate intermediate DNN layer and DNN partitioning to maximize the efficiency of DNN inference. They consider static and dynamic configurations depending on bandwidth stability during the DNN inference and they consider a threshold for latency requirement. A runtime optimization algorithm is proposed, whose goal is reaching a trade-off between latency and accuracy. The algorithm proceeds in a greedy way, initially setting the partition point to the last exit point (which has the higher accuracy). Next, the algorithm considers all possible layers partitioning between edge and cloud and calculates the latencies of running the different layers on device and cloud. The partition that leads to minimum latency is selected by inspection. If this latency is lower than an a priori fixed threshold, the partition found is the optimal result, otherwise, they compute the minimum latency for the previous exiting point and reiterate. In [Shiand2019], authors propose a 2-step pruning framework for DNN partition between mobile devices and edge servers. The DNN model is pruned once in the training phase where unimportant convolutional filters are removed iteratively. In pruning step s , they apply an iterative pruning algorithm to the entire original testing network. Since the data generated by the last layer in the front-end part of the network before the partition point needs to be transmitted via wireless channel, pruning can be applied to the layer just before the partition point to shorten the transmission latency. In pruning step 2, they apply the iterative pruning algorithm to each layer in the pruned network after pruning step 1 by restricting the pruning range to each layer. In this way, they identify a set of pruned network models that are generated by pruning each layer, which correspond to the partition point just after the layer. Next, they compute mobile and edge server computation latency and the transmission latency for each pruned network model and select a network model and its corresponding partition point with minimum latency. More recently, a lightweight collaborative DNN for the mobile web, named LcDNN, has been proposed in [Huang2020]. Authors design a lightweight DNN with a binary neural network (BNN) branch to execute the inference of DNN on the mobile web in order to decrease mobile energy consumption and latency. They also propose a joint training method and implement an inference library for running the BNN on the mobile web. Finally, they present an online scheduling based on Deep Reinforcement Learning, called DRLoS, to maximize

the resource utilization of the edge. Vice versa, some other works like [Eshratifar2021] use partitioning both in the inference and the training phase. Authors propose a method, called JointDNN, and reduce the mobile cloud computing optimal scheduling problem to the shortest path problem shown in Figure 4.2, in which top vertices represent the DNN layers on cloud and bottom vertices represent the DNN layers on edge. The weights on horizontal edges denote the computation cost of the current layer on edge or cloud and the weight on diagonal edges represent the communication cost from cloud to edge or vice versa. If the scenario does not have any constraint (mobile battery limitations, quality of service constraints, etc.), the optimal scheduling can be obtained by solving the shortest path problem in the JointDNN graph model. If the scenario has constraints, they proposed an integer linear programming (ILP) formulation of DNN partitioning for both training and inference phase for the different scenarios with different constraints.

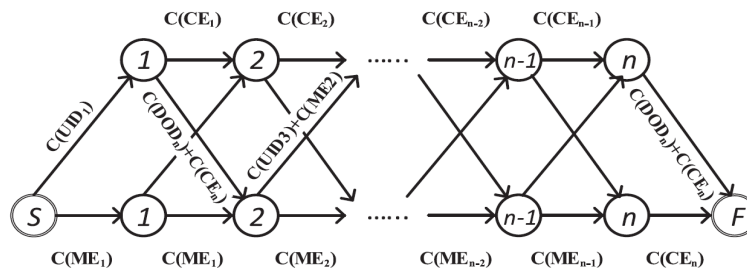


Figure 4.2 - Graph model of JointDNN approach [Eshratifar2021]

In addition to splitting the layers of DNN among devices, splitting the input data sample is another method to reduce computation load on a single device and distribute the load among devices in edge-cloud. A hybrid parallelism method to partition both the DNN and the data samples across the edge device, edge server and cloud centre is proposed in [Liu2020]. Authors considered three levels of tasks: *Original Task* which trains the full DNN with b_o data samples, *Short Task* which trains m_s consecutive layers from layer 1 to layer m_s with b_s data samples and *Long Task* which trains m_l consecutive layers from layer 1 to layer m_l with b_l data samples. By assigning each task to device, edge or cloud, there are 6 different states (permutation of 3). They solved an integer linear programming (ILP) optimization problem for each state to achieve the best state with minimum total time. In the optimization problem, the decision variables are m_s and m_l which are the partition points of layers and b_o , b_s and b_l which are mini-batch size of data sample assigned to device, edge or cloud.

Finally, a methodology to find an optimal placement of Convolutional Neural Networks (CNNs) on the IoT devices is proposed in [Disabato2015]. Authors formulate the problem as an optimization problem for assigning the layer of CNNs to N IoT devices in order to minimize the latency between the data-gathering (input read) and decision-making (inference result) phases within the memory and processing load constraints. Early-exits are also considered to reduce the network traffic and computation requirements for the whole system.

4.2 Network Architecture Search

Neural architecture search (NAS) is the process of automating machine learning solutions design. In the last decade, deep learning solutions like neural networks have been frequently used for their capability of extracting features automatically from data, avoiding the feature engineering step required by traditional machine learning algorithms. Through the years, many engineers developed new techniques to improve neural network accuracy on relevant problems, usually comparing the results obtained from their architectures on reference datasets. Good architectures are studied and developed by engineers, based on the given problem and the actual knowledge available in the literature. Due to the black box nature of the

neural networks, right architecture search can be a tedious procedure, requiring various trial and error steps to find an optimal structure for a given problem and dataset.

The main objective of neural architecture search is to automatize the architecture engineering step, providing algorithms that can search autonomously a set of architectures that provides good accuracy results for the considered problem. Even if NAS techniques were studied in the past, it is only recently that the topic gained lots of attention, thanks to the work of Zoph et al. [Zoph2017], that achieves state-of-the-art results on the CIFAR-10 dataset, a reference dataset for image classification problems, and the Penn Treebank dataset, a dataset used frequently as a benchmark for recurrent neural networks. NAS nowadays, considers the architecture search no more as a single-objective optimization problem, focused only on obtaining the best accuracy possible, but as multi-objective optimization problems, e.g., to reduce the time required by the NAS algorithm or to constraint the search on additional requirements like the latency or the memory required by the final architecture.

4.2.1 NAS dimensions

NAS algorithms can be extremely heterogeneous and are characterized by three dimensions: search space, search strategy, and performance estimation strategy [Elsken2019]. In Figure 4.3, an abstract illustration of NAS methods is depicted. Search space \mathcal{A} is defined by the set of operations defined by the NAS designer. Search strategies select architectures from \mathcal{A} , that are then evaluated with a performance estimation strategy, estimating their accuracy on unseen data.

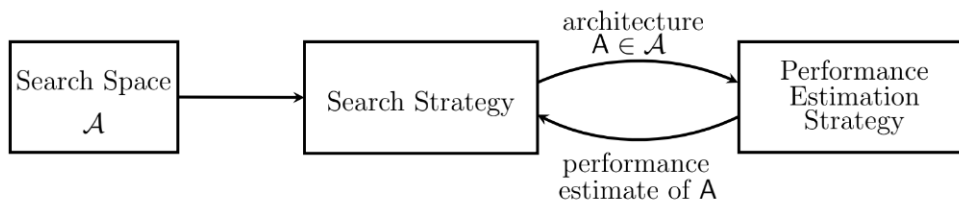


Figure 4.3 - Overview of the workflow used by neural architecture methods [Elsken2019-2]

Search space

The search space defines the space of all the possible architectures that can be produced by the NAS algorithm. This is commonly defined by the algorithm designer, through a set of operations that are considered well-suited for the task addressed by the output architectures. Defining a set of operations a priori reduces the size of the search space, simplifying the search, and also introduces a human bias in the algorithm. This is in part beneficial, because operations chosen by designers are often extensively used in already existing neural architectures and that proved to achieve very good performance but can also be seen in a limitation in finding new building block not currently discovered in the literature, especially for problems where actual architectures still do not perform well enough. Recent works on NAS incorporate modern design elements known from hand-crafted architectures such as skip connections, which allow to build complex, multi-branch networks. Hand-crafted architectures usually find a performing combination of operation and inputs, forming a “building block” that is then repeated multiple times across the network to build the actual architectures.

Also, NAS techniques embrace this motif repetition pattern, focusing on the search of good cells and blocks to be repeated inside the architecture, instead of searching for a whole architecture. Micro-search focuses on the cell structure that can be composed by multiple blocks combined in a multi-branch way. Usually, two types of cells are defined: a normal cell, that preserves the dimensionality of the input, and a reduction cell, which

reduces the spatial dimension. Macro-search focuses instead on how these cells are stacked to produce the final architecture. A common approach is to connect them in a single path chain-structured network, where multiple normal cells and single reduction cells are interleaved (see Figure 4.4). This search space has two major advantages:

- The size of the search space is drastically reduced since cells can be comparably small
- Cells can more easily be transferred to other datasets by adapting the number of cells used within a model

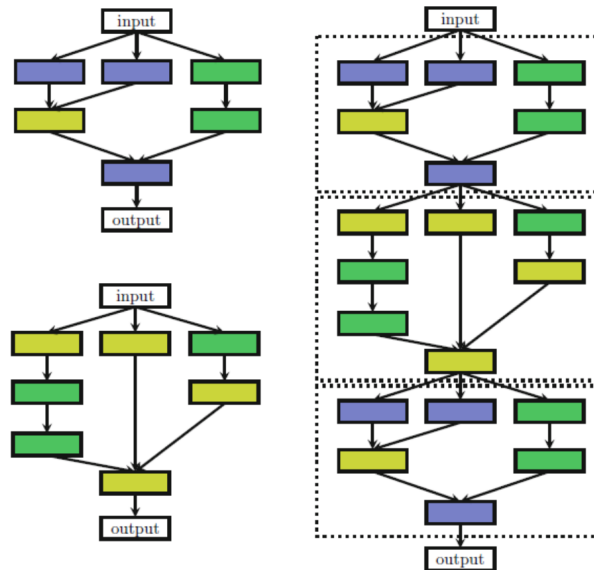


Figure 4.4 - Cells Architecture. On the left, a normal cell (top) and a reduction cell (bottom). Micro-search focuses on finding the best structure for these two cells. These cells are then stacked together to form the neural architecture on the right, according to macro-search results [Elsken2019-2]

Search Strategy

Search strategy is responsible for exploring the search space and selecting architectures to evaluate. Many different search strategies that have been employed, can be mainly organized in four categories:

- **Reinforcement learning (RL):** Reinforcement learning is based on a controller, usually a recurrent neural network (RNN) like a LSTM. The controller receives a string embedding as input and generates a neural architecture as output, considered as the agent's action, where the action space is the search space itself. The reward of the action is simply the performance estimation of the architecture produced. The main difference between approaches using reinforcement learning is how the controller learns to produce the architectures. The action reward is used in the agent's policy that is chosen by the NAS designer: some examples are REINFORCE [Williams1992] [Zoph2016] and Q-Learning [Beakcheol2019].
- **Evolutionary algorithms (EA):** Evolutionary methods use genetic algorithms to search for optimal architectures. Even if genetic algorithms can be used also to optimize the weights of an architecture, gradient-based methods like stochastic gradient descent (SGD) are usually employed as they are much faster. Evolutionary algorithms start from a population of promising architectures and gradually evolve them by crossover and mutation operations, to generate a new set of architectures to use in the next iteration of the algorithm. Neuro-evolutionary methods differ in how they sample parents, update populations, and generate offsprings. Usually, crossover and mutation rates are adaptive and

change during the search procedure, providing a way to balance exploration and exploitation at runtime.

- *Gradient-based*: Gradient-based methods relax the constraints of the search space to a continuous one, allowing the application of gradient-based techniques also to optimize the architecture search. Using this type of architecture search makes it possible to optimize both architecture weights and search procedure using the same technique, simplifying the NAS algorithm. Gradient-based method was introduced by DARTS [Liu2019] and it has been a very popular choice for NAS algorithms since its introduction.
- *Bayesian optimization (BO)*: Bayesian optimization is the most popular method for hyperparameter optimization tools for neural networks, but it has not been applied too much in NAS algorithms because they are focused on low-dimensional continuous optimization problems, where instead NAS search space is high-dimensional and discontinuous. Bayesian optimization methods start from a prior Gaussian distribution of the objective function, which is updated at each iteration in a new Gaussian distribution, called posterior. The distribution update is based on the evaluation of the sample extracted from the distribution, also keeping track of the confidence interval of the estimations to enable an exploration-exploitation trade-off.

Performance estimation strategy

Performance estimation is a necessary step to guide the search strategy into finding the most promising architectures in the considered search space. In fact, neural architectures selected by the search strategy cannot be fully trained to provide an accurate validation error, because it would require an immense GPU time, considering that search spaces of the NAS algorithms can be in order of 10^{15} architectures or more and that therefore many searches must be done to find the best architectures. The main goal of the estimation phase is to correctly rank the accuracy, so that the best architectures can be selected for next iterations, or the controller can be awarded proportionally to the potential accuracy of the architecture after full training. Since the exact accuracy is not required for performing the search, various techniques exist for performing estimation strategy, providing shallower results but in a very fast time.

The main ones are:

- *Low-fidelity estimation*: Some strategies simply focus on providing lower fidelity estimates compared to a full network training, by training on lower resolution images, a subset of data and/or training for a small number of epochs, using a high learning rate to compensate a bit for the fast training. Even if the computational cost is drastically reduced, this approach introduces bias towards smaller and simpler architectures, because they train more easily and therefore provide good results even with a short training. There is study evidence in the literature that the relative ranking of the architectures can change drastically because of this intrinsic bias, making this method not well-suited for general use.
- *Learning curve extrapolation*: Learning curve extrapolation methods focus on the initial learning curve, terminating the architectures that are predicted to perform poorly, speeding up the search.
- *Weight inheritance*: Weight inheritance speeds up the performance estimation of an architecture by initializing its weights with the ones of another compatible architecture that was previously trained. One way to allow weight inheritance is through network morphism, a procedure that allows one to change an architecture to make it more similar to the one from which will inherit the weights, but without changing the function of the network.
- *Weight sharing through pretrained supernet (one-shot model)*: One-Shot Architecture Search is another promising approach for speeding up performance estimation, which treats all architectures as different subgraphs of a supergraph (the one-shot model) and shares weights between

architectures that have edges of this supergraph in common. Only the weights of a single one-shot model need to be trained (in one of various ways), and architectures (which are just subgraphs of the one-shot model) can then be evaluated without any separate training by inheriting trained weights from the one-shot model. This greatly speeds up performance estimation of architectures since no training is required (only evaluating performance on validation data). This approach typically incurs a large bias as it under-estimates the actual performance of architectures severely; nevertheless, it allows ranking architectures reliably, since the estimated performance correlates strongly with the actual performance.

4.2.2 One-shot models

Training thousands of models is difficult or impossible for a typical machine learning practitioner. One promising direction is sharing weights between models: rather than training thousands of separate models from scratch, one can train a single large network capable of emulating any architecture in the search space. The work in [Bender2018] is considered the baseline work for a comprehensive analysis of one-shot architecture. It provides a very simple example for grasping the main concepts of a supernet.

In Figure 4.5, we can see that a set of 3 different operations is used: 3x3 convolution, 5x5 convolution and max pooling. A single network can be trained using all these 3 operations, instead of training 3 separate models. At evaluation time two paths can be zeroed, selecting an operation for the searched architecture. A complete search space is evaluated with the use of a large supernet composed of multiple cells; this approach is feasible also for quite large search spaces because an exponential growth of the search space is tied to a linear growth of the one-shot model. This can be intuitively seen in the example: stacking 4 cells would lead to $3^4 = 81$ possible architectures but requires only x4 the memory of a single cell.

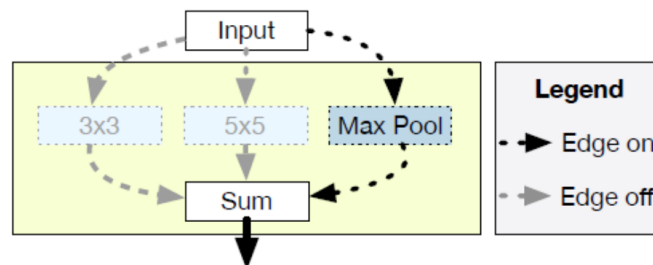


Figure 4.5 - Simple example of a one-shot model. In a choice block, we have 3 different operations, during training all the weights are trained, while at runtime only one path is chosen [Bender2018]

The main advantage of using a one-shot model is that, after the resource investment for training and storing the supernet, search space can be performed very efficiently thanks to weight sharing. Another interesting result of the paper is that neither a hypernetwork nor an RL controller are required to achieve good results. By properly training the supernet, the network focuses mainly on most important operations for having good results, so zeroing them lead to low rewards during search, instead zeroing out non-performing operations have a minor impact. This behaviour allows search strategies with gradient-based techniques, without controllers.

The common procedure for one-shot architecture search can be summarized as follows:

- Design a search space for the one-shot model.
- Train the one-shot model to make it predictive of the validation accuracies of the architectures.
- Evaluate candidate architectures on the validation set using the pre-trained one-shot model.
- Re-train the most promising architectures from scratch and evaluate their performance on the test set.

One-shot models provide proxy evaluation metrics by activating subgraphs of the supernet. The one-shot model is a standard large neural network trained using SGD with momentum but requires some extra considerations to provide good evaluation results.

The main problem is related to co-adaptation of model weights. Training naively the supernet leads to weights co-adapting during training, causing high accuracy drops also when removing unimportant operations during the model predictions. Incorporating path dropout during training mitigates this issue, allowing weights to train more independently. Dropout can alter the training speed and final accuracy, so should be tuned during training. Bender et al. [Bender2018] achieved good results by disabling path dropout at training start and gradually increasing over time, reaching $r^{\frac{1}{k}}$ at the end of supernet training, where $0 < r < 1$ is a hyperparameter and k is the number of incoming paths to a given network operation. In this way, the more inputs there are, the more likely they are to be dropped out, but the probability of dropping all inputs is constant to r for all operations. Training is very sensible to r , so it must be tuned properly (see Figure 4.6).

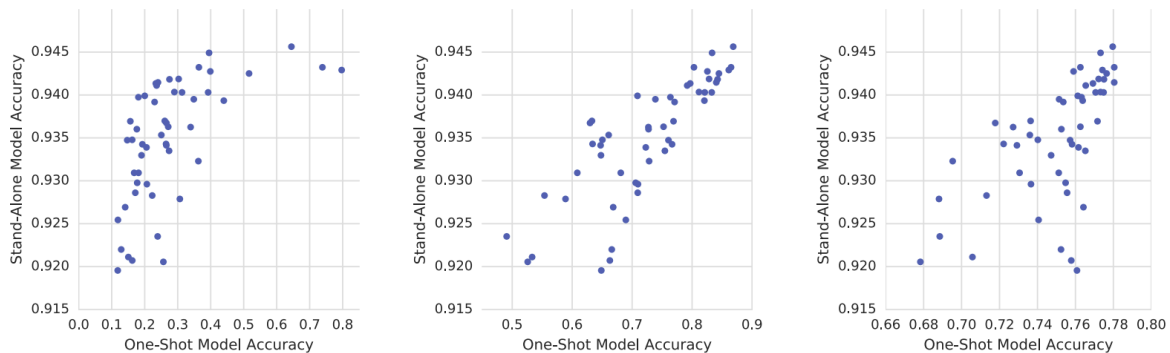


Figure 4.6 - One-shot models trained with constant dropout rates. Left: model training with a constant dropout rate of $r = 10^{-6}$; paths in the network are retained more than 85% of the time. Middle: model trained with a constant dropout rate of $r = 0.01$, a moderate value. Right: model trained with a constant dropout rate of $r = 0.3$, a high value where some paths are retained only 15% of the time [Bender2018]

Another problem is that training can be quite unstable. To stabilize training, batch normalization proved to be quite effective, but must be computed on the fly because during evaluation subgraphs will have different batch statistics. A variant of ghost batch normalization further stabilized the training in Bender work. The last addressed problem in the paper is over-regularization. To prevent over-regularization, L2 regularization is applied only to parts of the model that are used by the current architecture. Without this change, layers that are dropped out frequently are regularized more.

Existing one-shot methods, however, are hard to train and not yet effective on large scale datasets like ImageNet. The work of Guo et al. [Guo2020] proposes single path one-shot models to address these challenges. One-shot models solve the problem of joint optimization of architecture search and supernet training (that cause bias towards simpler architectures in general), because the architecture search problem is decoupled from the supernet training and addressed in a separate step, in a sequential manner. Still, there are coupled weights in the supernet, making optimization complicated and sensitive to hyperparameters. Guo cites Bender work, focusing on solving the training sensitivity on dropout rate parameter, that makes supernet training hard for complex datasets.

The key to the success of architecture search is that the accuracy of any architecture a on a validation set using inherited weight (without extra fine tuning) is highly predictive for the accuracy of a that is fully trained. This gives rise to the principle that the supernet weights should be optimized in a way that all architectures in the search space are optimized simultaneously. This can be expressed as:

$$W_{\mathcal{A}} = \underset{W}{\operatorname{argmin}} E_{a \sim \Gamma(\mathcal{A})} \left[\mathcal{L}_{\text{train}}(\mathcal{N}(a, W(a))) \right]$$

where \mathcal{A} is the search space, W are supernet weights, $W_{\mathcal{A}}$ are optimal weights for the supernet, $\mathcal{N}(a, W(a))$ is a subgraph of the supernet (representing architecture a) and $\Gamma(\mathcal{A})$ is a prior distribution of $a \in \mathcal{A}$.

Guo proposes to structure the supernet so that each architecture is a single path, reducing weight co-adaptation. This means that each choice block has only one choice, instead of multiple ones. Compared to dropout strategy, this approach allows an hyperparameter-free training, simplifying the procedure while achieving even better results. Also, only weights $W_{\mathcal{A}}$ are activated and updated at each step, making memory usage efficient. It also proposes to use a uniform sampling as a fixed prior $\Gamma(\mathcal{A})$, instead of learning it like other existing approaches, that would lead to difficult optimization and more correlation among supernet weights and architecture parameters.

4.2.3 HardCoRe-NAS: Hard Constrained differentiable NAS

A popular approach to find fitting networks is through constrained Neural Architecture Search (NAS), however, previous methods enforce the constraint only softly. Therefore, the resulting networks do not exactly adhere to the resource constraint and their accuracy is harmed. HardCoRe-NAS algorithm [Nayman2021] instead force hard constraints without any additional tuning required, achieving state of the art architectures. The motivation of the focus on hard constraints is motivated by the latest year's interest in deployment on edge devices and standard CPUs. These are more limited computing platforms, requiring lighter architectures that for practical scenarios have to comply with hard constraints on the real time latency or power consumption.

HardCoRe-NAS propose a gradient-based approach, using a pretrained one-shot model. It searches the space for sub-networks by solving a hard constrained optimization problem while keeping the one-shot model pretrained weights frozen. The constrained optimization is solved via the block coordinate stochastic Frank-Wolfe (BC-SFW) algorithm, which, according to tests, resulted faster in convergence than SGD, while also satisfying more tightly the constraints. The architecture search \mathcal{S} is parameterized by $\zeta \in \mathcal{S}$, governing the architecture structure in a differentiable manner, and by w , the convolution weights. In the paper, it is explained how a latency-constrained NAS can be seen as a bilevel optimization problem, where the architecture chosen must minimize its error on validation set, while satisfying the latency constraint strictly. An architecture $\zeta = (\alpha, \beta)$ is a subgraph of the supernet, where α is the set of parameters related to the micro-search and β the set of parameters related to the macro-search. Also, strict resource constraints can be parameterized using these sets.

Regarding micro-search, every block is an elastic version of the MBInvRes block, introduced in [Sandler2018], with expansion ratio $er \in A_{er} = \{3, 4, 6\}$ of the point-wise convolution, kernel size $k \in A_k = \{3 \times 3, 5 \times 5\}$ of the depth-wise separable convolution (DWS), and Squeeze-and-Excitation (SE) layer [Hu2018] $se \in A_{se} = \{\text{on}, \text{off}\}$. The blocks are configurable for each block b of stage s . Regarding macro-search instead, the output of each block of every stage is also directed to the end of the stage as illustrated in the middle of Figure. 4.7. In conclusion, an architecture of the search space can be fully defined by an assignment of these boolean parameters α and β , that can be seen as a mask to select the architecture subgraph from the supernet.

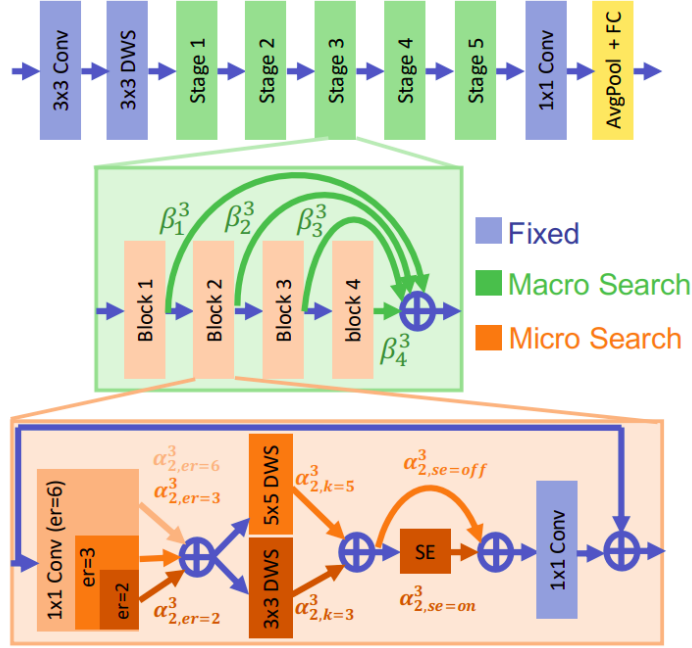


Figure 4.7 - HardCoRe-NAS search space [Nayman2021]

Any resource constraint can be estimated from the architecture configuration.

$$LAT(\alpha, \beta) = \sum_{s=1}^S \sum_{b'=1}^d \sum_{b=1}^{b'} \sum_{c \in C} \alpha_{b,c}^s \cdot t_{b,c}^s \cdot \beta_{b'}^s$$

In the paper is presented this example about latency constraint, that can be seen as the sum of latency measurements $t_{b,c}^s$ for each cell c , block b , depth b' and stage s . By replacing $t_{b,c}^s$ with any other resource measurement, it is possible to define additional constraints and insert them into the optimization problem solved by the NAS algorithm. Optimal weights w^* are obtained only once, by using a uniform sampling of the supernet. For every training dataset's image in the batch multiple distinctive paths of the supernet are sampled, using the Gumbel-Softmax trick. For every feature x going through a block b at stage s , two random variables $U_{b,c,x}^s, U_{b,x}^s$ are sampled, governing the path taken by feature map:

$$\hat{\alpha}_{\{b,c,x\}}^s = \frac{e^{\log(\alpha_{b,c}^s) - \log(\log(U_{b,c,x}^s))}}{\sum_{c \in C} e^{\log(\alpha_{b,c}^s) - \log(\log(U_{b,c,x}^s))}}$$

$$\hat{\beta}_{\{b,x\}}^s = \frac{e^{\log(\beta_b^s) - \log(\log(U_{b,x}^s))}}{\sum_{c \in C} e^{\log(\alpha_{b,c}^s) - \log(\log(U_{b,x}^s))}}$$

This approach leads with high probability in sampling a number of paths as high as the batch size. Using multiple paths also reduces the variance of the gradients.

Regarding objective constraints, HardCoRe-NAS do not use soft penalties in the loss function, instead it directly restricts the search space based on the budget limitations. For example, by imposing only a latency constraint: $S_{LAT} = \{\zeta | \zeta \in P_\zeta(S), LAT(\zeta) \leq T\}$. S_{LAT} is then relaxed to a discrete search space, where stochastic Frank-Wolfe (SFW) algorithm is applied to find the best architecture, i.e., the one that minimizes the error on the validation set.

