

DRAFT

## D3.3

# Enhanced IoT federated deep learning/ reinforcement ML

**WORKPACKAGE** WP3

**DOCUMENT** D3.3

**REVISION** V1.0

**DELIVERY DATE** 30/09/2022

**PROGRAMME IDENTIFIER** H2020-ICT-2020-1

**GRANT AGREEMENT ID** 957246

**START DATE OF THE PROJECT** 01/10/2020

**DURATION** 3 YEARS

© Copyright by the IoT-NGIN Consortium

This project has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No 957246



## DISCLAIMER

This document does not represent the opinion of the European Commission, and the European Commission is not responsible for any use that might be made of its content.

This document may contain material, which is the copyright of certain IoT-NGIN consortium parties, and may not be reproduced or copied without permission. All IoT-NGIN consortium parties have agreed to full publication of this document. The commercial use of any information contained in this document may require a license from the proprietor of that information.

Neither the IoT-NGIN consortium as a whole, nor a certain party of the IoT-NGIN consortium warrant that the information contained in this document is capable of use, nor that use of the information is free from risk, and does not accept any liability for loss or damage suffered using this information.

## ACKNOWLEDGEMENT

This document is a deliverable of IoT-NGIN project. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 957246.

The opinions expressed in this document reflect only the author's view and in no way reflect the European Commission's opinions. The European Commission is not responsible for any use that may be made of the information it contains.

<b>PROJECT ACRONYM</b>	IoT-NGIN
<b>PROJECT TITLE</b>	Next Generation IoT as part of Next Generation Internet
<b>CALL ID</b>	H2020-ICT-2020-1
<b>CALL NAME</b>	Information and Communication Technologies
<b>TOPIC</b>	ICT-56-2020 - Next Generation Internet of Things
<b>TYPE OF ACTION</b>	Research and Innovation Action
<b>COORDINATOR</b>	Capgemini Technology Services (CAP)
<b>PRINCIPAL CONTRACTORS</b>	Atos Spain S.A. (ATOS), ERICSSON GmbH (EDD), ABB Oy (ABB), NETCOMPANY-INTRASOFT SA (INTRA), Engineering-Ingegneria Informatica SPA (ENG), Robert Bosch Espana Fabrica Aranjuez SA (BOSCHN), ASM Terni SpA (ASM), Forum Virium Helsinki (FVH), ENTERSOFT SA (OPT), eBOS Technologies Ltd (EBOS), Privanova SAS (PRI), Synelxis Solutions S.A. (SYN), CUMUCORE Oy (CMC), Emotion s.r.l. (EMOT), AALTO-Korkeakoulusaatio (AALTO), i2CAT Foundation (I2CAT), Rheinisch-Westfälische Technische Hochschule Aachen (RWTH), Sorbonne Université (SU)
<b>WORKPACKAGE</b>	WP3
<b>DELIVERABLE TYPE</b>	REPORT
<b>DISSEMINATION LEVEL</b>	PUBLIC
<b>DELIVERABLE STATE</b>	FINAL
<b>CONTRACTUAL DATE OF DELIVERY</b>	30/09/2022
<b>ACTUAL DATE OF DELIVERY</b>	04/10/2022
<b>DOCUMENT TITLE</b>	Enhanced IoT federated deep learning/ reinforcement ML
<b>AUTHOR(S)</b>	S. Bourou (SYN), Z. Anastasakis, (SYN), A. Voulkidis (SYN), T. Velivassaki (SYN), J. Mira (ATOS), I. Moreno(ATOS), J. Gorroñoigoitia Cruz (ATOS), H. BARDISBANIAN (CAP)
<b>REVIEWER(S)</b>	J. Gorroñoigoitia Cruz (ATOS), D. Skias (INTRA)
<b>ABSTRACT</b>	SEE EXECUTIVE SUMMARY
<b>HISTORY</b>	SEE DOCUMENT HISTORY
<b>KEYWORDS</b>	Deep Learning, Reinforcement Learning, Online Learning, Federated Learning, Machine Learning, Privacy Preservation, AI, ML

## Document History

Version	Date	Contributor(s)	Description
V0.1	05/07/2022	SYN	Table of Contents
V0.1.1	07/07/2022	SYN	Updates on ToC, based on editors' feedback
V0.1.2	11/07/2022	SYN, ATOS	Minor updates on ToC
V0.2	27/07/2022	SYN	Description of Federated Learning frameworks
V0.3	31/08/2022	SYN, ATOS, CAP	Contributions to MLaaS architecture, Online, Reinforcement Learning and Federated Learning
V0.4	09/09/2022	SYN, ATOS, CAP	Description of MLaaS platform components, Implementations for LLs for the ML techniques
V0.5	16/09/2022	SYN, ATOS, CAP	Contributions to MLaaS architecture, Online, Reinforcement Learning and Federated Learning
V0.6	19/09/2022	SYN	Consolidation of inputs
V0.7	22/09/2022	SYN	Finalization of content; Ready for peer review
V0.7.1	27/09/2022	ATOS	Peer review comments
V0.7.2	29/09/2022	INTRA	Peer review comments
V0.8	04/10/2022	ATOS, CAP, SYN	Addressing peer review comments; Ready for quality check
V0.9	04/10/2022	CAP	Quality check
V1.0	04/10/2022	SYN	Final version

# Table of Contents

Document History .....	4
Table of Contents .....	5
List of Figures.....	7
List of Tables.....	10
List of Acronyms and Abbreviations.....	11
Executive Summary .....	13
1 Introduction.....	14
1.1 Intended Audience.....	14
1.2 Relations to other activities .....	15
1.3 Document overview .....	16
2 IoT-NGIN Machine Learning as a Service Framework.....	18
2.1 Introduction .....	18
2.2 Technical architecture overview .....	18
2.3 MLaaS platform architectural approach.....	21
2.4 MLaaS platform components.....	22
2.4.1 ML Components .....	23
2.4.2 Infrastructure Components.....	26
2.5 MLaaS reference architecture .....	29
2.5.1 Overall architecture .....	29
2.5.2 Access to the MLaaS platform .....	31
2.5.3 End-to-end services examples.....	32
2.6 Kubeflow .....	33
2.6.1 Kubeflow overview .....	33
2.6.2 IoT-NGIN MLaaS Kubeflow test environment.....	35
2.7 KServe.....	36
2.7.1 KServe overview.....	36
2.7.2 IoT-NGIN MLaaS KServe installation.....	38
3 IoT-NGIN Machine Learning Techniques.....	40
3.1 Online Learning Framework .....	40
3.1.1 Description.....	40
3.1.2 Technical Design.....	41
3.1.3 Implementation for IoT-NGIN LLs .....	46
3.2 Reinforcement Learning .....	52

3.2.1	Description.....	52
3.2.2	Technical Design.....	54
3.2.3	Implementation for IoT-NGIN LLs .....	56
4	IoT-NGIN Privacy-Preserving Federated Learning Framework.....	61
4.1	Description.....	61
4.2	Technical Design.....	62
4.2.1	NVIDIA FLARE.....	63
4.2.2	IoT-NGIN FedPATE: Flower & PATE .....	71
4.2.3	TensorFlow Federated.....	76
4.3	Implementation for IoT-NGIN LLs .....	78
4.3.1	NVIDIA FLARE.....	78
4.3.2	IoT-NGIN FedPATE .....	89
4.3.3	TensorFlow Federated.....	94
4.4	Discussion .....	99
5	Installation and User Guide.....	103
5.1	MLaaS Framework .....	103
5.1.1	Installation principles .....	103
5.1.2	Application installation .....	107
5.2	Online Learning Framework .....	110
5.3	Privacy-preserving Federated Learning Framework .....	116
5.3.1	NVIDIA FLARE.....	116
5.3.2	IoT-NGIN FedPATE.....	118
5.3.3	TensorFlow Federated.....	119
6	Conclusions.....	121
7	References.....	122

## List of Figures

Figure 1: Work packages structure.....	15
Figure 2: MLaaS framework technical architecture. ....	19
Figure 3: The Lego approach.....	21
Figure 4: "all-in-one" approach.....	22
Figure 5: MLaaS framework technical architecture. ....	29
Figure 6: MLaaS components interoperability.....	30
Figure 7: Infrastructure components interactions. ....	31
Figure 8: Access to the MLaaS platform.....	32
Figure 9: End-to-end inference pipeline example.....	32
Figure 10: End-to-end inference pipeline example with Kafka.....	33
Figure 11: Kubeflow conceptual overview. ....	34
Figure 12: Kubeflow components.....	35
Figure 13: Kubeflow User Interface.....	35
Figure 14: KServe architecture.....	36
Figure 15: KServe Control Plane.....	37
Figure 16: KServe <i>InferenceService</i> .....	37
Figure 17: KServe <i>InferenceService</i> .....	38
Figure 18: Online Learning Concept.....	40
Figure 19: Online learning architecture.....	42
Figure 20: Model training process.....	43
Figure 21: Model prediction process.....	44
Figure 22: OL service modules. ....	45
Figure 23: Smart Energy LL concept for UC9 and UC10.....	47
Figure 24: Pre-processing stage for Smart Energy LL OL service. ....	49
Figure 25: DL architectures' scheme for the Smart Energy LL. ....	49
Figure 26: Training results for power generation forecasting of UC9.....	51
Figure 27: Training results for power generation forecasting of UC10.....	52
Figure 28: Reinforcement Learning concept.....	53
Figure 29: Reinforcement Learning architecture. ....	55
Figure 30: UC10 Environment: EMOT Network Topology.....	57
Figure 31: RL-based optimization entities in UC10.....	58
Figure 32: States and Actions for ECSs.....	59
Figure 33: High-level architecture of PPFLaaS. ....	62

Figure 34: NVIDIA Flare Controller/Worker interactions.....	64
Figure 35: FL Training Workflow.....	65
Figure 36: NVIDIA FLARE filter for exclude/remove variables. ....	66
Figure 37: NVIDIA FLARE filter for Percentile Privacy. ....	67
Figure 38: NVIDIA FLARE filter for SVT Privacy. ....	67
Figure 39: NVIDIA FLARE filter for Homomorphic Encryption.....	68
Figure 40: NVIDIA FLARE provisioning Tool Workflow.....	69
Figure 41: Provisioning in NVIDIA FLARE by project.yml. ....	70
Figure 42: UI of provisioning tool helper in NV FLARE. ....	71
Figure 43: Overview of the noisy aggregation mechanism in PATE [35].....	72
Figure 44: Approach diagram: an ensemble of teachers is trained on disjoint subsets of the private data and a student model is trained on public data labelled using the ensemble. ....	72
Figure 45: Accuracy is higher throughout training, despite greatly improved privacy [34]. ...	73
Figure 46: Overview of FL-PATE framework [36].....	73
Figure 47: Teacher's and Student's accuracy [36].....	74
Figure 48: Overview of proposed FedPATE framework. ....	75
Figure 49: TensorFlow federated architecture [40]. ....	77
Figure 50: Annotations of PASCAL VOC dataset.....	80
Figure 51: Annotations of the MS Coco dataset.....	81
Figure 52: YOLOv5 annotation format.....	81
Figure 53: (a)Centralized Learning performance (b)FL training performance.....	84
Figure 54: Number of samples with different data Heterogeneity cases.....	85
Figure 55: Performance of the model with data heterogeneity. ....	85
Figure 56: FL with Percentile privacy, the comparison of mAP with various protection levels. ....	86
Figure 57: FL with SVT privacy, the comparison of mAP with various noise insertion levels. ....	88
Figure 58: FL with and without HE. ....	88
Figure 59: FL with Percentile Privacy with HE.....	89
Figure 60: Performance of the student baseline model of FedPATE.....	90
Figure 61: Performance of the aggregated model for each federated round.....	91
Figure 62: Student's test accuracy for 9000 queries, without noise injection. ....	91
Figure 63: Student's test accuracy. ....	93
Figure 64: Data heterogeneity scenarios.....	93
Figure 65: Accuracy of the FL system when applied data heterogeneity.....	94
Figure 66: Proposed architecture of FL in an intrusion detection system. ....	95



Figure 67: Testing accuracy for different number of clients. ....	96
Figure 68: Impact of noise on model's accuracy using 25 clients.....	97
Figure 69: Number of samples per client for different values of $\alpha$ .....	98
Figure 70: Impact of the component $\alpha$ on model's accuracy. ....	98
Figure 71: Testing accuracy for the realistic scenario with 25 clients.....	99
Figure 72: MLaaS installation principles. ....	103
Figure 73: ArgoCD projects. ....	106
Figure 74: ArgoCD application example. ....	107
Figure 75: ArgoCD application example. ....	108
Figure 76: IoT-NGIN MLaaS test applications. ....	109
Figure 77: ArgoCD application for ArgoCD.....	109
Figure 78: Baseline OL model in MinIO storage. ....	110
Figure 79: Kubeflow dashboard. ....	113
Figure 80: Creation of Jupyter's notebooks in Kubeflow interface. ....	113
Figure 81: Jupyter notebook defining a Kubeflow pipeline for deploying the OL service....	114
Figure 82: Kubeflow pipeline execution for deploying the OL service. ....	114

## List of Tables

Table 1: Relation of WP3 activities to other WPs and tasks.....	16
Table 2: IoT-NGIN MLaaS platform ML components.....	23
Table 3: Common Services components.....	26
Table 4: Underlying IoT-NGIN test infrastructure components.....	27
Table 5: installation and support components.....	28
Table 6: Smart Energy LL MQTT topics for forecasting services. ....	48
Table 7: Smart Energy LL training hyper-parameters.....	50
Table 8: Normality test results (p-values) for Smart Energy LL use cases.....	51
Table 9: Summary of AI services and features experimented for the 3 FL frameworks. ....	78
Table 10: Scenarios of centralized and FL training. ....	83
Table 11: Information about batch size and split of dataset.....	83
Table 12: Scenarios of FL training with Percentile privacy. ....	86
Table 13: Scenarios of FL training with SVT privacy .....	87
Table 14: Number of images in the CIFAR-10 dataset.....	89
Table 15: CIFAR10 dataset partitions for student's training. ....	89
Table 16: Lessons learnt from FL frameworks' experimental assessment.....	99
Table 17: Recommended uses of FL in the LL use cases.....	100

## List of Acronyms and Abbreviations

AP	Average Precision
API	Application Programming Interface
BCE	Binary Cross-Entropy
BDVA	Big Data Value Association
CA	Certificate Authority
CNN	Convolutional Neural Network
DL	Deep Learning
DP	Differential Privacy
ECG	Electric Charging Station
EG	Electric Grid
EIP	Enterprise Integration Patterns
EV	Electric Vehicle
FC	Federated Core
FL	Federated Learning
FQDN	Fully-Qualified Domain Name
GAN	Generative Adversarial Network
GRU	Gated Recurrent Unit
HE	Homomorphic encryption
IaC	Infrastructure as Code
IDS	Intrusion Detection System
IoT	Internet of Things
IoU	Intersection over Union
LL	Living Labs
mAP	mean Average Precision
ML	Machine Learning
MLaaS	Machine Learning as a Service
MLOps	ML Operations
MSE	Mean Squared Error

NNI	Neural Network Intelligence
NV FLARE	NVIDIA Federated Learning Application Runtime Environment
OL	Online Learning
PATE	Private Aggregation of Teacher Ensembles
POC	Proof-Of-Concept
PPFL	Privacy-Preserving Federated Learning
PPFLaaS	Privacy-Preserving Federated Learning as a Service
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SCR	Self-Consumption Ratio
SGD	Stochastic Gradient Descent
SRIA	Strategic Research and Innovation Agenda
SSL	Secure Sockets Layer
SSR	Self-Sufficiency Ratio
SVT	Sparse Vector Technique
TFF	TensorFlow Federated
UC	Use Case
UI	User Interface
VOC	Visual Object Class
WP	Work Package
YOLO	You Only Look Once

## Executive Summary

Machine Learning is the cornerstone for the evolution of our technological life, as it adds growing intelligence to entities and ecosystems in our surroundings. IoT-NGIN provides a set of tools aimed for enhancing IoT intelligence and thus closely relate to ML processes. Specifically, the main outcomes towards this goals reported in this deliverable include:

- The technical specification of the IoT-NGIN MLaaS platform. The MLaaS platform architecture adopts a microservices' based architecture, leveraging the state-of-the-art open sources tools, and supporting the complete ML lifecycle from data collection to model serving. Moreover, its instantiation for the purposes of the project is presented.
- The technical specifications of the IoT-NGIN enhanced ML techniques, namely the Online Learning framework, as well as the Reinforcement Learning framework. For both techniques, the technical design is accompanied by implementations in the context of the Smart Energy Living Lab.
- The technical design of the Privacy-Preserving Federated Learning framework. The report presents the high-level design for providing federated learning as a service and proceeds with the analysis of three state-of-the-art FL frameworks enhanced with privacy preservation techniques:
  - NVIDIA FLARE is experimented under an image classification task, investigating the model performance under 3 privacy preservation techniques.
  - The IoT-NGIN FedPATE framework is presented, as a result of integrating Flower with PATE, supported by experimental investigation of the model performance against privacy preservation under image classification.
  - Tensorflow Federated is investigated under privacy preservation for intrusion detection in network logs' datasets.
- Open source implementation of the aforementioned platform and frameworks, followed by comprehensive guidance on their deployment, allowing their deployment and testing by third parties. The IoT-NGIN open source developments are available at the project's public GitLab group at: <https://gitlab.com/h2020-iot-ngin>.

The future work refers to enhancements towards the finalization of the tools, as well as the integration of these developments within the IoT-NGIN framework. The updates towards these goals shall be reported in deliverable D3.4 "ML models sharing and Transfer learning implementation" due in the second quarter of 2022.

# 1 Introduction

Machine Learning (ML) has been disruptive across industries during the past years and obviously there is still long way to unleash the full potential of ML in adding intelligence in tiny or larger things embracing our personal and professional lives. Nowadays, reliable ML models may be found to detect anomalies in health-related images, predict the energy consumption for the next days, detect flaws in industrial production lines and in many more use cases. Nevertheless, whatever the initial performance of an ML model might be, it could still be rendered ineffective, when remaining static, i.e. when it does not learn any more, or, e.g. it is difficult to be deployed in different or across devices.

Putting ML models in production and managing them appropriately is achieved via MLOps, which caters for covering requirements surrounding the whole ML lifecycle, from data collection and analysis to serving the model to the desired devices. IoT-NGIN delivers integrated MLOps functionality via its Machine Learning as a Service (MLaaS) platform, leveraging State-of-the-Art open source tools and extending them appropriately to serve computing concepts and paradigms, such as Digital Twins, edge and fog computing,

Moreover, the way the machine learns affects both its efficiency and its “learning curve”, in the sense of how fast it learns. Indicatively, Deep Learning is inspired from human neural networks and allows the machine to learn from huge datasets. In Reinforcement Learning, the machine would learn out of a process similar to trial and error, while through online learning, the dynamic model training takes place on live data. In addition, collaborative distributed training can be performed through federated learning, exploiting the knowledge residing inside dispersed datasets without disclosing them outside their administrative domain. IoT-NGIN provides enhancements towards reinforcement, online and federated learning, integrating them into the MLaaS platform and considering privacy preservation in distributed schemes.

The present document, entitled “Enhanced IoT federated deep learning/ reinforcement ML”, is the third deliverable of WP3 (D3.3) and reports the activities of Task 3.1 “*Big Data and ML framework architecture*”, Task 3.2 “*Deep learning/reinforcement learning techniques to enhance training processes*” and Task 3.3 “*Confidentiality-preserving federated ML models*”. Moreover, the activities reported in D3.3 align with the objectives of WP3:

- Define, design and develop a big data management and privacy preserving federated ML layer, based on Big Data Value (BDV) Strategic Research and Innovation Agenda (SRIA)4.0 [1], to train and share ML models.
- Implement innovative deep learning/reinforcement learning techniques to enhance training processes with inline adaptive self-learning that will improve the resulting machine learning models automatically.

The present document is a technical report which provides technical specifications for the MLaaS platform of IoT-NGIN, the Deep and Reinforcement learning techniques developed, as well as the Privacy-Preserving Federated Learning framework of IoT-NGIN.

## 1.1 Intended Audience

The intended audience includes ML engineers and developers which may find this report inspiring for their research and development efforts as well as ML service providers who aim to adopt the IoT-NGIN MLOps paradigm and the enhancements towards reinforcement,

online and federated learning. The document provides technical specifications, as well as practical guidance allowing interested audience to test and adopt the IoT-NGIN developments. The MLaaS platform adopts cloud-native architecture and supports deployments towards edge computing. Moreover, the document describes the design and implementation of online learning and reinforcement learning techniques, as well as the experimental evaluation of federated learning techniques against privacy preservation techniques.

Moreover, the document might be of interest to business users wishing to adopt machine learning into their processes. The document provides insights for exploiting the IoT-NGIN tools in the context of the Living Lab use cases, which could be indicative for other use cases as well. Also, the FL frameworks' assessment and recommended use per LL use case can be helpful guidance for selecting appropriate privacy preserving FL framework in their cases.

Finally, this report is useful internally, for the project partners which develop ML related solutions, perform integration and validation activities, as well as for the Living Labs. Useful feedback could be also received from the Advisory Board, including both technical and impact creation comments.

## 1.2 Relations to other activities

The activities of WP3 are strongly connected to the rest IoT-NGIN activities, as indicated in Figure 1.

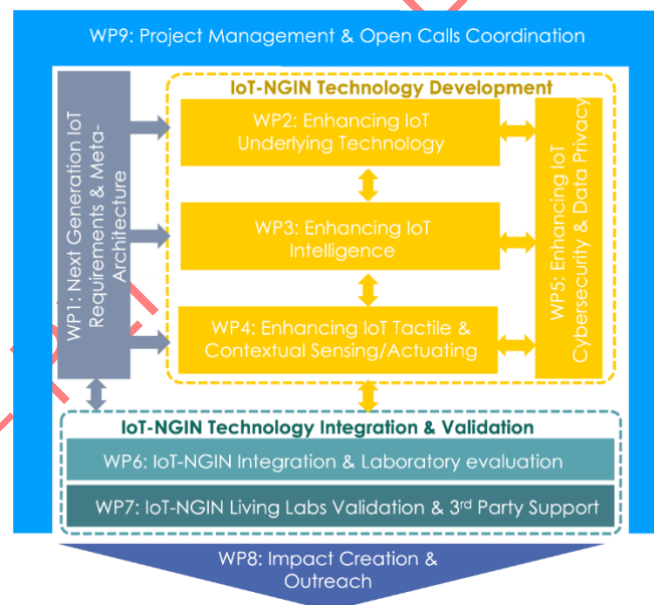


Figure 1: Work packages structure.

Considering this, WP3 is related to the work packages and tasks described in Table 1.

Table 1: Relation of WP3 activities to other WPs and tasks

WP	Relation to WP3 and D3.3
<b>WP1</b>	The definition of the Living Labs' use cases (UCs) and elicitation of user/system requirements has served D3.3 for defining the design and implementation of the MLaaS platform, of the online and reinforcement learning services and the federated learning framework, and providing tailored implementations in the context of these use cases. Also, the role of the tools and platforms deployed in WP3 shall be represented in the meta-architecture resulting by the activities in WP1.
<b>WP2</b>	The Secure Edge Cloud framework for IoT micro-services shall be integrated with MLaaS for secure execution of ML models.
<b>WP4</b>	The IoT Device Discovery module incorporates ML models, which may be trained and served via the MLaaS platform and WP3 learning tools.
<b>WP5</b>	WP5 develops cybersecurity tools for securing FL operation. These tools can work together with the FL modules of WP3 to ensure protection against data and model poisoning attacks. Moreover, the WP5 tools are based on ML functions, which may be trained and served via the MLaaS platform.
<b>WP6</b>	WP3 components will be integrated with the rest of project's technologies and frameworks within WP6, while the MLaaS platform and tools can be useful in the development of application logic for the use cases or the Open Calls.
<b>WP7</b>	<p>WP3 components will be implemented and used in several living labs and use cases.</p> <p>WP3 will support 3<sup>rd</sup> parties by offering ML models via MLaaS model training and sharing.</p>
<b>WP8</b>	WP3 will provide notable outcomes and results for supporting impact creation activities. Moreover, it will consider feedback (e.g. from the market analysis and business modelling tasks) which could be relevant for updating or enhancing the WP3 design and development.

## 1.3 Document overview

The rest of the document is organized as follows.

Section 2 presents the technical specifications of the MLaaS platform. It outlines the MLaaS architecture, which aims to provide a complete ML system, as well as a consistent reproducible and easy-to-install implementation.

Section 3 presents the IoT-NGIN techniques for online and reinforcement learning. Both the technical design and the implementation for the LL use cases are provided.



Section 4 is devoted to privacy-preserving Federated Learning. The IoT-NGIN Privacy-Preserving Federated Learning as a Service (PPFLaaS) is presented, providing access to three FL frameworks, which are further experimentally investigated in the context of the LLs.

Section 5 provides installation and user guidance, as the MLaaS platform and tools will be provided as open source.

Finally, section 6 draws conclusions and next steps.

DRAFT - PENDING EC APPROVAL

## 2 IoT-NGIN Machine Learning as a Service Framework

### 2.1 Introduction

Deliverable “D3.1” Enhancing deep learning / reinforcement learning” [2] gives an overview of the MLaaS state-of-the art, the concepts that applies in the context of IoT-NGIN project, and the various use cases. It also describes the expected functional and non-functional requirements and the high-level architecture of the MLaaS Platform Concept. This section introduces the technical architecture used for the actual implementation of the MLaaS framework. The components are discussed as well as the interactions between them.

There are several MLaaS platforms commercially available either from the big Cloud Service Provider (AWS, Azure, GCP) or from specialized companies. However, there are little MLaaS frameworks built around open-source projects that cover the whole spectrum of the need for a complete end-to-end ML system. Building such a system is a challenge because:

- Lot of different functions are required to build a complete MLaaS system
- For the same function, there could be several open-source projects available, and it could be a long and complex exercise to choose the right product
- Projects are usually created for a specific function with little or no consideration for integration in a larger system
- Projects can be very complex to install, manage and use
- Customizations and adaptations may be needed to the available open source to fit the functional and non-functional requirements expected from the MLaaS System

One of the main goals of the IoT-NGIN MLaaS project is to provide an architecture which gives a holistic view on how to possibly create a system that provide a complete ML platform in line with the functional and non-functional requirements and the high-level architecture of the MLaaS Platform Concept. This is realized by looking at the available projects in the open-source community, selecting components for specific functions and then exploring how these components could integrate together to form a comprehensive framework. Another goal is to be able to have a consistent reproducible implementation and simplify as much as possible the installation.

As part of the IoT-NGIN project a Minimum Viable Product (MVP) of the MLaaS platform will be implemented in order to test some of the selected components and their integrations in the platform. This installation will also be used to support the Living Labs tests.

### 2.2 Technical architecture overview

The technical architecture is introduced in figure 2. From a high-level view, it is split into the infrastructure hosting environment and the MLaaS components as such. The goal is to have the MLaaS components not tied to a specific hosting environment so that MLaaS can be deployed in various environments: in a private cloud, in a public cloud, in a simple or more complex IT setup. This document gives an overview of these infrastructure components used for the MLaaS implementation in the IoT-NGIN project (provided by WP6). However, there are some requirements that will have to be fulfilled by the hosting platform to accommodate for the MLaaS platform installation. The MLaaS platform does not mandate specific security,

## D3.3 - ENHANCED IOT FEDERATED DEEP LEARNING/ REINFORCEMENT ML

monitoring and managements tools, so the infrastructure provider is free to use its usual tools for managing and maintaining the infrastructure. For example, on Kubernetes Prometheus and Graphana are often used.

The MLaaS platform implementation aims to provide the following main functions to the users: training, deployment of models and prediction. This translates to the need of the following technical components:

- Data
  - Components for data acquisition
  - Components for data analysis
  - Components for data transformation
  - Components for data storage
- Modelling
  - Components to support training ML model
  - Components for model evaluation
  - Components for continuous learning
- Deployment
  - Components to deploy models
  - Components to share models
- Prediction
  - Components for model serving
  - Components for batch prediction
  - Components for real-time prediction

In the technical architecture these components are grouped in functional blocks as highlighted in Figure 2.

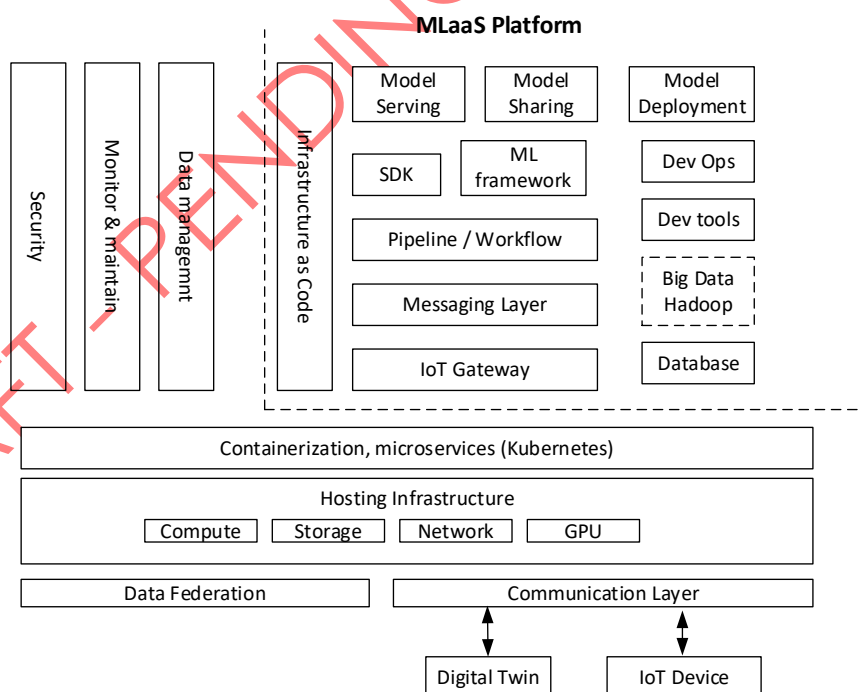


Figure 2: MLaaS framework technical architecture.

As previously highlighted the hosting infrastructure and monitoring/management tools are not part of the MLaaS Platform. However, in order to support the MLaaS platform, this

infrastructure must provide network access, compute resources, storage service, and preferably GPU. For IoT-NGIN, it has been decided to use containers and microservices, which means that the infrastructure must support containerization and microservices via a Kubernetes cluster. In addition, it is expected that the hosting environment will provide all the tools to manage and monitor the MLaaS services as well as some security services.

The blocks of the MLaaS platform are:

*IoT gateway:* this group includes components used to receive data from the IoT devices. Example of technology that can be used in that layer are: MQTT Broker, CoAP, HTTP/HTTPS, OPC-UA, REST proxy. The MLaaS for IoT-NGIN selected MQTT Broker and HTTP/HTTPS for this block.

*Messaging layer:* this group includes components for a messaging system for event streaming that allows passing data between the IoT gateway and some upper layers where the data will be consumed. There are also components to interface with the messaging layer, tools such as Apache Camel.

*Database:* this group includes components that provides data storage capabilities. Possible technologies in this layer are: SQL database, NoSQL database and time series services.

*Big Data:* this group includes the components of a Hadoop service. Given the complexity and the resources required to implement such a service it will not be included in the MLaaS platform from the IoT-NGIN project. Such an addition is left for future work.

*Pipeline/workflow:* this group allows the building and deployment of portable, scalable machine learning (ML) workflows. For example, the workflow can perform data pre-processing, data transformation, model training, and so on.

*ML Framework:* this group aims to provide the ML framework and libraries that a developer can use to train a model. Example of well-known frameworks are Tensorflow, Keras, PyTorch, TinyML, scikit-learn and XGBoost. In addition, components for Federated Machine Learning are implemented.

*SDK:* the SDK provides the necessary environment to develop and test programs. The intent of the MLaaS platform is not to provide state-of-the-art IDE so the platform will be limited to a simple development environment with Python and possibly the Rust compiler.

*Model Serving:* components in this block allow to expose ML model via API so that an application can request a prediction from this model. Example of technologies are KFServe, TensorFlow serving, TorchServe and Seldon Core.

*Model Sharing:* components in this block are used to share the ML models so that ML engineers, data scientists or AI developers can use them as-is or for transfer learning. MLaaS uses a Polyglot Model Sharing. MLaaS aims to support open format built to represent machine learning models and to allow for translating models between the format from popular ML framework. An external DLT system can be used to verify model integrity via blockchain technology.

*Model Deployment:* components that are used for placing a ML model in a live environment or in IOT devices and where it can then be used to perform predictions. At the time of this writing this part is still for future work.

*Dev tools:* this group includes components which intent is to help AI developers. The intent of the MLaaS platform is not to provide an state-of-the-art IDE, so the platform will be limited to

mainly providing notebook capabilities, some monitoring tools for the training of the ML model such as TensorBoard.

*Dev Ops*: components in this group are CI/CD tools used to deploy new apps and ML models. Examples of technologies are: GitLab runner, Jenkins, Spinnaker, Argo. At the time of this writing this part is still for future work.

*Infrastructure as Code (IaC)*: this block contains the possible YAML files that are used to configure the platform.

## 2.3 MLaaS platform architectural approach

One of the constraints for implementing the IoT-NGIN MLaaS platform is that it must be based on open-source software in order to make it vendor independent.

From an architectural point of view, several choices have been done for the implementation of the platform.

The first structural choice has been to rely on containers and microservices, with Kubernetes as the managing platform for running and managing containers. Using this microservice approach offers the flexibility to select the blocks and components that are deployed in the MLaaS platform, depending on the business needs and the hosting environment. Indeed, the MLaaS platform can be installed from a relatively lightweight infrastructure like a simple two nodes cluster, to a medium size environment like in a private cloud or up to a large infrastructure like in a public cloud setup.

The other main structural choice has been to select between integrating lot of “small” components or a single product incorporating the core functions the required in MLaaS platform. The first choice, the Lego approach, tries to integrate disparate components such as, for example, the Apache Beam, Airflow, TensorFlow, Apache Spark, etc. This approach is illustrated in Figure 3.

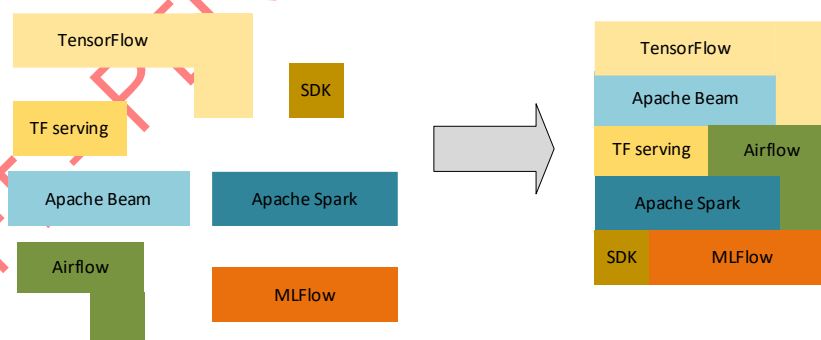


Figure 3: The Lego approach.

In this approach products for each step of the creation of the AI application are independently selected from each other. For example, one product is selected for training, another for auto-ML, another for creating pipeline and finally one for model serving. It allows a careful and ad-hoc choice, possibly tuned for a specific use case, in the selection of the components. However, it presents major integration and operating challenges both in term

of initial setup, interconnection and evolution as each product has its own logic, console, lifecycle and development team.

The second choice is to use a “all-in-one” approach, where all components are integrated and maintain as part of a single product by a single development team. Functions are well integrated together allowing for a coordinated lifecycle, simple installation and management via a single dashboard. This approach is illustrated in Figure 4.

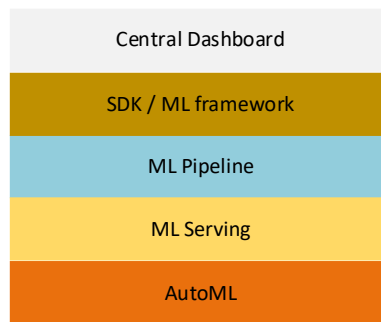


Figure 4: “all-in-one” approach.

The latter “all-in-one” choice as been chosen and Kubeflow [1] has been selected as the integrated platform. The Kubeflow project is dedicated to make deployments of machine learning (ML) workflows on Kubernetes simple, portable and scalable. The goal is to provide a straightforward way to deploy best-of-breed open-source systems for ML into diverse infrastructures running Kubernetes.

Another choice has been to use a GitOps approach for installation of the platform – see <https://www.gitops.tech/> for reference describing the GitOps concept. The core idea of GitOps is having a Git repository that always contains declarative descriptions of the infrastructure currently desired in the production environment and an automated process to make the production environment match the described state in the repository which for the MLaaS MVP is [https://gitlab.com/h2020-iot-ngin/enhancing\\_iot\\_intelligence/ml-framework-architecture.git](https://gitlab.com/h2020-iot-ngin/enhancing_iot_intelligence/ml-framework-architecture.git). In the case of the MLaaS platform, the goal is to be able to easily implement the platform in a consistent and reproductive way and also to allow for tracking the changes over the time.