At the end, a single feasible architecture is extracted by a discretization step. Setting α and β parameters equal to 1 when argmax for a given block, cell, stage combination is not enough, because could violate resource constraints. HardCoRe-NAS proposes an alternative projection step: considering solution (α^*, β^*) the credit assigned to each supernet configuration and a latency constraint $LAT(\zeta) \leq T$, it is possible to perform the discretization by solving two linear programs:

$$\begin{aligned} \max_{\alpha \in S^{\alpha_*}} \alpha^T * \alpha^* \\ \max_{\beta \in S^{\beta}} \beta^T * \beta^* \end{aligned}$$

This allows to maximize the credit while strictly satisfying the latency constraint. A theorem is provided by paper authors to guarantee that the projection produces a sparse solution, meaning that it represents a valid sub-graph of the one-shot model.

4.2.4 CARS: Continuous Evolution for Efficient NAS

CARS [Yang2020] proposes an efficient EA-based neural architecture search framework. A continuous evolution strategy is developed to maximally utilize the knowledge learned in the last evolution generation. CARS can provide a series of models on the Pareto front with high efficiency, by exploiting a supernet approach, which is initialized at search start and updated during the search, allowing to share the weight efficiently also in an evolutionary algorithm.

CARS search stage is composed of two stages: parameter optimization and architecture optimization, which are conducted alternatively. Genetic Algorithm (GA) is used for architecture evolution, because GA maintains a set of well-performed architectures that cover a vast space. At each search step, a set of P architectures $\{C_1, \dots, C_p\}$ is gradually updated according to the paper proposed pNSGA-III method. After that, the most promising P architectures are retained for the next search step.

Some experiments in the literature [Real2019] show that the evolutionary algorithm discovers better neural architectures than RL-based approaches, but the search cost of EA is much more expensive due to the evaluation procedure of each individual, i.e., a neural network in the evolutionary algorithm is independently evaluated. One of the main contributions of CARS is to adapt evolutionary algorithms for using a supernet approach, which is extremely dominant in recent works for its efficiency in search time. Initially, a parameter warm up step is required to correctly initialize the supernet and use it for efficient search. Since the shared weights of the supernet are randomly initialized, if the architectures in the population are also randomly initialized, the most frequently used operations for all the architectures would be trained more times compared with other operations. Thus, by following one-shot NAS methods, a uniform sampling strategy is used to initialize the parameters in the supernet. In this way, the supernet trains each possible operation with the same possibility.

At each step, different networks N_i are sampled from the supernet N having weights W . Each network N_i can be represented as a (W_i, C_i) , where C_i are binary connection parameters, used to mask connection in the supernet, while W_i are the weights of the architecture, obtained with formula: $W_i = W \odot C_i$.

Parameters W are updated through the SGD algorithm. The gradients over a mini-batch of B architectures are taken as an unbiased approximation of the averaged gradients of all the P different individuals. The time cost for each update could be largely reduced, and the appropriate mini-batch size leads to a balance between efficiency and accuracy.

$$dW \approx \frac{1}{B} \sum_{j=1}^B \frac{\vartheta L_{n_j}}{\vartheta W_{n_j}}$$

As for the architecture optimization procedure, the evolution algorithm is used together with a non-dominated sorting strategy. The non-dominated strategy can be defined as follows: Considering two networks \mathcal{N}_i and \mathcal{N}_j and a series of measurements $\{F_1, \dots, F_m\}$ we want to minimize. If:

$$\mathcal{F}_k(\mathcal{N}_i) \leq \mathcal{F}_k(\mathcal{N}_j) \forall k \in \{1, \dots, M\}$$

$$\mathcal{F}_k(\mathcal{N}_i) < \mathcal{F}_k(\mathcal{N}_j) \exists k \in \{1, \dots, M\}$$

\mathcal{N}_i is said to dominate \mathcal{N}_j ($\mathcal{N}_i \leq \mathcal{N}_j$). Applying the non-dominated strategy, it is possible to retain a set of non-dominated neural architectures, approximating the Pareto front.

Authors tried to apply the NSGA-III [Deb2014] algorithm for evolving the generations of architectures. NSGA-III (Non-dominated Sorting Genetic Algorithm) is an algorithm designed for general evolutionary multi-objective optimization (EMO) problems. Many difficulties must be faced by EMO to work properly in a large dimensional space (multi-objective with ≥ 4 dimensions), in particular:

- With a high number of objectives, a large fraction of the population becomes non-dominated. Most EMO algorithms emphasize non-dominated solutions in a population, trying to keep them between iterations. This can slow down the search process.
- Recombination operation can become quite inefficient in case only a handful of solutions are found, because solutions are likely widely distant from each other. In that case, recombination between distant solutions will produce solutions distant from the parents, producing questionable results. A special recombination operator with mating restriction must be used to handle this case efficiently.

Key idea used by NSGA-III and other EMO algorithms is to use a special domination principle that discretize adaptively the Pareto front, producing a set of finite points with similar distance between each other. This can result in quite difficult to achieve, so instead of searching the entire search space multiple targeted searches (reference points) can be set to alleviate this issue. Recombination issues are instead addressed with mating-restriction schemes.

In extreme synthesis, NSGA-III follows these steps:

- Use a domination principle to select non-dominated solutions at each step.
- At start, choose reference points equally distributed in the hyperspace or use user-defined ones.
- Normalize the hyper-space based on extremes individuated by actual population members at each step, adapting to different scales of objective values in a flexible way.
- Associate each member of the population to a reference point.
- Since multiple members can be associated to the same reference point and some reference point can have no association, prune useless points and choose a single member randomly for each reference point with multiple associations.
- Use crossover and mutation operations on randomly picked parents to generate an offspring.
- Repeat for a given number of iterations.

Note that NSGA-III is a parameter-less algorithm, only the classical genetic algorithm parameters must be defined such as the population size, termination parameter, crossover and mutation probabilities, and their associated parameters.

CARS authors tried to apply NSGA-III but found that it suffered from a “small model trap phenomenon”. This is caused by the fact that some smaller models with fewer parameters, but higher test accuracy tend to dominate those larger models which achieve a lower accuracy, but which have the potential for achieving higher accuracies later on. To solve this defect, CARS authors proposed the pNSGA-III method, an extension to the original NSGA-III algorithm. Besides, considering the number of parameters and accuracy, pNSGA-III also considers the increasing speed of the accuracy and the number of parameters. This allows to preserve larger promising models into the architecture generation of the evolutionary algorithm.

4.2.5 Pareto-Frontier-aware Neural Architecture Generation for Diverse Budgets

Many existing methods seek to design a model to obtain an architecture under a single budget. When considering diverse budgets, they have to conduct multiple search processes for each budget, which is very inefficient yet unnecessary. The proposed Pareto-Frontier-aware Neural Architecture Generator (NAG) instead takes an arbitrary budget as input and produces the Pareto optimal architecture for the different target budget under the input one. NAG considers the same search space when designing architectures for different budgets. Based on a common architecture under some budgets, it is possible to obtain promising architectures for a larger/smaller budget by slightly modifying some of its elements/layers. In this sense, it is possible to find effective architectures for diverse budgets simultaneously, exploiting the knowledge provided by other architectures to benefit the search process.

NAG is composed mainly by an architecture generator, based on reinforcement learning techniques, and an architecture evaluator, that evaluates architectures under different budgets and utilizes a Pareto dominance rule to determine whether an architecture is better than another one. It is also based on a supernet approach to share weights. All resource constraints (e.g., latency and multiply-adds, MAdds) are defined in budget B . Given a search space Ω , for each architecture $\alpha \in \Omega$ it is possible to define $c(\alpha)$ and $Acc(\alpha)$ as cost and validation accuracy respectively.

The architecture generator is based on an LSTM network. It takes a budget B as input and outputs an architecture α_B that satisfies budget constraints ($c(\alpha_B) \leq B$). The LSTM controller is updated based on the reward received after training.

To approximate the Pareto front, the input budget is evenly sampled K times. Budget embedding vectors are learnable and incorporated into the parameters of the architecture generator to train them jointly. To accommodate all the budgets that belong to a continuous space, an embedding interpolation method is used to represent a budget with any possible value. The interpolation formula is:

$$d(\beta_1, \beta_2, B) = \begin{cases} 1, & \text{if } (c(\beta_1) \leq B \wedge c(\beta_2) \leq B) \wedge (Acc(\beta_1) \geq Acc(\beta_2)); \\ -1, & \text{else if } (c(\beta_1) \leq B \wedge c(\beta_2) \leq B) \wedge (Acc(\beta_1) < Acc(\beta_2)); \\ 1, & \text{else if } c(\beta_1) \leq c(\beta_2); \\ -1, & \text{otherwise.} \end{cases}$$

$$b = g(B) = \xi g(B_1) + (1 - \xi)g(B_2), \text{ where } \xi = \frac{B_2 - B}{B_2 - B_1}$$

where $\xi \in [0, 1]$ is the weight of the interpolation.

To guide the training of the NAG, an evaluator composed of three fully-connected layers is used to predict the performance of the trained network under the given budget. The evaluator makes use of a Pareto dominance function, which simply states that, given two architectures β_1 and β_2 , β_1 dominates β_2 in case $c(\beta_1) \leq B$ and $c(\beta_2) \leq B$ but $Acc(\beta_1) \geq Acc(\beta_2)$. Formally, we define the Pareto dominance function $d(\beta_1, \beta_2, B)$ to reflect the above rules.

The evaluator collects the results of M architectures at each training step, recorded as triplets $\beta_i, c(\beta_i), Acc(\beta_i) \forall i \in 1, \dots, M$, then trains using a pairwise ranking loss, based on the $M(M - 1)$ pairs extracted. Figure 4.8 illustrates the scheme of the NAG algorithm.

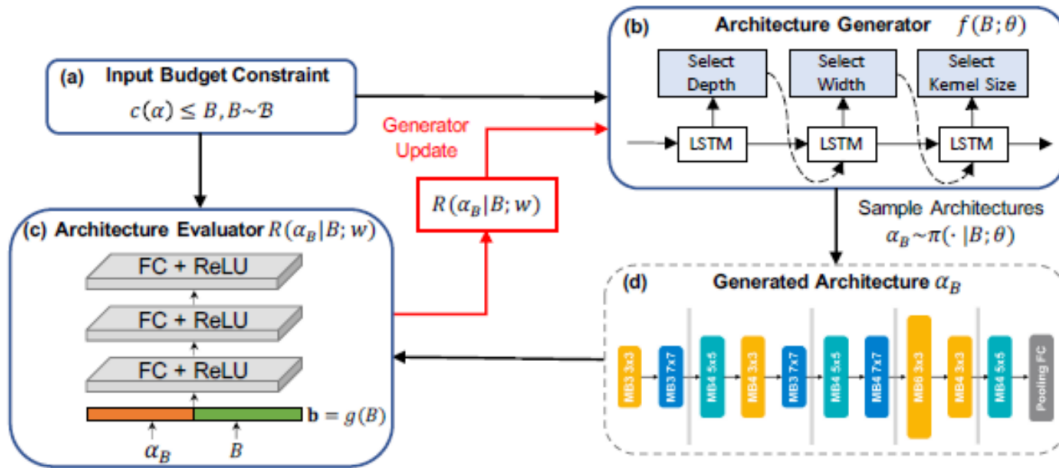


Figure 4.8 - Recap of NAG algorithm. A budget B (sampled from \mathcal{B}) is given as input to the architecture generator, which produce an architecture α_B . Both α_B and budget embedding b are then used to generate the reward of the architecture, which is used for both architecture evaluator and architecture generator for training

4.2.6 Weak NAS predictors

Most recent NAS methods attempt to model the performance distribution over the whole architecture space using a strong predictor. However, since the architecture space is often exponentially large and highly non-convex, modelling the whole space is very difficult. The work in [Wu2021] investigates the approach of utilizing a few weak predictors on small local spaces and progressively moving to the space where good architecture habits, based on the assumption that the search area is subdivided into some sort of clusters: some search space areas produce good architectures, while some areas produce relatively bad architectures. Similar architectures have been proved to have similar accuracies in studies of NAS-bench-101 and NAS-bench-201 datasets. An effective representation is provided by Figure 4.9.

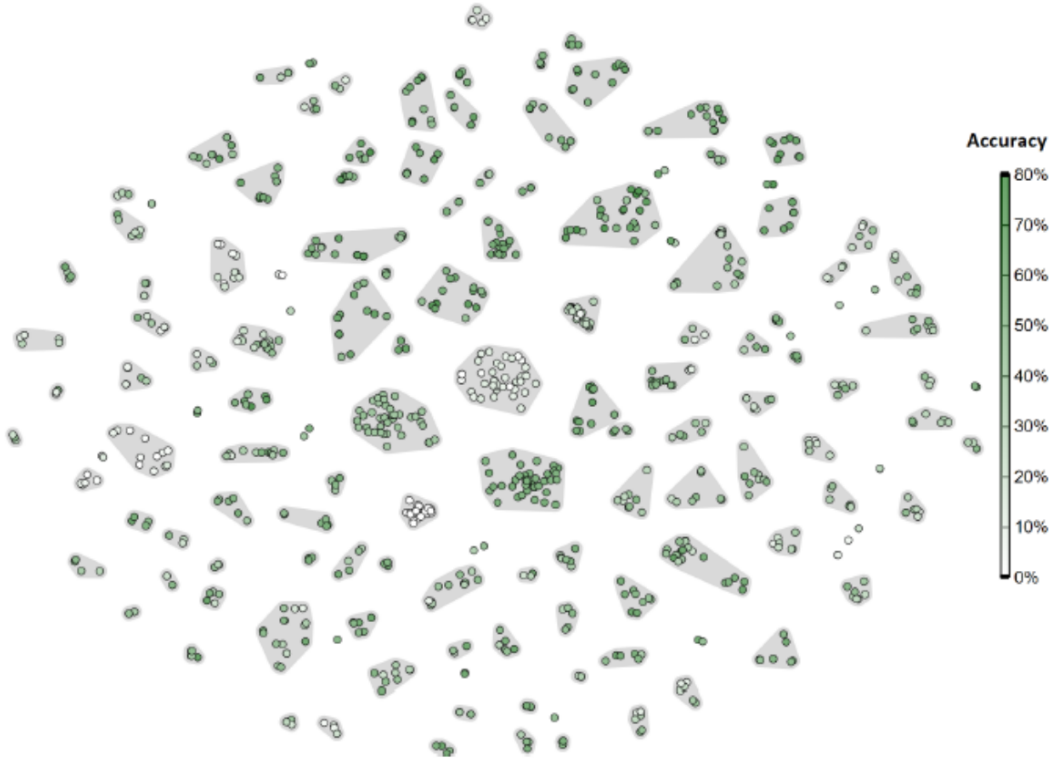


Figure 4.9 - t-SNE visualization of NAS-bench-201 on CIFAR100. Accuracies are encoded by color [Wu2021]

Rather than expecting a strong predictor to model the whole space, weak NAS predictors method seeks a line of progressive evolving weak predictors that can connect a path to the best architecture. In this way, it greatly simplifies the learning task of each predictor. To ensure moving towards the best architecture along the path, at each iteration, the sampling probability of better architectures keep increasing through the guidance of the previous weak predictor. Then, the consecutive weak predictor with better samples will be trained in the next iteration. The iteration proceeds until it arrives at an embedding subspace where the best architectures reside. The weak predictor achieved at the final iteration becomes the dedicated predictor focusing on such a fine subspace and the best performing architecture can be easily predicted.

As it is not possible to train all searched architectures to get their accuracy, predictor-based NAS try to learn a proxy predictor to approximate an architecture accuracy. This can be expressed as:

$$x^* = \underset{x \in X}{\operatorname{argmax}} \tilde{f}(x|S)$$

$$s. t. \tilde{f} = \underset{S, \tilde{f} \in \tilde{\mathcal{F}}}{\operatorname{argmax}} \sum_{s \in S} \mathcal{L}(\tilde{f}(s), f(s))$$

where \mathcal{L} is the loss of the predictor \tilde{f} , $\tilde{\mathcal{F}}$ is the set of all possible approximations and S is the set of training pairs (architecture, accuracy) for predictor \tilde{f} .

Many predictor-based NAS methods attempt to learn a proxy predictor by sampling uniformly the search space. A biased sampling strategy is instead more desirable given previous considerations. Also, training such a predictor \tilde{f} is very complex because the architecture space is incredibly large and samples S are limited. Predictor \tilde{f} and sampling S should therefore co-evolving. The paper proposes to jointly optimize sampling and learning stages progressively, which can be formulated as follows:

$$\begin{aligned}
\text{Sampling Stage: } \tilde{P}^k &= \{\tilde{f}_k(s) | s \in X \setminus S^k\} \\
S^{k+1} &= \{s_i\}_{i=1}^M \cup S^k, \\
\text{where } \tilde{f}_k(s_i) &\in \text{Top}_N(\tilde{P}^k),
\end{aligned}$$

$$\begin{aligned}
\text{Learning Stage: } x^* &= \underset{x \in X}{\operatorname{argmax}} \tilde{f}(x | S^{k+1}) \\
\text{s. t. } \tilde{f}_{k+1} &= \underset{\tilde{f} \in \tilde{\mathcal{F}}}{\operatorname{argmin}} \sum_{s \in S^{k+1}} \mathcal{L}(\tilde{f}(s), f(s))
\end{aligned}$$

Training is initialized by randomly sampling the search space X to get S^1 to train the initial predictor f_1 . After that, S^k and predictor \tilde{f}_k are jointly optimized for K iterations.

At iteration $k + 1$, all architectures in the search space X (excluding all the samples already in S^k) are sorted according to predicted performance \tilde{P}^k at the previous iteration. For the given sample budget, M architectures are randomly sampled from the top N ranked architectures in \tilde{P}^k . Sampled architectures are then added to S^k to form S^{k+1} . Predictor \tilde{f}^{k+1} is learned, such that it minimizes loss \mathcal{L} of the sampled architectures S^{k+1} . All architectures in the search space are then evaluated by \tilde{f}^{k+1} to form \tilde{P}^{k+1} . Through these stages of alternate iteration, predictor \tilde{f}_k guides the sampling process to zoom into the promising architecture samples. In addition, samples S^{k+1} improve \tilde{f}^{k+1} because they are focused on areas of promising architecture.

4.2.7 FBnetV2: Differentiable NAS for Spatial and Channel Dimensions

Differentiable Neural Architecture Search (DNAS) has demonstrated great success in designing state-of-the-art, efficient neural networks. DNAS methods search space is small when compared to other search methods, since they are based on a supernet that must reside in memory. FBnetV2 [Wan2020] proposes a DNAS variant called DMaskingNAS, that supports searches over spatial and channel dimensions that are otherwise prohibitively expensive: input resolution and number of filters. Supernet approaches suffer from two significant limitations:

- Memory costs bound the search space, as the supergraph must reside in GPU memory for training.
- Cost grows linearly with the number of options per layer, limiting search dimension options.

DMaskingNAS propose some optimizations, allowing a memory and computationally efficient DNAS that optimizes both macro-architectures (resolution, channels) and micro-architectures (building blocks) jointly in a $10^{14}x$ larger search space using differentiable search. It also uses a masking mechanism and effective shape propagation for feature map reuse. This is applied to both the spatial and channel dimensions in DNAS.

DMaskingNAS allows searching over spatial and channel dimensions. To support searches over varying numbers of channels, previous DNAS methods simply instantiated a block for every channel option in the supergraph. The main issues related to channel search are:

- Incompatible dimensions: each cell output is a weighted sum of all operations that each the output. This means that all block outputs must align dimensions, disallowing different filter sizes.
- Slower training, increased memory cost: each convolution with a different channel option must be run separately, resulting in a $O(k)$ increase in FLOP cost. Furthermore, each output feature map must be stored separately in memory.

FBnetV2 proposes to consider convolution blocks that can have a different number of filters as blocks with fixed dimensions (max possible dimension) and use Gumbel Softmax output to mask the extra channels. Since all blocks have the same number of filters, they can be represented by a single convolution b using weight sharing. Finally, when combining multiple convolution blocks, it is possible to aggregate also the Gumbel mask, running the computation only once on the block b (see Figure 4.10). Using these optimizations, it is possible to perform channel search with negligible cost.

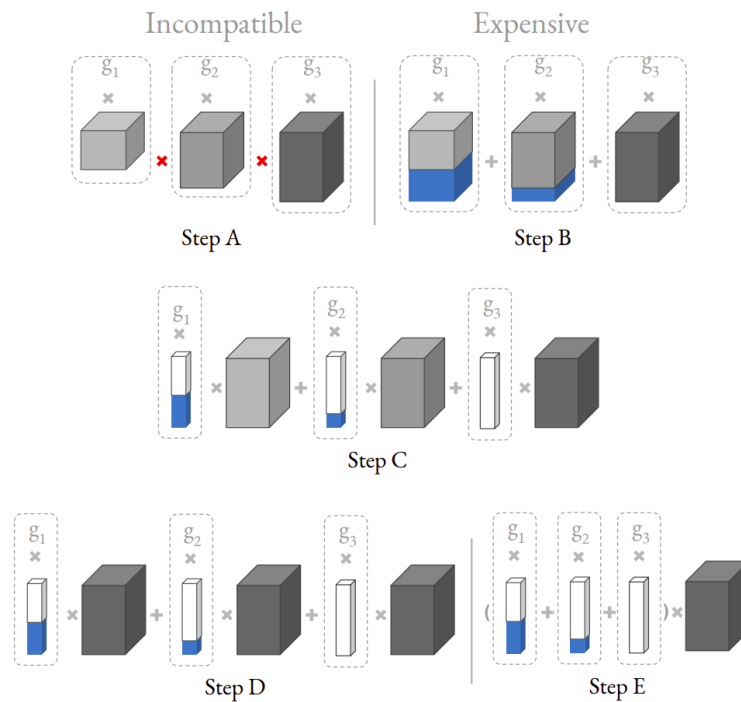


Figure 4.10 - Channel Search Challenges: **Step A:** Consider 3 convolutions with varying numbers of filters. Each output (gray) will have varying numbers of channels. Thus, the outputs cannot be naively summed. **Step B:** Zero-padding (blue) outputs allows them to be summed. However, both FLOP and memory cost increases sub-linearly with the number of channel options. **Step C:** This is equivalent to running three convolutions with equal numbers of filters, multiplied by masks of zeros (blue) and ones (white). **Step D:** We approximate using weight sharing – all three convolutions are represented by one convolution. **Step E:** This is equivalent to summing the masks first, before multiplying by the output. Now, FLOP and memory cost are effectively constant w.r.t. the number of channel options [Wan2020]

As with channels, previous DNAS methods would simply instantiate each block with every input resolution. This approach has two main issues, which are addressed by FBnetV2:

- Pixel misalignment: peripheral padding causes pixel misalignment in sum output. To handle pixel misalignment, FBnetV2 uses zero-pad such that zeros are interspersed spatially. For example, a 2x increase in size would involve zero-padding every other row and column.
- Receptive field misalignment: since subsets of the feature map correspond to different resolutions, naively convolving over the full feature map would result in a reduced receptive field. To handle receptive field misalignment, FBnetV2 convolves over subsampled input instead.

With the above insights (see also Figure 4.11), the input resolution search thus incurs constant memory cost, regardless of the number of input resolutions. On the other hand, computational cost increases sub-linearly as the number of resolutions grows.

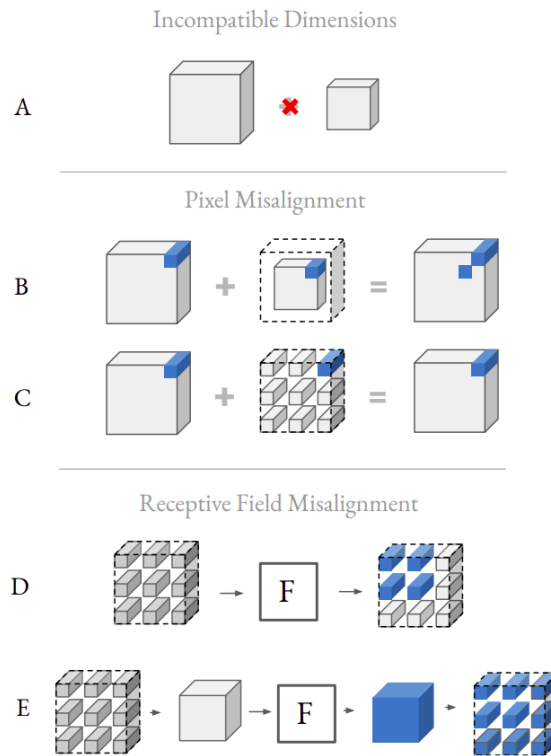


Figure 4.11 - Spatial Search Challenges: **A:** Tensors with different spatial dimensions cannot be summed due to incompatible dimensions. **B:** Zero-padding along the periphery of the smaller feature map makes summing possible. However, the top-right pixels (blue) are not aligned correctly. **C:** Interspersing zero-padding spatially results in a sum that aligns pixels correctly. Note the top-right pixels of both feature maps are correctly overlapping in the sum. **D:** Say F is a convolution with 3×3 kernels. Convolving naively with the feature map, containing a subset (gray), results in reduced receptive field (2×2 , blue) for the subset. **E:** To preserve the receptive field for all searched input resolutions, the input must be subsampled before convolving. Note the receptive field (blue) is still 3×3 . Furthermore, note it is possible to achieve the same effect, without the need to construct a smaller tensor, with appropriately-strided dilated convolutions; subsampling is performed to avoid modifying the operation F [Wan2020]

4.3 Federated Learning and Privacy Preservation

TinyML implementation (see Section 3.2.1) paved the way to the idea of training ML algorithms by exploiting local data directly from the edge devices where it is acquired, leading to an emerging field of AI called Federated Machine Learning (FML or FL) [IEEEGuide2021]. FML defines a ML framework that allows a collective model to be constructed from data distributed across repositories owned by different organizations or devices.

Companies and organizations are collecting increasingly more detailed information about users. This information is exploited by ML techniques to improve products, services, and welfare, and it is a consensus that valuable information can be extracted from raw data belonging to various organizations. However, due to the potential misuse and adversarial attacks, in a distributed ML paradigm as such there can be severe challenges to the protection of data privacy and security. FML represents a technology that aims to build and use ML models by collectively exploiting the data at each data owner's location without compromising user privacy and information security.

Data privacy and information security constitute significant challenges to the big data and AI community as these communities are increasingly under pressure to adhere to regulatory requirements like the European

Union’s General Data Protection Regulation (GDPR). Many routine operations in big data applications, such as merging user data from various sources to build a ML model, are considered to be illegal under current regulatory frameworks. The purpose of FML is to provide a feasible solution that enables ML applications to utilize the data in a distributed manner that does not exchange raw data directly and does not allow any party to infer the private information of other parties.

4.3.1 Reference architecture

FML represents a distributed machine-learning framework that enables multiple participants to collaboratively train and use a ML model for a given task, e.g., classification, prediction and recommendation. A FML framework consists of data, users, and systems as illustrated in Figure 4.12.

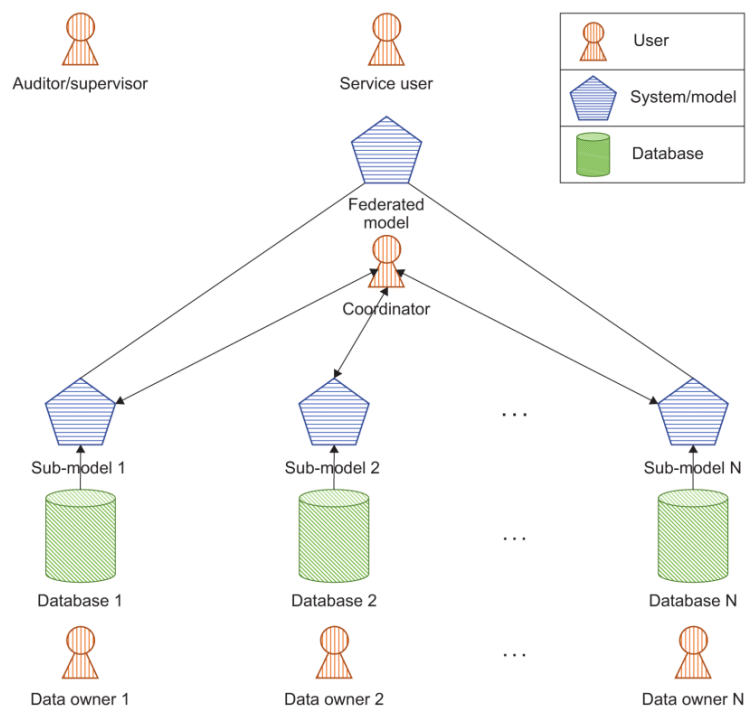


Figure 4.12 - Federated Learning Framework

In the FML framework, data are distributed across various repositories and are utilized to build FML sub-models that are, subsequently, integrated to result in a federated model. All of this in a secure and privacy-preserving manner. An FML user can be a person, a corporation, or other organizations with the legal capacity to participate in the FML framework. In addition, such users may play four roles, which are the data owners, the model users, the coordinators, and the auditors (see Figure 4.13). The FML system consists of multiple functional modules that provide FML services to users.

FML user	Role
Model user	Can establish business relations with federated ML coordinators
Data owner	Contribute their data sources for building an FML system. They provide model parts to FML coordinators or to each other in a privacy-preserving manner and receive rewards as a result of these contributions
Coordinator	Initiate, maintain, and provide FML services for both data owners and model users
Auditors	Responsible for checking the correctness of the FML process and verifying that the process complies with the system constraints in terms of performance and security requirements, as well as regulatory requirements

Figure 4.13 - Roles of Federated ML users

4.3.2 Data view

FML data is often stored in a standard database format, i.e., tables, whereby each row represents a data sample and each column represents a feature or label of the sample. In supervised learning, a complete training dataset consists of both features, denoted by X , and labels, denoted by Y . A set of feature attributes are typically represented as a feature vector (X_1, X_2, \dots, X_n) . Furthermore, a unique sample ID is associated with each data sample. In an FML system, data from multiple datasets may have overlaps in sample IDs and/or feature attributes. Depending on the extent of overlapping along either sample IDs or features, the following three cases are of interest for an FML framework:

- The overlap of feature attributes (X_1, X_2, \dots, X_n) is substantially larger than the overlap of sample IDs (U_1, U_2, \dots, U_m)
- The overlap of sample IDs (U_1, U_2, \dots, U_m) is substantially larger than the overlap of feature attributes (X_1, X_2, \dots, X_n)
- The overlap of sample IDs (U_1, U_2, \dots, U_m) and the overlap of features (X_1, X_2, \dots, X_n) are both small

In the list above, “substantially larger” (and “significant overlap”) is judged by whether the overlapping data can be utilized to build a high-quality ML model, where the measure of quality is determined by applications. Depending on the application scenarios, FML is categorized as Horizontal FML, Vertical FML, and Federated Transfer Learning. Also, depending on whether the datasets used have the same format or different formats, FML is categorized into homogeneous FML or heterogeneous FML.

- *Horizontal Federated Machine Learning:* Horizontal FML (Figure 4.14) refers to building a model in the scenario where datasets have significant overlaps on the feature spaces (X_1, X_2, \dots, X_n) but not on the ID spaces. In this case, an FML model can be built as if the data are split and joined horizontally. Horizontal FML may apply to scenarios where the number of sample IDs from data owners is too small to build a high-quality model. An FML model should perform better than the sub-models built by one single dataset, and the performance of the FML model is very close to that of the model built when all data were put together in one location.

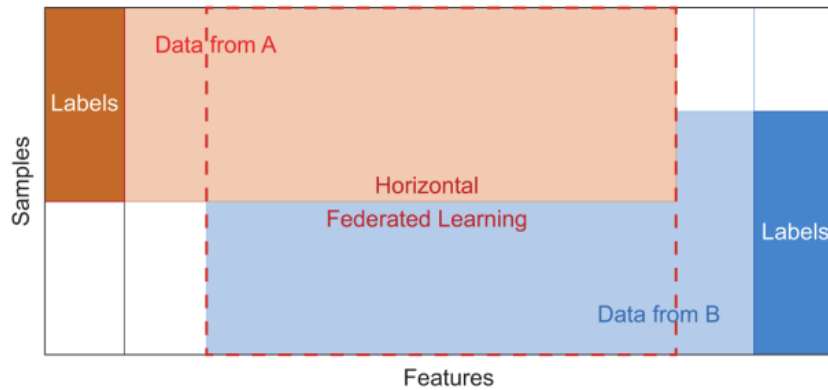


Figure 4.14 - Horizontal Federated Machine Learning [Yang2019]

- Vertical Federated Machine Learning:** Vertical FML, see Figure 4.15, refers to building a model in the scenario where datasets have significant overlaps on the sample space D , but not on the feature spaces (X_1, X_2, \dots, X_n) . In this case, an FML model can be built as if the data is split and joined vertically. Vertical FML may apply to scenarios where there are insufficient features or labels to build a high-quality model. Still, An FML model should perform better than the sub-models built by one single dataset, and the performance of the FML model is very close to that of the model built when all data were put together in one location.

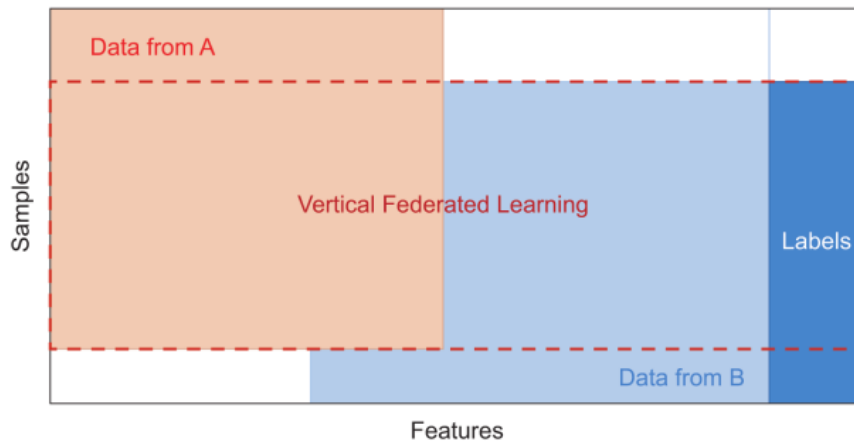


Figure 4.15 - Vertical Federated Machine Learning [Yang2019]

- Federated Transfer Learning:** Federated Transfer Learning (FTL), see Figure 4.16, refers to the FML technique designed for application scenarios where datasets have no significant overlap on neither the sample space nor the feature space. FTL takes advantage of transfer learning techniques to exploit reusable knowledge across different feature domains, and consequently, results in high-quality FTL models despite small data and weak supervision difficulties.

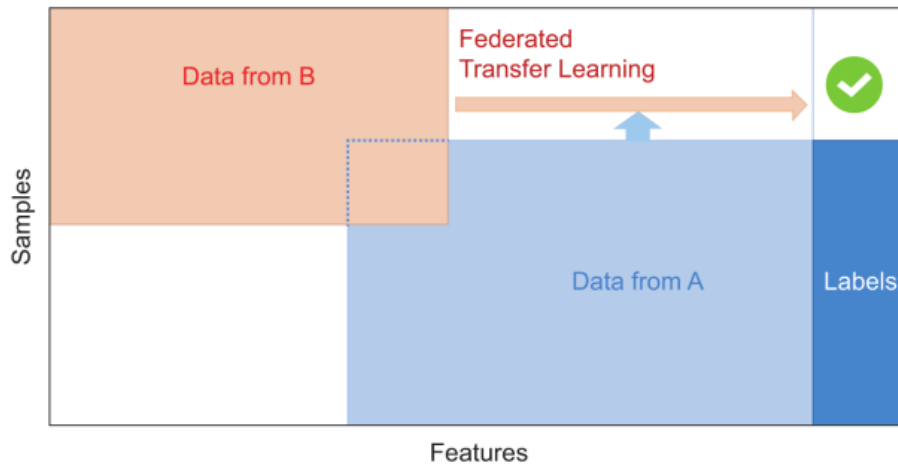


Figure 4.16 - Federated Transfer Learning [Yang2019]

4.3.3 Machine Learning View

Figure 4.17 illustrates a reference hierarchical framework of FML, in which functional modules define elementary FML activities. Modules are grouped into layers according to the relevance of their activities, while modules in different layers can interact with each other through cross-layer functions. Note that depending on varying requirements in different use cases, functional modules may be included in or omitted from a specific FML framework. Note that any FML systems may include one or more modules encapsulated in each layer of the reference framework.

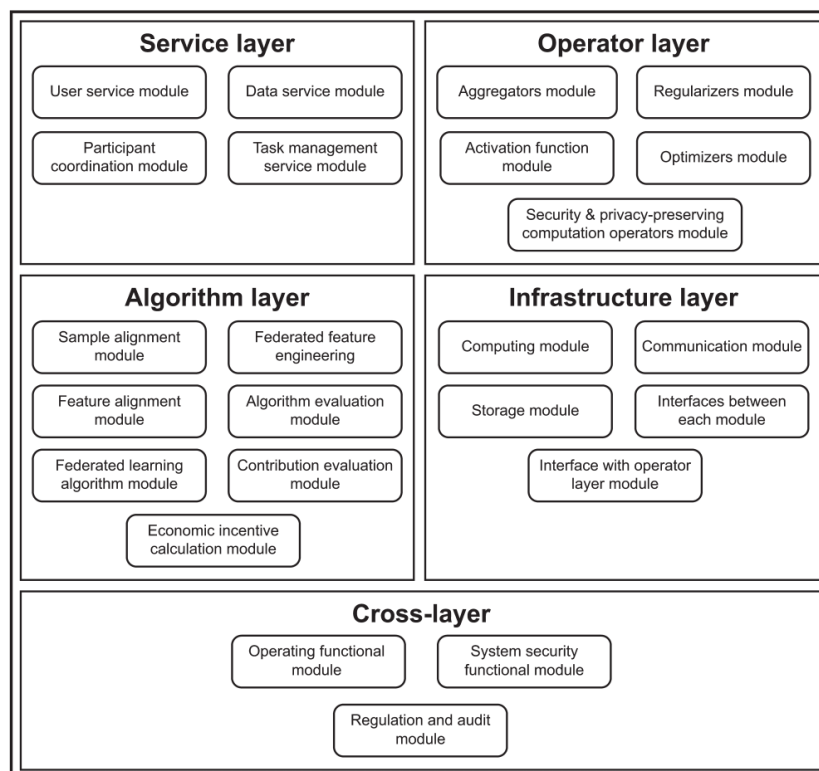


Figure 4.17 - Hierarchical framework of Federated Machine Learning

For the purposes of the project, we do not go into the details of each one of the elements of this provided overview. Conversely, as a last analysis on FML, it is of paramount importance to highlight some concerns about privacy and security, model performance efficiency, and economic viability that FML users should take into account (see Figure 4.18).



Figure 4.18 - Common concerns of Federated Machine Learning

Privacy and Security

Adhering to the privacy and security principles posed by both the ethical and regulatory requirements, this standard aims to provide a guide for the design, development, and implementation of FML frameworks. It should also be used as a baseline in the monitoring and measurement of performance, benchmarking, and auditing aspects of privacy and security management programs of a FML system.

Adhering to the *privacy principle* means avoiding information leakage. The information received from the sub-models does not leak the source data information, and attackers cannot infer the participants' information by eavesdropping gradient during the training or inference stages.

Adhering to the *security principle* means that a FML model should be sufficiently robust to defend against malicious attacks, such as testing time adversarial examples and training time data poisoning. Such a model should be robust to other threats such as backdoor attacks, which aim to induce the model over-fitting to backdoor patterns that trigger misjudgment or misbehavior of FML models.

Adhering to the *transmission security* means that no private information about raw data and model parameters is tampered or disclosed under transferring procedure. Attackers cannot infer private information even if they captured the transferred data.

Efficiency

Adhering to the computational and communication efficiency means:

- An FML algorithm should be analysed to determine its resource usage so that the user of the algorithm can have an early estimate of the possible overhead and be aware of appropriate resources
- The design of the algorithm concerning the amount of data transferred and the computational logic utilized should be optimized according to the use case requirements
- An FML algorithm is deemed efficient if its resource consumption, or computational cost, is at or below some acceptable level

Benchmarks can be used to assist with gauging an algorithm's relative performance.

Economic viability

Beyond data security and privacy preserving consideration, FML may need to economically incentivize users to join and stay in the federation. The economic incentive calculation module determines how to reward contributions made by FML users, e.g., data owners. The objectives of this module include keeping all participants in the federation (individual rationality), leading to a non-negative profit for the coordinator (weak budget balance), maximization of the total utilities of all data owners (Pareto efficiency), incentivizing data owners to provide all their data (incentive compatibility), and equal unit price of effective data (fairness).

Note that it is hard to attain all these objectives simultaneously. Depending on different use cases, FML coordinators should choose one or more objectives to optimize. The attainment of these objectives should be quantified by specific economic measures.

Performance evaluation

A performance evaluation scheme is introduced to confirm the conformance of the performance of FML ecosystems and algorithms. The performance evaluation scheme allows an FML system to be evaluated by using specific metrics, which are widely adopted by academic and industrial research, concerning efficiency, security as well as the performance of FML systems. Notice that measures specified in this clause are not exclusive and more testing specifications can be made for different settings of influential factors (like Dataset, hardware, deployment technique, etc.).

Adhering to the quality principle means verifying that FML systems are well-behaved, up-to-date, adequate, and relevant for the purpose of use. It is, therefore, mandatory to establish evaluation mechanisms and periodically check the performance and quality of FML models through appropriate means. FML models are expected to achieve an equivalent or more competitive performance to that of a data centralized model, which is a model trained on the data collected from all parties. The difference, denoted as FML model performance discrepancy, between FML model performance with respect to that of the data centralized model, should be evaluated. Model performance in varying applications may be measured in terms of accuracy, precision, recall, image quality, or any other measures relevant to the application tasks.

4.3.4 Implementations

Numerous tools are available to facilitate the building and the maintenance of Federated Learning systems on edge devices. In the following, a brief overview of the main ones is provided.

TensorFlow Federated

TensorFlow Federated¹²² (TFF) is the open-source framework based on TensorFlow, which provides the main building blocks for enabling FL. TFF allows developers to deploy ML models and to train on decentralized data across multiple sources. In addition, TFF provides also support for non-learning computations, such as

¹²² <https://www.tensorflow.org/federated>

Federated Analytics¹²³, which re-uses the FL infrastructure to analyse decentralized data with data science methods. TFF interfaces are organized in two layers: Federated Learning API¹²⁴ and Federated Core (FC) API¹²⁵.

FL API provides high-level interfaces to perform federated training, or evaluation, starting from TensorFlow ML models. The interfaces, defined in the *tff.learning* namespace, include three main components:

- **Models:** provide the core functions to build models in TFF. This is done by defining custom models via subclassing *tff.learning.Model*, or simply wrapping an existing Keras model (*tff.learning.from_keras_model*).
- **Federated Computation Builders:** provide the main functions for federated training and evaluation.
- **Datasets:** provide collections of data that can be used in a simulated FL scenario at development stage, together with the functions to access local datasets.

FC API represents the core of TFF, providing low-level interfaces through which developers can implement new functions and interfaces. It supports different types, such as Tensor, sequence, tuple and function types, and provides several programming abstractions, such as TensorFlow computations, federated operators, e.g., *tff.federated_sum* or *tff.federated_broadcast*, lambda expressions, i.e., the TFF equivalent of lambda or def in Python, placement literals, to define simple client-server computations, and functions invocations.

PySyft

PySyft¹²⁶ is an open-source framework which enables FL with the main ML frameworks such as PyTorch, Keras, or TensorFlow. It provides support to implement and integrate privacy-preserving methods including multi-party computation and differential privacy. PySyft allows to perform dynamic computation directly on decentralized data or to create static graph to be deployed at a later date on different compute. The PyGrid¹²⁷ library allows to deploy PySyft at scale, allowing data owners in the network to collectively train AI models and serving as a central server for conducting both model-centric and data-centric FL. At the moment, PySyft provides a set of tutorials, but a detailed documentation is still missing.

Open Federated Learning

Open Federated Learning¹²⁸ (OpenFL) is a Python 3 library designed for conducting federated learning, thus enabling collaborative training among data owners. The framework was developed by Intel Labs and Intel Internet of Things Group and supports models developed with the main ML frameworks, i.e., PyTorch and TensorFlow. A remote data owner is defined in the OpenFL network as a *collaborator* node. A collaborator node hosts its own dataset and its hardware is used to train the models locally. Data remains on collaborator nodes, which share only weights and metrics through the *aggregator* node. Aggregator and collaborators connect through a mutually-authenticated transport layer security (TLS) network connection. The coordination of the federation is provided by the *FL plan*, which is a YAML file describing the collaborator and

¹²³ <https://ai.googleblog.com/2020/05/federated-analytics-collaborative-data.html>

¹²⁴ https://www.tensorflow.org/federated/federated_learning

¹²⁵ https://www.tensorflow.org/federated/federated_core

¹²⁶ <https://github.com/OpenMined/PySyft>

¹²⁷ <https://github.com/OpenMined/PyGrid>

¹²⁸ <https://github.com/intel/openfl>

aggregator settings and other training parameters. In addition, the FL plan specifies the algorithm the aggregator will use to combine the updates from collaborators.

KubeFATE

KubeFATE¹²⁹ manages federated learning workloads using cloud native technologies like Docker Compose and Kubernetes. KubeFATE enables federated learning jobs to run across public, private and hybrid cloud environments. FATE¹³⁰ (Federated AI Technology Enabler) is an open-source project from Webank's AI Department to provide a secure federated computing framework. It implements secure computation protocols based on homomorphic encryption and multi-party computation (MPC). FATE supports several FL algorithms, including horizontal and vertical FL, and federated transfer learning. Computation of ML algorithms, such as logistic regression and tree-based algorithms, is supported, together with DL models. The main components of KubeFATE are:

- **FATE Cloud**¹³¹: an Infrastructure which enables FATE to be managed in multi-cloud, providing support for a secure federated data network across or within organizations.
- **FATE Serving**¹³²: is a high-performance serving system for federated learning models. It offers:
 - High-performance online FL algorithms
 - Dynamic loading federated learning models
 - Can serve multiple models, or multiple versions of the same model
 - Real-time inference
 - Support pre- and post-processing, and data-access adapters for obtaining host feature data

4.4 AI Applications Performance Modelling

DL popularity is steadily increasing thanks to its impact on many application domains (ranging from image and voice recognition to text processing) and DL application performance prediction has received a lot of interest from many academic and industrial groups. Since usually both training and inference of NNs are run on GPUs, it is important to understand how such hardware can influence performance, but unfortunately only a few studies are available in the literature.

Jia et al. [Jia2012] propose the Stargazer framework to build performance models for a simulator running on GPU, so as to correlate several GPU parameters to the simulator execution time. Given the daunting size of the design space (which considers very low-level parameters like the number of thread blocks concurrently running on one core or how many intra- and inter-warp memory requests may be coalesced into one memory access), authors exploit sparse random sampling and iterative model selection, thus creating step by step an accurate linear regression model. Another approach to the issue is proposed in [Liu2007], where the authors elaborate a detailed analytical model of general-purpose applications on GPUs. The proposed model consists of three general expressions to estimate the time taken for common operations, according to their dependence on data size or computational capabilities. Similar analytical modelling approaches, see, e.g., [Song2013], rely on micro-architecture information to predict GPU performance. As GPU architectures

¹²⁹ <https://github.com/FederatedAI/KubeFATE>

¹³⁰ <https://github.com/FederatedAI/FATE>

¹³¹ <https://github.com/FederatedAI/FATE-Cloud>

¹³² <https://github.com/FederatedAI/FATE-Serving>

continue to evolve, the main issue of analytical models is that a minor change in the architecture may require extensive work to adapt the model to hardware enhancements [Dao2015].

Given the complexity of GPU hardware (due to the large number of processors, the thread context switching mechanism, and the on-board hierarchical memory subsystem), recently black box approaches based on ML are favoured over analytical models. Indeed, black box approaches can derive performance models from data to make predictions without a priori knowledge about the internals of the target system. On the other hand, ML models [Lu2017], [Gupta2018], [Peng2018], [Dube2019] require performing an initial profiling campaign to gather training data to learn the mapping among the features of an application and its execution time. An overview and a quantitative comparison among analytical and ML-based model proposals is reported in Madougou et al. [Madougou2016].

The contribution closest to the approach we will follow in AI-SPRINT can be found in Dao et al. [Dao2015]. The work considers the OpenCL benchmark suite and obtains ϵ -SVR models able to predict each OpenCL kernel performance with an average 5–10% error. However, the approach is rather low-level since it requires to instrument each kernel's code and to identify, through an ad-hoc profiling activity, low-level features such as the maximum number of active work groups, the number of registers used per work item, the number of branches and divergent branches per work item. Moreover, for some specific kernels the error is up to 70%. Kerr et al. [Kerr2010] profile and build models for a range of applications, run either on CPUs or GPUs, exploiting the Ocelot framework [Diamos2010]. Relying on 37 performance metrics, they exploit principal component analysis and linear regression in order to highlight those features that are more likely to affect performance on heterogeneous processors. The approach is rather general since the Ocelot infrastructure allows for simulating GPUs, gathering performance metrics, and instrumenting kernels. The work models CUDA kernels' performance using only metrics that are available at compilation time. Unfortunately, the Ocelot infrastructure is not actively developed anymore, limiting its support to old GPU generations. Along the same lines, Gupta et al. [Gupta2018] propose an online modelling approach to predict the single frame processing time starting from GPU frequency and performance counters for mobile devices. In the DL applications area, Hadjis et al. [Hadjis2016] present solutions to minimize the total training time of CNNs, given the network architecture of the DL model, the data set used for training, the set of available computational resources and their throughput, and the network speed. The framework is able to identify the optimal split of the DL model across multiple compute groups and the optimal number of servers per group. They also propose a solution to improve the stochastic gradient descent momentum update to compensate for the staleness introduced by the adoption of multiple compute groups. However, their system cannot predict a priori the execution time of a given number of iterations nor the training time in experiments with different batch size, which will be one of AI-SPRINT goals. A higher level approach, which considers matrix multiplication computation time as the core feature, is presented in Lu et al. [Lu2017], where the authors focus on the deployment of CNNs on mobile devices. According to this consideration, they focus only on the forward pass, whose overall execution time is predicted starting from an estimate of the time taken by matrix multiplications when the CNNs are deployed on a CPU or a GPU. This approach entails extracting tensor sizes from network specifications and associating them to their expected response times, according to platform-specific benchmarks. In the end, the approach is more accurate when CPUs are considered (the percentage error they obtained is in the range 6–15%, while they obtain 16–21% for the GPU case). Similarly, in [Gianniti2018] which will be our starting point in AI-SPRINT, we derived linear regression models for estimating CNN training times by relying on the layers computational complexity in terms of simple primitives available on GPUs. This enables performance prediction even on networks never considered during the learning phase. However, our initial study is limited to CNNs trained on a single GPU with the Caffe framework. In a more recent work [Gianniti2019], we compared our per layer model in Gianniti et al. [Gianniti2018] with a pure black box ML end-to-end model, but the study was still limited to a single DL framework and could not generalize prediction to different GPU hardware. Finally, the work in Dube et al. [Dube2019] proposed AI

Gauge, a framework based on ML where models are continuously calibrated by processing job traces. The proposed models achieve less than 10% relative error on average, but, they are limited to single GPU deployments.

More recently, authors in [Mendoza2021] developed ML models to predict the performance slowdown due to inference DL models colocation on the same hardware. Predicting the slowdown in inference serving system is extremely challenging since inference tasks have few milliseconds duration and determining what models can be safely co-located together on a hardware platform avoiding QoS violations would require a combinatorial growth of the profiling data (the system should assess how each DL model inference latency is affected by being co-located with one or more models across different machine types). Authors proposed a unified predictor based on random forests able to leverage similarity across machine types for accurate prediction which considers as main features the DL models metrics (e.g., CPU and GPU utilization, GPU Global buffer usage, PCIe read and write bandwidth) and machine encodings. In edge systems [Disabato2015], [Kang2017] propose some linear regression models to predict the execution time of DNN or their partitions in edge devices or mobile phones, respectively, by considering the number floating point operation per seconds (FLOPS) or simply the number of multiplications.

The performance research community has also put a lot of effort in defining specific benchmarks for AI applications deployed in computing continua. In particular, [Tianshu2019] proposed an EdgeAI bench which includes intensive care unit patient monitor, surveillance camera, smart home, autonomous vehicles, and a federated learning framework as reference applications with the purpose of measuring and optimizing computing continua architectures. Similarly, EEMBC¹³³ developed MLMark, a proprietary ML benchmark suite for embedded edge computing platforms including image classification, object detection, language translation, and speech recognition tasks. EdgeBench [Das2018] has a specific focus on Cloud providers edge solution and compares AWS Greengrass and Microsoft Azure IoT Edge while AI Benchmark [Ignatov2018] is a benchmark suite specific for smartphones.

Broadening the scope of our analysis and considering general applications running in computing continua, traditional models based on Queueing Networks have demonstrated to be reasonably accurate for estimating the performance of edge systems. In particular, [Bures2018] compared the performance of a QN model with a simulator of a Cyber-Physical Systems for parking slots monitoring achieving a percentage error between 6% and 31%, while [Tadakamalla2021] adopted M/G/1 queueing networks for studying the performance of an application counting the number of bicyclists in Seattle, monitoring the taxi trips in Chicago, and monitoring GPS trajectories of Beijing taxis achieving an error below 11%.

Finally, for what concerns microservices and FaaS system performance modelling, machine learning is becoming popular also in this area. In particular, [Grohmann2021] (extending the initial work in [Grohmann2019]) proposes SuanMing an integrated framework for learning regressors (based on random forest, k -Nearest Neighbor Regression, Ridge Regression, and Support Vector Regression) of microservice based systems running in public and private clouds. The final goal is to identify the root cause of performance degradation of complex application and the learned regressors are used to predict the violation of predefined thresholds. Similarly, [Gan2019] adopts deep learning (mixing fully connected layers, CNNs, and LSTMs) to predict QoS violations for microservices based applications by relying on distributed RPC-level tracing data and detailed low-level hardware monitoring.

As an example of performance modelling of FaaS systems, [Mahmoudi2020a] proposes a temporal performance model to predict some performance metrics by considering the application average response

¹³³ <https://www.eembc.org/>

time for warm and cold requests, the requests arrival rate, and the system expiration threshold. The same authors also proposed a steady-state performance model in [Mahmoudi2020b] for improving the QoS and utilization of FaaS and reducing their operating costs. The proposed model provides a trade-off between the cost and performance by making the platform workload-aware. Authors leverage a continuous time Semi-Markov Process (SMP), where each state represents a warm instance, and it is modelled with an M/G/m/m queuing system where m is the number of concurrent function instances. In this way, authors compute some parameters like cold start rate, arrival rate and server expiration rate. After computing the parameters for each function instance in the warm pool, they solve the SMP to calculate the steady-state characteristics like the probability of rejection, average response time, mean utilization, etc. They validate the proposed model on AWS Lambda. Vice versa, [Gias2020] proposes a performance modelling technique based on layered queueing networks (LQN) for the FaaS platform and developed the COCOA approach, a sizing method for private FaaS platforms which rely on their LQN model to solve the cold start issue. Each function is modelled as a Continuous Time Markov Chain (CTMC) and the cold start probability is obtained by solving the CTMC. In this way an LQN model can predict the response time of each function.