Finally, a number of components have then been selected to create the blocks needed to complete the framework and to fulfil the functional requirements.

## 2.4 MLaaS platform components

There are two categories of MLaaS platform software components:

1. ML Components that are used to develop and run the ML applications. These components part of in the blocks shown in the MLaaS Platform in Figure 2.
2. Infrastructure components that are used to support the implementation of the MLaaS components.

The following sections list the various components used in the IoT-NGIN MLaaS platform. Not all the components are mandatory and have to be installed. The users can select only what they need.

Note that given the extensive number of components and complexity, not all of the components will be tested as part of the IoT-NGIN MLaaS MVP. Installation and test of the components not included into the MVP may be tested in the future as the platform develops over time.




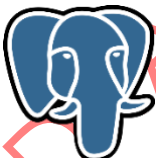

## 2.4.1 ML Components

Table 2 gives a summary of the ML components selected for the IoT-NGIN MLaaS platform.






Table 2: IoT-NGIN MLaaS platform ML components.



Product	Usage	Description
 Kubeflow	ML framework Pipeline/workflow Dev tools	From <a href="https://www.kubeflow.org/">https://www.kubeflow.org/</a> : "Kubeflow is a platform for data scientists who want to build and experiment with ML pipelines. Kubeflow is also for ML engineers and operational teams who want to deploy ML systems to various environments for development, testing, and production-level serving". Kubeflow is the central components of the In the MLaaS platform providing the following functions: Notebooks, model training, Pipelines, Multi-framework (PyTorch, Apache MXNet, MPI, XGBoost).
 KServe	ML serving	From <a href="https://kserve.github.io/website/">https://kserve.github.io/website/</a> : "KServe is a standard Model Inference Platform on Kubernetes, built for highly scalable use cases". In the MLaaS platform, KServe is used for model serving allowing users and applications to request predictions from a model.
 Kafka	Messaging	From <a href="https://kafka.apache.org/">https://kafka.apache.org/</a> : "Apache Kafka is an open-source distributed streaming system used for stream processing, real-time data pipelines, and data integration at scale". In the MLaaS platform Kafka to exchange messages between an IoT gateway and Kubeflow.
 Camel-k	Messaging	From <a href="https://camel.apache.org/">https://camel.apache.org/</a> : "Apache Camel K is a lightweight integration framework built from Apache Camel that runs natively on Kubernetes and is specifically designed for serverless and microservice architectures. Users of Camel K can instantly run integration code written in Camel DSL". Camel



		<p>implements the Enterprise Integration Patterns (EIP) which are based on messaging principles.</p> <p>In the MLaaS platform Camel-k can be used for integration code between Kafka and Kubeflow or KServe.</p>
Minio 	Storage	<p>From <a href="https://min.io/">https://min.io/</a>: "MinIO [4] is high-performance Kubernetes-native object storage that is compatible with the S3 API. MinIO provides a single global namespace and a consistent object storage interface across multiple cloud providers, on premise and at the edge".</p> <p>In the MLaaS platform MinIO is used to store models and artifacts.</p>
Flower 	FL framework	<p>From <a href="https://flower.dev/">https://flower.dev/</a>: "Flower is a Friendly Federated Learning Framework. It provides a unified approach to federated learning. Federate any workload, any ML framework, and any programming language".</p> <p>In the MLaaS platform Flower is planned to bring FL capabilities.</p>
MLFlow 	ML framework	<p>From <a href="https://mlflow.org/">https://mlflow.org/</a>: "MLflow is an open-source platform to manage the ML lifecycle, including experimentation, reproducibility, deployment, and a central model registry".</p> <p>In the MLaaS platform MLFlow is planned to complement Kubeflow for training and parameter tuning.</p>
PostgreSQL 	Database	<p>From <a href="https://www.postgresql.org/">https://www.postgresql.org/</a>: "PostgreSQL is a powerful, open-source object-relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads".</p> <p>In the MLaaS platform PostgreSQL is planned to store and extract structured data for model training for example.</p>
InfluxDB 	Database	<p>From <a href="https://www.influxdata.com/">https://www.influxdata.com/</a>: "InfluxDB is an open-source time series platform. This includes APIs for storing and querying data, processing it in the</p>



		<p>background for ETL or monitoring and alerting purposes, user dashboards, and visualizing and exploring the data and more".</p> <p>In the MLaaS platform InfluxDB is planned to store and extract time series data for model training for example.</p>
Cassandra  <b>cassandra</b>	Database	<p>From <a href="https://cassandra.apache.org/">https://cassandra.apache.org/</a>: "Apache Cassandra is an open-source NoSQL distributed database".</p> <p>In the MLaaS platform Cassandra is planned to store and extract structured data for model training for example.</p>
Mosquito MQTT 	IoT Gateway	<p>From <a href="https://mosquitto.org/">https://mosquitto.org/</a>: "Eclipse Mosquitto is an open source (EPL/EDL licensed) message broker that implements the MQTT protocol versions 5.0, 3.1.1 and 3.1. Mosquitto is lightweight and is suitable for use on all devices from low power single board computers to full servers".</p> <p>In the MLaaS platform MQTT is planned to ingest data from IoT Devices or Digital Twins.</p>
Nginx 	IoT Gateway	<p>From <a href="https://www.nginx.com/">https://www.nginx.com/</a>: "Nginx (pronounced "engine-ex"), is an open-source web server that, is now also used as a reverse proxy and ingress gateway".</p> <p>In the MLaaS platform Nginx is planned to ingest data from IoT Devices or Digital Twins.</p>
ONNX  ONNX	Model Sharing	<p>From <a href="https://onnx.ai/">https://onnx.ai/</a> : "ONNX is an open format built to represent machine learning models. ONNX defines a common set of operators - the building blocks of machine learning and deep learning models - and a common file format to enable AI developers to use models with a variety of frameworks, tools, runtimes, and compilers".</p> <p>In the MLaaS platform ONNX is used as library by Model Sharing service.</p>
Hadoop 	Big Data	<p>From <a href="https://hadoop.apache.org/">https://hadoop.apache.org/</a> : "The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across</p>

		clusters of computers using simple programming models".
GitLab Runner 	Dev Ops	From <a href="https://docs.gitlab.com/runner/">https://docs.gitlab.com/runner/</a> : "GitLab Runner is an application that works with GitLab CI/CD to run jobs in a pipeline."  In the MLaaS platform GitLab Runner is planned for setting up CI/CD pipeline which is used for example to upload new model into an IoT device.
Rust 	SDK	Rust is a language a choice for development targeting low-resource devices. <a href="https://www.rust-lang.org/">https://www.rust-lang.org/</a>  In the MLaaS platform Rust is planned for development of application to be uploaded into IoT device.

## 2.4.2 Infrastructure Components




The infrastructure components can be classified into three categories:

1. Components which are required for Kubeflow installation. They are the Common Services maintained by the Kubeflow Manifests WG [5].
2. The components used by the underlying infrastructure provided by WP6 to test the MLaaS platform. Other components may be used as long as they provide a similar service.
3. The components that have been selected to help administrator install and support the MLaaS platform

The components for Common Services are listed in Table 3.




Table 3: Common Services components.



Product	Usage	Description
Istio 	Ingress	Istio is an open-source service mesh. It is used by several components of Kubeflow as well as by the ingress gateway. <a href="https://istio.io/latest/">https://istio.io/latest/</a>
Cert-manager	Security	From <a href="https://cert-manager.io/">https://cert-manager.io/</a> : "Cert-manager adds certificates and certificate issuers, as resource types, in the Kubernetes clusters, and simplifies the process of obtaining, renewing and using those certificates. It can issue certificates from a variety of supported sources,"

		<p>including Let's Encrypt, HashiCorp Vault, and Venafi as well as private PKI".</p> <p>In the MLaaS platform Cert-manager is used to generate certificates for the Istio and NGINX ingress gateway. It used the Let's Encrypt issuer.</p>
Knative 	Serverless	<p>From <a href="https://knative.dev/docs/">https://knative.dev/docs/</a> : "Knative is an Open-Source Enterprise-level solution to build Serverless and Event Driven Applications".</p> <p>In the MLaaS platform Knative is used by some components of Kubeflow and by KServe.</p>
Dex 	IAM	<p>From <a href="https://dexidp.io/">https://dexidp.io/</a>: "Dex is an identity service that uses OpenID Connect to drive authentication for other apps".</p> <p>In the MLaaS platform Dex is used the default authentication service used by Kubeflow and is planned to be replaced by Keycloak.</p>

The components used for the underlying IoT-NGIN test infrastructure are listed in Table 4.


Table 4: Underlying IoT-NGIN test infrastructure components.



Product	Usage	Description
Nginx 	Ingress	<p>From <a href="https://www.nginx.com/">https://www.nginx.com/</a>: "Nginx (pronounced "engine-ex"), is an open-source web server that, is now also used as a reverse proxy and ingress gateway".</p> <p>In the MLaaS platform Nginx is used as an Ingress gateway for several services.</p> <p><a href="https://www.nginx.com/">https://www.nginx.com/</a></p>
Ceph 	Storage	<p>From <a href="https://ceph.com/en/">https://ceph.com/en/</a> : "Ceph provides a unified storage service with object, block, and file interfaces from a single cluster built from commodity hardware components".</p> <p>In the MLaaS platform, Ceph provide the underlying storage for the services with the Block storage being the default storage class.</p>
Rook 	Storage	<p>From <a href="https://rook.io/">https://rook.io/</a> : "Rook is an open-source cloud-native storage orchestrator for Kubernetes, providing the platform, framework, and support for a diverse set of storage solutions</p>

		<p>to natively integrate with cloud-native environments. Rook turns storage software into self-managing, self-scaling, and self-healing storage services".</p> <p>In the MLaaS platform, Rook is used to manage Ceph storage.</p>
Rook Toolbox 	Storage	<p>From <a href="https://rook.io/docs/rook/v1.9/Troubleshooting/ceph-toolbox/">https://rook.io/docs/rook/v1.9/Troubleshooting/ceph-toolbox/</a> : "The Rook toolbox is a container with common tools used for rook debugging and testing".</p> <p>In the MLaaS platform, Rook is used to verify and diagnose Ceph storage.</p>
Kubernetes Metrics Server	Metrics API	<p>From <a href="https://github.com/kubernetes-sigs/metrics-server">https://github.com/kubernetes-sigs/metrics-server</a> : "Metrics Server is a scalable, efficient source of container resource metrics for Kubernetes built-in autoscaling pipelines. It is required for Horizontal Autoscaling".</p> <p>In the MLaaS platform, metric server is used to collect data required for Horizontal Pod Autoscaling (HAP) used by some of the Kubeflow components. In a production environment this component would likely be replaced with Prometheus.</p>
Metallb 	Ingress	<p>From <a href="https://metallb.universe.tf/">https://metallb.universe.tf/</a> : "MetalLB is a load-balancer implementation for bare metal Kubernetes clusters, using standard routing protocols".</p> <p>In the MLaaS platform, Metallb is used to allocate external load-balancer ip address to the Istio gateway and to the Nginx gateway.</p>

The components for installation and support of the MLaaS Platform are presented in Table 5.

Table 5: installation and support components.

Product	Usage	Description
ArgoCD 	Dev Ops	<p>From <a href="https://argo-cd.readthedocs.io/en/stable/">https://argo-cd.readthedocs.io/en/stable/</a> : "ArgoCD is a declarative, GitOps continuous delivery tool for Kubernetes."</p> <p>In the MLaaS platform, ArgoCD is used to deploy and maintain the platform from a Git repository.</p>
Keycloak	IAM	<p>From <a href="https://www.keycloak.org/">https://www.keycloak.org/</a>: "Keycloak adds authentication to applications and secure services.</p>

		<p>Keycloak provides user federation, strong authentication, user management, fine-grained authorization, and more".</p> <p>In the MLaaS platform, keycloak is planned to replace Dex for the authentication of the Kubeflow users.</p>
<p>GitLab</p> 		<p>From <a href="https://about.gitlab.com/">https://about.gitlab.com/</a>, GitLab is the "The One DevOps Platform".</p> <p>In the IoT-NGIN MLaaS testing, GitLab is used to store the configuration files for installing the platform with ArgoCD.</p>

## 2.5 MLaaS reference architecture

### 2.5.1 Overall architecture

Figure 5 gives a view of the reference architecture for the IoT-NGIN MLaaS platform.

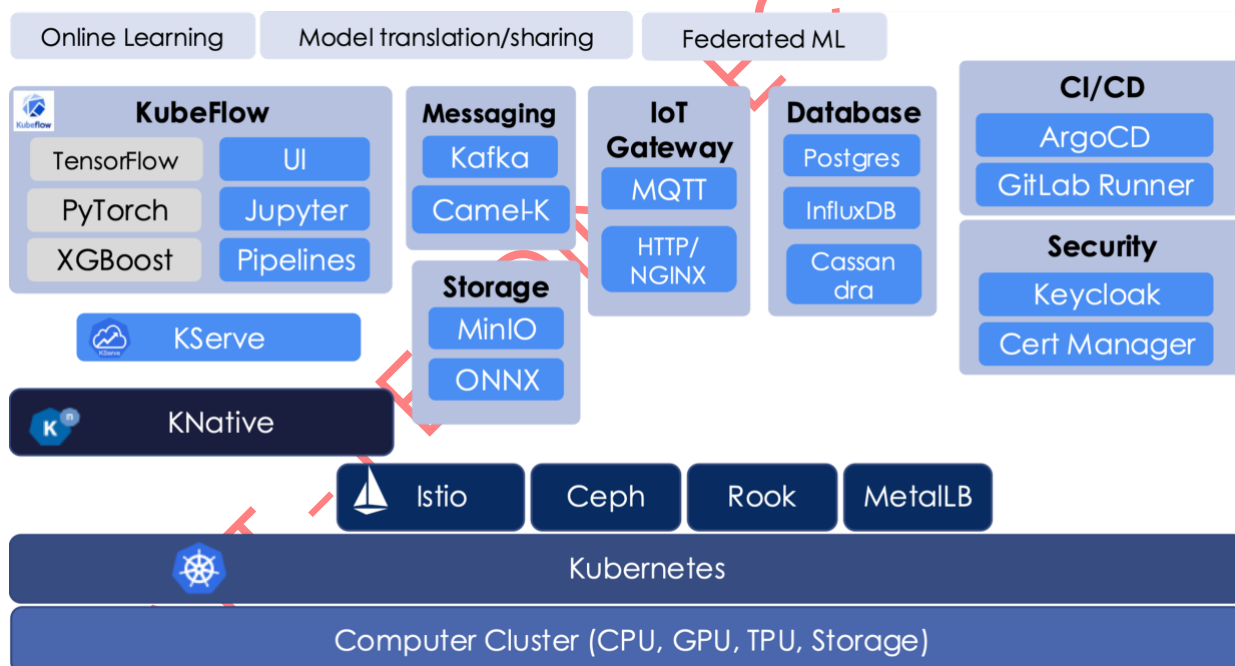


Figure 5: MLaaS framework technical architecture.

Figure 6 gives an indication of the main organization and interoperability of the different components of the MLaaS platform with Kubeflow and Kserve highlighted as the central services.

Note that in the diagram this is the IoT Device that sends data to the platform. However it could also be done by the Digital Twin.

Similarly, Figure 7 gives a high-level view of the interactions between the various infrastructure components.

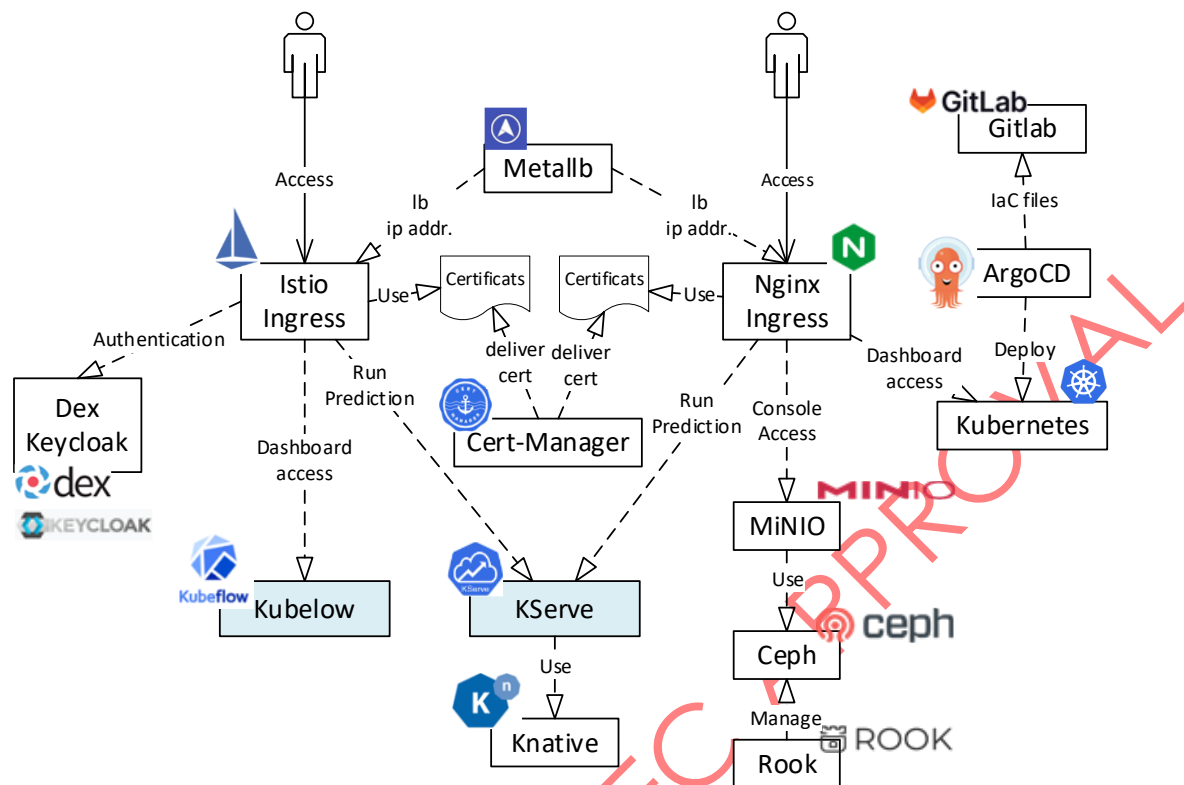


Figure 7: Infrastructure components interactions.

## 2.5.2 Access to the MLaaS platform

Access to the platform is via either a Istio Ingress gateway for some components such as the Kubeflow dashboard and Kserve or via the Nginx ingress gateway for other components such as MinIO.

For Kubeflow, the Istio Ingress gateway is responsible for checking authentication and authorization before granting access to the upstream services. Authentication is performed using an OpenID Connect (OIDC) provider. Dex is used in the Kubeflow standard installation. Keycloak is planned to be used for the IoT-NGIN MLaaS platform.

For Kserve refer to section 2.7 for explanation how to the inference services are exposed to the Internet.

Figure 8 illustrates the access to the platform from the Internet.

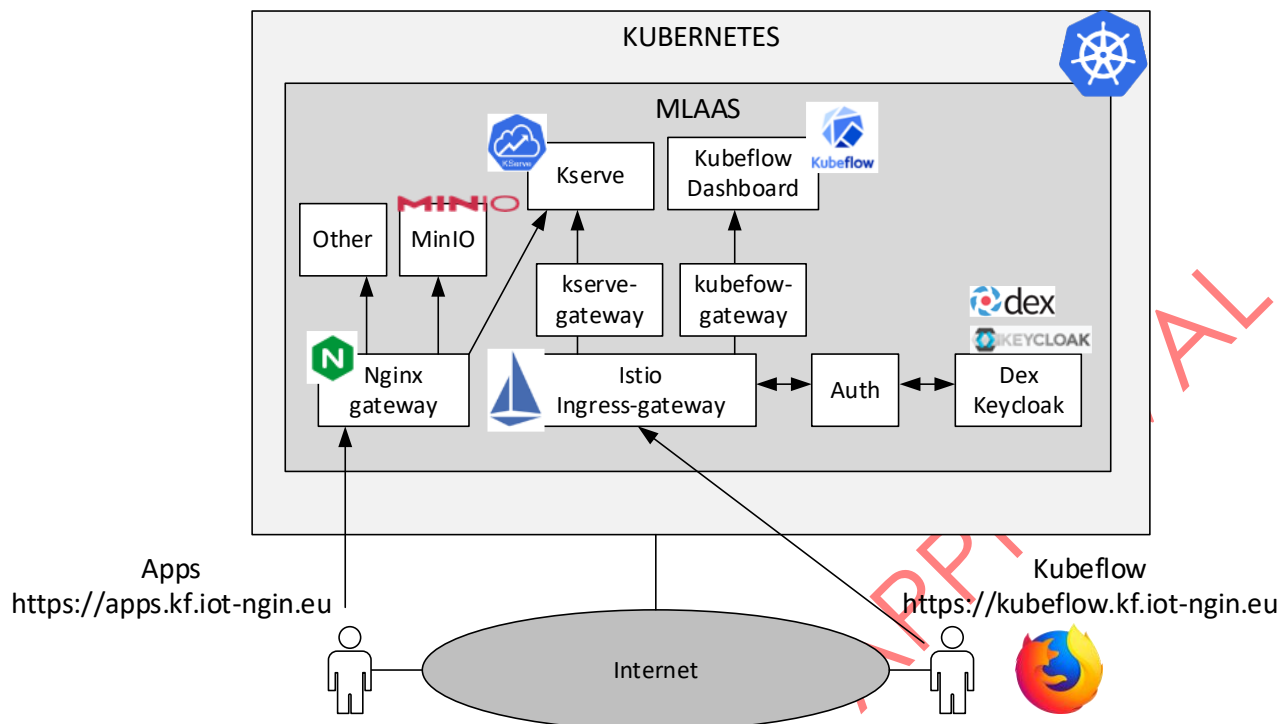


Figure 8: Access to the MLaaS platform.

### 2.5.3 End-to-end services examples

Figure 9 shows an example of end-to-end inference pipeline that could be implemented with the platform.

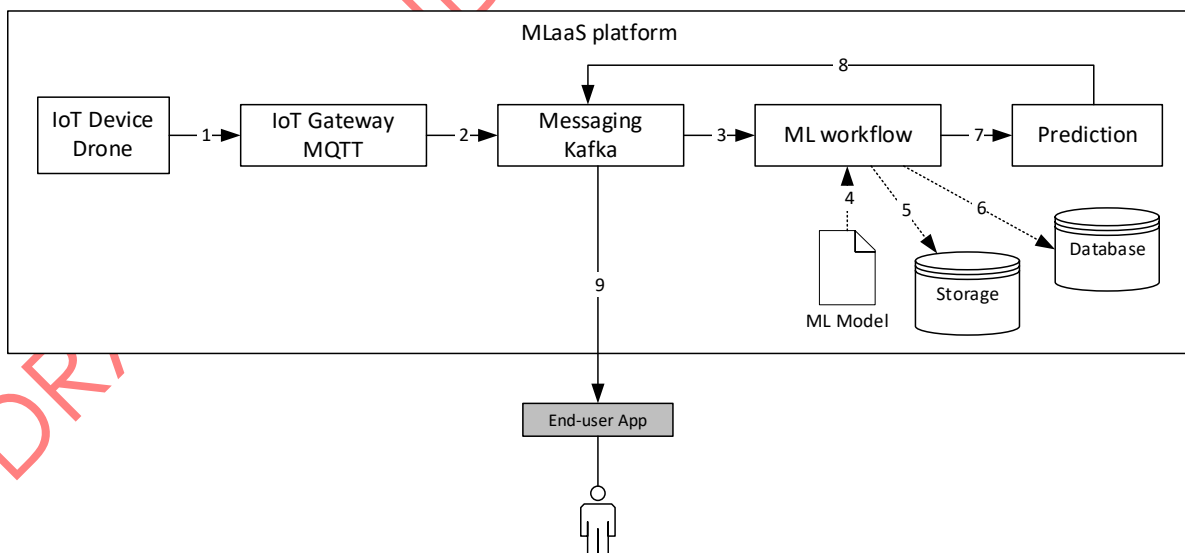


Figure 9: End-to-end inference pipeline example.

In this example:



1. An IoT device is sending data to an IoT gateway. It could be for example a drone sending images to a MQTT gateway via a 5G connectivity.
2. The IoT Gateway send the data to a Kafka system on a specific topic.
3. The Kafka system relays the message which triggers a ML workflow. The workflow performs the required processing and transformation of the data.
4. A previously trained model is used for by the ML Workflow to perform prediction.
5. The data are saved in MinIO for example for possible continuous learning (re-train model).
6. Metadata are saved to keep track of the incoming data, for example as a time series in InfluxDB, for reference and possible future use
7. The ML workflow output a prediction inferred from the received data
8. The prediction is sent to the kafka messaging system
9. The prediction is then relay to an end user application

Figure 10 from [6] shows another example of end-to-end inference pipeline which processes a Kafka event and invoke the inference service to get the prediction with provided pre/post processing code.

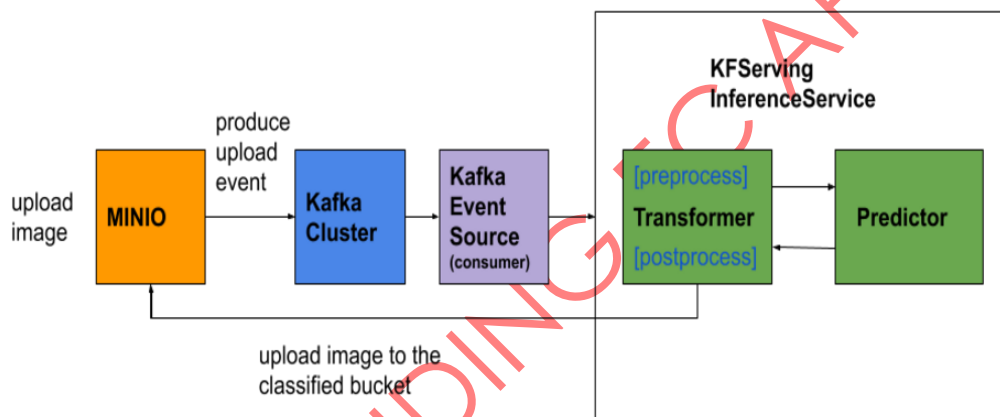


Figure 10: End-to-end inference pipeline example with Kafka.

## 2.6 Kubeflow

### 2.6.1 Kubeflow overview

Kubeflow is the central component of the IoT-NGIN MLaaS framework. It provides a Central Dashboard, Kubeflow Notebooks, Kubeflow Pipelines, Katib and Training Operators. Kubeflow support add-ons such as KServe for Model serving. Kubeflow support multi-tenancy allowing different users to use the platform in isolation. and be part of a group. Kubeflow is said to be “the ML toolkit for Kubernetes”.

Figure 11 from [7] shows a conceptual overview of Kubeflow.

## D3.3 - ENHANCED IOT FEDERATED DEEP LEARNING/ REINFORCEMENT ML

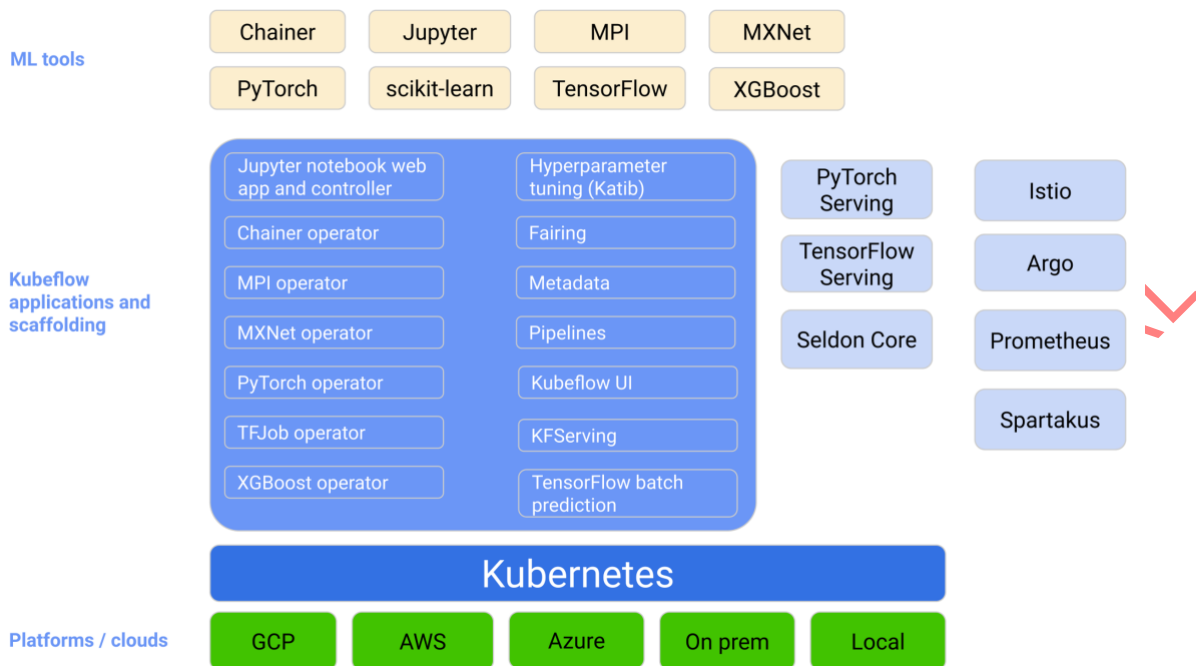


Figure 11: Kubeflow conceptual overview.

Figure 12 from [7] shows which Kubeflow components are useful at each stage.

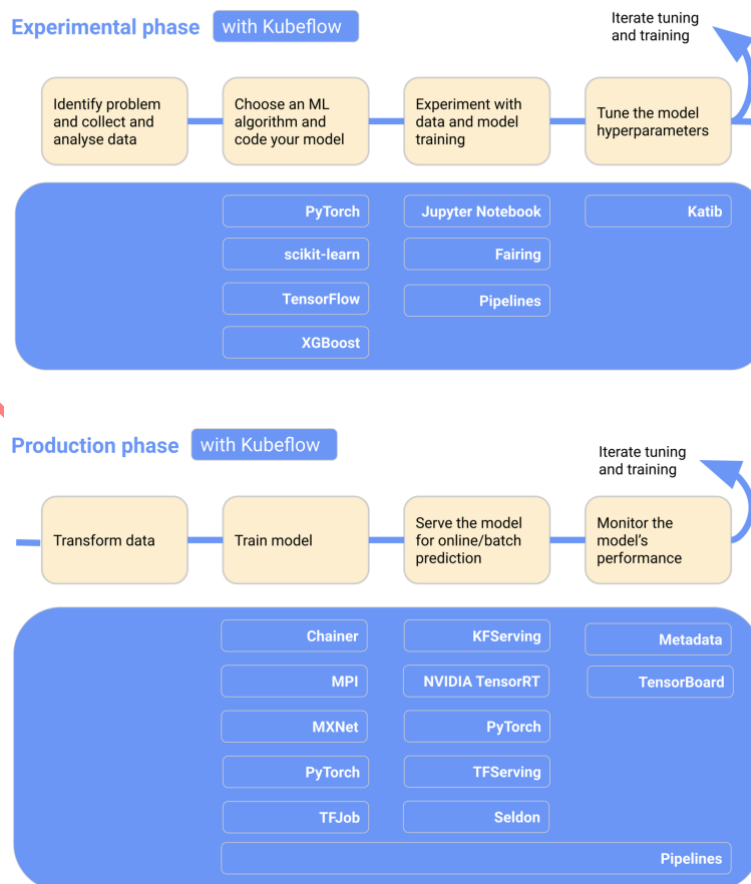


Figure 12: Kubeflow components.

## 2.6.2 IoT-NGIN MLaaS Kubeflow test environment

A Minimal Viable Product (MVP) version of the MLaaS platform is deployed to test the functionalities of the platform, the integration with the other IoT-NGIN tasks and the installation methodology. At the time of this writing the following components have been deployed:

- For the infrastructure part: Istio, Knative, Nginx, Cert-manager, Rook, Ceph, ArgoCD, Dex
- For the ML part: Kubeflow, KServe, MinIO, Camel-K

In the IoT-NGIN MLaaS test installation, the Kubeflow is accessible via the Istio Ingress gateway. The certificate is generated by the cert-manager component by using the Let's Encrypt issuer.

Figure 13 shows the Kubeflow User Interface from the MLaaS IoT-NGIN installation.

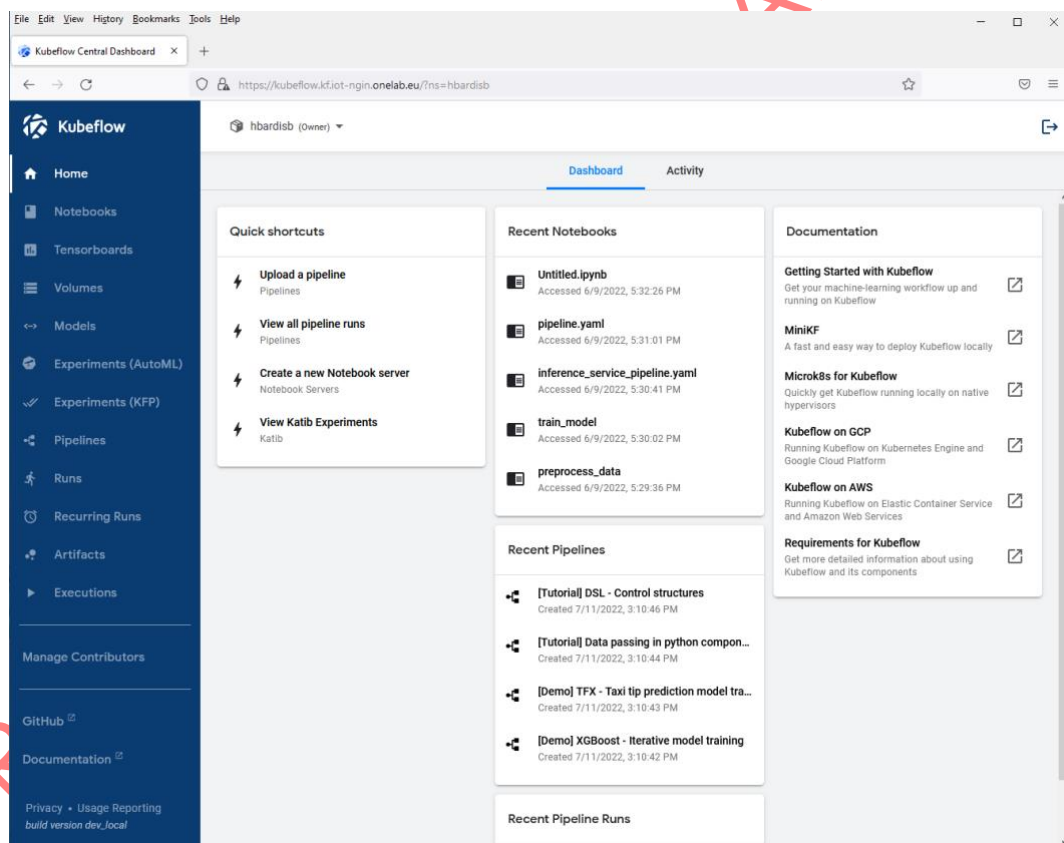


Figure 13: Kubeflow User Interface.

For M30, it is planned to have Keycloak implemented for the authentication of the Kubeflow users. Other components may be added depending on the needs of the Living Labs and Open Calls partners. Also it is planned to improve the installation of the MLaaS platform by using configuration files listing the personalized parameters (like domain names) and by adding more automation.

## 2.7 KServe

### 2.7.1 KServe overview

With Kubeflow, KServe plays a critical role in the MLaaS platform as it is the component that allows applications to request ML predictions via API.

Figure 14 from [8] shows the KServe architecture.

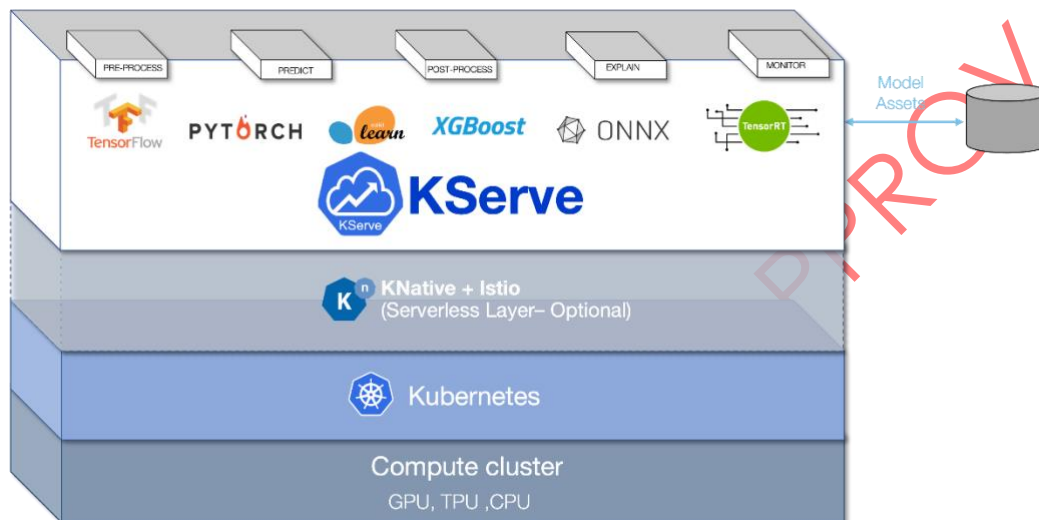


Figure 14: KServe architecture.