4.5 Edge Applications Components Placement and Design Exploration

Component placement and design exploration in computing continua have recently received a lot of attention from the research community. A recent survey is provided in [Bellendorf2020] where authors propose a classification of the literature proposals according to the layers involved (cloud/fog-edge nodes/end devices), the purpose of the placement (e.g., end devices offload, fog nodes offload, fog nodes cooperation, data distribution among fog nodes or cloud, etc.), the decisions taken (e.g., tasks priority, resource to task assignment, hardware resources placement, etc.), the relevant metrics (e.g., latency, energy, profit-cost, and device-specific) and the general goal (e.g., energy minimization, latency-throughput trade-off, privacy and security of data).

Many works focus on Mobile Edge Clouds (MECs) with a particular emphasis on the network resources. For example, the work in [Xu2016] formulates a cloudlet placement problem in a Wireless Metropolitan Area Network (WMAN) as an Integer Linear Programming (ILP) which locates some cloudlets to some strategic locations in the WMAN minimizing the average delay between mobile users and the cloudlets. The authors propose a greedy heuristic algorithm to solve the ILP and provide also two approximation algorithms with guaranteed approximation ratios for two special cases where all cloudlets have identical or different resource demands. In MECs, user mobility is also particularly relevant. Authors in [Balevi2018] consider a square planar where the cloud is located at its centre and n end-users device nodes are randomly and uniformly distributed around the cloud. The goal is to determine the number and the placement of fog nodes with additional computing capabilities so as to maximize the average data rate and minimize the transmission delay of the system. Authors consider a variable $0 \leq p \leq 1$ and choose the number of fog nodes based on it: $n_0 = n(1-p)$, $n_1 = np$ where n_0 is the number of end devices and n_1 is the number of fog nodes. At first, all nodes are ordinary, and the problem is how to find a unique optimum global value of p so as to minimize the transmission delay which is a concave function based on signal-to-interference-plus-noise-ratio and distance factors. Since the transmission delay is related to path loss exponents α , the authors solved the optimization problem for $\alpha=1$, $\alpha=2$ and $\alpha=4$ to obtain the optimal p for each case.

Similarly, [Santoyo2018] models the service infrastructure placement problem in a fog computing environment as a latency and capacity-constrained location selection problem. The reference use case is a 5G scenario where the latency is the main metric to be addressed to support mission-critical and delay-sensitive

environments. To achieve this goal, service infrastructure placement optimization is needed in the interest of minimizing the delays in the service access layer. The placement problem in a Fog Computing/NFV environment is modelled as a Mixed-Integer Linear Programming (MILP) problem. The authors, considering 5G mobile network requirements, proposed a heuristic solution based on Simulated Annealing (SA) algorithm mixed with a Tabu Search (TS) to take the advantage of the flexibility of SA and combining it with the memory structures used in TS.

[Bahreini2017] and [Wang2017] are among the proposals closest to the problems we will face in AI-SPRINT. Authors in [Bahreini2017] formulate an offline version of a multi-component application placement problem (MCAPP) as a MILP which is solved by the CPLEX solver. The authors also propose an online algorithm and consider the solution of the offline problem as a lower bound to estimate the performance of the online version. The online solution is based on simple heuristic techniques such as iterative matching and local search. In particular, the online version first neglects component communication cost, and the best matching between components and the edge servers is determined through the Hungarian algorithm. In a second step, the communication cost among the components is considered and a local search algorithm is applied to improve the solution.

On the other hand, some works like [Wang2017] investigates the placement of multi-component applications only in the edge. Authors model the application components as an application graph and the physical devices on the edge as a physical graph and propose both online and offline algorithms to optimally map the application to the physical graph while providing performance guarantees for the end applications. In particular, an optimal offline algorithm maps a single application linear graph (i.e., including only sequential components). Another solution is provided for single or multiple applications with branches, modelled as tree application graphs, however, under the assumption that the placements of all branch nodes are given. Finally, another variant where the placement of only some branch nodes is given is also proposed.

Component placement problems are considered also in FaaS systems. [Das2020b] proposes a scheduling framework for multi-function serverless applications over a hybrid public-private cloud. The authors assume that there is a batch of jobs with the same deadline and the objective is to minimize the cost of public cloud by scheduling the function executions over hybrid cloud. They proposed a greedy algorithm for their NP-hard problem by considering job priorities. If the priority is the job cost, the scheduler orders the jobs based on their execution cost and offloads the less expensive jobs to the public cloud first. If the priority is the processing time, the scheduler orders the jobs based on their processing time and offloads the jobs with longer processing time to the public cloud. On the other hand, some works consider both budget limitation and QoS in their systems. One example is proposed in [Lin2021], in which authors present an analytical model and optimize the performance and cost to reach a trade-off between limited budget and the desired QoS of serverless applications. They introduce a formal definition of the serverless workflow by considering four types of structures for the workflow composition (i.e., sequence, parallelism, branch, and loop). Then, they propose different analytical models to predict the average response time and cost of complex workflows. Furthermore, they present a heuristic approach to optimize performance and cost of serverless applications with four greedy algorithms and they evaluate their proposed approach on AWS Lambda and Step Functions.

Finally, in [Elgamal2018] the goal is to optimize the cost of serverless applications in AWS Lambda while keeping the latency under a certain threshold. The authors fuse multiple functions together in order not to incur in the transition cost from one function to another. They propose a price model for AWS Lambda and an execution time model to estimate the response time of a function. Then, they define a cost graph in which each node represents a fused function and the weight of the edge between two nodes i and j includes two costs: 1) The price of fused function i and the transition cost from i to j , 2) Delay cost which is the execution time of node and the transmission time of output data from i to j . Each path in this graph represents a solution for function placement and fusion. Finally, they formulate the problem as a shortest path problem and

propose an algorithm (Dijkstra's shortest path) to find the optimal application's price solution with the threshold latency and vice versa.

4.6 Gaps filled by AI-SPRINT

AI-SPRINT will develop solutions to enable developers with limited ML expertise to train high-quality deep network models specific to their needs, also in terms of QoS requirements. Training DNNs and designing the right network topology is still a matter of art craft and designer experience and it becomes very challenging to jointly design the network architecture and to optimize its deployment, possibly on multiple layers of a continuum infrastructure. To this aim, the PyCOMPSs programming framework and the dislib library will be adopted to provide ready to use distributed training mechanisms.

In AI-SPRINT, network architecture search, deep learning models partitioning, and components placement will be jointly addressed. AI-SPRINT will provide tools for developing learning as a service solutions, that, starting from a training set with labelled training examples (images or temporal data series which are of interest for use cases) will automatically identify the most accurate DNN which provides execution time guarantees. Moreover, AI-SPRINT will automatically identify the optimal partitioning of DNNs and their minimum cost deployment fulfilling a priori fixed deadlines for the execution time of single or multiple components considering also multiple devices candidate for the components execution and/or network settings.

AI-SPRINT will leverage its design time and run-time tools to implement federated learning algorithms in a transparent manner for the user allowing for secure and privacy preserving training of deep models. As part of the contribution to federated learning, AI-SPRINT will design methods for the seamless update of models either via weights updates or via model redeployment. Network architecture search, deep learning models partitioning, and components placement will leverage performance models for estimating DNNs, or their partitions, execution time. The performance modelling approach will be mainly based on ML. ML-based performance models will be developed to predict the execution time of AI applications running on heterogeneous multi-devices systems composed of IoT & AI enabled sensors, edge, and cloud resources. The tools will provide support and automate AI applications performance profiling. Solutions to identify the ML model providing the highest performance prediction accuracy supporting model selection and hyperparameters tuning will be also developed.

5. Runtime Frameworks for Computing Continua management

Chapter 5 analyses the enabling technologies for the runtime management of computing continua. In particular, Section 5.1 provides an overview on virtualization, with a particular focus on container orchestration technologies. Section 5.2 is focused on deployment technologies and cloud orchestration while Section 5.3 addresses the topic of resources monitoring. The runtime management is presented in Section 5.4 while scheduling solutions for GPU-accelerated clusters are described in Section 5.5. Finally, Section 5.6 outlines the strategies and solutions that AI-SPRINT will propose to fill the gaps left in the proposed literature.

5.1 Virtualization, Containers and Containers Orchestrators

Virtualization techniques allow to run isolated operating systems on top of other operating systems (OS), or directly on the underlying hardware, by means of a hypervisor. There exist several hypervisor technologies by the most popular open-source alternatives are KVM and XEN. Its ability to provide isolation and fine-tuned performance has fostered the adoption of this technology to consolidate workloads and increase the utilization of hardware resources. This enabled the surge of Cloud Computing on which unused computer resources are offered on a pay-as-you-go basis, with prominent players such as Amazon Web Services (AWS), Microsoft Azure and Google Cloud Platform as discussed in Section 3.4. Indeed, new hypervisors are now available as a result of this trend. This is the case of AWS Nitro [AWSNitro], a lightweight hypervisor that provides performance indistinguishable from bare metal. Also, in the quest for combining lightweight virtualization but strong isolation, new developments have surged, such as Firecracker [FireCracker], an open-source virtualization technology that is purpose-built for creating and managing secure, multi-tenant container and function-based services. This is the underlying technology employed in the commercial FaaS (Functions as a Service) offering from AWS Lambda.

The trend towards lightweight virtualization paved the way for container technology to considerably evolve, exemplified by the advances in LXC and Docker. Linux containers are able to run multiple isolated processes in one host without the overhead caused by the hypervisor layer introduced by Virtual Machines in CPU, memory and storage. In particular, Docker turned containers into a mainstream technology, contributing: i) a global shared repository of Docker containers, i.e., Docker Hub; ii) a procedure to create Docker images out of Dockerfiles, and iii) the usage of a layered file system that reduces the footprint of Docker images. The ecosystem of tools around Docker has exploded in the last years, with contributions in many areas such as Continuous Integration/Continuous Delivery (CI/CD), application packaging and Container Orchestration Tools.

Indeed, there are many applications to manage the execution of containers across multiple hosts (e.g., Docker Swarm, Nomad, Kubernetes). Docker Swarm represents the native clustering approach proposed by Docker, which controls a cluster of Docker hosts and exposes it as a single "virtual" host. This way, a container-based virtual cluster can be easily created on top of virtual or physical resources. However, the most widely used Container Orchestration Platform (COP) is Kubernetes, an open-source system for automating the deployment, scaling and management of containerized applications. The Kubernetes ecosystem has blossomed in the last few years, and it is turning into the de-facto standard for running fleets of containers at

scale. Indeed, the report “State of Container and Kubernetes Security”¹³⁴ from StackRox revealed that 91% of respondents had adopted Kubernetes in favour of other alternatives such as Docker Swarm or Amazon Elastic Container Service.

A good alternative to Docker, specifically designed for embedded and IoT devices is BalenaOS¹³⁵. It puts emphasis on reliability over long periods of operation. By using Balena.io platform, a fleet of IoT devices with BalenaOS can be managed remotely. BalenaOS is an embedded operating system and shares architectural principles with many existing systems, but also aims to unify the approach to containers with Docker. BalenaOS has the added benefit of focusing on portability, with 20 device types already supported, and production-readiness, with thousands of devices already deployed for business purposes. For resource constrained devices, there also exist alternatives to Kubernetes like the minified Kubernetes k3s¹³⁶.

5.2 Deployment Technologies and Solutions

The scientific exploitation of cloud resources is nowadays a reality. However, despite this large adoption, cloud computing still presents several functionality gaps that make it difficult to deliver its full potential, especially for scientific usage. Furthermore, in the last years, with the rise of edge computing, cloud orchestration is being considered more and more important, as it will play the role needed to perform the abstractions needed to deploy complex service architectures for a wide range of application domains.

Cloud orchestration involves the automated arrangement, coordination, and management of cloud resources (i.e., compute, storage, and network) to meet user’s needs and requirements [Caballer2018]. It addresses the lack of elasticity and transparent interoperability and portability across different cloud technologies and infrastructures. It provides the users with seamless dynamic elasticity over a large pool of computing resources across multiple cloud and edge providers.

Several open-source orchestration tools and services exist in the market, but most of them come with the limitation of only supporting their own Cloud Management Platforms (CMPs) as they are developed within those project ecosystems. As an example, we can cite some of them: OpenStack Heat [HEAT] and its YAML-based Domain Specific Language (DSL) called Heat Orchestration Template (HOT), native to OpenStack. OpenNebula also provides its own JSON-based multi-tier cloud application orchestration called OneFlow [ONEFLOW]. Eucalyptus [EUCALYPTUS] supports orchestration via its implementation of the AWS CloudFormation web service. All of them are focused on their own CMPs and furthermore they rely on their own DSL languages (open or proprietary ones such as CloudFormation).

However, the usage and promotion of open standards such as the Topology Open Specification for Cloud Applications (TOSCA) [TOSCA]) in the cloud is a way to obtain more interoperable, distributed, and open infrastructures. As a matter of fact, the European Commission recommended, back in 2004, the usage of Open Standards in its “European Interoperability Framework for pan-European eGovernment Services”. So, CMP-agnostic tools tend to move away from specific DSLs in order to adopt open standards such as TOSCA.

So, moving from the CMP specific tools and focusing on other open-source orchestration stacks that support Cloud standards we can find: Cloudify [CLOUDIFY], which provides TOSCA-based orchestration across different Clouds. Alien4Cloud [A4C] enables the composition of TOSCA templates and their deployment by means of a

¹³⁴ Kubernetes and Container Security and Adoption Trend. <https://www.stackrox.com/kubernetes-adoption-security-and-market-share-for-containers/>

¹³⁵ <https://www.balena.io/os/>

¹³⁶ <https://k3s.io>

TOSCA runtime engine (Orchestrator). Alien4Cloud provides plugins to connect to external TOSCA runtime engines. In particular, it can delegate on Cloudify, but it also provides its own TOSCA runtime, Puccini in the initial versions, Yorc in more recent ones. Ystia Orchestrator (Yorc) [YORC] is a hybrid cloud/HPC TOSCA orchestrator that aims to support the whole application lifecycle, from deployment, scaling, monitoring, self-healing, self-scaling to application upgrade, over hybrid infrastructures (IaaS, HPC schedulers, CaaS). TORCH [Tomarchio2021] is a very recent project, not mature enough and with a reduced number of cloud providers and topologies supported. Project CELAR used an old TOSCA XML version using SlipStream as the orchestration layer (this project has no activity in the last years and SlipStream is now deprecated by Nuvla) with the limitation of being opencore, thus not supporting commercial providers in the open-source version. The OpenTOSCA Container [OPENTOSCA] is the TOSCA Runtime Environment to deploy and manage Cloud applications. It enables the automated provisioning of applications that are modeled using TOSCA and packaged as CSARs. It still uses the old TOSCA XML version and also currently only supports OpenStack and EC2 providers. MiCADOscale [MICADO] is an application-level multi-cloud orchestration and auto-scaling framework centered on Kubernetes clusters. It uses Occopus [OCCOPUS] and Terraform as the orchestration tools. The SeaClouds platform is a multi-cloud orchestration based on a modified version of Apache Brooklyn [BROOKLYN] with TOSCA support.

In contrast, the Infrastructure Manager (IM) [Caballer2015] which will be AI-SPRINT baseline, supports TOSCA-based deployments over a wide variety of cloud backends including OpenNebula, CloudStack and OpenStack; commercial cloud providers such as Microsoft Azure, Amazon Web Services (AWS), Google Cloud Platform (GCP), Open Telekom Cloud (OTC), Linode, Orange Flexible Engine and the EGI Federated Cloud, the large-scale pan-european federated IaaS Cloud to support scientific research.

5.3 Monitoring

Monitoring of resources in modern cloud deployments is a challenging task. The situation is even more complex when the edge devices are taken into consideration due to a bigger number of devices, bandwidth throughput limitation and often devices limitation. There are several aspects that may be analysed but the two most important are: (i) logs monitoring (e.g., Elasticsearch¹³⁷ Logstash¹³⁸ & Kibana¹³⁹, and (ii) metrics calculation and gathering (e.g., Telegraf¹⁴⁰, InfluxDB¹⁴¹, Grafana¹⁴²). Additionally, there are tools focused on the analysis of machine generated data (e.g., Splunk¹⁴³). A second classification can consider the deployment model since there exist SaaS solutions (like DataDog¹⁴⁴) or on premise/along private cloud deployments (Zabbix¹⁴⁵, Nagios¹⁴⁶). Next, tools can be divided into: open source ones and closed source/commercial tools. Finally, there is a set of tools connected with certain implementations of cloud, like Azure Monitoring¹⁴⁷ or Amazon CloudWatch¹⁴⁸.

¹³⁷ <https://www.elastic.co/>

¹³⁸ <https://www.elastic.co/logstash>

¹³⁹ <https://www.elastic.co/kibana>

¹⁴⁰ <https://www.influxdata.com/time-series-platform/telegraf/>

¹⁴¹ <https://www.influxdata.com/products/influxdb/>

¹⁴² <https://grafana.com/>

¹⁴³ <https://www.splunk.com/>

¹⁴⁴ <https://www.datadoghq.com/>

¹⁴⁵ <https://www.zabbix.com/>

¹⁴⁶ <https://www.nagios.org/>

¹⁴⁷ <https://azure.microsoft.com/en-us/services/monitor/>

¹⁴⁸ <https://aws.amazon.com/cloudwatch/>

As the promotion of open-source tools and standards is an important principle, further analysis will be focused on the open source and cloud agnostic tools. Zabbix and Nagios are general purpose and versatile monitoring tools focused on metrics gathering and calculation. They have many predefined templates and as mature tools their integration is quick and painless. On the other hand, they were designed to work well with physical or virtual machines while working in edge and container-based deployments is rather challenging. Additionally, these tools are rather centralized and do not scale well.

The next step of development in modern monitoring solutions was an ELK stack: ElasticSearch, LogStash, Beats and Kibana. ElasticSearch is a database that allows for rapid data searching and analysis and is used for storing data. An additional advantage is that ElasticSearch can be scaled easily. Unfortunately, ElasticSearch is not prepared well for time series data storing and analysis, which is the case for many calculated metrics. The Beats is a set of small data gathering and sending tools specialized to send on data type. They can be easily installed on edge devices. On the other hand, LogStash allows for custom data transformation and routing. Kibana is a data and metrics visualization tool that provides dashboards and allows for visual analysis. Finally, this set of tools integrates well with container based and cloud deployments.

Other important tools are the ones provided by AWS and Azure cloud. They are very comprehensive end to end solutions supporting both edge data collection, time series analysis, and seamless integration with containers and serverless tools. Unfortunately, their usage is limited only to a certain cloud provider which causes vendor lock in and removes them from further analysis.

The most suitable solution for time series analysis are newest solutions like stacks build time series databases. The most comprehensive solution is the stack built on the InfluxDB. InfluxDB is a modern database specialized in computing, analysing and storing time series. It has its own domain specific language designed to support such operations - Flux DSL. One of the important aspects of the Flux language is built-in moving average calculation which is one of key aspects of resource monitoring. Another aspect is support for anomalies detection, e.g., Median Absolute Deviation algorithm. Another part of the InfluxDB stack is Telegraf. This is a plugin-driven server agent that is responsible for collecting and sending metrics and events from systems, edge devices, IoT sensors, and more. This is a lightweight tool that may be deployed as a Kubernetes Sidecar and can be easily integrated via plugins with multiple data sources. Metrics and data can be visualized on dashboards using Grafana which is a dashboarding solution having views focused on the time series data presentation. Finally, all mentioned tools have an open-source version that may be customized to specific project needs (automatic deployments, configuration injection, etc).

Figure 5.1 illustrates the architecture of the monitoring solution based on the InfluxDB stack.

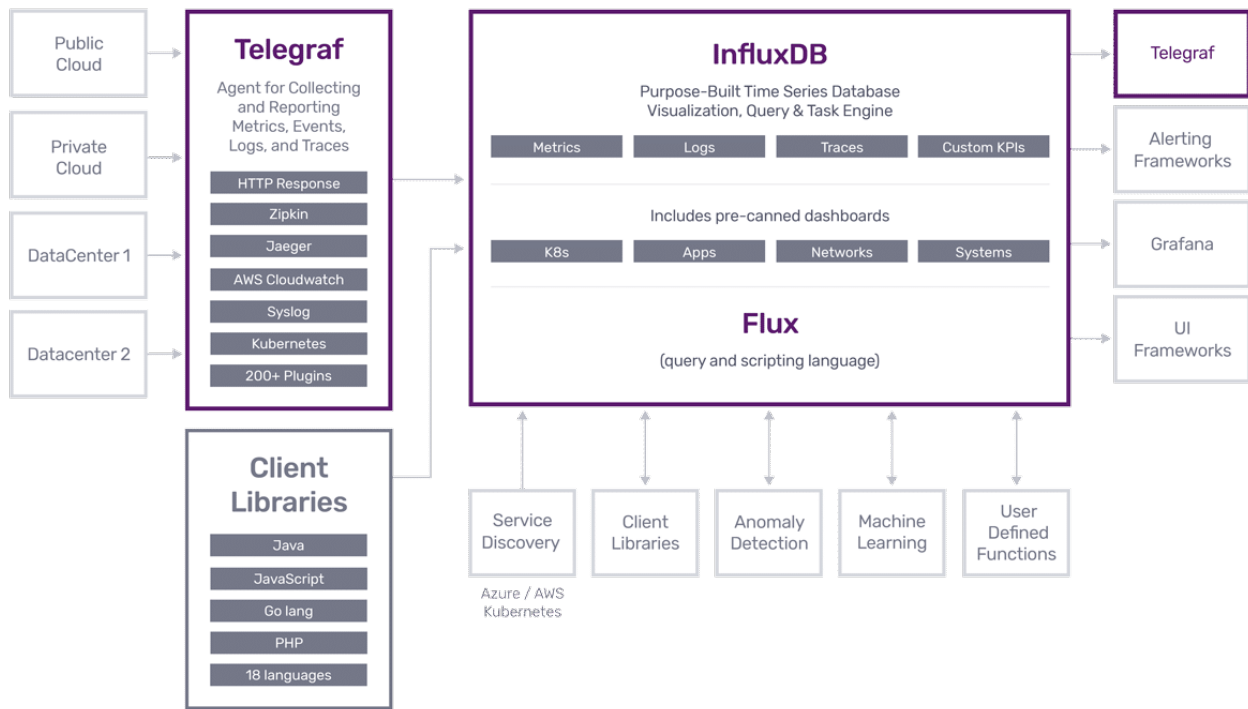


Figure 5.1 - InfluxDB architecture¹⁴⁹

The first step of monitoring is data collection. This is the responsibility of Telegraf which is a plugin-driven server agent for collecting and sending metrics. Telegraf provides many plugins that allow for seamless integration with many servers and applications. Moreover, Telegraf is written entirely in Go, and it compiles into a single binary with no external dependencies and a minimal memory footprint.

Telegraf metrics are the representation used to model data during processing. Telegraf metrics are closely related to the InfluxDB data model and contain four main components:

1. Measurement name: Description and namespace for the metric.
2. Tags: Key/Value string pairs and usually used to identify the metric.
3. Fields: Key/Value pairs that are typed and usually contain the metric data.
4. Timestamp: Date and time associated with the fields.

Telegraf behaviour is shown in Figure 5.2. As we can see, there are two steps in the monitoring process: the first is performed during data collection (transform, decorate, filter) while the second one is done during aggregation (to limit the amount of data pushed to InfluxDB).

¹⁴⁹ Figure source: <https://www.influxdata.com/wp-content/uploads/APM-Diagram-1.png>

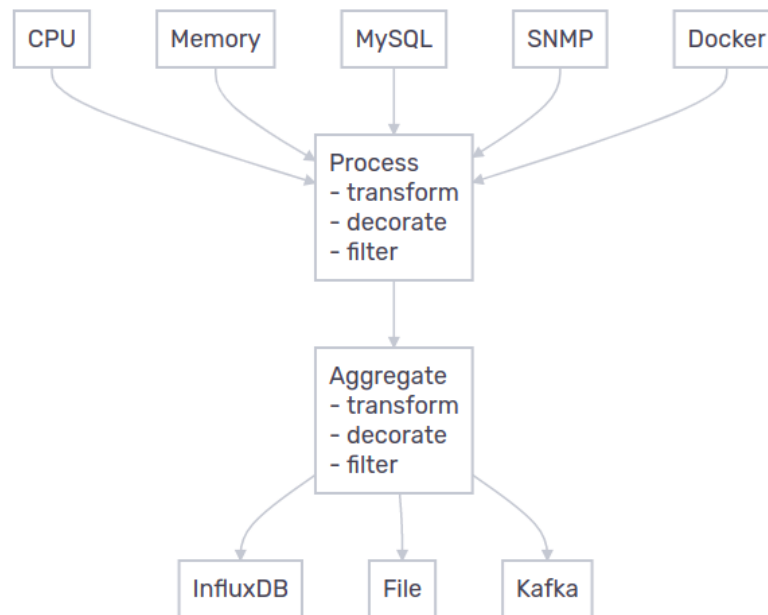


Figure 5.2 - Telegraph Processing¹⁵⁰

Finally, monitoring data is stored into the InfluxDB time series database. Internally, InfluxDB has an organization that optimizes time-related queries. Data is organized in buckets which may be perceived as tables that resemble a column-like data organization style. Every measurement is called a point and has a timestamp and measurement name. Additionally, there are two key-value field types of metadata connected with a point: tags that are used for indexing and fields that are used for storing values. Points that share a measurement, tag set, and field key are organized into series. In other words, we can imagine that every measurement, tag set, and field key have they own key-values table associated with them. The framework is completed with a DSL declarative language, i.e., Flux which is built and optimized to query time related data. Additionally, there are tasks that help data processing and calculations. InfluxDB tasks are scheduled Flux scripts that modify or analyse data streams which are then stored in new buckets or are used to trigger other actions.

The last data visualization step is supported by dashboards which can be implemented by multiple tools, e.g., Grafana, which provides multiple functionalities like zoom in/out, data marking, charting etc.

5.4 Runtime Management

Managing the runtime of the services deployed on distributed infrastructures becomes even more challenging when considering computing continua: mobility inherent to smart mobile devices adds dynamicity to the pool of available resources while variations of the number of users connected to the system lead to changes in the generated data volumes and workload fluctuations which might lead to edge resources saturation.

As of today, all commercial solutions offered by the major cloud vendors simplify the technical challenge by making the edge totally reliant on the Cloud: Amazon IoT Greengrass [AWSGreengrass], Microsoft Azure IOT Edge [MSIoTEdge] and Google Cloud IOT [GOOGLEIoTCloud]) maintain the cloud as a necessary part and only allow the deployment and execution of some functions on edge devices. This problem reaches out also to

¹⁵⁰ Figure source: https://docs.influxdata.com/telegraf/v1.18/concepts/aggregator_processor_plugins/

academic solutions like the Osmosis framework [Villari2019] which also follows this bottom-up approach. The developer defines microElements (MELs) and describes how these MELs relate to each other. From the cloud, Osmosis orchestrates MEL deployments and migrations considering resource availability, each MEL's QoS and the infrastructure topology.

The edge should become autonomous and work even when disconnected from the Cloud. Ramachandran et al. identified the challenges to provide a peer-to-peer standing for the edge to the cloud [Ramachandran2019]. Departing from a real-world case study, Beckman et al. [Beckman2020] diagnose the technical shortcomings to implement a solution; they consider Twister2 for data analytics and indicate the necessary implementations.

Also, the COMPSs ecosystem has taken steps towards the distribution of computation across Computing continua with Colony. This framework proposes to consider each device as an autonomous individual (agent) able to host events/data processing on its embedded computing resources in a FaaS manner. Such agents can interact with each other to share data and computing capabilities to tackle larger problems, achieve higher-performance and, thus, enabling services otherwise not viable. For doing so, the framework proposes to establish a dynamic device hierarchy and define agent colonies, disjoint groups of agents sharing their workload and resources where one of the devices acts as the endpoint that other devices/colonies within the infrastructure contact to offload part of their computation onto the colony. The project natively includes support for the COMPSs programming model; thus, any function executed in an agent can be converted into a workflow and its inner tasks offloaded onto other devices of the infrastructure in the same FaaS manner. In turn these tasks can be further decomposed in other tasks and, thus, a single computation can flood throughout the whole infrastructure.