As explained in KServe documentation [9] “KServe enables serverless inferencing on Kubernetes and provides performant, high abstraction interfaces for common machine learning (ML) frameworks like TensorFlow, XGBoost, scikit-learn, PyTorch, and ONNX to solve production model serving use cases”. Conceptually, KServe has two planes: a control plane and a data plane.

The Control plane is responsible for reconciling the InferenceService custom resources. It creates the Knative serverless deployment for predictor, transformer, explainer. Figure 15 from [9] shows the architecture of the control plane.

## D3.3 - ENHANCED IOT FEDERATED DEEP LEARNING/ REINFORCEMENT ML

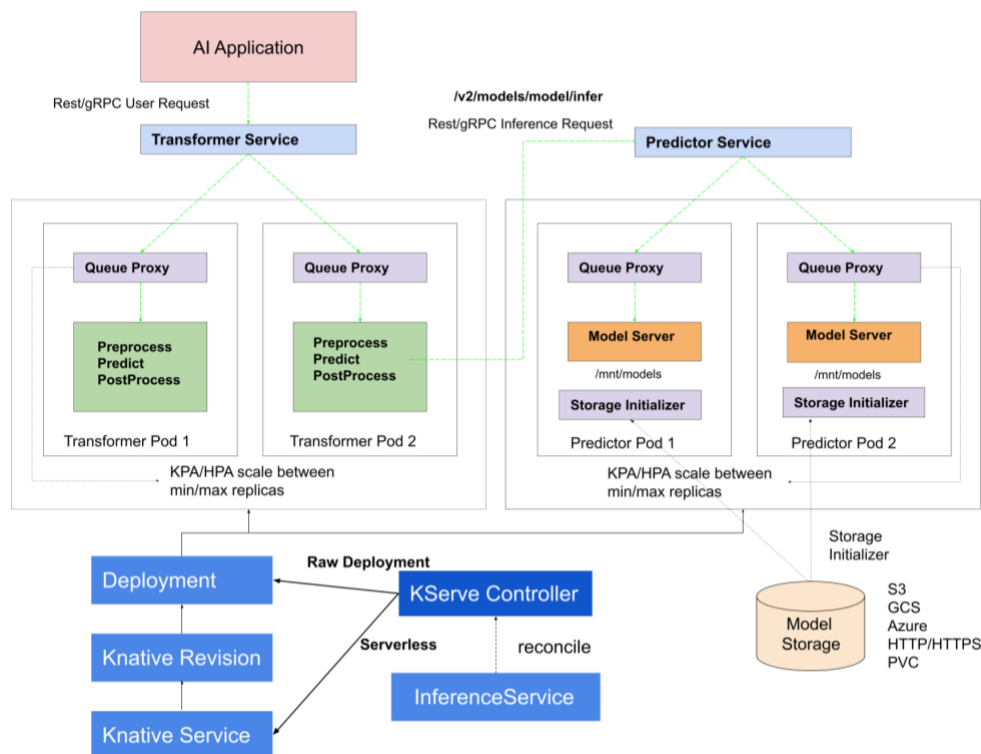


Figure 15: KServe Control Plane.

The InferenceService Data Plane architecture consists of a static graph of components which coordinate requests for a single model. This is illustrated in Figure 16.

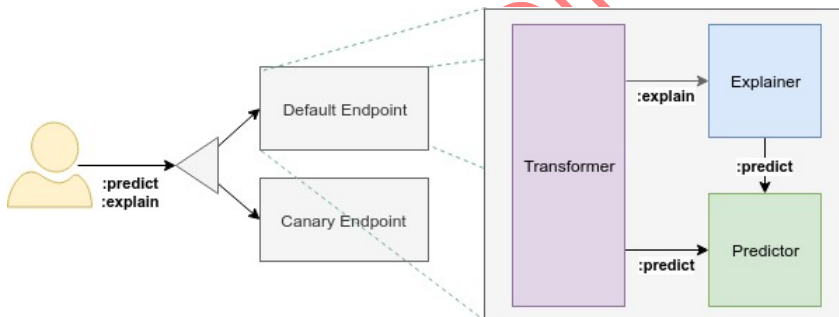


Figure 16: KServe InferenceService.

As explained in [9], the data plane uses the following concepts:

- **Component:** Each endpoint is composed of multiple components: "predictor", "explainer", and "transformer". The only required component is the predictor, which is the core of the system.
- **Predictor:** The predictor is the workhorse of the *InferenceService*. It is simply a model and a model server that makes it available at a network endpoint.
- **Explainer:** The explainer enables an optional alternate data plane that provides model explanations in addition to predictions. Users may define their own explanation

- container, which configures relevant environment variables such as the prediction endpoint. For common use cases, KServe provides out-of-the-box explainers like Alibi.
- **Transformer:** The transformer enables users to define a pre and post processing step before the prediction and explanation workflows. Like the explainer, it is configured with relevant environment variables too. For common use cases, KServe provides out-of-the-box transformers like Feast.

## 2.7.2 IoT-NGIN MLaaS KServe installation

In IoT-NGIN MLaaS platform, KServe is configured to be access from the outside either via the `kserve.kf.iot-ngin.onelab.eu` domain or via the `apps.kf.iot-ngin.onelab.eu` domain.

In the first case, access is through the Istio Ingress gateway, via the domain name `kserve.kf.iot-ngin.onelab.eu` and by using the hostname created by Knative, in the form `<isvc-name>.<namespace>.kserve.kf.iot-ngin.onelab.eu`.

In the second case, access is through the Nginx ingress via the hostname `apps.kf.iot-ngin.onelab.eu`. As the name of the inference service and of the namespace are not in the hostname, they have to be specified in the URI. The hostname is the of the form `apps.kf.iot-ngin.onelab.eu/v1/models/<isvc-name>.<namespace>`. For example, to access inference "sklearn-iris" created in the "kserve-test" namespace the URL is `https://apps.kf.iot-ngin.onelab.eu/v1/models/sklearn-iris/kserve-test:predict`. From there the request is redirected to the Istio Ingress gateway and the same service flow as with Istio ingress is then executed.

The two possible service flows are shown in Figure 17. Note that it means that at least two IP addresses are available for the ingress gateway.

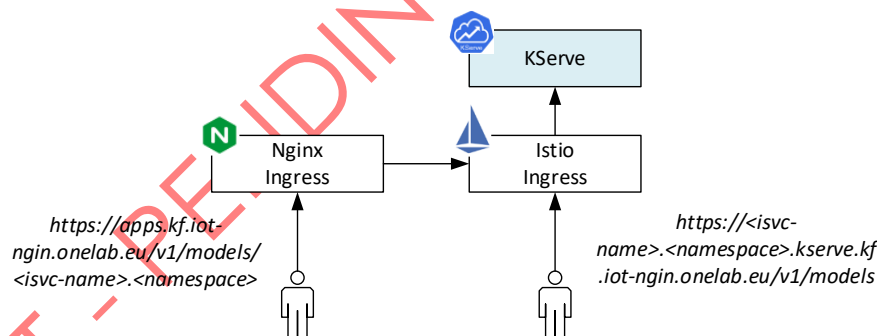


Figure 17: KServe InferenceService.

The benefit of using the Nginx ingress is that, as a fixed hostname is used, a single certificate and DNS entry can be used for all inference services. With the Istio ingress, from a DNS perspective, a wildcard DNS entry can be used that will direct all hostnames `*.kserve.kf.iot-ngin.onelab.eu` to the Istio ingress IP address. However, each inference service would need its own certificate, unless using a wildcard `*.kserve.kf.iot-ngin.onelab.eu` certificate which could be more complex to generate and required careful review as it creates significant security risks as highlighted in [10].

The Istio ingress gateway always asks for authentication before redirecting to the KServe service. So, when https is used, a token must be passed in the prediction request, for example in the form of the `-H "Cookie:authservice_session=...."` option in curl.

Listing 1 shows a prediction request via the *kserve.kf.iot-ngin.onelab.eu* hostname.

```
kserve-test>curl -v -k -H
"Cookie:authservice_session=MTY1ODMyMTMyOHxOd3dBTKZOWFUxRlNSMVExTWtGWVdrSkRWelpZTWt0SlJEV5RbFkxU1VsS
VFVSk9SMDlYVmxnelEwOURXbGczUkZaRFZqWTNTa0U9fNi0W2y5D3SwCswvHUz2zn_UeZzsxs1iJ2vxBmdTDMwK"
https://sklearn-iris.kserve-testb.kserve.kf.iot-ngin.onelab.eu/v1/models/sklearn-iris:predict -d
@./iris-input.json
* Trying 132.227.122.44:443...
* Connected to sklearn-iris.kserve-testb.kserve.kf.iot-ngin.onelab.eu (132.227.122.44) port 443 (#0)
* schannel: disabled automatic use of client certificate
* ALPN: offers http/1.1
* ALPN: server accepted http/1.1
> POST /v1/models/sklearn-iris:predict HTTP/1.1
> Host: sklearn-iris.kserve-testb.kserve.kf.iot-ngin.onelab.eu
> User-Agent: curl/7.83.1
> Accept: */*
>
Cookie:authservice_session=MTY1ODMyMTMyOHxOd3dBTKZOWFUxRlNSMVExTWtGWVdrSkRWelpZTWt0SlJEV5RbFkxU1VsS
VFVSk9SMDlYVmxnelEwOURXbGczUkZaRFZqWTNTa0U9fNi0W2y5D3SwCswvHUz2zn_UeZzsxs1iJ2vxBmdTDMwK
> Content-Length: 76
> Content-Type: application/x-www-form-urlencoded
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< content-length: 23
< content-type: application/json; charset=UTF-8
< date: Wed, 20 Jul 2022 13:48:42 GMT
< server: istio-envoy
< x-envoy-upstream-service-time: 38
<
{"predictions": [1, 1]}* Connection #0 to host sklearn-iris.kserve-testb.kserve.kf.iot-ngin.onelab.eu
left intact
```

Listing 1: KServe prediction request with kserve hostname.

Listing 2 shows a prediction request via the *apps.kf.iot-ngin.onelab.eu* hostname.

```
kserve-test>curl -v -k -H
"Cookie:authservice_session=MTY1ODMyMTMyOHxOd3dBTKZOWFUxRlNSMVExTWtGWVdrSkRWelpZTWt0SlJEV5RbFkxU1VsS
VFVSk9SMDlYVmxnelEwOURXbGczUkZaRFZqWTNTa0U9fNi0W2y5D3SwCswvHUz2zn_UeZzsxs1iJ2vxBmdTDMwK"
https://apps.kf.iot-ngin.onelab.eu/v1/models/sklearn-iris/kserve-testb:predict -d @./iris-input.json
* Trying 132.227.122.43:443...
* Connected to apps.kf.iot-ngin.onelab.eu (132.227.122.43) port 443 (#0)
* schannel: disabled automatic use of client certificate
* ALPN: offers http/1.1
* ALPN: server accepted http/1.1
> POST /v1/models/sklearn-iris/kserve-testb:predict HTTP/1.1
> Host: apps.kf.iot-ngin.onelab.eu
> User-Agent: curl/7.83.1
> Accept: */*
>
Cookie:authservice_session=MTY1ODMyMTMyOHxOd3dBTKZOWFUxRlNSMVExTWtGWVdrSkRWelpZTWt0SlJEV5RbFkxU1VsS
VFVSk9SMDlYVmxnelEwOURXbGczUkZaRFZqWTNTa0U9fNi0W2y5D3SwCswvHUz2zn_UeZzsxs1iJ2vxBmdTDMwK
> Content-Length: 76
> Content-Type: application/x-www-form-urlencoded
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Wed, 20 Jul 2022 13:51:29 GMT
< Content-Type: application/json; charset=UTF-8
< Content-Length: 23
< Connection: keep-alive
< x-envoy-upstream-service-time: 17
< Strict-Transport-Security: max-age=15724800; includeSubDomains
<
{"predictions": [1, 1]}* Connection #0 to host apps.kf.iot-ngin.onelab.eu left intact
```

Listing 2: KServe prediction request with apps hostname.



## 3 IoT-NGIN Machine Learning Techniques

### 3.1 Online Learning Framework

#### 3.1.1 Description

The Online Learning (OL) service is responsible for i) the dynamic training of ML models for IoT applications, such as those implemented for the different Living Lab apps, as data become available, and also ii) for providing inferences (i.e., predictions), from these models, when requested. It is designed to support both features on demand (i.e., through REST API) and real time (e.g., through MQTT and/or Kafka) interactions (see Figure 18).

Since most Living Labs generate data in real time, the OL service needs to be able to receive streaming data so that it processes and prepares it for machine learning models.

As discussed above, the OL service works in 2 different scenarios, depending on the request focus: streaming and REST API. However, the process only differs in receiving data and returning results. When data comes from a broker, the OL service must be subscribed to the topic where the data is being published by IoT devices and to return the result, the OL service dumps the output in another broker topic. When data comes from discrete requests, the OL service presents a REST API endpoint that is waiting for incoming requests with enclosed data in JSON format, as a common request/response HTTP service.

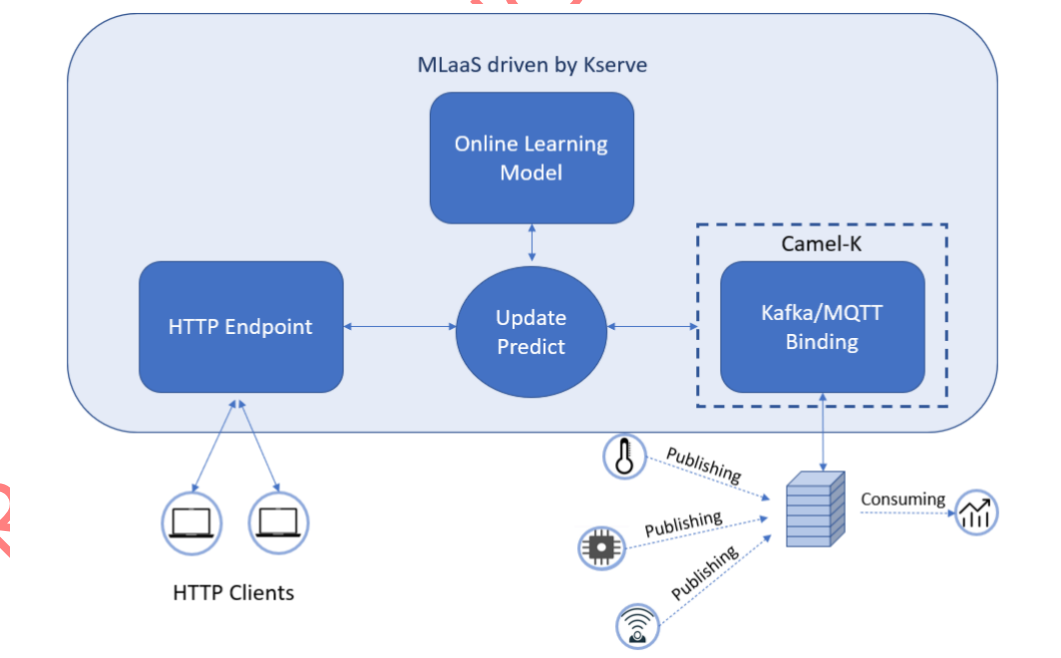


Figure 18: Online Learning Concept.

Once the data is received, the pre-processing stage begins so that the incoming data is prepared for the requested action. Depending on the needs, the pre-processed data is used for updating the model or performing a prediction. When the model has been updated, the



OL service returns a confirming ACK (acknowledgement) message. In the case of having received an inference request, the service returns its prediction.

Moreover, the updated model is saved in the cloud model storage and integrated with the *Model Sharing* service (see [2]) so that the trained model can be reused.

### 3.1.2 Technical Design

The OL service is deployed using the *Kserve*<sup>1</sup> framework through *Kubeflow*<sup>2</sup>. *Kubeflow* is utilized to deploy and execute ML workflows and *Kserve* allows to serve machine learning models on arbitrary frameworks. The ML workflow that is used to deploy the OL service through *Kserve* is declared within a *Kubeflow* pipeline. After executing this pipeline, an OL service instance is created and exposed through a HTTP REST API endpoint. Each OL service instance contains a specific machine learning model implementation, and it is waiting for receiving incoming requests, containing data in JSON format, in order to either update the model or perform predictions. IoT devices can send data to the OL service through this HTTP endpoint.

However, the data is often sent through streaming flows from IoT devices. For this reason, it is necessary to implement a binding that allows the OL service to receive and send data from/to the Kafka/MQTT broker. The binding is implemented using *Camel-K*<sup>3</sup>. Whenever new data is available from the broker, the *Camel-K* binding receives it and redirects it to the OL service REST API endpoint.

When the model is updated after a training process, the OL service may eventually decide to store it, if its performance is improved, in the cloud model storage (e.g. based on *MinIO*<sup>4</sup>). As a planned feature for M30 release, other strategies will be designed and implemented to store models based on the features they present. The OL service technical architecture design is shown in Figure 19.

---

<sup>1</sup> <https://kserve.github.io/website/0.7/>

<sup>2</sup> <https://www.kubeflow.org/>

<sup>3</sup> <https://camel.apache.org/camel-k/1.8.x/index.html>

<sup>4</sup> <https://min.io/>

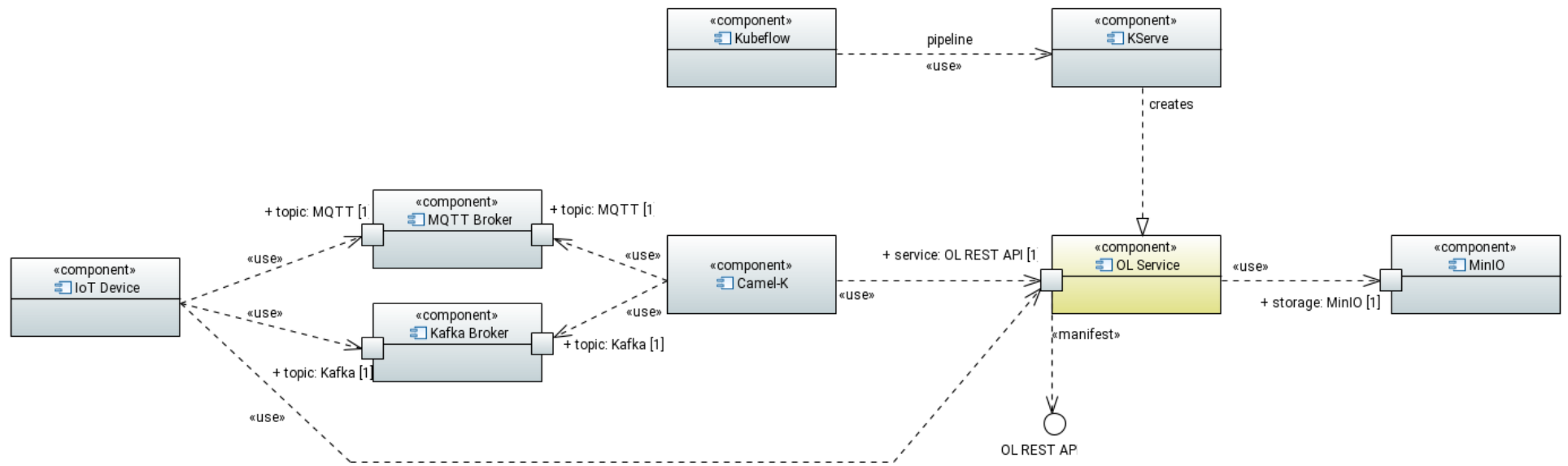


Figure 19: Online learning architecture.

As aforementioned, two main actions are supported by the OL service: i) online model training, ii) model inference or prediction. Figure 20 shows the model training process.

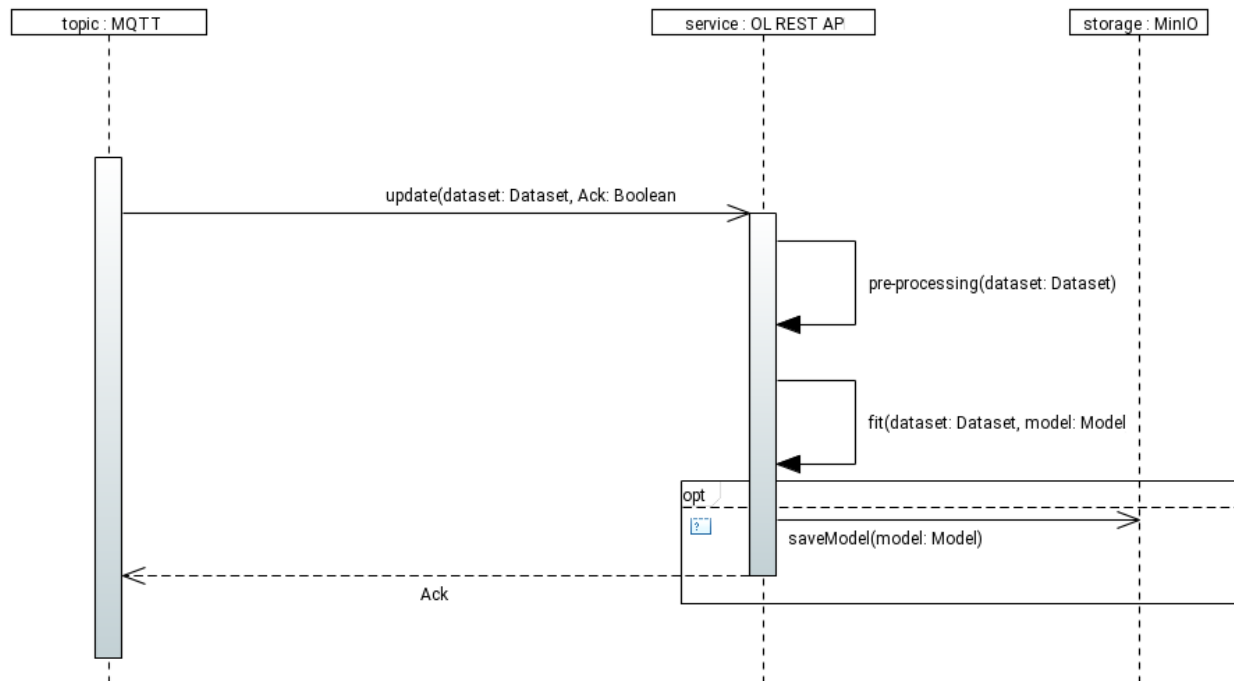


Figure 20: Model training process.

Online model training, or update model, is triggered on demand, by IoT devices or applications, either by directly accessing the OL REST API endpoint or by streaming through an MQTT topic broadcasting. In the former case, a dataset is provided in batches, while in the latter case, dataset is provided in streaming, so that the dataset batch is created by the OL service, once enough data is received. Once the dataset is available, it is pre-processed. Data pre-processing process depends on the ML model architecture to be trained and the structure of the training dataset, and it is use case specific. See section 3.1.3 for further details on dataset pre-processing for the Smart Energy LL UCs. After data pre-processing, the model is trained (i.e., fit) with the dataset, and the model performance computed. Eventually, if the gain in the model performance is over a given threshold, a trained model snapshot is stored in the shared model repository (e.g., MinIO) by using the Model Sharing service.

Model inference or prediction process is shown in Figure 21. In this case, the IoT device, application or streaming topic requests the predict API in the OL endpoint. Request can include a dataset containing the X array or being empty. If X array is given, this API will return the predicted Y array for given X. If empty, this API returns the next predicted point (in case of regression for time series). OL service uses the trained model to compute the prediction.

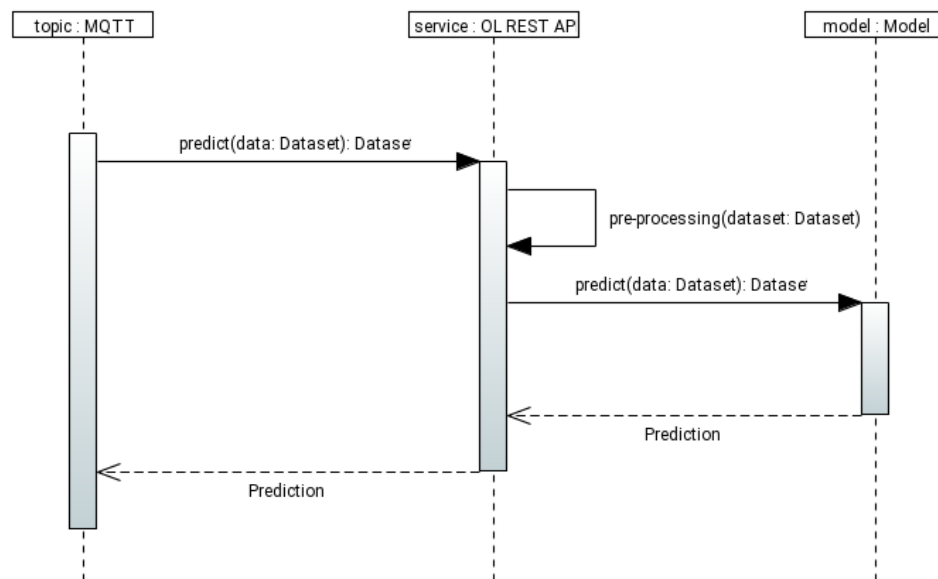


Figure 21: Model prediction process.

OL service is implemented by using the *Python* language since it offers a large ecosystem of libraries for artificial intelligence. OL service is composed of the following modules (see Figure 22):

- **Create Kserve Service.** It is responsible for deploying the REST API service with the OL service. It creates an HTTP endpoint which is waiting for new data in JSON format in order to update the model or perform a prediction. Moreover, in the first version of the OL framework, this module also contains the data pre-processing stage. Next M30 version will present an additional module which will be responsible for data pre-processing. The main library used in this module is Kserve.
- **Online Learning Module.** This API links the previous module to the Backend module. It is responsible for choosing the correct backend and transmitting the model update or prediction requests.
- **Streaming connector.** This module aims to provide tools to support real-time protocols. This module is not being used because Kserve only supports HTTP connections, but it is implemented in case future versions of Kserve start working with streaming data. The main libraries that have been used are *Kafka*<sup>5</sup> and *Paho-MQTT*<sup>6</sup>.
- **MinIO Connector.** Provides the required tools to download and upload the ML models. This version stores the ML model in MinIO storage each time it is updated by the OL service. Future versions will incorporate strategies to update the model in MinIO only when evaluation metrics improve. This module is based on the *MinIO*<sup>7</sup> library.
- **Backend.** Modules responsible for including required functions to perform ML model updates or predictions in each framework. The first version includes the following frameworks: *Sklearn*, *Vowpal Wabbit*, *TensorFlow* and *Pytorch*.

<sup>5</sup> <https://kafka-python.readthedocs.io/en/master/>

<sup>6</sup> <https://pypi.org/project/paho-mqtt/>

<sup>7</sup> <https://github.com/minio/minio-py>

The OL implementation is available at [https://gitlab.com/h2020-iot-ngin/enhancing\\_iot\\_intelligence/t3\\_2/online\\_learning](https://gitlab.com/h2020-iot-ngin/enhancing_iot_intelligence/t3_2/online_learning).

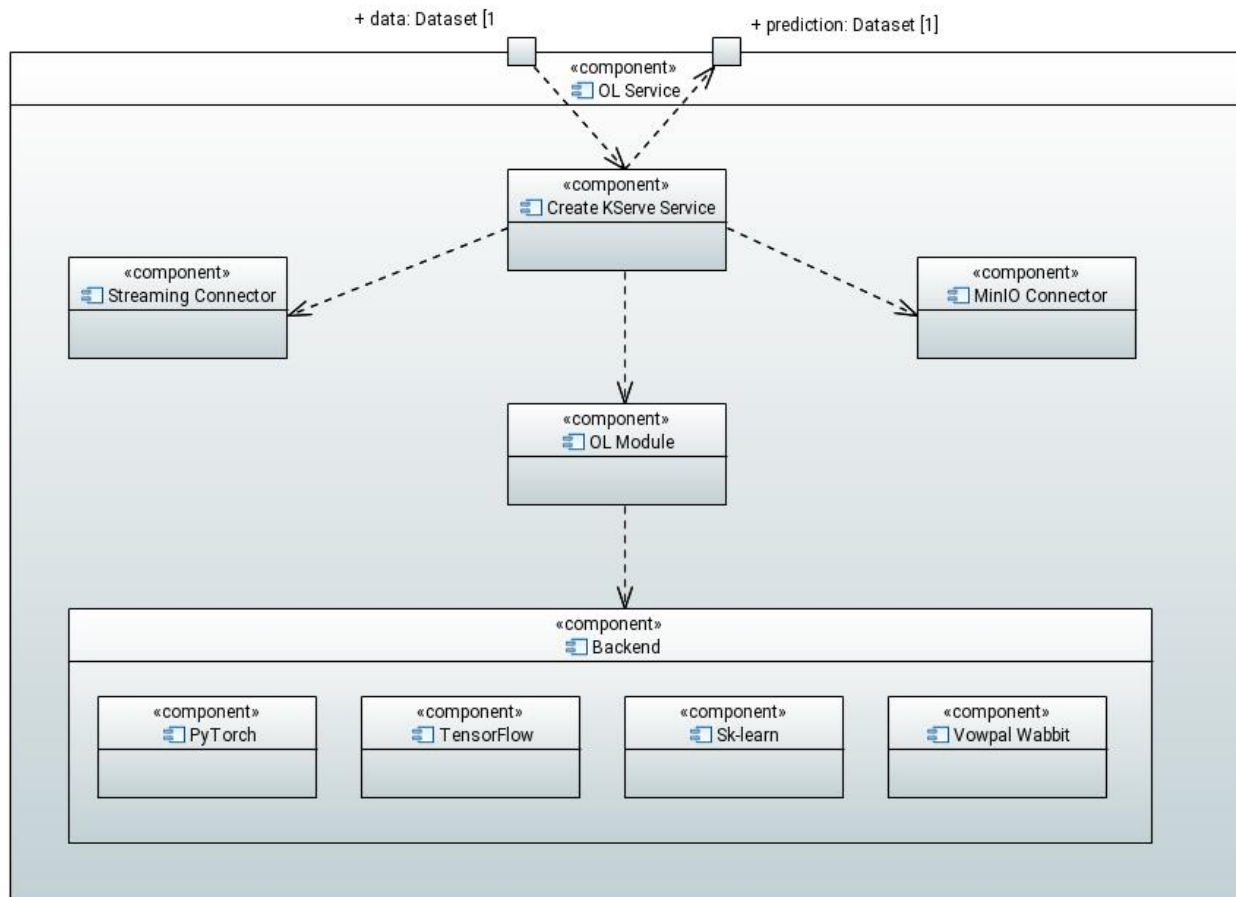


Figure 22: OL service modules.

Apart from the main OL service implementation, additional developments are also required for having the service deployed. They are listed below along with a brief description.

- 1- **OL Service Adaptation:** This is the initial step and consists of configuring the OL service to set different parameters such as the MinIO host and the buckets where the ML models are stored, the backend (framework which was used to implement the model) to use in order to perform the model update or the prediction.
- 2- **Create the Docker image:** Once the OL service is configured, it is required to wrap it within a Docker image that will be uploaded in a Docker registry so that Kubeflow can include it into the pipeline. The Dockerfile is available at [https://gitlab.com/h2020-iot-ngin/enhancing\\_iot\\_intelligence/t3\\_2/online\\_learning/-/blob/main/Dockerfile\\_Kubeflow](https://gitlab.com/h2020-iot-ngin/enhancing_iot_intelligence/t3_2/online_learning/-/blob/main/Dockerfile_Kubeflow).
- 3- **Define Kserve YAML manifest:** This manifest defines the configuration of the OL service when deployed. It defines the name of the inference service, the number of replicas, the CPU limits or the Docker image to use, among others. An example can be found at:

[https://gitlab.com/h2020-iot-ngin/enhancing\\_iot\\_intelligence/t3\\_2/online\\_learning/-/blob/main/kubeflow/kserve\\_isvc.yaml](https://gitlab.com/h2020-iot-ngin/enhancing_iot_intelligence/t3_2/online_learning/-/blob/main/kubeflow/kserve_isvc.yaml).

- 4- Create Kubeflow pipeline:** At this point, we have the Docker image ready to use and the Kserve YAML manifest that defines the OL service. The next step is to create a Kubeflow pipeline to incorporate the Kserve YAML manifest and thus be able to run it. This pipeline is developed in a *Jupyter Notebook*<sup>8</sup> available at [https://gitlab.com/h2020-iot-ngin/enhancing\\_iot\\_intelligence/t3\\_2/online\\_learning/-/blob/main/kubeflow/pipeline.ipynb](https://gitlab.com/h2020-iot-ngin/enhancing_iot_intelligence/t3_2/online_learning/-/blob/main/kubeflow/pipeline.ipynb).
- 5- Run Kubeflow pipeline:** This step deploys the OL service as an HTTP inference service. Instructions to run the pipeline in Kubeflow are summarized in section 5.2.
- 6- Define Camel-k binding (if needed):** This step is required only when data comes from real time protocols (e.g., MQTT or Kafka). Camel-K binding consists of a YAML file that defines the broker and topics in which data is being dumped and the prediction service deployed in the previous step in order to resend the data.

Next M30 release of OL service will include a number of planned features:

- Interoperability with the Polyglot Model Sharing (MS) service: the OL service will leverage the MS service REST API to store models into the MLaaS model repository (e.g., MinIO)
- KServe based multi-model serving. The OL Service architecture and its implementation will be refactored for enabling multi-model serving, by leveraging the KServe pipeline: transformer, predictor, explainer (see section 2.7.1). This pipeline enables a flexible combination of pre-post processing (with the transformer), model training and inference (with the predictor) and the generation of Explanatory AI (XAI) metadata (with the explainer) as a way to learn multiple ML models applied to multiple use case datasets.
- Prototype implementation of transfer learning, leveraging on the OL process to learn existing trained models on different learning scenarios (i.e., datasets and learning objectives). This feature could be applied to the Smart Energy LL use cases, to learn models predicting other metrics of the Electric Grid, from models predicting the generated power.
- Model saving strategies based on performance gains. Model snapshots will be stored in the MLaaS model storage when performance gains over a given threshold are achieved through the OL process. Different criteria for automatic model storage will be investigated and implemented.

### 3.1.3 Implementation for IoT-NGIN LLs

As described in D3.1, several use cases conducted in the IoT-NGIN Living Labs (LLs) have expressed the need for creating ML models for different forecasting purposes. Some of them require OL-based model training as their training datasets are fed online on streaming, like those in the Smart Energy LL. The first application of OL, described in this section, focused on the Smart Energy LL use cases, but other applications to other use cases from Smart Cities,

---

<sup>8</sup> <https://jupyter.org/>

Smart Agriculture and Smart Industry LLs will come in next M30 release for those cases where OL model training is required.

Smart Energy LL owns UC9 and UC10, namely “Move from Reacting to Acting in Smart Grid Monitoring” and “Control, and Driver-friendly dispatchable EV charging”, respectively, as described in [2]. These use cases present services that require online learning, namely the *Power Generation Forecasting*, the *Power Consumption Forecasting* and the *Energy Demand Forecasting*. They forecast the behavior of certain aspects of the electric grid. For such purposes, the grid includes metering devices that are measuring several parameters such as the power, voltage or current, among others, in distinct places of the electric grid and publishing all the collected information in a MQTT broker.

Figure 23 depicts a simplified description of the Smart Energy LL scenario.

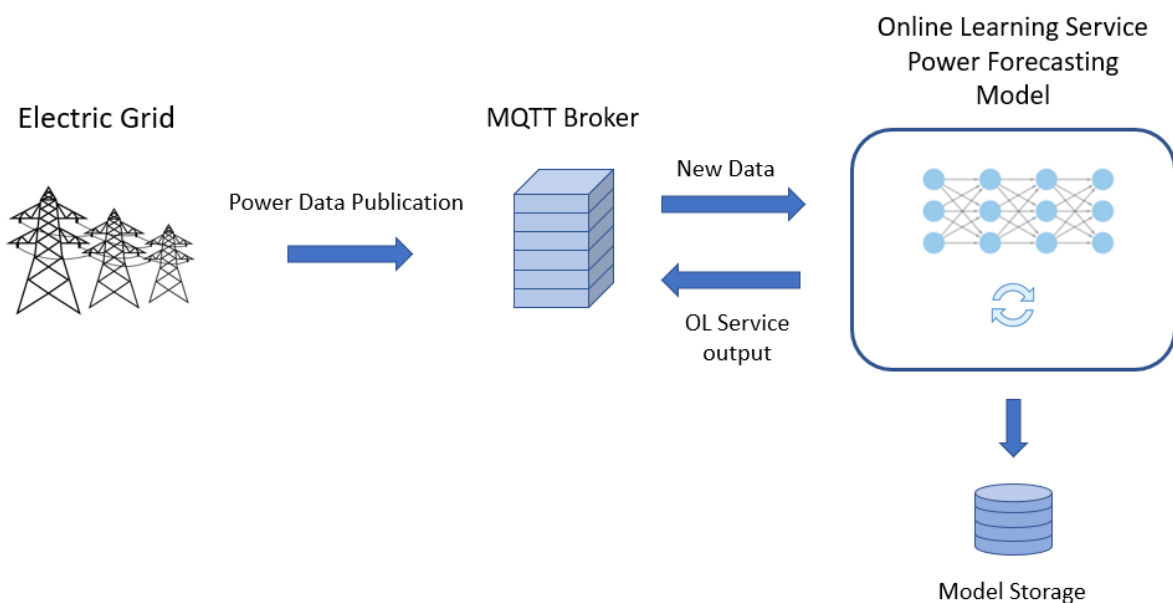


Figure 23: Smart Energy LL concept for UC9 and UC10.