Application component placement and resource allocation across the computing continuum are other important runtime problems. A recent survey of resource runtime management methods in edge-cloud is provided in [Hong2019]. Authors classify the literature proposals according to *architectures*, *infrastructure*, and *algorithms* for resource management in fog/edge computing. In this classification, *architectures* include dataflow (e.g., aggregation, sharing, offloading), control (e.g., centralized, distributed, hierarchical) and tenancy (e.g., single, multi), *infrastructure* includes hardware (e.g., computing equipment and network devices), system software (e.g., system and network virtualization) and middleware (e.g., volunteer edge computing, hierarchical fog/edge computing, mobile fog/edge computing, cloud orchestration management.) while *algorithms* include discovery (e.g., programming infrastructure, handshaking protocol, message passing), benchmarking (e.g., evaluating functional properties, application benchmarking, integrated benchmarking), load-balancing (e.g., cooperative load balancing, graph based, etc.) and placement (e.g., dynamic condition aware, iterative techniques).

Many works focus on component placement only on edge during the runtime. One example is provided in [Skarlat2017a] that proposes a fog computing architecture. The architecture has some partitions, called *fog cells*, which are software components running on IoT devices, and control and monitor the virtualized resources of these devices. A *control node* and a few fog cells construct a *fog colony* which acts as micro datacentres and share services between its fog cells and other fog colonies. Fog colonies are connected to a *fog computing management system* that is a middleware running in the cloud. The authors model the Fog Service Placement Problem (FSPP) as an Integer Linear Programming optimization problem which is solved at runtime and aims to determine an optimal mapping between IoT applications and computational resources while considering QoS constraints like deadlines on the execution time of applications. The same authors in [Skarlat2017b] propose a deadline-aware optimization problem to model the placement of IoT applications over edge resources, maximizing the resource usage. This problem is solved directly, and through a first-fit greedy approximation and a genetic algorithm, and a comparison with existing approaches that run all services in a centralized cloud is also provided. Similarly, [Yi2017] proposes LAVEA, a system to offload computation

between client and edge nodes to reduce latency in video analytics tasks, adopting an edge-first design and a scheduling method based on an adaptation of flow job shop model and Johnson's rule. [Puthal2019] introduces a dynamic load-balancing technique to optimize resource utilization and response times in edge datacentres, considering a secure authentication method to identify the edge datacentre before the task's deployment.

Vice versa, some works investigate the runtime component placement both on edge and cloud. Among them, [Souza2018] proposes a service placement approach for Fog-to-Cloud architectures with *parallel* and *distributed* execution. In order to improve the system performance, the proposed method guarantees higher quality of service in terms of transmission and processing delays, by splitting a large service into some sub-services. In this way, the workload will be distributed between the fog and the cloud, and all the sub-services can be run on fog and cloud, in parallel. Similarly, [Taneja2017] presents a resource aware placement algorithm in fog-cloud. The algorithm first sorts the network nodes and application components ascendingly according to their capacity and requirement respectively, then places each component on an appropriate node that has enough resources, iteratively, starting from the available fog nodes to cloud. Finally, [Mouradian2019] focuses on application component placement in Network Function Virtualization (NFV)-based hybrid cloud/fog systems. Authors implement application components as Virtual Network Function (VNFs) and model their execution sequences by a combination of substructures like *sequence*, *parallel*, *selection*, and *loops*. They formulated the application component placement problem as an Integer Linear Programming (ILP) problem which minimizes the weighted sum of execution time and cost and then they proposed a Tabu Search algorithm to find a suboptimal solution.

In addition to component placement, resource allocation and scheduling are other important issues for runtime management and many works focus on this context. Among them, some works focus on cloud resources, such as [Bao2019] which develops performance modelling and task scheduling for independent microservices-based applications. Processing time (PT) is used as the main performance metric and authors formulate a Microservice-based Application Workflow Scheduling (MAWS) problem in the cloud-side under a user-specified budget constraint (MAWS-BC) to minimize end-to-end delay. Authors demonstrate that the MAWS-BC is essentially a multiple-choice knapsack problem which is NP-complete.

Vice versa, some other works such as [Zhang2015] and [Guo2018] focus on resource allocation in edge resources while the others like [Du2018] investigate resource management in mixed edge and cloud. Authors in [Zhang2015], consider that a mobile application includes some sections, and each section can be executed in a local device or offloaded to the edge. They also consider that offloading failures can happen due to both user mobility and edge admission control. Therefore, the goal is to obtain an optimal offloading policy to minimize the mobile users' cost. In order to achieve the goal, authors propose a Markov decision process (MDP) based model to describe the optimal offloading problem, with a set of states and actions, describing the operating statuses and offloading decisions of the mobile user. Then, they formulate the MDP optimization problem by a Bellman equation and solve it by a value iteration algorithm. Authors assume that mobile user distribution follows a Poisson point process (PPP), and they consider mobility of users in the state transitions and cost function. They achieve successful offloading actions' probabilities and use these probabilities in the MDP. They prove the existence of the threshold policy and propose an equal-division decision making algorithm for mobile users. [Guo2018] proposes a queueing network to model a system in which a base station (BS) offers pure traditional cellular uplink transmission (UT) service, pure traditional cellular downlink transmission (DT) service and mobile edge computing (MEC) service at the same time. They consider three types of users who are served by different types of servers. Pure UT and DT users only need the service at queue UT servers and DT servers, respectively, while MEC users need service at all three queues. Since MEC users consume both UT and DT transmission resources to offload/download to/from the same BS, the admission control of MEC users becomes important. Given the utilization factor of each queue and

average delay requirement of each user class of each service type, they formulate a polynomial optimization algorithm to find the optimal admission control of MEC users that maximizes the revenue of service providers for providing MEC service. The underlying problem is NP-hard, and they obtained an approximate solution by applying *GloptiPoly* solver. Moreover, authors formulate another optimization problem to minimize the cost of all three types of servers, in which the decision variables are the number of the three server types to be deployed. The optimization problem is an integer programming problem and proposed a greedy algorithm to solve it.

As an example, for resource allocation in both edge and cloud, in [Du2018] the authors' goal is to make a decision for computation offloading and resource allocation to execute the mobile applications in mixed edge/cloud computing systems. In order to achieve their goal, they formulate the computation offloading problem as a mixed integer non-linear programming (MINLP) problem. To guarantee the fairness of all the users equipment (UEs), they try to minimize the maximum cost (weighted sum of energy consumption and latency) among all smart mobile UEs while a tolerable deadline for each application is guaranteed. They proposed an algorithm to solve the problem, called computation offloading and resource allocation algorithm (CORA). Then, they proposed a bisection method called BCRA to solve the nested resource allocation problem in CORA. Finally, they use fractional programming and Lagrangian dual decomposition to optimize radio bandwidth and transmit power allocation among all UEs.

Moreover, some other works like [Lin2018] focus on the runtime network capacity and traffic allocation. In particular, [Lin2018] proposes a two-phase iterative optimization (TPIO) approach to adjust capacity and traffic allocation. Authors assume that the traffic arrival rate from each device follows a Poisson Process and each packet needs two services, one provided by the core and the other by the devices. They define four types of traffic among core (a single network device), edges and devices and their goal is to distribute the latency among these traffic types. According to the TPIO algorithm, the percentages of four types are initially set to be the same. Then, the algorithm computes the overall percentage of traffic and if it is less than a threshold, it starts the first phase and chooses the devices characterized by the percentage of traffic lower than a predefined threshold as a victim. Then the approach computes the amount of shifted traffic to the victims to adjust the traffic. The algorithm iterates the first phase until the overall percentage of traffic becomes greater than the threshold. Then it starts the second phase and computes the capacity of nodes and determines which capacity should be reduced. The algorithm will stop when the capacity reduction ratio is less than a small value.

Finally, on the FaaS management side, [Das2020a] tackles the problem of determining which tasks should be deployed on edge or cloud resources, in the context of the FaaS paradigm. It proposes a dynamic task placement framework to minimize latency subject to cost constraints or to minimize costs subject to latency constraints. Similarly, in [Shahrad2020] authors characterize the FaaS workload of Azure Functions and then propose a resource management policy, called hybrid histogram policy, to decrease the number of cold starts while also minimizing the resource waste. The policy addresses some challenges such as hard-to-predict invocations, heterogeneous applications, applications with infrequent invocations, tracking overhead and execution overhead. The policy is implemented and validated in Apache OpenWhisk.

5.5 Accelerated Clusters for AI

Artificial Intelligence (AI) and Deep Learning (DL) algorithms are extensively adopted nowadays to tackle increasingly complex problems in the context of, e.g., predictive maintenance, machine vision and healthcare. Massive datasets must be processed to train Deep Neural Networks before they achieve reasonable prediction accuracy and generality. The adoption of Graphic Processing Units (GPUs) to support General Purpose computation and massive parallelism is effective in increasing the set of previously intractable problems that

can be solved within a reasonable computing time, achieving speed-up factors in the range of 5-40x compared to the CPU-only systems [Bahrapour2015, Madougou2016]. Nevertheless, despite the clear advantages of GPU-accelerated clusters in terms of performance, these are characterized by high operational costs and energy footprint. Their effective management comes therefore with great challenges. Among these, designing efficient scheduling and resource allocation solutions is crucial to maximize performance and minimize operational costs of datacentres. Even if an increasing effort has been put in recent years in tackling both problems, they are often addressed separately, focusing either on jobs scheduling or on resource allocation. In the following, we briefly review previous work in both scenarios.

The common way in which GPUs are used in high-performance supercomputers and datacentres is to install one or more of these accelerators at each node of a cluster. However, although this configuration is interesting from a performance point of view, from a power consumption point of view it is not efficient, since a single GPU can easily consume 25% of the total power of a computer, and GPUs are typically never used 100% of the time. In fact, average GPU utilization is typically not greater than 20%. Therefore, the configuration used today to exploit the computational resources of GPUs is very inefficient, both at the energy level and at the level of the acquisition cost of the equipment. A more interesting cluster configuration would be to reduce the number of GPUs installed in the cluster and to concurrently share the installed GPUs among applications. This would result in a higher utilization of the installed GPUs, a lower acquisition cost and also a lower energy consumption. Disaggregated hardware architectures offer to the existing applications the possibility of accessing several remote GPUs connected through a fast network. Enabling GPU sharing is particularly efficient when training simple models, as well as in the inference phase, where the average GPU utilization is significantly lower than 100%.

Considerable attention has been devoted to GPU scheduling in HPC systems to improve load balance and performance of CPU and GPUs, but there are a handful of solutions targeting specifically DL training applications. A scheduling algorithm based on collocating CPU-only jobs with GPU-assisted jobs is presented in [Wu2013]. GPU scheduling in HPC is also considered in [Reaño2017]: it improves resource usage through an optimized scheduling algorithm allowing multiple applications to share the same GPU.

The resource budget that needs to be shared among different applications encompasses different dimensions, such as time, GPU memory, bandwidth, and I/O channels. A mechanism that keeps track of all these multiple dimensions is developed in [Sheikhalishahi2016]. It analyses the scheduling process in the framework of multi-dimensional bin-packing, which offers an efficient mechanism to treat the job scheduling process, determining the best deployment for applications while minimizing the used resources.

A priority-based technique, named BR-SRS (Budget Reservation Spatial Resource Sharing) scheduling, is proposed in [Kang2017]. It bonds the number of GPU cores assigned to each application to its priority so that the number of resources allocated to high priority jobs is always more significant than the amount given to low-priority ones. The relation between assigned resources and priority will be also considered in AI-SPRINT, where jobs closest to the due date are processed first. However, BR-SRS does not admit pre-emption and determines a priori a fixed resource budget that should be statically reserved for high-priority jobs.

Several algorithms start from the information provided by running applications to improve the scheduling process. An example is SLAQ [Zhang2017], a cluster scheduling system for ML jobs. It exploits a token-bucket algorithm to implement a resource allocation policy that, having designed a schedule for the current time period, collects resource usage information from running jobs. This information is used to determine how the resources assigned to different applications affect the model quality to generate a prediction for future iterations that improves the process effectiveness.

A resource allocation strategy for DL training jobs is proposed in [Saxena2020]. Authors determine the optimal jobs batch size according to their scaling efficiency and combine a FIFO scheduling with a dynamic

programming algorithm to determine an effective solution. Similarly, Optimus [Peng2018] is a Kubernetes scheduler designed to manage DL jobs on a shared distributed containerized environment, which minimizes jobs training time by relying on online resource-performance models. DL² [Peng2019], also based on Kubernetes, dynamically allocates resources by exploiting Deep Reinforcement Learning instead of relying on analytical performance models and job profiling.

Domain-specific knowledge is exploited in Gandiva [Xiao2018], which focuses on tasks' recurrent behaviours during the sequence of mini-batches iterations and proposes a pre-emptive, time-sharing scheduler for DL jobs. It can operate in reactive mode, exploiting a job placement policy with oversubscription to deal with job arrivals, departures, and failures, and in introspective mode, continuously monitoring and adapting jobs placement to improve GPUs and nodes usage and reduce jobs completion time.

Integrated frameworks supporting the whole training process are implemented in Tiresias [Gu2019], Philly [Jeon2019], and FfDL [Jayaram2019]. They deal with job submissions and execution and focus only on the scheduling of DL training jobs, while the resources to be assigned to all applications are specified by the users. Specifically, Tiresias has the primary goal of reducing jobs completion times, in a framework where the actual execution times are unpredictable. Moreover, it identifies and applies some model-specific criteria to determine when the worker GPU collocation constraints should be relaxed. Philly is a service in Microsoft for training machine learning models. Based on the data collected through it, the authors analyse how queuing delays and GPU utilization are influenced by local constraints and characterize the causes of failures in their system. Furthermore, FfDL is developed as a middleware between the deep learning frameworks and the hardware resources and supports scalability and fault tolerance of applications.

Finally, [Chiang2021] proposes DynamoML to support efficient ML pipeline workload executions on GPU clusters. Implemented as an extended framework for Kubernetes, it involves three modular components: a fine-grained GPU allocator, that enables GPU sharing among light-loaded jobs, an application-aware scheduler and a performance-driven auto-scaling controller, which together cooperate in building a dynamic and pre-emptive scheduler for heterogeneous workflows.

GPU sharing is the main purpose of the vGPU [NVIDIA-vGPU] and MIG [NVIDIA-MIG] solutions recently proposed by NVIDIA. Basically, these two solutions virtualize GPUs in order to concurrently provide them to multiple applications. However, although these virtualization solutions effectively address the aforementioned concerns, it is possible to use a better approach: make use of the remote GPU virtualization mechanisms. Remote GPU virtualization adds flexibility and efficiency to the local virtualization solutions proposed by NVIDIA.

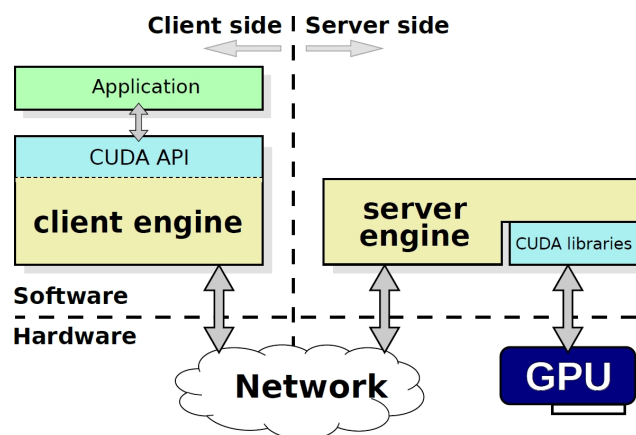


Figure 5.3 - Typical architecture of remote GPU virtualization frameworks

Figure 5.3 shows how remote GPU virtualization works. Remote GPU virtualization frameworks follow a client-server approach. The middleware server runs on the cluster computers where GPUs are installed, which serve the requests that come from the middleware clients. Clients run on the cluster computers that do not have a GPU and are presented to applications as the CUDA library. In this way, accelerated applications can be executed in any of the cluster nodes and when they need the services of a GPU they use the client library of their computer, in a transparent way from the application point of view, to send their requests to the actual GPU that is installed on another computer in the cluster. It should be noted that the application is not aware that it is interacting with a virtual remote GPU, but it believes that it has exclusive access to a real GPU installed on the local computer. Obviously, the overhead introduced by the remote GPU virtualization middleware depends on the capabilities of the network fabric connecting clients and servers.

Among remote GPU virtualization solutions, we can name, for example, rCUDA [Reaño2017], [Reaño2015], vCUDA [Shi2009], GVIM [Gupta2009], GVirtuS [Giunta2010], GridCUDA [Liang2011] and V-GPU [VGPU2021] (the latter only worked on InfiniBand) which pursue the virtualization of the CUDA runtime API (Application Programming Interface). In the case of OpenCL, VCL [Barak2010] and SnuCL [Kim2012] present work environments with similar characteristics to the environments created for CUDA. Regarding the environments focused on the CUDA API, it should be noted that, except for rCUDA (one of the most important AI-SPRINT asset in this area), the rest of the existing solutions only provide partial support for obsolete CUDA versions, what makes them non-interesting from a practical point of view and, at an industrial level, are not usable.

In the case of rCUDA, it presents an overhead in the execution times of applications, when the InfiniBand high-performance network is used, around 3% compared to the usual configuration in which the GPU is used locally with CUDA. rCUDA can also work with any TCP/IP compatible network and its overhead will depend on the performance capabilities of the underlying network.

5.6 Gaps filled by AI-SPRINT

For what concerns AI application deployment and continuous integration, AI-SPRINT will leverage the Infrastructure Manager (IM) developed by UPV, an open-source TOSCA compliant deployment environment which uses Ansible recipes to configure and update complex infrastructures automatically. Relying on this platform-agnostic tool, compatible with multiple providers and standards (including cloud OCCI), AI-SPRINT will ensure interoperability with most current public and private clouds, will avoid cloud vendor lock-in, and will ease the deployment on edge servers.

To provide serverless support for compute-intensive applications, AI-SPRINT will leverage the open-source tool SCAR [SCAR][Pérez2018] which provides the ability to transparently execute containers out of Docker images in AWS Lambda, supporting complex generic applications in various areas and deep learning frameworks. SCAR relies on udocker [udocker], an open-source development that allows to execute containers from Docker images in user space, that is, without requiring any root privileges and without requiring a Docker installation, thus being able to run containers also on restricted execution environments. SCAR also supports automatic delegation of event-driven functions into AWS Batch jobs, for workloads that exceed the computing capabilities of AWS Lambda. This allows the automatic provisioning of an elastic compute cluster to execute long-running jobs or accelerated workloads using GPU-based computing [Risco2021].

On the other hand, to provide a portable and vendor agnostic FaaS approach we will use OSCAR [OSCAR] which supports the very same computing model offered by SCAR but within an on-premises CMP. OSCAR can be automatically deployed on multi-Clouds in order to create highly-parallel event-driven file-processing serverless applications that execute on customized runtime environments provided by Docker containers than

run on an elastic Kubernetes cluster. OSCAR uses both EC3 and the IM to automatically provision a configured Kubernetes cluster on several Cloud back-ends (including EGI FedCloud) that can dynamically scale in and out in terms of the number of virtual machines. In AI-SPRINT, OSCAR will be extended to run on minified Kubernetes distributions such as K3s and ARM-based processors in order to support inferencing of pre-trained Machine Learning models at the edge. Through the combination of SCAR and OSCAR we will support data-driven inference-based workflows that can run along the Cloud continuum.

Application components will be continuously monitored by a monitoring platform which will be based on an extension of the InfluxDB technology and ecosystem to support network partitioning and edge devices disconnections. The monitoring platform will be able to gather metrics at every layer (including the application level and AI models inference accuracy) of the computing continuum.

Within AI-SPRINT, runtime management will leverage the PyCOMPSs runtime that will be extended to support the serverless computing paradigm through OSCAR, in a way application components or COMPSs workers will be run as event-triggered functions orchestrated within a workflow. Application components migration will leverage Krake (see *AI-SPRINT D1.3 - Initial Architecture Design*) which provides a central point to manage application components at any layer of the computing continuum and re-evaluates components placement at regular intervals. While Krake currently only supports infrastructure metrics (e.g., resource utilization, hardware energy efficiency) it will be extended to support performance-oriented application metrics and to be able to take optimal decisions on how many resources (e.g., number of Docker containers/total cores) need to be allocated to provide application components execution time guarantees.

Finally, for what concerns the scheduling of accelerated cluster, AI-SPRINT will propose advanced techniques to solve the joint resource planning (i.e., how many GPUs to assign to a training job) and scheduling problems (i.e., determine the job ordering leading to optimal resource access) both for private and public clouds. The final goal is to obtain the efficient use of shared public cloud VMs/private infrastructures while observing the completion time goals of training jobs (taking into account, possibly, AI model partitioning between cloud and edge servers). GPU sharing and access mechanisms will be based on rCUDA.

6. Security and Privacy for Edge AI applications

Chapter 6 is devoted to the security and privacy issues that arise in AI applications, and on the strategies that can be adopted to tackle and reduce them.

In particular, the trusted execution environment is introduced and presented in Section 6.1. Section 6.2 is focused on the patch management, while Section 6.3 describes Secure Boot standard. The network security and attacks in edge systems are overviewed in Section 6.4, while Section 6.5 is focused on privacy concerns for AI applications. Finally, Section 6.6 presents the strategies adopted by AI-SPRINT to tackle such security and privacy issues, with a particular focus on how it addresses the shortcomings of the proposed literature.

6.1 Trusted Execution Environment

Edge computing infrastructure has been introduced to fulfil the low latency requirement of emerging applications. There are several studies on the performance and applications of the AI-edge computing paradigm [Tianshu2019], however, less studies have focused on the security and privacy issues of this paradigm. In addition, the complexity introduced by recent systems, makes the traditional security methods incapable of solving the new systems and applications' security demands [Sabt2015].

The software-based security approaches such as microkernels, sandboxes, and virtualizations cannot fulfil the desired security level for data confidentiality and integrity [Pinto2017]; therefore, nowadays, vendors have moved towards integrating hardware-assisted Trusted Execution Environment (TEE)s to the edge computing platform [Ning2018]. Trusted Execution Environment is a secure, integrity-protected environment with processing and storage capabilities which is isolated from the regular processing environment [Asokan2014]. TEE guarantees the confidentiality of the code, data and provides remote attestation to prove its trustworthiness for third-parties. It can resist against software and the physical attacks which are performed on the system main memory [Sabt2015].

Due to the heterogeneous architecture of edge nodes, the hardware vendors with different architectures provide several hardware-assisted TEEs such as Intel Software Guard eXtension (SGX), Arm TrustZone, and AMD Memory Encryption Technology [Ning2018]. In the following, we survey some of the studies on these popular hardware-assisted TEEs.

SGX, which was introduced by intel in 2013, provides TEE for users and applications by applying a new memory access mechanism. The application in SGX is executed in a secure container called Enclave. The sensitive data and related functions of an application are placed inside the enclave which is safe from possible attacks by prohibition of the external access. SGX provides a small, trusted computing base (TCB) unlike the other TEE systems [Wei2021].

Haven introduced in [Baumann2015] is one of the first methods to use Intel SGX for isolation. In Haven, unmodified apps such as SQL Server and Apache can be run inside an enclave and kept protected from the host which can be the cloud operator's OS, VM, and firmware. To keep the host safe from untrusted AI services, Haven also uses a secure container which does not have access to normal OS services [Meurisch2021]. Another beginner approach is Verifiable Confidential Cloud Computing (VC3) [Schuster2015] which uses the SGX enclave concept to protect the confidentiality and integrity of user data in distributed MapReduce computations in the cloud. To provide this confidentiality and integrity in VC3, the unmodified running Hadoop, operating system and the hypervisor are kept out of the TCB.

Using SGX, a secure cloud method is proposed in [Kelbert2017] as a layered architecture that provides functions such as secure creation and deployment of micro-services, the secure integration of individual

micro-services for big data applications and the safe execution of applications in untrusted cloud environments. In this work, authors handle security issues from a full stack perspective and the micro-services are deployed by donating a secure container at the top of the non-secure stack [Wei2021].

Focusing on securing file and network communication for non-encrypted applications, SCONE [Arnautov2016] reduces enclave transition costs by using asynchronous system calls. SCONE is a secure containerization method that uses the SGX to protect container processes from external attacks. It uses the C standard library inside the SGX enclave, and all system calls of the application are assigned to the non-secure host [Priebe2018].

TEEMon, presented in [Krahn2020], is a performance monitoring and analysis framework used to measure the overhead of Intel SGX frameworks such as SCONE [Arnautov2016]. This framework helps developers by providing the analysis to identify the performance bottlenecks caused, especially considering tight storage resources such as EPC and the expensive enclave operations [Pericas2020].

There are a few works which have considered AI services and algorithms in TEE. Chiron, implemented using SGX enclaves proposed in [Hunt2018], is a system for privacy-preserving ML as a service. The training data is kept hidden from the service operator and the user has no access to the training algorithm and the model structure and access to the user data is within an extended Ryoan sandbox. Two types of enclaves are determined in this work namely, training one containing the standard ML training toolchain (including the popular Theano framework and C compiler) and parameter enclave for exchanging AI model parameters [Meurisch2021]. Finally, authors in [Chandra2017] proposed a secure defence strategy while achieving higher computational efficiency and reduced performance overhead. This method introduces proper resistance to side-channel attacks. However, due to adding noise to the data, the accuracy of the AI algorithms is reduced [Meurisch2021].

6.2 Patch management

Besides providing confidentiality and integrity protection for applications when running in untrusted public environments, it is equally important to provide secure means of software patch management such that the integrity of the used software artefacts can be always maintained.

Although Intel SGX provides remote attestation which allows to verify if application code has not been modified while being loaded in the so-called enclave, there are currently no means to ensure that software upgrades such as transitioning to a newer version will not introduce security flaws due to malicious introduced code.

One approach to achieve this, is the monitoring of the source code evolution as well as the compilation processes of the artifacts by a source code repository as provided through, e.g., git. These mechanisms allow establishing a verifiable and testifiable source code history through signed commits as well as a new mechanism, called secure boot, as compilation steps and intermediated outputs can be versioned as well. We will therefore review work in this area of providing a secure history utilizing TEEs as well as version control systems.

A number of publications covered mechanisms for integrity protection of persisted data and how to provide secure application logging using secure file systems [Burns2005, Peterson2007, Opera2007]. For example, for Ext3Cow [Burns2005], Burns et al. describe techniques to provide secure audits and compute MACs incrementally while Oprea et al. [Opera2007] focus on aspects such as detecting integrity violations using Merkle-Trees. In AI-SPRINT, we plan to go beyond those works as we assume these techniques as state of the art, and we will utilize them either directly or indirectly. For example, we envision to use the mature LIBGIT

library [Lbgit2021] implementation that uses Merkle-Trees internally to detect integrity violations etc. that can help to detect injection of malicious code in source code or in intermediated output files as part of the compilation process. Note that, we plan also to use a trusted compiler chain such that the compilation process itself runs in enclaves and cannot be tampered as well.

An alternative approach to our envisioned path was proposed by authors of SGX-Log Karande et al. [Karande2017] through the so-called secure system logs that utilizes a combination of security mechanisms of Intel SGX for sealing persisted data and hash-chaining for efficient integrity verification of logs. In AI-SPRINT we envision to use similar techniques, however, with the addition that we provide protection for arbitrary application data on file systems rather than solely system logs. Furthermore, our goal is to provide complete transparency and to stay application agnostic, such that we do not require any source code or application modification and we use well established algorithms for recording history.

Secure patch management can also be considered as a service that requires the detection of integrity protection with each roll out of a new version. Hence, another approach that can be taken into consideration is the approach described by Aublin et al [Aublin2018]. The authors describe a tool to provide an auditing library for internet-based services that captures client-server interactions and securely logs them prior to relaying the messages to their destinations. The replying step could be considered as a final commit when rolling out a new release. Although this system is a promising direction as it provides an audit log that can be conveniently inspected, in AI-SPRINT we also require auditing on file system level rather than network traffic/interaction level as proposed in these works as compilation and source code modifications are solely occurring on file system level.

6.3 Secure Boot

Trusted computing mainly splits into two parts. The first part can be described as a hardware root-of-trust, whereas the second part is based mainly on software and is called Secure Boot. An example for hardware root-of-trust is the Trusted Platform Module (TPM). It is an industry standard and comes as a chip which uses cryptographic services like hashing and encryption to make sure each TPM device can be booted without being malicious [Microsemi2013, Smith2021]. There are also software-based TPM emulators that can be used, but it is said that such emulators offer nowhere near as much security benefit as a real hardware TPM [Keylime2021].

Secure boot is a security standard that ensures that only trusted software is booted when turning on a computer by using signature verification. This software is often referred to as OEM (Original Equipment Manufacturer). To be able to use Secure Boot, it is necessary that the motherboard's firmware is running on UEFI (Unified Extensible Firmware Interface). The reason behind this is that Secure Boot is an industry standard developed by the non-profit UEFI Forum and was launched in 2012. When the computer is turned on, the firmware checks every signature of each boot software, including the UEFI firmware itself and the operating system via public/private key pairs. The developer of the software "signs" the code with their specific private key so that the matching public key can be used to check the software origin. This happens even before anything else in the software is running. However, the whole process can also be referred to as a chain-of-trust, where each segment is checked one after the other and, if one fails, the whole boot process is interrupted. If the checks are successful, the firmware gives the control to the booted operating system. The reason secure boot was invented in the first place is that some of the earliest viruses for PCs were boot sector viruses, hiding from the OS by taking action pre-boot. The general purpose of secure boot is to bring the system to a known and trusted state [Wilkins2013, Smith2021].

Besides secure boot in general, there are other mostly hybrid boot process security measurements. Sau et al. [Sau2017] mention some other forms of the secure boot process in their work like Trusted Boot, Certified Boot, Measured Boot, Verified Boot and Authenticated Boot. Whereas Trusted Boot refers to the hardware root-of-trust design mentioned in the first part of this section. It is described as taking integrity measurements in a system with the results stored in a TPM. Certified Boot, on the other hand, is described as the combination of both, Secure Boot and Trusted Boot. Measured Boot sits on top of Certified Boot and uses TPM's Platform Configuration Register (PCR) to store its integrity measurements. It relies on the idea that each core code component is measured, like hashed with the information of the next component in the chain of trust and stored in the PCR before it is executed. Verified Boot is very similar to Trusted Boot in its functionality but developed from Google for Chrome OS. Authenticated Boot basically works like Certified Boot, but, in some papers, it is called differently [Sau2017].

Another work states that Secure Boot itself is not able to protect against all sorts of attacks or in that case against LoJax, a boot-kit [Kuzminykh2019]. The case is that LoJax is deployed to the SPI Flash of Secure Boot. However, because Secure Boot starts the "chain of trust" via the platform key that resides in the SPI Flash, Secure Boot only checks actions after the SPI Flash firmware is loaded. Thus, it is not able to defend against attacks that target the SPI Flash firmware. The authors also discuss alternative methods or extensions, such as Boot Guard or BIOS Guard, to boot the firmware securely, since Secure Boot alone is not enough. They also mention that with each additional security method, the system's boot mechanism is slowed down further. The conclusion is that they cannot give any universal recommendations for rootkit detection since there is not an optimal solution. The given solutions like Boot Guard or BIOS Guard work but need to be reconfigured manually for proper detection. However, their future research will be focused on developing proper security features.

Due to the growth of IoT-related use cases today, many academic papers can be found that address this type of topic. Field Programmable Gate Arrays (FPGA) are mentioned again and again in the scientific papers of various authors. The reason for this is that these little "things" also bring new possibilities for attacking vulnerabilities with malware. FPGAs are edge devices on which, for example, sensor technology can be present. Siddiqui et al. [Siddiqui2019] mention several risks of attacks on FPGAs like IP cloning, hardware trojans or man in the middle attacks. There are ways to attack IoT devices directly on-site and remotely. The authors discuss further that with proper Secure Boot mechanisms, those dangers could be minimized. Therefore, they introduce a so-called Multilayer Camouflaged Secure Boot process that uses a unique device-level physical unclonable function to unlock the design and update the look-up table (LUT) frame that is unique to each device to mitigate the security vulnerabilities of maliciously modifying the boot image of the programmable logic programming bitstream. This layered secure boot process allows the remote Attestation Server to mutually authenticate and verify the design running on the fabric with logic locking and LUT frame modification [Siddiqui2019].

In another work by Elrabaa et al. [Elrabaa2019], a new scheme for providing secure enclaves on FPGAs for users' data and applications in data centres and clouds was developed. The proposed protocol provides FPGA authentication, protection of user application confidentiality and integrity, and confidentiality of their data. A full proof-of-concept prototype implementation by the authors has shown that the solution can be easily implemented on existing FPGAs. In addition, the solution is said to be efficient in terms of resource usage and take less time to boot compared to traditional software-based virtual machines. Thus, even on slower FPGA devices with, say, 100 MHz clock speed, a secure application should run faster on the FPGA than a switch version running on "bare metal" (i.e., without containerization, virtualization, or a secure enclave like Intel's SGX).

There are also works which address booting speed based on Secure Boot mechanics. For example, Profentzas et al. [Profentzas2019] mention potential attacks based on slow booting times due to Secure Boot methods,

like DoS attacks especially at the IoT level. They conclude that secure boot is one of the primary tools for securing IoT applications, but also find that software-based secure boot increases boot time by 4%, while hardware-based tools such as TPM increase boot time by 36%. In the case of Secure Boot itself, Sarmah et al. use PCIe to improve booting times significantly, with up to 20 times faster booting compared to standard booting mechanisms [Sarmah2017].

In addition, in a recent survey, Rashmi et al. [Rashmi2018] discuss various methods of Secure Boot. They address Secure Boot of embedded Linux, Secure Boot mechanisms of FPGA based IoT devices, Secure Boot based on Mobile Trusted Modules (MTM) and Secure Boot of wireless sensor networks and therefore provide additional references on this topic with a wide variety of works. This paper provides a good foundation for further development in the Secure Boot area.

In another paper by Bashun et al. from 2013, the problems and security gaps of Secure Boot are discussed and summarised. For this purpose, each segment of the Secure Boot process was examined in more detail by the authors. They also discuss various threats and potential attacks on each segment or boot phase. The conclusion, as the title of the paper itself says (Title: Too Young to be Secure: Analysis of UEFI Threats and Vulnerabilities), is more negative than positive. It says that Secure Boot is mandatory, but also has enough disadvantages. For most of the drawbacks, the conclusion comes with a guideline on what needs to be addressed or implemented to ensure full security. This paper shows good starting points for further research to close some of the gaps. Figure 6.1 shows a summary of all potential danger spots with a brief explanation [Bashun2013].

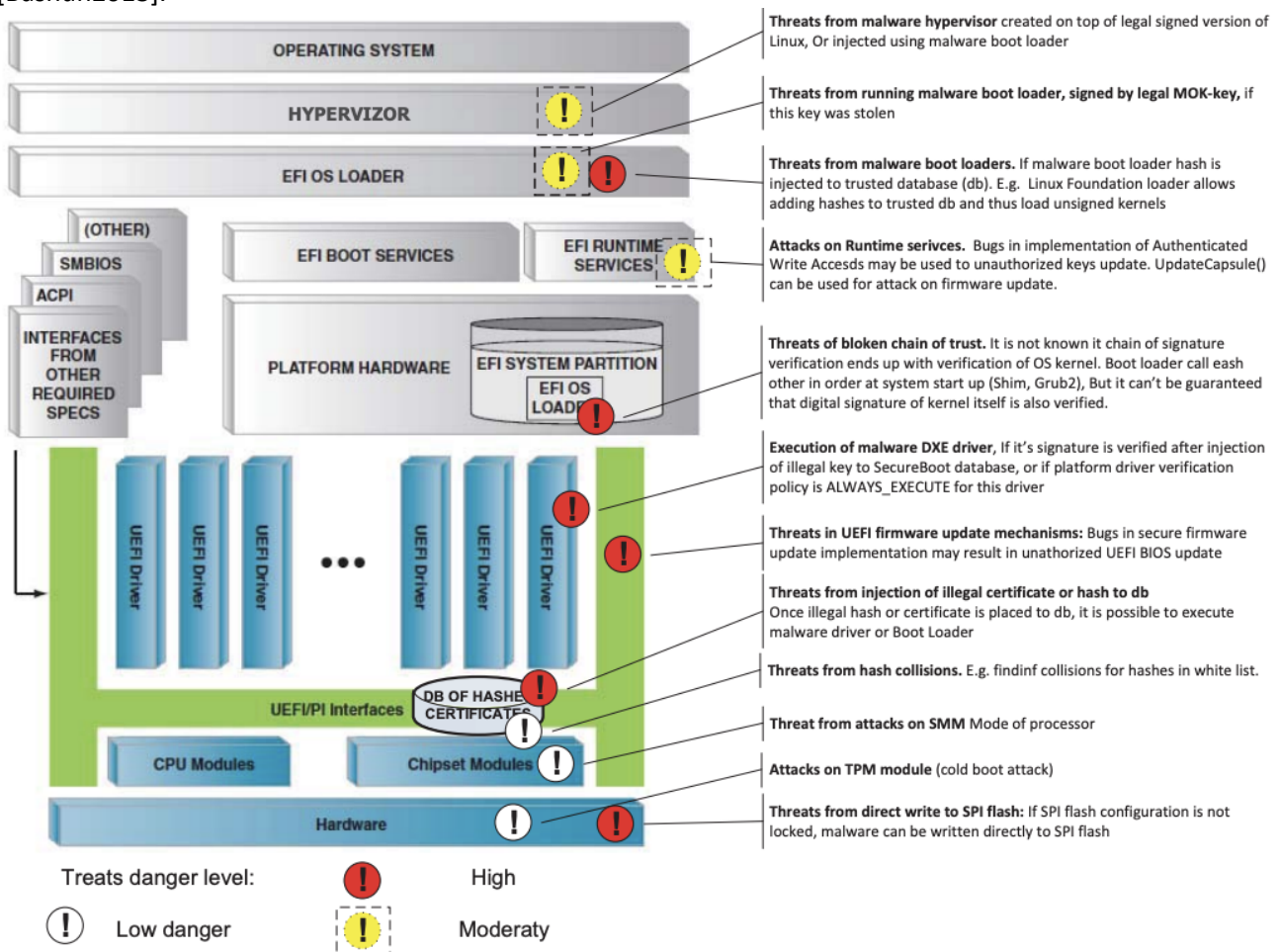


Figure 6.1 - Summary of threats, threat level and phase or location of occurrence [Bashun2013]

It is worth mentioning at this point there is a scalable open-source trust system which uses TPM technology. This system is called Keylime; it is developed in python and is open source with Apache 2.0 license. It supports TPM 2.0 and can be used in Edge, Cloud and IoT environments. It also can be used with TPM emulated solutions, but the security is way weaker compared to hardware-based TPM solutions [Keylime2021]. The best way to describe Keylime is to cite the introduction from Keylime's GitHub readme [Keylime-Github2021]: "Keylime provides an end-to-end solution for bootstrapping hardware rooted cryptographic trust for remote machines, the provisioning of encrypted payloads, and run-time system integrity monitoring. It also provides a flexible framework for the remote attestation of any given PCR (Platform Configuration Register). Users can create their own customized actions that will trigger when a machine fails its attested measurements."

6.4 Network Security and Attacks in Edge Systems

The edge paradigm can be used on multiple, diverse network architectures. Consequently, the relevant security threats are multiple and diverse (see, e.g., Roman et al. [Roman2018]). The impact and feasibility of such threats is different depending on the specific edge architecture, i.e., whether the edge node is public, such as in Mobile Edge Computing or Mobile Cloud Computing [Suo2013], or it is private.

A first group of security threats involve external attackers compromising the communications infrastructure. This group includes Denial-of-Service attacks, such as jamming of wireless channels, Man-in-the-Middle attacks, and Rogue Gateways. In the Man-in-the-Middle attack, an attacker takes control of part of the network and may eavesdrop or modify messages [Stojmenovic2016]. In the Rogue Gateway attack, an attacker deploys a malicious node and attracts legitimate traffic, similarly to the Man-in-the-Middle attack or can even inject malicious traffic to legitimate nodes.

A second group of threats consists in compromising the service infrastructure. This group includes theft or misuse of resources, privacy leakage, service manipulation, up to rogue infrastructure. Examples of misuse of resources are the deployment of bots for mining cryptocurrencies, for sending spam, or for distributing malware. Since edge nodes tend to store more sensitive information, privacy leakage is a particularly dangerous attack [Yi2015]; in addition, the same data is also an appealing target for ransomware. Service manipulation consists in modifying the result of an in-cloud computation. Such an attack can be critical in the context of the Industrial IoT because it can result in misuse or damage of equipment or even in life threatening injuries. In a public service infrastructure, the attacker could deploy a rogue service and steal user information or inject bogus data.

Finally, a last group of threats consists in compromising the user devices either by gaining control or a legitimate device or by inserting a rogue device in the system. Compromised devices can distribute bogus data and alter the result of a service.

In order to address these threats, there has been growing interest in using the 5G network protocols and their extensive security features, also in a private setting [Parikh2020], [Pronto2020]. In order to address these threats, a number of countermeasures have been designed. Access control systems are essential to identify and authorize legitimate users. The common practice is based on centralized admission control and cryptographic exchange, which provide mutual authentication between the user or the user's device and the network or the service. As the authentication server is remote, the authentication delay may be unacceptably high for edge computing applications. Recent literature introduces fast authentication mechanisms that leverage 5G procedures [Duan2015] or blockchain [Cui2020].

A second trend is the usage of micro-segmentation, which is a common technique to prevent lateral movement of attackers in datacenters [VmWare2014]. In the context of edge computing, micro-segmentation can be used to separate the data of different services or even of different users of the same service

[Mammela2016]. Micro-segmentation is also often used along with Zero-trust authentication, by which every node must be explicitly authorized to attach to a micro-segment. This can be achieved by using a policy server at each edge node [Kantola2016].

End-to-end security is then achieved by chaining micro-segments from multiple networks or network slices, which exposes multiple challenges such as guaranteeing consistent updates throughout the network is a critical issue for patch management [Foerster2018]; policy checking by injecting probe packets makes it possible to assess the behaviour of hybrid networks [Zhang2019]; automatic conflict resolution is a necessary tool to minimize the impact of security policies on network performance [Li2018].

Micro-segmentation results in significant demands on the network infrastructure because it requires pervasive policy checks. Therefore, many researchers have analysed offloading use cases of packet processing logic to programmable network hardware. Authors of [Sanvito2017] propose to optimize the load of Deep Packet Inspection (DPI) software by offloading the connection tracking logic on programmable switches based on the Open Packet Processor (OPP) model [Bianchi2016], [Petrucci2017] uses the same programming model to accelerate Linux iptables. Other works exploit the P4 Language as the means to offload the packet processing to programmable hardware. The authors of [Miao2017] offload the load balancing logic with P4 implementing a key-value store on programmable switches. Other scenarios consider datacentre coordination services [Jin2018] or MapReduce acceleration [Sapio2017] by exploiting programmable data planes.

Other relevant attacks that are worth mentioning are rollback attacks and remote attestation attacks. One of the risks when running applications in untrusted environments such as public clouds as well as on edge devices is the possibility of becoming a victim of a so-called rollback attack. Although TEEs as mentioned above provide measures to ensure confidentiality and integrity of application code and data, there is currently no support to detect if a root user provides the application with an older snapshot of data in order to drive so-called version downgrade attacks or to use e.g., expired licenses etc. [Möller2014, Chen2017, Alashwali2018]. Matetic et al. [Matetic2017] argue that Intel SGX is susceptible to rollback attacks and that hardware counters, such as TPMs, offer low performance. Their framework ROTE [Matetic2017] establishes a distributed counter service by incorporating multiple machines to safeguard the current value of a counter in the protected memory of distributed SGX enclaves. Through a sophisticated messaging protocol Matetic et al. reduce the number of necessary replicas to $f + 1$ compared to $3f + 1$ in typical Byzantine fault-tolerant systems. An alternative would be the use of the so-called Lightweight Collective Memory (LCM) [Brandurger2017], which is a protocol that maintains sequence numbers and hashes to ensure that a TEE service's data cannot be rolled back unnoticed by its clients. With each request, clients send their current context, consisting of a sequence number and a hash of all previous requests, to the TEE service on the untrusted server via an encrypted channel. Clients are trusted and never lose their latest context. The TEE service maintains the latest context of all clients and can ensure that it has not been rolled back before the previous request. However, the use of this service is quite cumbersome and requires developer effort while the approach in AI-SPRINT targets a transparent approach with no to only little developer effort. Finally, remote attestation is an important measure to ensure that the application running inside an Intel SGX enclave is indeed genuine. Unfortunately, several TEEs are currently vulnerable to so-called relay attacks which target the remote attestation procedure. In these types of attacks, an adversary can control the OS (or other software) on the target host/device to relay incoming attestation requests to another host/device. For example, in the context of TPM attestation, the relay attack is called cuckoo attack [Parno2008]. In this attack, the adversary uses malicious software on the target host to redirect the attestation to a host that he controls. Thus, the provisioning of confidential data is directed to the host controlled by the adversary. To handle this attack, several defence mechanisms have been proposed however they still contain several limitations [Fink2011]. Meanwhile, in another approach Flicker [McCune2008] attests to only a single application at the time, requiring shutting down the entire OS and all other services and also requires a trusted administrator. Recently, several attacks against Intel SGX remote attestation have been

presented: ProximiTEE [Dhar2020] presents a relay attack on SGX attestation, or SGAXe [Schaik2020] shows an attack that bypasses the SGX remote attestations. However, these attacks require to first perform micro-architectural attacks such as side-channel attacks [Bulck2018, Chen2019, Kocher2019, Lipp2018], thereafter use the gained secrets to attack the remote attestation. In the context of AI-SPRINT, we target to implement techniques to circumvent an attack against TEEs remote attestation without requiring performing any microarchitectural attacks.

6.5 Privacy issues in Edge systems for AI applications

Data privacy is increasingly becoming one of the most relevant issues in the DL era. While it represents a fundamental right for the individual, it is also considered the most significant obstacle for sharing information, especially if sensitive, and thus also a limitation for learning purposes.

In the context of Federated Learning (FL), see also Section 4.3, several scenarios are characterized by huge flows of information to centralized servers which can potentially disclose private or sensitive information. Despite the high-security levels to ensure safe data exchanges and the trustworthiness of the central server, it has been proved that DL models can leak information about learned data. Indeed, different privacy attacks have been developed with the goal of showing that unintended disclosure of training dataset information is possible. These attacks can be classified into two main families, respectively called Membership Inference [Shokri2017, Rahman2018] and Model Inversion [He2019] attacks.

- *Model Inversion*, is an attack developed to find feature missing values in a dataset row, given all the remaining ones. The attack leverages access to the model, and in many cases, it is necessary also only its black-box knowledge. Further developments extend the attack to a full reconstruction of a dataset example, addressing two major themes: white-box decision tree attacks and black-box reconstruction attacks to facial recognition models.
- *Membership Inference*, instead, is a kind of attack that, given a data record and black-box access to a model, tries to determine if that record belonged to the training dataset or not. This attack methodology builds different proxy training datasets, either mimicking the distribution of the original dataset or using other datasets closely related to the private one, used to train the target model. These datasets are also called shadow datasets. Membership Inference attacks have also been developed and customized for generative models.

Another type of privacy attack, called the GAN-based Collaborative Learning attack [hitaj2017], has demonstrated the possibility of creating information leaks even in a Horizontal Federated Learning scenario: an attacker, as shown in Figure 6.2, with access to the sharing gradients party, could exploit the information contained in the models updates to reconstruct the training data by using a Generative Adversarial Network (GAN) [Goodfellow2014].

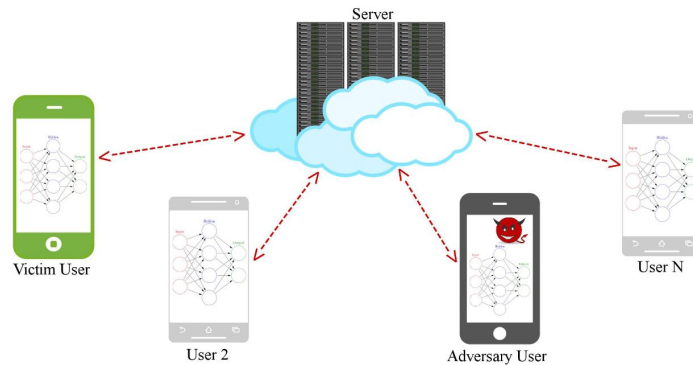


Figure 6.2 - Schema of a GAN-based Collaborative Learning attack [Hitaj2017]

One of the most adopted solutions to privacy attacks relies on the idea introduced by Abadi *et al.* [Abadi2016] which aimed at improving Differential Privacy (DP) and measuring it via the introduction of noise during different stages of neural networks training.

DP [Dwork2006], [Dwork2014] is a privacy model providing strong guarantees against the disclosure of sensitive information related to individual samples. In detail, data can be considered differentially private if each system that deals with it can only share information that describes the patterns of the groups within it, keeping private any information linked to the individual. According to the definition introduced by Dwork *et al.*, we call (ϵ, δ) -DP a randomized mechanism $M: D \rightarrow R$ with domain M and range R which is satisfied if for any two adjacent inputs $d, d' \in D$ and for any subset of outputs $S \subseteq R$ it holds that:

$$P[M(d) \in S] \leq \exp^{-\epsilon} P[M(d') \in S] + \delta$$

where ϵ is a privacy budget balancing the accuracy of the mechanism with sensitive disclosures. The smaller the budget, the higher the privacy guarantees. The δ parameter relaxes the constraints of (ϵ, δ) -DP introduced in [Dwork2006] by allowing violations with probability δ .

In the DL field, several approaches have been proposed to integrate DP guarantees within training and inferential procedures. As anticipated, to protect the sensitivity of training data, Abadi *et al.* [Abadi2016] designed a Differentially-Private version of the Stochastic Gradient Descent algorithm (DP-SGD) for training deep neural networks with non-convex objectives. Papernot *et al.* [Papernot2016] addressed the problem of the implicit storage of sensitive training data within a machine learning model, proposing the PATE framework, which relies on the separation of teacher models, directly trained on disjoint sensitive datasets and not published, and student models, learning from the teachers to predict an output retrieved by noisy voting among them.

Synthetic data generation is another important aspect from DP perspective. Deep generative models, e.g., GAN and Variational Autoencoder (VAE) [Kingma2013], are powerful tools learning to generate realistic data largely resembling the training datasets. While achieving high data utility, these models do not implement any countermeasure against the disclosure of sensitive data possibly provided during training. Chen *et al.* [Chen2018] introduced a Differentially-Private VAE-based generative model, while a Differentially-Private version of GAN is defined by Xie *et al.* [Xie2018]. Jordon *et al.* [Jordon2018] extended the PATE framework to GANs, while Mugunthan *et al.* [Mugunthan2021] applied Differential Privacy to the InfoGAN model.

DP-auto-GAN, a framework for synthetic data generation applicable to unlabeled mixed-type data, has been proposed by Tantipongpipat *et al.* [Tantipongpipat2019] and it is able to combine the flexibility of GANs with

the dimensionality-reduction capabilities of VAEs. Torkzadehmahani *et al.* [Torkzadehmahani2019] applied Differential Privacy to Conditional GANs, relying on the recent Rényi Differential Privacy model [Mironov2017]. Takahashi *et al.* [Takahashi2020] showed that by customizing the VAE ELBO loss with different divergence terms, the sensitivity of the model to the noise introduced by DP-SGD scales with the batch size. To solve this issue, the authors define a novel training procedure, term-wise DP-SGD, that keeps the VAE sensitivity low when attaching divergence.

Finally, Acs *et al.* [Acs2018], instead, separated the Differentially-Private data synthesis process in two steps: first they carry out Differentially-Private kernel K-means clustering over the sensitive training dataset, and then they train K separate generative models, one for each cluster, achieving higher data utility through the models mixture with respect to a single generative model.

6.6 Gaps filled by AI-SPRINT

The Trusted Execution Environment frameworks presented in the previous section address the execution of applications in secure enclaves but with significant limitations. AI-SPRINT will simplify the deployment, portability, and secure execution of applications on heterogeneous target platforms including GPUs. In order to widen the security aspects not yet covered by the use of TEEs, AI-SPRINT will address the following attacks within the course of the project:

- Rollback attacks: we will make use of TPM modules to safely store counter values across reboots and utilize a set of distributed machines comprising cloud as well as edge nodes to increase performance as well as minimizing the so-called vulnerability window, i.e., the number of transactions that can be rolled back.
- Patch management: furthermore, we will develop novel techniques to provide a trusted compilation environment such that frequent software updates and patches can be trusted and attested. The attestation process will be hardened in order to avoid relay attacks.

On the secure boot side, AI-SPRINT will ensure that every possible VM instance is secure boot-enabled. In addition, AI-SPRINT will possibly leverage hardware root-of-trust security such as TPM in the future. Secure Boot and TPM are security measures that have known and unknown vulnerabilities and must be well configured to provide adequate protection. Our approach will be to ensure that every single configuration and update aspect, as well as Secure Boot itself, is enabled and running at boot times to ensure a high level of security. Small gaps can lead to malicious behaviour of the booted software. Edge devices in particular, where Secure Boot can be enabled, should be included in an automated test procedure to boot securely. However, to successfully implement such high-security measures, there are some costs to overcome. Keylime or a similar system could also provide a good entry point into TPM-based secure boot mechanisms.

For what concerns privacy protection AI-SPRINT will study novel techniques leveraging data generation which exploit the concept of generative privacy as an improvement beyond the current state of the art in differential privacy. Data generation via generative models, e.g., Variational AutoEncoders or Generative Adversarial Networks, allows exchanging an undefined number of samples without explicit disclosure of single individual information and it allows the synthetic increase and rebalancing of data when data scarcity is an issue.

Finally, the AI-SPRINT secure network component will use the 5G authentication procedures to identify the end users, authorize them to access the allowed services, and automatically deploy the relevant traffic filtering rules, therefore, achieving microsegmentation in the access to edge services. It will also use smart contracts to provide access control to the data collected from IoT devices.