The first implemented prototype of each forecasting service consumes information from one single topic, as shown in Table 6. Next M30 planned version will process more information to perform improved forecasting, such as the date/time frame or weather conditions if available, for example.

Table 6: Smart Energy LL MQTT topics for forecasting services.

Service	Description	MQTT Meter/Topic	
		UC9	UC10
<b>Power Consumption Forecasting</b>	Use power data over the time to predict the consumption in next 24-36 hours.	Smart Meter/ BBB60XX	Power Quality Analyzer/ W4
<b>Power Generation Forecasting</b>	Use power data over the time to predict the generation in next 24-36 hours.	PMU/ 3640f24ba43a 423188979372 bae6277a	Power Quality Analyzer/ W6

Once the OL service receives new data within a request, it conducts data-acquisition and pre-processing steps to prepare the data, before it is being used either for ML model training or prediction. The following procedures, applied to power consumption/generation forecasting, can be included within this stage:

- **Extraction of power value:** power time series are collected by subscribing to specific MQTT topics.
- **Resampling of the power data:** Since the sampling rate is too low, around 1 second, is needed to resample the power data by aggregating all the values received within 1 hour and computing its average power. For this purpose, the *Pandas*<sup>9</sup> library is used.
- **Data scation:** ML and DL present higher performance and stability when all values are scaled between 0 and 1. Therefore, the power data is scaled using the *max-min* scale strategy by using preprocessing functions from the *scikit-learn*<sup>10</sup> library.
- **Time series windowing:** The univariate time series forecasting algorithms take vectors as input. This step creates an input vector that contains the scaled averaged power per hour that is used to update the model or perform a prediction. The vectors are created by using different tools from *Pandas* and *Numpy*<sup>11</sup>.

Figure 24 summarizes the data preparation stage.

<sup>9</sup> <https://pandas.pydata.org/docs/>

<sup>10</sup> <https://scikit-learn.org/stable/>

<sup>11</sup> <https://numpy.org/>



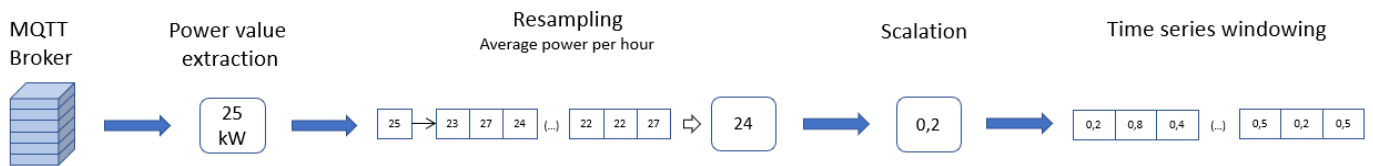


Figure 24: Pre-processing stage for Smart Energy LL OL service.

After the pre-processing stage, the data is ready to train a machine learning model. Model architectures adopted for these power forecasting services consist of recurrent neural networks (RNNs) [11]. RNNs have demonstrated to work well when facing a time series forecasting problem, although they present some disadvantages such as the vanishing gradient problem [12]. For this reason, different layers and architectures are being analyzed to reach the best possible inferences. The type of RNN used for power forecasting services is Gated Recurrent Unit (GRU) [13], since it solves the vanishing gradient problem suffered by the original RNN and converges faster than other types of RNN (e.g., Long-Short Term Memory variant). After the recurrent layers, fully connected layers [14] are added for applying linear transformations to the outputs of the GRU layer. Figure 25 depicts the layers of the two implemented architectures.

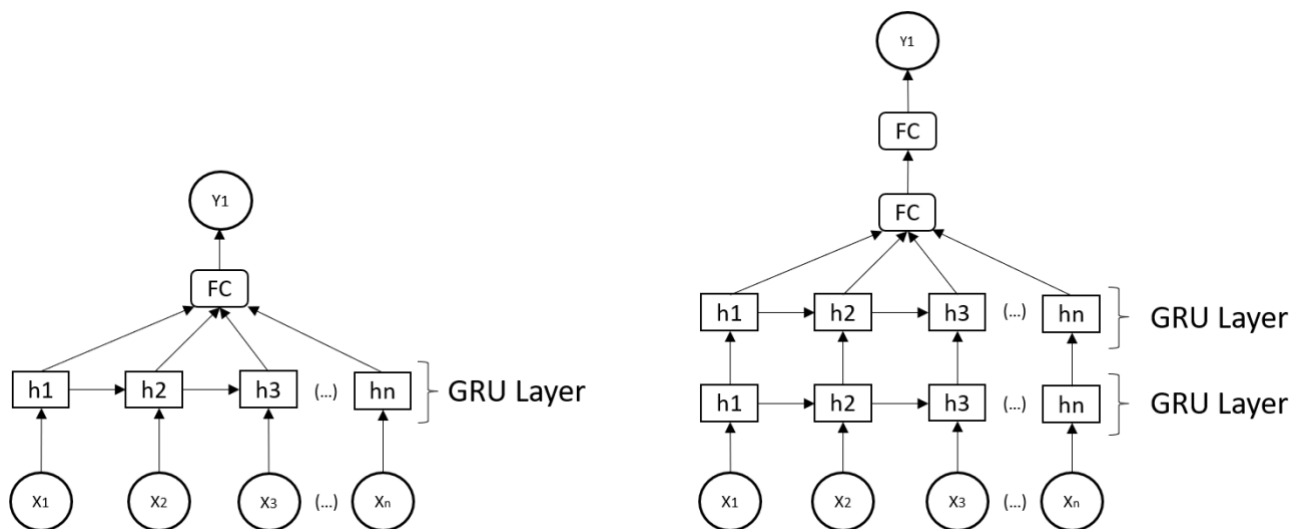


Figure 25: DL architectures' scheme for the Smart Energy LL.

These models are trained with the goal of minimizing the mean squared error<sup>12</sup> (MSE) between real values and model outputs.

The neural networks and the training procedure have been implemented with *Pytorch*<sup>13</sup> library since it provides an easy-to-use framework with a large number of tools for DL.

<sup>12</sup> Mean Squared Error (MSE) measures the average of the squared errors between the actual data and the predictions performed by the DL model.  $MSE = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$ , where n is the number of data points, and  $Y_i$  and  $\hat{Y}_i$  correspond to the actual values and inferences, respectively.

<sup>13</sup> <https://pytorch.org/>

On the one hand, when new data is available to update the model, the OL service gets notified by the broker, processes the model training with the received data, returns an acknowledgement message to confirm that the model has been updated and eventually, if a noticeable (over a give threshold) performance gain is achieved, stores the new model in the model storage deployed with MinIO.

On the other hand, the prediction of the generated/consumed power in the next 24/36 hours is requested. As mentioned above, a resampling is performed during the data preparation stage to compute the averaged generated power in 1 hour. Therefore, the service can perform a new prediction each hour.

As commented in a previous section, Kserve tool is used to deploy the OL service exposed through a HTTP endpoint. However, the service described in this section needs to receive and transmit data using the MQTT protocol. To solve this situation, Camel-K based bindings are developed to permit OL services to work with MQTT.

To verify that the selected architectures are valid solutions for this power forecasting scenario, and to set up the service with a pre-trained baseline model, generated power datasets for 20 days have been collected. A data analysis has been carried out in order to find trends seasonality and correlation between the power samples. After the analysis, it has been found out a daily seasonal component, so the power time series presents a period of 24 hours. Data analysis, models implementation and model performance evaluation have been developed in a Jupyter Notebook available at <https://gitlab.com/h2020-iot-ngin/enhancing-iot-intelligence/-/tree/3-create-smart-energy-ml-models>.

In this M24 version of OL, the training hyper-parameters present same values for all services. Table 7 shows the most significant training hyper-parameters.

Table 7: Smart Energy LL training hyper-parameters.

Hyper-parameter	Value
<b>Epochs</b>	50
<b>Learning rate</b>	0.005
<b>Optimizer</b>	Adam [15]
<b>Loss function</b>	Mean Squared Error
<b>Batch size</b>	128

Figure 26 and Figure 27 show the actual power data (orange line), inferences performed by the ML model (blue points) and the forecasting intervals with a 90% of confidence (blue area). It is important to note that the forecasting intervals can be computed since the errors between the actual data and the model predictions present a distribution that can be considered as Gaussian. To assume errors that come from the gaussian distribution, they have been subjected to normality tests: Shapiro-Wilk [16], Anderson-Darling [17], and D'Agostino-Pearson [18]. These tests consist of statistical hypothesis tests and allow checking whether the

data contains certain property. Thus, 2 hypotheses are defined: the null hypothesis and the alternative hypothesis. The null hypothesis supports that the data probably comes from a normal distribution while the alternative hypothesis defends that the data present a different distribution. The statistical test returns a probability known as  $p$ -value. If this result presents a value lower than the defined significance level (0,05 in this case), the null hypothesis must be rejected, so the data distribution cannot be assumed as normal. Table 8 shows the  $p$ -values obtained. These normality tests have been implemented by using the *Statsmodels*<sup>14</sup> and *Scipy*<sup>15</sup> libraries.

Table 8: Normality test results ( $p$ -values) for Smart Energy LL use cases.

Normality Test	UC9	UC10
Shapiro-Wilk	0.47	0.52
Anderson-Darling	0.76	0.61
Agostino-Pearson	0.10	0.16

For both use cases, the model can learn the seasonal variations that the generated power seems to have. Moreover, the inferences performed using the validation subset (data not included during training) are promising since the MSE obtained for the UC9 and UC10 are 0.009 and 0.021, respectively (see Figure 26 and Figure 27).

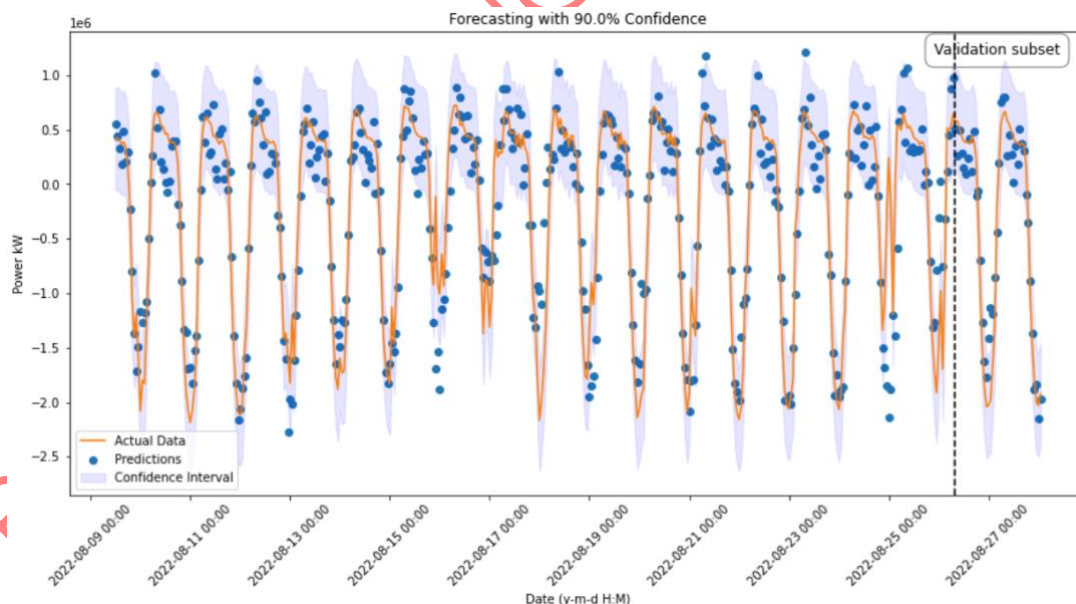


Figure 26: Training results for power generation forecasting of UC9.

<sup>14</sup> <https://www.statsmodels.org/stable/index.html>

<sup>15</sup> <https://docs.scipy.org/doc/scipy/index.html>

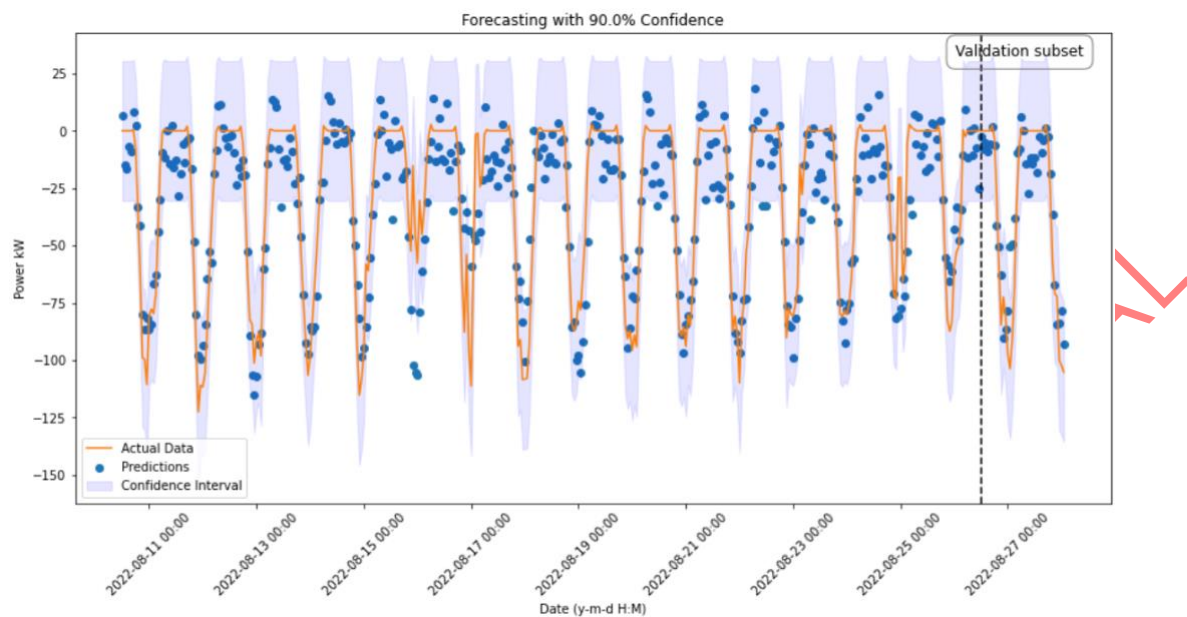


Figure 27: Training results for power generation forecasting of UC10.

These models are stored in MinIO so that the OL services can load them and perform an update when new data is being received.

The deployment of the OL service for this LL works in the same way described in section 3.1.2. OL service implementation is configured so that the service loads and saves the models in the specific MinIO buckets and uses the *Pytorch* backend to train or predict, since the models have been defined by using this library, as mentioned above. Then the OL service is wrapped in a Docker image which is uploaded to a Docker registry. Later, the Kserve YAML file is created for the service. Next the KubeFlow pipeline is created and executed; and the OL service is deployed. Finally, the Camel-K binding is created by indicating the MQTT broker host of this LL with the topics shown in Table 6.

## 3.2 Reinforcement Learning

### 3.2.1 Description

Reinforcement learning is a machine learning discipline centered on the learning of optimal behavioral policies for decision making of a group of agents interacting in a common environment, leading to a maximization of a cumulative reward (i.e., expert-defined performance metric). In the context of control systems, the learned policy allows for the deployment of deterministic or stochastic control logic / instructions for agents interacting in the end system, such as, e.g., optimizing the grid power generation, by allowing EV charging stations to autonomously adjust their charging rate depending on the current state of the grid and the attached vehicle's battery.

This discipline has been widely adopted in the automation industry, most notoriously in the field of robotics, but also in other fields such as in the energy, finance, advertising or healthcare sectors.

The main advantage of reinforcement learning is that it offers a set of algorithms that can learn independently, from their interactions with the environment, with a great deal of flexibility, in terms of the acquisition and usage of their experience, thus allowing its implementation in a wide range of environments, in both physical and simulated systems.

Some of the main concepts of reinforcement learning involve the environment, agents, states, actions, rewards, observations, and policies (see Figure 28):

- The environment refers to the physical or simulated space which the agents interact with.
- Agents are the entities which are affected by, and are in position of interacting with the environment by taking actions.
- Agents take a state (e.g., vector) that represents their status at every point of time. States are defined as a discrete or a continuous, closed set.
- A set of actions is defined for the agents to take. This group is defined as a discrete or a continuous, closed set.
- Rewards are given by the environment after the undertaking of actions by the autonomous agents.
- Observations are pre-processed snapshots (e.g., in the form of vectors) collected after each transition, that gather relevant variables of the environment, as well as the previous state and the actions taken, resulting on the state and the observed reward.
- Policies, in broad terms, are the learned (deterministic or stochastic) mapping between the set of states and the set of actions

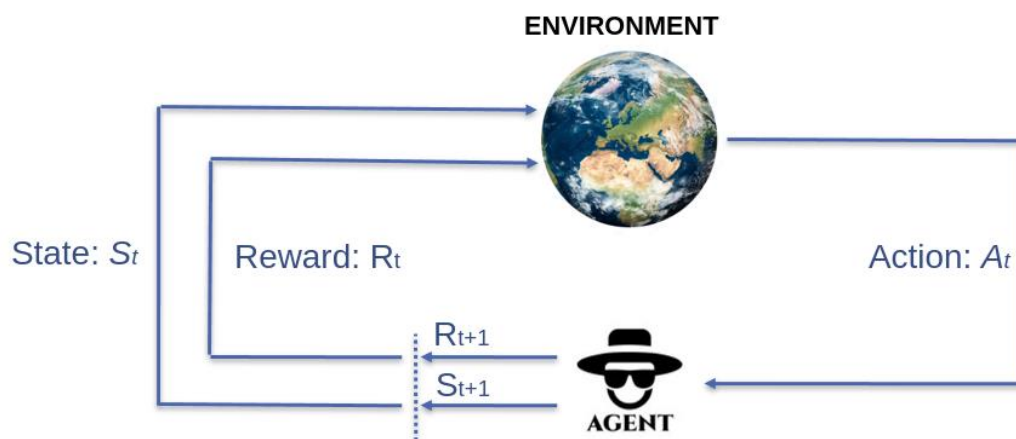


Figure 28: Reinforcement Learning concept.