7. Conclusions

In this deliverable, we have provided a comprehensive overview of the state-of-the-art in several areas related to the AI-SPRINT project including, but not limited to: AI development frameworks, hardware for intelligent sensors, deployment and management solutions for computing continua and performance, privacy, and security issues. One of the main takeaways that emerge from this deliverable is the breadth, heterogeneity and complexity of the challenges that exist in building AI applications running in computing continua:

- First, we notice a generalised lack of abstractions for defining quality requirements (in terms of AI models accuracy and system performance). This will require AI-SPRINT to define novel abstractions to impose constraints on the, e.g., execution time of DN models or their partitions, or to predicate on the energy consumptions of devices running on the edge.
- Next, the analysis on standardisation fills gaps in the ICT Standardisation Rolling Plan 2021, revealing that much of the work on edge computing is taking place around global mobile communications, primarily 3GPP and ETSI MEC. Standards are needed in this area to build solutions involving multiple stakeholders. Open source also plays a role in shaping and accelerating deployments of edge clouds, where a variety of cloud-related OS projects could equally apply to edge clouds. However, it is not yet clear if and how other standards organisations will ride the wave of the rapidly expanding edge-related initiatives as the edge-cloud ecosystem is still being developed. Further work will continue in Task 6.2 Standards and Open Source Collaborations and Clustering as part of an on-going assessment.
- Moreover, we observed that the technology stack of intelligent sensors and cloud technologies is very heterogeneous, and this poses the question on how a 3-years research project like AI-SPRINT could produce a useful contribution. In fact, AI and IoT technologies are evolving fast and it does not seem always possible to come up with general solutions. It seems therefore more interesting for AI-SPRINT to focus on a few technologies of clear impact and diffusion (e.g., Keras, TensorFlow, PyTorch, Tensorflow Lite, NVIDIA RT) rather than supporting a plethora of diverse technologies for the sake of comprehensiveness. Extensibility of the approach to other technologies should still be assessed and, where possible, supported via appropriate guidelines.
- The investigation in Chapter 3 has led us to assess and compare the commercial offerings of the big cloud players and European cloud providers. Such solutions, lack in providing quality guarantees in the final models and limited support, especially in European scale providers, for advanced AI application design (e.g., DN partitioning or NAS) and performance guarantees for the final application.
- Lastly, this review has led to outline the AI-SPRINT goals and to the definition of *D1.2 - Requirement analysis* deliverable, where a detailed requirement analysis is provided. Moreover, the analysis performed in this deliverable has led to the development of an initial architecture for AI-SPRINT, supporting the different areas of interest of the project. This has led to the definition of deliverable *D1.3 - Initial Architecture Design*, which provides an overview of the general technical approach.

In conclusion, AI and computing continua are diverse and heterogeneous areas where AI-SPRINT can lead to significant innovation by providing novel tools for building AI applications with quality guarantees.

References

- [A4C] Alien4Cloud: <https://alien4cloud.github.io/>
- [Abadi2016] M. Abadi, et al. Deep learning with differential privacy. Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. 2016.
- [Acs2018] G. Acs, et al. Differentially private mixture of generative neural networks, IEEE Transactions on Knowledge and Data Engineering 31.6: 1109-1121, 2018.
- [Parikh2020] A. Parikh, Enabling a New Era of Smart Enterprises, White Paper, Open Networking Foundation 2020.
- [Ajami2015] S. Ajami, F. Teimouri. Features and application of wearable biosensors in medical care. J Res Med Sci, 20:1208–1215, 2015.
- [Alashwali2018] E. S. Alashwali, K. Rasmussen. What’s in a downgrade? A taxonomy of downgrade attacks in the TLS protocol and application protocols using TLS. In International Conference on Security and Privacy in Communication Systems (SecureComm ’18), 2018.
- [Alvarez2019] J. Álvarez Cid-Fuentes, S. Solà, P. Álvarez, A. Castro-Ginard, and R. M. Badia. dislib: Large Scale High Performance Machine Learning in Python. In Proceedings of the 15th International Conference on eScience, 96-105, 2019.
- [Ardagna2018] D. Ardagna, C. Cappiello, W. Samà, M. Vitali. Context-aware Data Quality Assessment for Big Data. Future Generation Computer Systems. Elsevier. 89, 548-562. 2018.
- [Arnautov2016] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O’keeffe, M.L. Stillwell, D. Goltzsche. {SCONE}: Secure linux containers with intel {SGX}. 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16). 2016.
- [Asokan2014] N. Asokan, et al. Mobile trusted computing. Proceedings of the IEEE 102.8, 1189-1206, 2014.
- [Aublin2018] P-L. Aublin, et al. LibSEAL: revealing service integrity violations using trusted execution. In Proceedings of Thirteenth EuroSys Conference, 2018
- [AWSGreengrass] AWS IOT GreenGrass. <https://aws.amazon.com/greengrass/>
- [AWSLambda] AWS Lambda. <https://aws.amazon.com/lambda/>
- [AWSNitro] AWS Nitro System. <https://aws.amazon.com/ec2/nitro/>
- [AzureFunctions] Azure Functions. <https://azure.microsoft.com/en-us/services/functions/>
- [Bahrapour2015] S. Bahrapour, N. Ramakrishnan, et al. Comparative study of deep learning software frameworks, arXiv, 2015.
- [Bahreini2017] T. Bahreini, D. Grosu. Efficient placement of multi-component applications in edge computing systems. The Second ACM/IEEE Symposium SEC’17, 2017.
- [Balevi2018] E. Balevi, R. D.Gitlin. Optimizing the Number of Fog Nodes for Cloud-Fog-Thing Networks. IEEE Access, 6: 11173-11183, 2018.
- [Bao2019] L. Bao, C. Wu, X. Bu, N., Ren, M., Shen. Performance modeling and workflow scheduling of microservice-based applications in clouds. IEEE Transactions on Parallel and Distributed Systems, 30(9): 2114–2129, 2019.
- [Barak2010] A. Barak, T. Ben-Nun, E. Levy, and A. Shiloh. A package for OpenCL based heterogeneous computing on clusters with many GPU devices. In 2010 IEEE International Conference On Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS), 2010.
- [Bashun2013] V. Bashun et al. Too Young to be Secure: Analysis of UEFI Threats and Vulnerabilities. In 14th Conference of Open Innovations Association. 2013.
- [Baumann2015] A. Baumann, M. Peinado, G. Hunt. Shielding applications from an untrusted cloud with haven. ACM Transactions on Computer Systems (TOCS) 33.3: 1-26, 2015.
- [Bayoumy2021] K. Bayoumy, M. Gaber, A. Elshafeey, O. Mhaimed, EH. Dineen, FA. Marvel, et al. Smart wearable devices in cardiovascular care: where we are and how to move forward. Nat Rev Cardiol 1–19, 2021.
- [Beakcheol2019] J. Beakcheol, K. Myeonghwi, H.a Gaspard, K. Jong, Q-Learning Algorithms: A Comprehensive Classification and Applications. IEEE Access. PP. 1-1. 10.1109/ACCESS.2019.2941229. 2019.
- [Beckman2020] P. Beckman, J. Dongarra, N. Ferrier, G. Fox, T. Moore, D. Reed, M. Beck. Harnessing the Computing Continuum for Programming Our World. IEEE Fog Computing: Theory and Practice:215-230, 2020.

- [Bellendorf2020] J. Bellendorf, Z. Á. Mann. Classification of optimization problems in fog computing. *Future Gener. Comput. Syst.* 107: 158-176. 2020.
- [Bender2018] G. M. Bender et al. Understanding and Simplifying One-Shot Architecture Search. In: 2018. url: <http://proceedings.mlr.press/v80/bender18a/bender18a.pdf>.
- [Bernstein2014] D. Bernstein. Containers and Cloud: From LXC to Docker to Kubernetes. In: *IEEE Cloud Computing* 1.3, pp. 81–84, 2014.
- [Bianchi2016] G. Bianchi, M. Bonola, S. Pontarelli, D. Sanvito, A. Capone, C. Cascone. Open packet processor: a programmable architecture for wire speed platform-independent stateful in-network processing. *arXiv preprint arXiv:1605.01977*, 2016.
- [Bishop2006] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.
- [Brandenburger2017] M. Brandenburger, C. Cachin, M. Lorenz, R. Kapitza. Rollback and forking detection for trusted execution environments using lightweight collective memory. In *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '17)*. 2017.
- [Brasier2019] N. Brasier, C.J. Raichle, M. Dorr, et al. Detection of atrial fibrillation with a smartphone camera: first prospective, international, two-centre, clinical validation study (DETECT AF PRO). *Europace*.21:41–47, 2019.
- [BROOKLYN] Apache Brooklyn. <https://brooklyn.apache.org/>
- [Bulck2018] J.V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, R. Strackx. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.
- [Bures2018] T. Bures, V. Matena, R. Mirandola, L. Pagliari, C. Trubiani. Performance Modelling of Smart Cyber-Physical Systems. Companion of the 2018 ACM/SPEC International Conference on Performance Engineering, ICPE 2018.
- [Burns2005] R. Burns, Z. Peterson, G. Ateniese, S. Bono. Verifiable audit trails for a versioning file system. In *Proceedings of the ACM Workshop on Storage Security and Survivability*, 2005.
- [Caballer2015] M. Caballer, I. Blanquer, G. Moltó, et al. Dynamic Management of Virtual Infrastructures. *Journal of Grid Computing*, 13: 53–70, 2015.
- [Caballer2018] M. Caballer, S. Zala, Á. López García, G. Moltó, P. Orviz Fernández, M. Velten. Orchestrating Complex Application Architectures in Heterogeneous Clouds. *Journal of Grid Computing*, 16: 3–18, 2018.
- [Campbell2020] Campbell BCV, Khatrri P. Stroke. *The Lancet*, 396:129–42. 2020.
- [Chandra2017] S. Chandra, V. Karande, Z. Lin, L. Khan, M. Kantarcioglu, B. Thuraisingham. Securing data analytics on sgx with randomization. *European Symposium on Research in Computer Security*. Springer, Cham, 2017.
- [Chapelle2006] O. Chapelle, B. Schölkopf, A. Zien, *Semi-Supervised Learning*. MIT Press, Cambridge, MA. 244, 544. 2006.
- [Chen2017] Y. Chen, Y. Zhang, Z. Wang, T. Wei. Downgrade attack on TrustZone. *arXiv preprint arXiv:1707.05082*, 2017.
- [Chen2018] Q. Chen, et al. Differentially private data generative models. *arXiv preprint arXiv:1812.02274*, 2018.
- [Chen2019] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai. SgxPectre: Stealing Intel Secrets from SGX Enclaves Via Speculative Execution. In *IEEE European Symposium on Security and Privacy (Euro S&P)*, 2019.
- [Cherariet2020] A. Cherariet. Modélisation expérimentale et statistique des relations entre les caractéristiques morphologiques de « la vigne et les dépôts de pulvérisation: application à l’agriculture de précision. www.theses.fr/s194690. 2020.
- [Chiang2021] M. Chiang, J. Chou, DynamoML: Dynamic Resource Management Operators for Machine Learning Workloads. In *CLOSER 2021 proceedings, Volume 1*, pages 122-132, 2021.
- [Chugh2014] S.S. Chugh, R. Havmoeller, K. Narayanan, D. Singh, M. Rienstra, E.J. Benjamin, et al. Worldwide epidemiology of atrial fibrillation: a Global Burden of Disease 2010 Study. *Circulation*, 129:837–47, 2014.
- [CLOUDIFY] Cloudify. <https://cloudify.co/>
- [Cui2020] Z. Cui et al. A Hybrid Blockchain-Based Identity Authentication Scheme for Multi-WSN. In *IEEE Transactions on Services Computing*, 13(2), 241-251, 2020.
- [Dagher2020] L. Dagher, H. Shi, Y. Zhao, N.F. Marrouche. Wearables in cardiology: here to stay. *Heart Rhythm* 17, 889–895, 2020.

- [Dao2015] T.T. Dao, J. Kim, S. Seo, B. Egger, J. Lee. A performance model for GPUs with caches. *IEEE TPDS* 26(7), 1800–1813. 2015.
- [Das2018] A. Das, S. Patterson, M. Wittie. EdgeBench: Benchmarking Edge Computing Platforms. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*. 2018.
- [Das2020a] A. Das, S. Imai, S. Patterson, M. P. Wittie, 2020. Performance Optimization for Edge-Cloud Serverless Platforms via Dynamic Task Placement. *20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. 2020.
- [Das2020b] A. Das, A. Leaf, C. A. Varela, S. Patterson. Skedulix: Hybrid Cloud Scheduling for Cost-Efficient Execution of Serverless Applications. *IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, 2020.
- [David2021] R. David, J. Duke, A. Jain, et al. TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems. arXiv: 2010.08678 [cs.LG]. 2021.
- [Deb2014] K. Deb, H. Jain. An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints. In: *IEEE Transactions on Evolutionary Computation*, 18(4), 577–601. 2014.
- [Denil2013] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, N. de Freitas. Predicting Parameters in Deep Learning. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2. NIPS'13. Lake Tahoe, Nevada: Curran Associates Inc., 2148–2156*, 2013.
- [Dhar2020] A. Dhar, I. Puddu, K. Kostianen, S. Capkun. ProximiTEE: Hardened SGX attestation by proximity verification. In *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*, 2020.
- [Diamos2010] G.F. Diamos, A. Kerr, S. Yalamanchili, N. Clark N. Ocelot: A dynamic optimization framework for bulk-synchronous applications in heterogeneous systems. In: *19th Int'l Conf. Parallel Architecture and Compilation Techniques (PACT 10)*, IEEE, 353–364. 2010.
- [Disabato2015] S. Disabato, M., Roveri, C., Alippi. Distributed deep convolutional neural networks for the internet-of-things. *IEEE Transactions on computers*, 14(8):1–14, 2015.
- [Dong2001] X. Dong, Y. Yang. NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search. arXiv: 2001.00326 [cs.CV]. 2020.
- [Dörr2019] M. Dörr, V. Nothurfft, N. Brasier, et al. The WATCH AF Trial: SmartWATCHes for Detection of Atrial Fibrillation. *JACC Clin Electrophysiol.* 5:199–208, 2019.
- [Du2018] J. Du, L. Zhao, J. Feng, X. Chu. Computation Offloading and Resource Allocation in Mixed Fog/Cloud Computing Systems With Min-Max Fairness Guarantee. *IEEE Transactions on Communications*, 66(4): 1594-1608, 2018.
- [Dua2014] R. Dua, A. R. Raja, D. Kakadia. Virtualization vs Containerization to Support PaaS. In *IEEE International Conference on Cloud Engineering Proceedings*, 610–614, 2014.
- [Duan2015] X. Duan, X. Wang. Authentication handover and privacy protection in 5G hetnets using software-defined networking. In *IEEE Communications Magazine*, 53(4), 28-35, 2015.
- [Dube2019] P. Dube, T. Suk T, C. Wang. AI gauge: Run-time estimation for deep learning in the cloud. In: *31st International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2019*, IEEE, 160–167. 2019.
- [Dwork2006] C. Dwork, et al. Our data, ourselves: Privacy via distributed noise generation. *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, Berlin, Heidelberg, 2006.
- [Dwork2014] C. Dwork, A. Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science* 9.3-4: 211-407, 2014.
- [Elgamal2018] T. Elgamal, A. Sandur, K. Nahrstedt, G. Agha. Costless: Optimizing Cost of Serverless Computing through Function Fusion and Placement. In *IEEE/ACM Symposium on Edge Computing (SEC)*, 2018.
- [Elrabaa2019] M. E. S. Elrabaa et al. Secure Computing Enclaves Using FPGAs. In: *IEEE Transactions on Dependable and Secure Computing*. 2019.
- [Elsken2019-2] T. Elsken, J.H. Metzen, F. Hutter. Neural Architecture Search: A Survey. *ArXiv, abs/1808.05377*. 2019.
- [Elsken2019] T. Elsken, J. H. Metzen, F. Hutter. Neural Architecture Search. In: *Automated Machine Learning: Methods, Systems, Challenges*, 2019.
- [Eshratifar2021] A. Eshratifar, M. Abrishami, M. Pedram. Jointdnn: An efficient training and inference engine for intelligent mobile cloud computing services. *IEEE Transactions on Mobile Computing*, 20(2):565–576, 2021.
- [EUCALYPTUS] Eucalyptus: <https://www.eucalyptus.cloud/>.

- [EuropeanCommission2020] European Commission, WHITE PAPER On Artificial Intelligence - A European approach to excellence and trust, 19 February 2020, COM(2020) 65 final, https://ec.europa.eu/info/sites/default/files/commission-white-paper-artificial-intelligence-feb2020_en.pdf
- [Filippini2021a] F. Filippini et al. Andreas: Artificial intelligence training scheduler for accelerated resource clusters. ArXiv abs/2105.05080. 2021.
- [Filippini2021b] F. Filippini et al. Artificial intelligence training scheduler for accelerated resource clusters. ArXiv abs/2105.05080. 2021.
- [Fink2011] R.A. Fink, A.T. Sherman, A.O. Mitchell, D. Challener. Catching the cuckoo: Verifying tpm proximity using a quote timing side-channel. In International Conference on Trust and Trustworthy Computing. 2011.
- [FireCracker] FireCracker. <https://firecracker-microvm.github.io/>
- [Flich2020] J. Flich, C. Hernandez, E. Quiñones, R. Paredes. Distributed Training on a Highly Heterogeneous HPC System. In: Embedded Computer Systems: Architectures, Modeling, and Simulation. SAMOS 2020.
- [Foerster2018] K. T. Foerster et al. Survey of Consistent Software-Defined Network Updates. IEEE Communications Surveys and Tutorials, 2018.
- [Gan2021] Y. Gan, Y. Zhang, K. Hu, D. Cheng, Y. He, M. Pancholi, C. Delimitrou. Leveraging Deep Learning to Improve Performance Predictability in Cloud Microservices with Seer. ACM SIGOPS Oper. Syst. Rev. 53(1): 34-39. 2019.
- [GBD 2016] The GBD 2016 Lifetime Risk of Stroke Collaborators. Global, Regional, and Country-Specific Lifetime Risks of Stroke, 1990 and 2016. N Engl J Med, 379: 2429–37. 2018.
- [Gianniti2018] E. Gianniti, L. Zhang, D. Ardagna. Performance prediction of GPU-based deep learning applications. In: 30th Int'l Symp. Computer Architecture and High Performance Computing (SBAC- PAD 18). 2018.
- [Gianniti2019] E. Gianniti, L. Zhang, D. Ardagna. Performance prediction of gpu-based deep learning applications. In: Proceedings of the 9th International Conference on Cloud Computing and Services Science, CLOSER 2019, 279–286. 2019.
- [Gias2020] A. Gias, G. Casale. COCOA: Cold Start Aware Capacity Planning for Function-as-a-Service Platforms. 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), 2020.
- [Giunta2010] G. Giunta, R. Montella, G. Agrillo, G. Coviello. A GPGPU Transparent Virtualization Component for High Performance Computing Clouds. In Proceedings of the Euro-Par Parallel Processing, Euro-Par, 2010.
- [Goodfellow2014] I.J. Goodfellow, et al. Generative adversarial networks. arXiv preprint arXiv:1406.2661, 2014.
- [Goodfellow2016] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.
- [GOOGLEIoTCloud] Google IoT Cloud. <https://cloud.google.com/solutions/iot/>
- [Gousev2020] E. Gousev. Tiny Machine Learning, Applied Machine Learning for Embedded IoT Devices Overview. 2020.
- [Grohmann2019] J. Grohmann, P. K. Nicholson, J. Omana Iglesias, S. Kounev, D. Lugones. Monitorless: Predicting Performance Degradation in Cloud Applications with Machine Learning. Middleware 2019. 2019.
- [Grohmann2021] J. Grohmann, M. Straesser, A. Chalbani, S. Eismann, Y. Arian, N. Herbst, N. Peretz, S. Kounev. SuanMing: Explainable Prediction of Performance Degradations in Microservice Applications. ICPE 2021.
- [Gu2019] J. Gu, et al. Tiresias: A GPU cluster manager for distributed deep learning. In: 16th USENIX NSDI Symposium, 2019.
- [Guo2018] S. Guo, D. Wu, H. Zhang, D. Yuan. Resource Modeling and Scheduling for Mobile Edge Computing: A Service Provider's Perspective. IEEE Access, 6: 35611-35623, 2018.
- [Guo2020] Z. Guo et al. Single Path One-Shot Neural Architecture Search with Uniform Sampling. arXiv: 1904.00420 [cs.CV]. 2020.
- [Gupta2009] V. Gupta, A. Gavrilovska, K. Schwan, H. Kharche, N. Tolia, V. Talwar, P. Ranganathan. GVIM: GPU-accelerated virtual machines. In Proceedings of the ACM Workshop on System-level Virtualization for High Performance Computing, HPCVirt, 2009.
- [Gupta2018] U. Gupta, M. Babu, R. Ayoub, M. Kishinevsky, M. Paterna, S. Gumussoy, U.Y. Ogras. An misc learning methodology for performance modeling of graphics processors. IEEE Transactions on Computer. 2018.
- [Hadjis2016] S. Hadjis, C. Zhang, I. Mitliagkas, C. Ré. Omnivore: An optimizer for multi-device deep learning on CPUs and GPUs. arXiv:1606.04487. 2016.

- [Hassibi1992] B. Hassibi, D. Stork. Second Order Derivatives for Network Pruning: Optimal Brain Surgeon. In: NIPS. 1992.
- [He2019] Z. He, T. Zhang, R. B. Lee. Model inversion attacks against collaborative inference. Proceedings of the 35th Annual Computer Security Applications Conference. 2019.
- [HEARTLINE2020] US National Library of Medicine. ClinicalTrials.gov, <https://clinicaltrials.gov/ct2/show/NCT04276441>, 2020.
- [HEAT] OpenStack Heat. <https://docs.openstack.org/heat/>
- [Hitaj2017] B. Hitaj, G. Ateniese, and F. Perez-Cruz. Deep models under the GAN: information leakage from collaborative deep learning. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017.
- [Hitaj2017] B. Hitaj, G. Ateniese, F. Perez-Cruz. Deep models under the GAN: information leakage from collaborative deep learning. Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 2017.
- [Hochreiter1997] S. Hochreiter, J. Schmidhuber. Long short-term memory. Neural computation, 9(8), 1735–1780, 1997.
- [Hong2019] C. Hong, B. Varghese. Resource Management in Fog/Edge Computing: A Survey on Architectures, Infrastructure, and Algorithms. ACM Computing Surveys, 52(5): 97(1-37), 2019.
- [Hu2018] J. Hu, L. Shen, G. Sun, Squeeze-and-excitation networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 7132–7141, 2018.
- [Huang2020] Y. Huang, X. Qiao, P. Ren, L. Liu, et al. A lightweight collaborative deep neural network for the mobile web in edge cloud. IEEE Transactions on Mobile Computing, 2020, DOI: 10.1109/TMC.2020.3043051.
- [Hunt2018] T. Hunt, et al. Chiron: Privacy-preserving machine learning as a service. arXiv preprint arXiv:1803.05961, 2018.
- [ICInsight2020] IC Insight. 2020. <https://www.icinsights.com/news/bulletins/MCUs-Expected-To-Make-Modest-Comeback-After-2020-Drop--/>
- [IEEEGuide2021] IEEE Guide for Architectural Framework and Application of Federated Machine Learning. In: IEEE Std 3652.1-2020, pp. 1–69, 2021.
- [IEEEstandard2018]. IEEE 1934-2018 - IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing, <https://standards.ieee.org/standard/1934-2018.html>
- [Ignatov2018] A. Ignatov, R. Timofte, W. Chou, K. Wang, M. Wu, T. Hartley, L. Van Gool. AI benchmark: Running deep neural networks on android smartphones. In Proceedings of the European Conference on Computer Vision (ECCV). 2018.
- [IM] IM: Infrastructure Manager: <https://www.grycap.upv.es/im/index.php>
- [Jaderberg2014] M. Jaderberg, A. Vedaldi, A. Zisserman. Speeding up Convolutional Neural Networks with Low Rank Expansions. arXiv: 1405.3866 [cs.CV], 2014.
- [Jayaram2019] K.R. Jayaram, et al. Ffdl. Proceedings of the 20th International Middleware Conference. 2019.
- [Jeon2019] M. Jeon, et al. Analysis of large-scale multi-tenant GPU clusters for DNN training workloads. In: 2019 USENIX ATC. 2019.
- [Jia2012] W. Jia, K.A. Shaw, M. Martonosi. Stargazer: Automated regression-based GPU design space exploration. In: Int’l Symp. Performance Analysis of Systems & Software (ISPASS 12). 2012.
- [Jin2018] X. Jin, X. Li, H. Zhang, N. Foster, J. Lee, R. Soulé, C. Kim, I. Stoica. Netchain: Scale-free sub-rtt coordination. In 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18), 35–49, 2018.
- [Jordon2018] J. Jordon, J. Yoon, M.V.D Schaar. PATE-GAN: Generating synthetic data with differential privacy guarantees. International Conference on Learning Representations. 2018.
- [Kang2017] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, et al. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. In ACM ASPLOS ’17, 2017.
- [Kang2017] Y. Kang, W. Joo, et al. Priority-driven spatial resource sharing scheduling for embedded graphics processing units. JSA, vol. 76, pp. 17–27, 2017.
- [Kantola2016] R. Kantola, J. Llorente Santos, N. Beijar. Policy-based communications for 5G mobile with customer edge switching. Sec. and Commun. Netw. 9(16), 2016.
- [Karande2017] V. Karande, E. Bauman, Z. Lin, L. Khan. Sgx-log: Securing system logs with sgx. In Proceedings of the ACM on Asia Conference on Computer and Communications Security, 2017.

- [Kelbert2017] F. Kelbert, F. Gregor, R. Pires, S. Köpsell, M. Pasin, A. Havet, V. Schiavoni, P. Felber, C. Fetzer, P. Pietzuch. Securecloud: Secure big data processing in untrusted clouds. Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2017.
- [Kerr2010] A. Kerr, G. Damos, S. Yalamanchili. Modeling GPU-CPU workloads and systems. In: Proc. 3rd Workshop General-Purpose Computation on Graphics Processing Units (GPGPU-3), ACM. 2010.
- [Keylime-Github2021] Keylime Github. <https://github.com/keylime/keylime>
- [Keylime2021] Keylime Website. <https://keylime.dev/>
- [Kim2012] J. Kim, S. Seo, J. Lee, J. Nah, G. Jo, J. Lee. SnuCL: An OpenCL Framework for Heterogeneous CPU/GPU Clusters. In Proceedings of the 26th ACM International Conference on Supercomputing, ICS '12, 2012.
- [Kingma2013] D. Kingma, M. Welling. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114, 2013.
- [Kocher2019] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, Y. Yarom. Spectre Attacks: Exploiting Speculative Execution. In 40th IEEE Symposium on Security and Privacy (S&P'19). 2019.
- [Krahn2020] R. Krahn, D. Dragoti, F. Gregor, D.L. Quoc, V. Schiavoni, P. Felber, C. Souza, A. Brito, C. Fetzer. TEEMon: A continuous performance monitoring framework for TEEs. Proceedings of the 21st International Middleware Conference. 2020.
- [Krivoshei2017] L. Krivoshei, S. Weber, T. Burkard, et al. Smart detection of atrial fibrillation. *Europace*. 2017;19:753–757. 2017.
- [Kuehn2016] BM. Kuehn. Telemedicine Helps Cardiologists Extend Their Reach. *Circulation*, 134:1189–91, 2016.
- [Kurtser2020] P. Kurtser, et al. In-field grape cluster size assessment for vine yield estimation using a mobile robot and a consumer level RGB-D camera. *IEEE Robotics and Automation Letters*, 5.2 (2020): 2031–2038, 2020.
- [Kuzminykh2019] I. Kuzminykh. Analysis of Security of Rootkit Detection Methods. In 2019 IEEE International Conference on Advanced Trends in Information Theory. 2019.
- [Lane2016] N. Lane, S. Bhattacharya, A. Mathur, C. Forlivesi, F. Kawsar. DXTK: Enabling Resource-efficient Deep Learning on Mobile and Embedded Devices with the DeepX Toolkit. In: *MobiCASE*. 2016.
- [LeCun1989] Y. LeCun, J. Denker, S. Solla. Optimal Brain Damage. In: *NIPS*. 1989.
- [Li2018] E. Li, Z. Zhou, X. Chen. Edge intelligence: On-demand deep learning model co-inference with device-edge synergy. In Proceedings of the 2018 Workshop on Mobile Edge Communications, ACM. 2018.
- [Li2018] Q. Li et al. Security Policy Violations in SDN Data Plane. *IEEE/ACM Trans. on Networking*, 99, 1–13, 2018.
- [Li2020] E. Li, L. Zeng, Z. Zhou, X. Chen. Edge AI: On-demand accelerating deep neural network inference via edge computing. *IEEE Transactions on Wireless Communications*, 19(1):447–457, 2020.
- [Liang2011] T. Y. Liang, Y. W. Chang. GridCuda: A Grid-Enabled CUDA Programming Toolkit. In Proceedings of the IEEE Advanced Information Networking and Applications Workshops, WAINA, 2011.
- [Libgit] <https://libgit2.org/>
- [Lin2018] Y.D. Lin, Y.C. Lai, J.X. Huang, H.T. Chien. Three-Tier Capacity and Traffic Allocation for Core, Edges, and Devices for Mobile Edge Computing. *IEEE Transactions on Network and Service Management*, 15(3): 923–933, 2018.
- [Lin2021] C. Lin, H. Khazaei. Modeling and Optimization of Performance and Cost of Serverless Applications. *IEEE Transactions on Parallel and Distributed Systems*, 32(3): 615–632, 2021.
- [Lipp2018] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, et al. Meltdown: Reading Kernel Memory from User Space. In 27th USENIX Security Symposium (USENIX Security 18). 2018.
- [Liu2007] W. Liu, W. Muller-Wittig, B. Schmidt. Performance predictions for general-purpose computation on GPUs. In: *Int'l Conf. Parallel Processing (ICPP 07)*, IEEE. 2007.
- [Liu2019] H. Liu, K. Simonyan, Y. Yang. DARTS: Differentiable Architecture Search. arXiv: 1806.09055 [cs.LG]. 2019.
- [Liu2020] D. Liu, X. Chen, Z. Zhou, Q. Ling. Hiertrain: Fast hierarchical edge AI learning with hybrid parallelism in mobile-edge-cloud computing. *IEEE Open Journal of the Communications Society*, 1:634–645, 2020.
- [Lordan2014] F. Lordan, E. Tejedor, J. Ejarque, R. Rafanell, J. Álvarez, F. Marozzo, D. Lezzi, R. Sirvent, D. Talia, R. M. Badia. ServiceSs: an interoperable programming framework for the Cloud. *Journal of Grid Computing*. 12(1), 67–91. 2014.
- [Lu2017] Z. Lu, S. Rallapalli, K. Chan, T. La Porta. Modeling the resource requirements of convolutional neural networks on mobile devices. In: *Proc. Conf. Multimedia (MM 17)*. 2017.

- [Madougou2016] S. Madougou, A. Varbanescu, C. de Laat, et al. The landscape of GPGPU performance modeling tools. *J Parallel Computing* 56:18– 33. 2016.
- [Madougou2016] S. Madougou, A. Varbanescu, et al. The landscape of GPGPU performance modeling tools. *PC*, vol. 56, pp. 18–33, 2016.
- [Mahmoudi2020a] N. Mahmoudi, H. Khazaei. Temporal Performance Modelling of Serverless Computing Platforms. Sixth International Workshop on Serverless Computing (WoSC'20). 2020.
- [Mahmoudi2020b] N. Mahmoudi, H. Khazaei. Performance Modeling of Serverless Computing Platforms. *IEEE Transactions on Network and Service Management*, 2020, DOI: 10.1109/TCC.2020.3033373.
- [Mammela2016] M. Olli, J. Hiltunen, J. Suomalainen, K. Ahola, P. Mannersalo J. Vehkaperä. Towards Micro-Segmentation in 5G Network Security. *EUCNC 2016*.
- [Matetic2017] S. Matetic, M. Ahmed, K. Kostianen, A. Dhar, D. Sommer, A. Gervais, A. Juels, S. Capkun. Rote: Rollback protection for trusted execution. In *Proceedings of the 26th USENIX Conference on Security Symposium (SEC '17)*. 2017.
- [McCune2008] J.M. McCune, B.J. Parno, A. Perrig, M.K. Reiter, H. Isozaki. Flicker: An Execution Infrastructure for Tcb Minimization. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems (EuroSys)*, 2008.
- [McMahan2017] Brendan McMahan and Daniel Ramage. Federated Learning: Collaborative Machine Learning without Centralized Training Data. 2017. <https://ai.googleblog.com/2017/04/federated-learning-collaborative.html>
- [McManus2013] DD. McManus, J. Lee, O. Maitas, et al. A novel application for the detection of an irregular pulse using an iPhone 4 S in patients with atrial fibrillation. *Heart Rhythm*, 10:315–319, 2013.
- [Mendoza2021] D. Mendoza, F. Romero, Q. Li, N. J. Yadwadkar, C. Kozyrakis. Interference-Aware Scheduling for Inference Serving. *1st EuroMLSys@EuroSys 2021*, 80-88, 2021.
- [Meurisch2021] C. Meurisch, M. Mühlhäuser. Data Protection in AI Services: A Survey. *ACM Computing Surveys (CSUR)* 54.2: 1-38, 2021.
- [Miao2017] R. Miao, H. Zeng, C. Kim, J. Lee, M. Yu. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017.
- [Microsemi2013] Microsemi. Overview of Secure Boot with Microsemi SmartFusion2 FPGAs. https://www.microsemi.com/document-portal/doc_download/132863-overview-of-secure-boot-with-microsemi-igloo2-fpgas. 2013.
- [MIKADO] MiCADOscale: <https://micado-scale.eu/>
- [Mironov2017] I. Mironov. Rényi differential privacy. 2017 IEEE 30th Computer Security Foundations Symposium (CSF). IEEE, 2017.
- [Mitchell1997] T. M. Mitchell, *Machine Learning*, McGraw-Hill. 1997.
- [Möller2014] B. Möller, T. Duong, K. Kotowicz. This POODLE bites: exploiting the SSL 3.0 fallback. *Security Advisory*, 2014.
- [Morales2020] M. Morales, J. Rydning. AI and the Edge Are Driving Adoption of Innovative Enabling Technologies in the ComputeSphere and StorageSphere, IDC. <https://www.idc.com/getdoc.jsp?containerId=US46790120&pageType=PRINTFRIENDLY>
- [Mouradian2019] C. Mouradian, S. Kianpisheh, M. Abu-Lebdeh, F. Ebrahimnezhad, et al. Application component placement in NFV-based hybrid cloud/fog systems with mobile fog nodes. *IEEE Journal on Selected Areas in Communications*, 37(5):1130–1143, 2019.
- [MSlotEdge] Azure IoT-Edge. <https://azure.microsoft.com/en-us/services/iot-edge/>
- [Mugunthan2021] V. Mugunthan, et al. DPD-InfoGAN: Differentially Private Distributed InfoGAN. *Proceedings of the 1st Workshop on Machine Learning and Systems*. 2021.
- [Nayman2021] N. Nayman et al. HardCoRe-NAS: Hard Constrained differentiable Neural Architecture Search. arXiv: 2102.11646 [cs.LG]. 2021
- [Ning2018] Z. Ning, et al. Preliminary study of trusted execution environments on heterogeneous edge platforms. 2018 IEEE/ACM Symposium on Edge Computing (SEC). IEEE, 2018.
- [NVIDIA-MIG] NVIDIA Multi-Instance GPU User Guide. <https://docs.nvidia.com/datacenter/tesla/mig-user-guide/index.html>

- [NVIDIA-vGPU] NVIDIA Virtual GPU Software User Guide. <https://docs.nvidia.com/grid/latest/grid-vgpu-user-guide/index.html>
- [OCCOPUS] Occopus. <https://occopus.readthedocs.io/en/latest/index.html>.
- [ODonnell2010] MJ. O'Donnell, D. Xavier, L. Liu, H. Zhang, SL. Chin, P. Rao-Melacini, et al. Risk factors for ischaemic and intracerebral haemorrhagic stroke in 22 countries (the INTERSTROKE study): a case-control study, 376:112–23, 2010.
- [ONEFLOW] OpenNebula OneFlow. https://docs.opennebula.io/6.0/installation_and_configuration/opennebula_services/oneflow.html.
- [OpenFog2017] OpenFog Consortium Architecture Working Group, OpenFog Reference Architecture for Fog Computing, February 2017.
- [OPENTOSCA] OpenTOSCA. <http://www.opentosca.org/>
- [Opera2007] A. Oprea, M. K. Reiter. Integrity checking in cryptographic file systems with constant trusted storage. In USENIX Security Symposium, 2007.
- [OscaAsensi2020] J. Osca Asensi, et al. The RITHMI study: diagnostic ability of a heart rhythm monitor for automatic detection of atrial fibrillation. *Rev Esp Cardiol (Engl Ed)*. 10:S1885-5857(20)30309-1, 2020.
- [OSCAR] OSCAR (Open Source Serverless Computing for Data-Processing Applications). <https://github.com/grycap/oscar>
- [OSCAR] OSCAR. <https://github.com/grycap/oscar>
- [Papernot2016] N. Papernot, et al. Semi-supervised knowledge transfer for deep learning from private training data. *arXiv preprint arXiv:1610.05755*, 2016.
- [Parno2008] B. Parno. Bootstrapping Trust in a "Trusted" Platform. In Proceedings of the 3rd Conference on Hot Topics in Security. 2008.
- [Peng2018] Y. Peng, Y. Bao, et al. Optimus: An efficient dynamic resource scheduler for deep learning clusters. In EuroSys, 2018.
- [Peng2018] Y. Peng, Y. Bao, Y. Chen, C. Wu, C. Guo. Optimus: An efficient dynamic resource scheduler for deep learning clusters. In EuroSys 2018 proceedings, 2018.
- [Peng2018] Y. Peng, Y. Bao, Y. Chen, C. Wu, C. Guo. Optimus: an efficient dynamic resource scheduler for deep learning clusters. In: Proceedings of the Thirteenth EuroSys Conference, 3:1–3:14. 2018.
- [Peng2019] Y. Peng, Y. Bao, et al. DL2: A deep learning driven scheduler for deep learning clusters. *ArXiv*, vol. abs/1909.06040, 2019.
- [Pérez2018] A. Pérez, G. Moltó, M. Caballer, A. Calatrava. Serverless computing for container-based architectures. *Future Generation Computer Systems*, 83: 50–59. <https://doi.org/10.1016/j.future.2018.01.022>. 2018.
- [Pérez2019] A. Perez, S. Risco, D.M. Naranjo, M. Caballer, G. Molto, 2019. On-Premises Serverless Computing for Event-Driven Data Processing Applications, in: 2019 IEEE 12th International Conference on Cloud Computing (CLOUD). IEEE, pp. 414–421, 2019. <https://doi.org/10.1109/CLOUD.2019.00073>.
- [PerezApple2019] M.V. Perez, et al. Large-scale assessment of a smartwatch to identify atrial fibrillation. *N. Engl. J. Med.* 381, 1909–1917, 2019.
- [Pericas2020] M. Pericàs, T.U.D., Do Le Quoc. REPORT ON EVALUATION AND OPTIMIZATIONS IN THE RUNTIME STACK.
- [Peterson2007] Z. N. Peterson, R. C. Burns, G. Ateniese, S. Bono. Design and implementation of verifiable audit trails for a versioning file system. In Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST), 2007.
- [Petrucci2017] L. Petrucci, M. Bonola, S. Pontarelli, G. Bianchi, R. Bifulco. Implementing iptables using a programmable stateful data plane abstraction. In Proceedings of the Symposium on SDN Research. ACM, 2017.
- [Pinto2017] S. Pinto, et al. IloTEED: an enhanced, trusted execution environment for industrial IoT edge devices. *IEEE Internet Computing* 21(1): 40-47, 2017.
- [Priebe2018] C. Priebe, K. Vaswani, M. Costa. EnclaveDB: A secure database using SGX. 2018 IEEE Symposium on Security and Privacy (SP). IEEE, 2018.
- [Profentzas2019] C. Profentzas. Performance of Secure Boot in Embedded Systems. In 15th International Conference on Distributed Computing in Sensor Systems. 2019.

- [Pronto2020] N. Foster, N. McKeown, J. Rexford, G. Parulkar, L. Peterson, O. Sunay. Using deep programmability to put network owners in control. *SIGCOMM Comput. Commun. Rev.* 50, 4. 2020.
- [Puthal2019] D. Puthal, R. Ranjan, A. Nanda, P. Nanda. Secure authentication and load balancing of distributed edge data centers. *J. Parallel Distributed Comput.* 124: 60-69, 2019.
- [Rahman2018] M.A. Rahman, et al. Membership Inference Attack against Differentially Private Deep Learning Model. *Trans. Data Priv.* 11.1: 61-79, 2018.
- [Ramachandran2019] U. Ramachandran, H. Gupta, A. Hall, E. Saurez, Z. Xu. Elevating the Edge to Be a Peer of the Cloud. 2019 IEEE 12th International Conference on Cloud Computing (CLOUD):17-24, 2019.
- [Rashmi2018] R. V. Rashmi, A. Karthikeyan. Secure boot of embedded Applications - A Review. In *Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*. 2018.
- [Real2019] E. Real et al. Regularized Evolution for Image Classifier Architecture Search. arXiv: 1802.01548 [cs.NE]. 2019.
- [Reaño2015] C. Reaño, F. Silla, G. Shainer, S. Schultz. Local and Remote GPUs Perform Similar with EDR 100G InfiniBand. In *Proceedings of the Industrial Track of the 16th International Middleware Conference, Middleware Industry '15*, 2015.
- [Reaño2017] C. Reaño, F. Silla, and J. Duato. Enhancing the rCUDA Remote GPU Virtualization Framework: From a Prototype to a Production Solution. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid '17*, pages 695–698. IEEE Press, 2017.
- [Reaño2017] C. Reaño, F. Silla, D.S. Nikolopoulos, B. Varghese. Intra-node memory safe gpu co-scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 29(5), 1089-1102, 2017.
- [Risco2021] S. Risco, G. Moltó. GPU-Enabled Serverless Workflows for Efficient Multimedia Processing. *Applied Sciences*, 11(4):1438, 2021. <https://doi.org/10.3390/app11041438>.
- [Roberti2020] G. Roberti, A. Buss. Key Developments in the European Edge Market in 2020, IDC. <https://www.idc.com/getdoc.jsp?containerId=EUR146468520>
- [Roman2018] R. Roman, J. Lopez, M. Mambo. Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenge. *Future Generation Computer Systems*, 78(2), 2018.
- [Rumelhart1986] D. Rumelhart, G. Hinton, and R. Williams. Learning representations by back-propagating errors. *Nature*, 323: 533–536, 1986.
- [Sabt2015] M. Sabt, et al. Trusted execution environment: what it is, and what it is not. 2015 IEEE Trustcom/BigDataSE/ISPA. 2015.
- [Samson2021] A. Samson. Imagerie et intelligence artificielle au service de la détection des maladies, où en est-on? www.genie.aladin.farm. 2021.
- [Sandler2018] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.
- [Santos2020] T. Santos, et al. Grape detection, segmentation, and tracking using deep neural networks and three-dimensional association. *Computers and Electronics in Agriculture* 170: 105247, 2020.
- [Santoyo2018] A. Santoyo-González, C. Cervelló-Pastor. Latency-aware cost optimization of the service infrastructure placement in 5G networks. *Journal of Network and Computer Applications*, 114: 29-37, 2018.
- [Sapio2017] A. Sapio, I. Abdelaziz, A. Aldilajjan, M. Canini, P. Kalnis. Innetwork computation is a dumb idea whose time has come. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*. ACM, 2017.
- [Sarmah2017] M. Sarmah. Methods for Booting an All Programmable System-on-Chip over PCI Express Link. In *IEEE International Conference on Computational Intelligence and Computing Research*. 2017.
- [Sau2017] S. Sau et al. Survey of Secure Processors. In *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. 2017.
- [Saxena2020] V. Saxena, K. R. Jayaram, et al. Effective elastic scaling of deep learning workloads. arXiv: 2006.13878, 2020.
- [Saxena2020] V. Saxena, K.R. Jayaram, S. Basu, Y. Sabharwal, A. Verma. Effective elastic scaling of deep learning workloads. In *MASCOTS 2020 proceedings*. 2020.
- [SCAR] SCAR (Serverless Container-aware ARchitectures). <https://github.com/grycap/scar>
- [SCARCNCf] SCAR Entry in the CNCf's Serverless Landscape. <https://landscape.cncf.io/serverless?selected=scar>

- [SCARDEEP] Event-Driven Execution of DEEP Open Catalog Modules for Prediction on Amazon Web Services. <https://deep-hybrid-datacloud.eu/2020/03/25/event-driven-execution-of-deep-open-catalog-modules-for-prediction-on-amazon-web-services/>
- [Schaik2020] S.V Schaik, A. Kwong, D. Genkin, Y. Yarom. SGAXe: How SGX fails in practice, 2020.
- [Schuster2015] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, M. Russinovich. VC3: Trustworthy data analytics in the cloud using SGX. 2015 IEEE Symposium on Security and Privacy. IEEE, 2015.
- [Shahrad2020] M. Shahrad, R. Fonseca, I. Goiri, et al. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a LargeCloud Provider. In USENIX Annual Technical Conference, 2020.
- [Sheikhalishahi2016] M. Sheikhalishahi, R. M. Wallace, et al. A multidimensional job scheduling. FGCS, vol. 54, 2016.
- [Shi2009] L. Shi, H. Chen, J. Sun. vCUDA: GPU accelerated high performance computing in virtual machines. In Proceedings of the IEEE Parallel and Distributed Processing Symposium, IPDPS, 2009.
- [Shiand2019] W. Shiand, Y. Hou, S. Zhou, Z. Niu, et al. Improving Device-Edge Cooperative Inference of Deep Learning via 2-Step Pruning. In IEEE INFOCOM 2019 Proceedings. 2019.
- [Shokri2017] R. Shokri, et al. Membership inference attacks against machine learning models. 2017 IEEE Symposium on Security and Privacy (SP). IEEE, 2017.
- [Siddiqui2019] A. S. Siddiqui et al. Multilayer camouflaged secure boot for SoCs. In 20th International Workshop on Microprocessor/SoC Test, Security and Verification. 2019.
- [Simonyan2014] K. Simonyan, A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv 1409.1556. 2014.
- [Skarlat2017a] O. Skarlat, M. Nardelli, S. Schulte, S. Dustdar, 2017. Towards QoS-Aware Fog Service Placement, in 2017 IEEE International Conference on Fog and Edge Computing (ICFEC), 2017.
- [Skarlat2017b] O. Skarlat, M. Nardelli, S. Schulte, S. Stefan, et al. Optimized IoT Service Placement in the Fog. Serv. Oriented Comput. Appl, 11(4):427–443, 2017.
- [Smith2021] R. Smith. Managing EFI Boot Loaders for Linux: Dealing with Secure Boot. <https://www.rodsbooks.com/efi-bootloaders/secureboot.html>. 2021.
- [Song2013] S. Song, C. Su C, R. Rountree, K.W. Cameron. A simplified and accurate model of power- performance efficiency on emergent GPU architectures. In: 27th Int’l Symp. Parallel and Distributed Processing (IPDPS 13), IEEE. 2013.
- [Souza2018] V. B. Souza, X. Masip-Bruin, E. Marín-Tordera, S. Sánchez-López, et al. Towards a proper service placement in combined Fog-to-Cloud (F2C) architectures. Future Generation Computer Systems, 87: 1-15, 2018.
- [Steinhubl2018] S.R Steinhubl, et al. Effect of a home-based wearable continuous ECG monitoring patch on detection of undiagnosed atrial fibrillation. JAMA 320, 146, 2018.
- [Stojmenovic2016] I. Stojmenovic, S. Wen, X. Huang, H. Luan. An overview of fog computing and its security issues, Concurr. Comput. Pract. Exper. 28 (10), 2991–3005, 2016.
- [Suo2013] H. Suo, Z. Liu, J. Wan, K. Zhou. Security and privacy in mobile cloud computing. In Proceedings of the 9th International Wireless Communications and Mobile Computing Conference, IWCMC, 2013.
- [Sutton1998] R. S. Sutton, A. G. Barto, Reinforcement Learning: An Introduction, MIT Press, 1998.
- [Szegedy2015] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E.Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich. Going deeper with convolutions. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1–9, 2015.
- [Tadakamalla2021] U. Tadakamalla, D. A. Menasce. Autonomic Resource Management for Fog Computing. In IEEE Transactions on Cloud Computing, doi: 10.1109/TCC.2021.3064629, 2021.
- [Takahashi2020] T. Takahashi, et al. Differentially Private Variational Autoencoders with Term-wise Gradient Aggregation. arXiv preprint arXiv:2006.11204, 2020.
- [Taneja2017] M. Taneja, A. Davy. Resource aware placement of IoT application modules in Fog-Cloud computing paradigm. In Proceedings of the IFIP/IEEE Symposium on Integrated Network and Service Management (IM’17), p 1222–1228, 2017.
- [Tantipongpipat2019] U. Tantipongpipat, et al. Differentially private mixed-type data generation for unsupervised learning. arXiv preprint arXiv:1912.03250, 2019.
- [Teerapittayanon2017] S. Teerapittayanon, B. McDanel, H. Kung. Distributed Deep Neural Networks Over the Cloud, the Edge and End Devices. In IEEE 37th ICDCS, 2017.

- [Tianshu2019] H. Tianshu, et al. Edge AIBench: towards comprehensive end-to-end edge computing benchmarking. International Symposium on Benchmarking, Measuring and Optimization. Springer, Cham, 2019.
- [Tomarchio2021] O. Tomarchio, D. Calcaterra, G. Di Modica, et al. TORCH: a TOSCA-Based Orchestrator of Multi-Cloud Containerised Applications. *J Grid Computing*, 19(5), 2021. <https://doi.org/10.1007/s10723-021-09549-z>
- [Torkzadehmahani2019] R. Torkzadehmahani, P. Kairouz, B. Paten. Dp-cgan: Differentially private synthetic data and label generation. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops. 2019.
- [TOSCA] OASIS Topology and Orchestration Specification for Cloud. <https://www.oasis-open.org/committees/tosca/>.
- [Tu2010] HTH. Tu, BCV. Campbell, S. Christensen, M. Collins, DA. Silva, KS. Butcher, et al. Pathophysiological determinants of worse stroke outcome in atrial fibrillation. *Cerebrovasc Dis*, 30:389–95, 2010.
- [udocker] udocker. <https://github.com/indigo-dc/udocker>
- [Vanhoucke2011] V. Vanhoucke, A. Senior, and Mark Z. Mao. Improving the speed of neural networks on CPUs. In: 2011.
- [VGPU2021] V-GPU: GPU virtualization. https://github.com/zillians/platform_manifest_vgpu
- [Villari2019] M. Villari, M. Fazio, S. Dustdar, O. Rana, D. Jha, R. Ranjan. Osmosis: The osmotic computing platform for microelements in the cloud, edge, and internet of thing. *IEEE Computer*, 52(8):14-26, 2019.
- [VIMONT20218DRONE] L. Vimont. 2018/09/06. Détecter les maladies par drone. www.reussir.fr. 2018
- [VmWare2014] VMware, Data Center Micro-Segmentation: A Software Defined Data Center Approach for a Zero Trust Security Strategy, Tech. Rep., <https://blogs.vmware.com/networkvirtualization/files/2014/06/VMware-SDDC-Micro-Segmentation-White-Paper.pdf>, 2014.
- [Wan2020] A. Wan et al. FBNetV2: Differentiable Neural Architecture Search for Spatial and Channel Dimensions. arXiv: 2004.05565 [cs.CV]. 2020.
- [Wang2017] S. Wang, M. Zafer, K. K.Leung. Online Placement of Multi-Component Applications in Edge Computing Environments. *IEEE Access*, 5: 2514-2533, 2017.
- [Wang2019] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, M. Chen. In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning. *IEEE Network*, 33(5), 156–165, 2019.
- [Wei2021] W. Zheng, et al. A survey of Intel SGX and its applications. *Frontiers of Computer Science* 15(3): 1-15, 2021.
- [Wilkins2013] R. Wilkins, B. Richardson. UEFI SECURE BOOT IN MODERN COMPUTER SECURITY SOLUTIONS. 2013. https://media.kasperskycontenthub.com/wp-content/uploads/sites/63/2014/06/21032725/UEFI_Secure_Boot_in_Modern_Computer_Security_Solutions_2013.pdf.
- [Williams1992] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*, 1992.
- [Wu2013] J. Wu, B. Hong. Collocating cpu-only jobs with gpu-assisted jobs on gpu-assisted hpc. In *CCGrid*, 2013.
- [Wu2021] J. Wu et al. Weak NAS Predictors Are All You Need. 2021. arXiv: 2102.10490 [cs.LG].
- [Xiao2018] W. Xiao, R. Bhardwaj, et al. Gandiva: Introspective cluster scheduling for deep learning. In *USENIX OSDI*, 2018.
- [Xie2018] L. Xie, et al. Differentially private generative adversarial network. *arXiv preprint arXiv:1802.06739*, 2018.
- [Xu2016] Z. Xu, W. Liang, W. Xu, M. Jia, S. Guo. Efficient Algorithms for Capacitated Cloudlet Placements. *IEEE Transactions on Parallel and Distributed Systems*, 27(10):2866-2880, 2016.
- [Xu2017] M. Xu, X. Liu, Yunxin Liu, F. Lin. Accelerating Convolutional Neural Networks for Continuous Mobile Vision via Cache Reuse. In: *ArXiv abs/1712.01670*, 2017.
- [Xu2020] D. Xu, T. Li, Y. Li, X. Su, S. Tarkoma, T. Jiang, J. Crowcroft, P. Hu. Edge intelligence: Architectures, challenges, and applications. 2020, arXiv:2003.12172, 2020. [Online] Available: <https://arxiv.org/abs/2003.12172>
- [Yadwadkar2019] N. Yadwadkar, F. Romero, Q. Li, C. Kozyrakis. A Case for Managed and Model-less Inference Serving. In *HotOS'19*, 2019.
- [Yang2019] Q. Yang, Y. Liu, Ti.Chen, Y. Tong. 2019. Federated Machine Learning: Concept and Applications. *ACM Trans. Intell. Syst. Technol.* 10(2), DOI:[http Ds://doi.org/10.1145/3298981](https://doi.org/10.1145/3298981). 2019.

- [Yang2020] Z. Yang et al. CARS: Continuous Evolution for Efficient Neural Architecture Search. arXiv: 1909.04977 [cs.CV]. 2020.
- [Yi2015] S. Yi, Z. Qin, Q. Li. Security and privacy issues of fog computing: A survey. Lecture Notes in Computer Science, vol. 9204, Springer International Publishing, 685–695, 2015.
- [Yi2017] S. Yi, Z. Hao, Q. Zhang, Qu. Zhang, W. Shi, Q. Li. LAVEA: Latency-Aware Video Analytics on Edge Computing Platform. ACM/IEEE Symposium on Edge Computing (SEC), 2017.
- [Ying2019] C. Ying et al. NAS-Bench-101: Towards Reproducible Neural Architecture Search. arXiv: 1902.09635 [cs.LG]. 2019.
- [YORC] Ystia Orchestrator (Yorc). <https://yorc.readthedocs.io/>.
- [Yosinski2014] J. Yosinski, J. Clune, Y. Bengio, H. Lipson. How Transferable Are Features in Deep Neural Networks? In: Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2. NIPS'14. Montreal, Canada: MIT Press, 3320–3328, 2014.
- [Yun2009] A. Yun, C. Shi, Y. Kim. On protecting integrity and confidentiality of cryptographic file system for outsourced storage. In Proceedings of the ACM Workshop on Cloud Computing Security, 2009.
- [Zhang2015] Y. Zhang, D. Niyato, P. Wang. Offloading in Mobile Cloudlet Systems with Intermittent Connectivity. IEEE Transactions on Mobile Computing, 14(12): 2516-2529, 2015.
- [Zhang2017] H. Zhang, L. Stafman, et al. Smaq: Quality-driven scheduling for distributed machine learning. Ser. SoCC'17, ACM, 390–404, 2017.
- [Zhang2019] P. Zhang et al. Fast Data Plane Testing for Software-Defined Networks With RuleChecker. IEEE/ACM Trans. on Networking, 2019.
- [Zoph2016] B. Zoph, Q. V. Le, Neural Architecture Search with Reinforcement Learning, *CoRR*, *abs/1611.01578*, 2016.
- [Zoph2017] B. Zoph, Q. V. Le. Neural Architecture Search with Reinforcement Learning. arXiv: 1611.01578 [cs.LG]. 2017.