The reinforcement learning service oversees the collection, generation and deployment of policies based on the observations of an environment and the interactions of its agents.

The main goal of the reinforcement learning service is to develop an optimal policy for a set of agents which interact in an environment by means of an end-to-end solution, which gathers, and processes experience obtained from interactions of agents in the environment, as well as develops and deploys the learned policy to the agents [19].

Reinforcement learning algorithms model systems in terms of the interactions of agents in an environment. The possible interactions are limited by the actions available for agents to take, which are defined on a problem-specific basis.

Depending on the problem to be tackled, experience is gathered by means of interactions in either a simulated or in a real (either digital, or physical) environment. Depending on the knowledge of the environment, we can distinguish between model-free and model-based approaches [19].

For the reinforcement algorithm to learn / approximate an optimal policy, it needs to buffer enough experience from the environment. The experience collected by the service is made up of transitions, which are made up from observations, including relevant variables from the environment, the agent previous status, the action taken, the next status and the observed reward.

In order to be able to use the accumulated experience for the training of a reinforcement learning algorithm, it needs to be pre-processed. This pre-processing step is highly dependent on the choice of the reinforcement algorithm to be implemented.

With this buffer of experience, the system can proceed with the training phase of the reinforcement algorithm of choice. Depending on the algorithm's approach, we have several categories of reinforcement learning algorithms – starting with the already mentions model-free and model-based approaches. Our service is focused on the delivery of model-free algorithms, in which we can further categorize them in policy-based and q-learning approaches [19]. Depending on the algorithm of choice, the service will handle the lifecycle of the necessary components for the algorithm development and implementation.

### 3.2.2 Technical Design

The implementation concept of the Reinforcement Learning (RL) service is depicted in Figure 29.



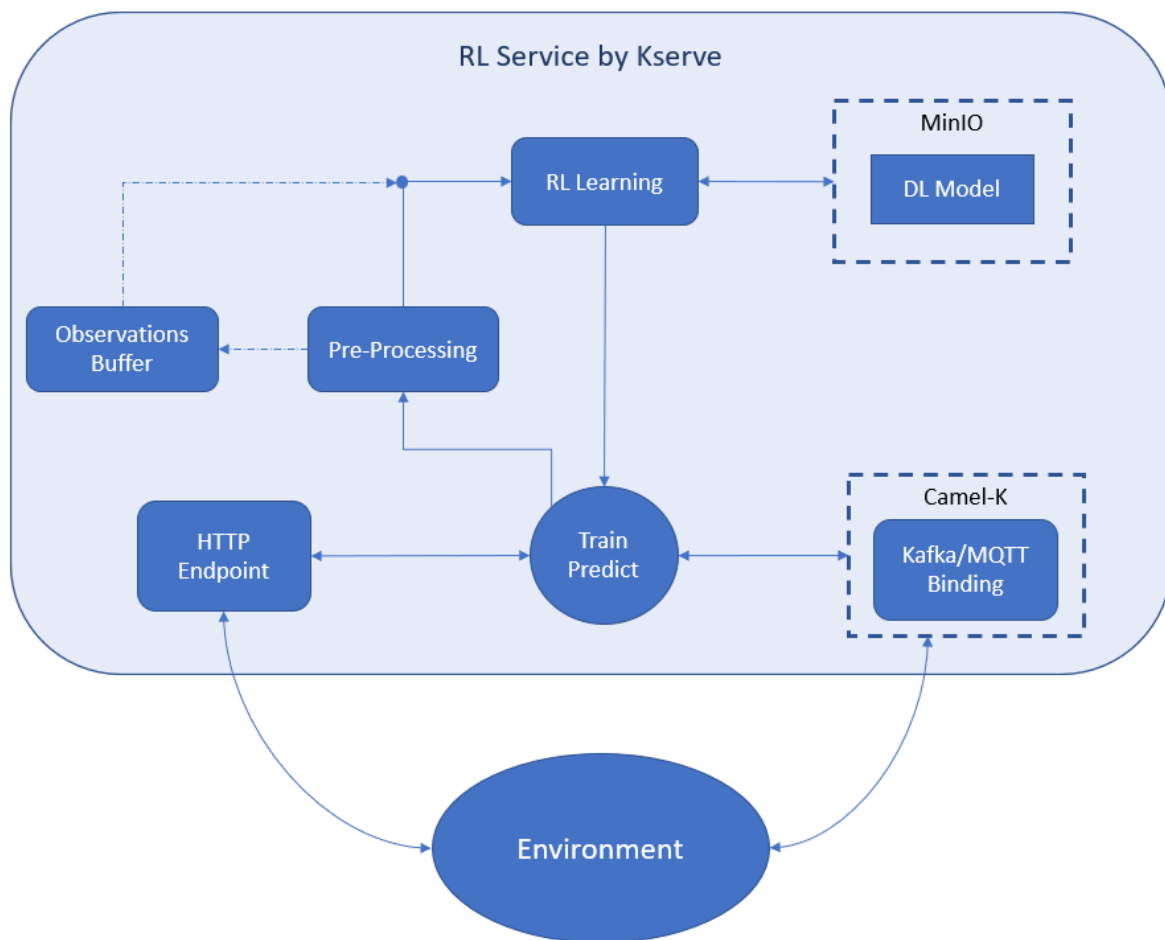


Figure 29: Reinforcement Learning architecture.

The service is deployed using Kserve through Kubeflow, like for the OL service described in the previous section. The procedure to perform the deployment is the same, the ML workflow used to deploy the RL service is declared in a Kubeflow pipeline. After executing the pipeline, an RL service instance is created and exposed through a HTTP REST API endpoint. Each instance presents a specific ML model that is trained using a determined RL algorithm (e.g., Deep Q Learning [20], Advantage Actor-Critic [21]).

The environment must send a set of parameters that monitor the agent state. These parameters are pre-processed so that the ML can generate the corresponding output and be trained. The pre-process step depends on the ML model architecture and the structure of the data, so it is subject to the use case characteristics. The model output is the action that must be performed to maximize the reward.

It is possible that environment observations come through streaming flows, so a Camel-K binding is needed to support Kafka/MQTT communications, as for the OL service.

The RL service aims to support different families of RL algorithms. A distinction of these algorithms can be made based on whether the ML model needs to interact with the environment in the training phase such as *On-Policy* or *Off-policy* RL. There may be times

when interaction with the environment is required but the environment is not available. In these situations, a simulator that mimics the environment behaviour is needed.

On the contrary, if the ML model does not interact with the environment and just uses a dataset with observations this is *Offline RL*. The RL service needs to store all the observations in a buffer so that the ML model can be trained later.

In addition, the RL service offers RL algorithms regardless of the framework used to implement the ML model. Thus, a ML model can be trained with all the RL algorithms present in the platform. In order to provide this benefit, the ML model must be stored on MinIO so that it can be loaded by the service.

The implementation of RL service, alike the OL service, is carried out with Python since this programming language presents good ML and data handling libraries. The most relevant libraries are listed below for different tasks:

- *Pre-processing*: This task needs to manipulate the data and perform transformations such as data scalation. The most relevant libraries are *Pandas* and *Sklearn*.
- *ML Train/Inference*: This task depends on the framework that have been used to implement the ML model. The RL service must support *Sklearn*, *Tensorflow* and *Pytorch* among others.
- *Model deployment*: As mentioned above, the service is deployed through Kserve, so Kserve SDK is needed.
- *Model storing*: MinIO offers a Python SDK to create a MinIO client so that the ML model can be uploaded and downloaded.

As for the OL service, apart from the RL service implementation, other developments are required for having the service deployed. The additional developments are similar to the ones needed to deploy the OL service (see section 3.1.2).

The implementation of the RL service integrated in the MLaaS platform will be released in M30 version. A concrete instantiation will be customized to the Smart Energy LL UC9/UC10 described in next section.

### 3.2.3 Implementation for IoT-NGIN LLs

Like for OL based model training, several use cases conducted in the IoT-NGIN Living Labs (LLs) have expressed the need for creating RL models for different optimization and control purposes. The first application of RL-based optimization, described in this section, focused on the Smart Energy LL use cases, but other applications to other use cases from Smart Cities, Smart Agriculture and Smart Industry LLs will come in next M30 release for those cases where RL-based optimization is required.

Smart Energy LL owns UC9 and UC10, namely "Move from Reacting to Acting in Smart Grid Monitoring" and "Control, and Driver-friendly dispatchable EV charging", respectively, as shown in [2]. These use cases present an AI-based service that requires reinforcement learning (RL) based optimization control, namely the Grid Operation Optimization, which is relevant for both UC9 and UC10.

The application of RL-based optimization requires a pre-analysis of these Smart Energy LL use cases to specify their environment, agents, states, actions and rewards (see section 3.2.1).



In the following, we conduct the analysis of the RL-based optimization scenario for the UC10, which is the one better understood at the time of writing. Analysis of the optimization scenario for UC9 will be provided in the next WP3 deliverable, D3.4.

### Environment

Figure 30 shows the environment for the UC10 optimization scenario.

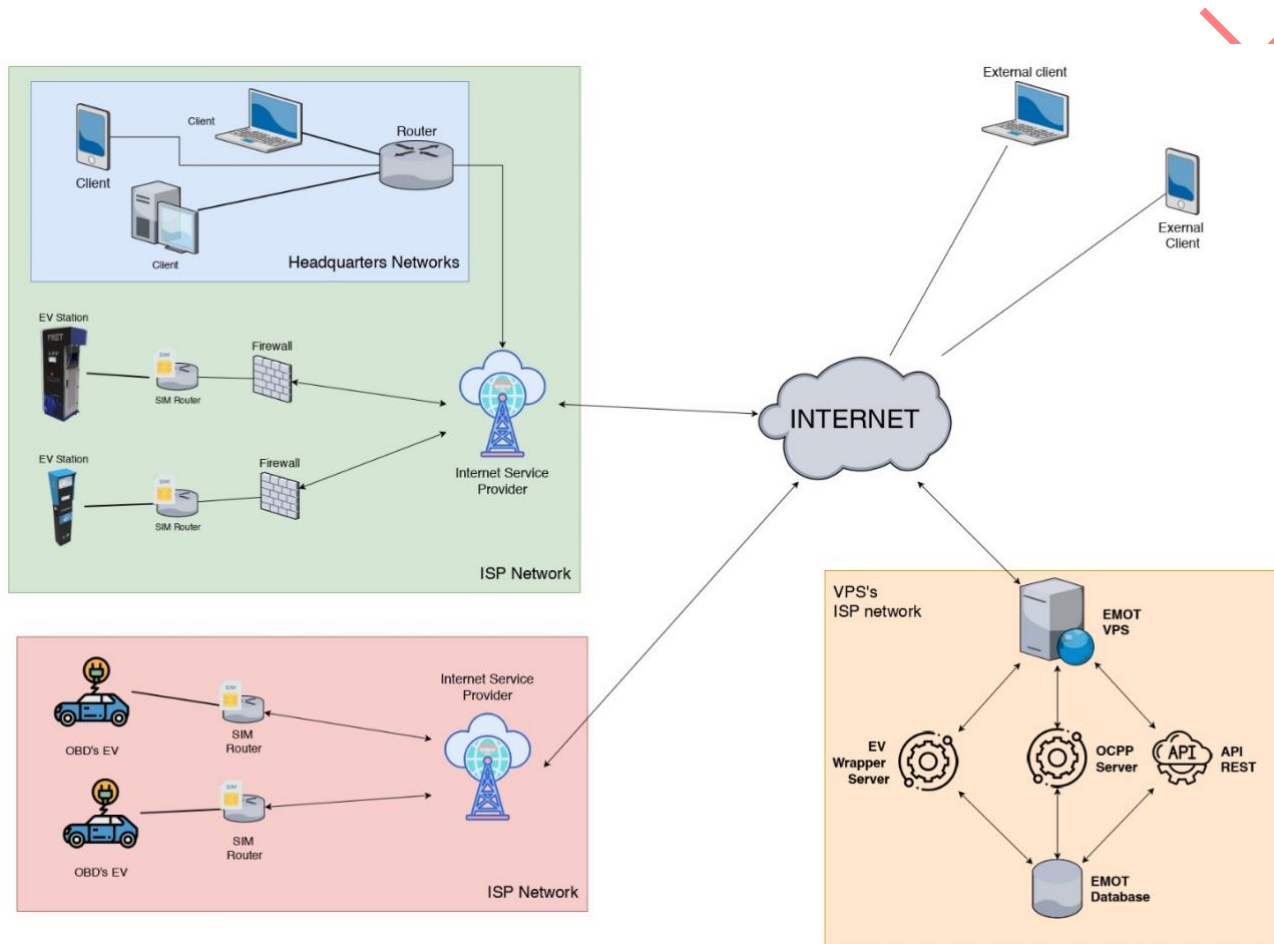


Figure 30: UC10 Environment: EMOT Network Topology.

In this representation is not included the Electric Grid (UC9) which the EV Charging Stations are connected to. In the scenario three main entities which are relevant for the RL-based optimization control are identified, as identified in Figure 31, namely:

- **Electric Grid (EG)**, provides electric power to the Electric Charging Stations (ECs) for charging Electric Vehicles (EVs). There is only one EG in this scenario,
- **Electric Vehicles (EVs)**, acting as consumers of the EG. There are multiple EVs in this scenario,
- **Electric Charging Stations (ECs)**, provide electric power to the EVs, taken from the EG, acting as mediators. There are multiple ECs in this scenario.

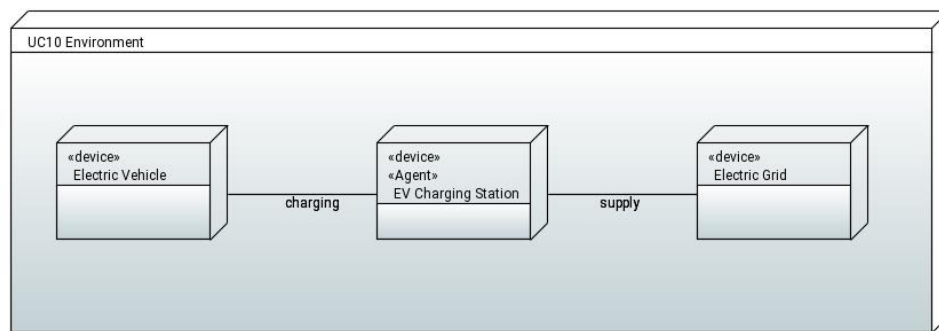


Figure 31: RL-based optimization entities in UC10.

## Agents

Among the entities in the environment, we need to identify those acting as agents. These are the ECSs. ECSs can be actioned through the EMOT VPS REST API, to instruct the charging process for EVs connected to them.

## States

EVs connected to ECSs can transition through the following states:

- **Not charged:** for simplicity this state merges into two situations: the EV's battery is not fully-charged and it is not being charged in this state,
- **Charging:** the EV's battery is being charged,
- **Fully-charged:** the EV's battery is fully charged.

The EG can transition through the following states:

- EG shows disturbances,
- EG does not show disturbances

The main disturbances EG can show are:

- **Overloads:** the power flowing in the transformers is greater than the rated power,
- **Over-voltages:** the voltage at a node is more than 1.1 times the nominal voltage,
- **Under-voltage:** the voltage at a node is less than 0.9 times the rated voltage.

Under these conditions the network can face serious consequences such as the damage of certain components, so circuit breakers must be opened to prevent this situation. UC10 considers over-voltages and under-voltages only, as there are no sensors to monitor the transformers or lines.

## Actions

The possible actions the RL-based optimization control can issue to the EGC agent are those supported by the EMOT VPS REST API. For an EGC managed by the EMOT VPS and identifiable by an ID, the following actions can be issued:

- Start charging the connected EV,
- Stop charging the connected EV,

- Change the charging rate, in percentage, in the range [0, 100]. This continuous action set can be partitioned in steps of a given delta in order to obtain a discrete action set.

Figure 32 shows the states of EV and the ECS actions that transition from one state to another.

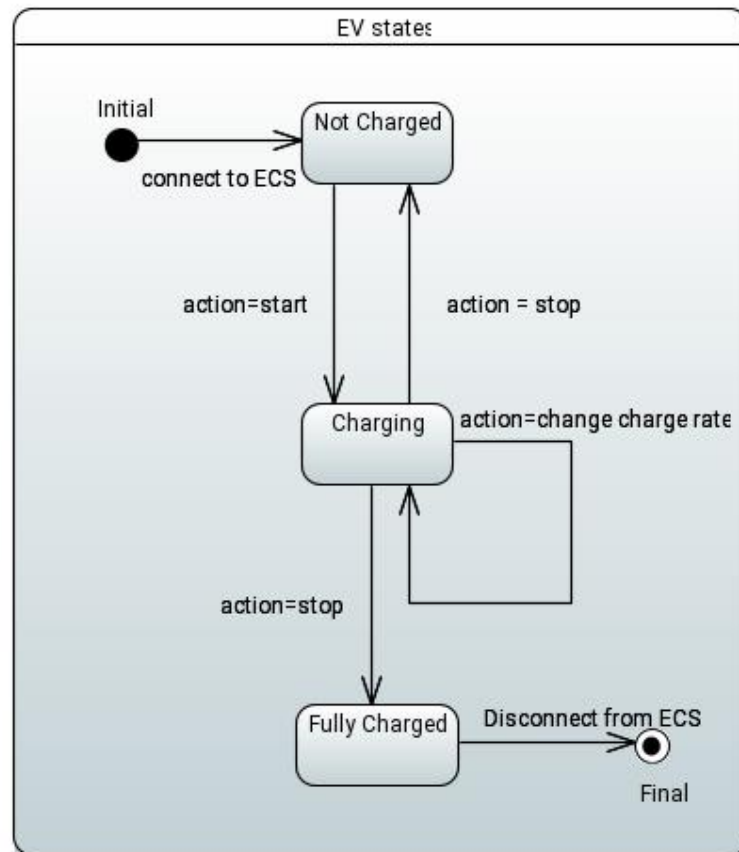


Figure 32: States and Actions for ECSs.

## Rewards

Rewards are identified for different entities in the UC10 optimization scenario.

For the EV the optimization the objective is to maximize the battery level in the lowest time. Associated rewards can be defined as:

- Positive reward  $+R_0$  if the EV battery level increases as a result of applying a given action.  $R_0$  could be computed as proportional to the battery level increase.
- Negative reward  $-R_0$  if the EV battery level remains unchanged.

For the EG, there are different optimization objectives, with associated rewards:

- Self-consumption ratio (SCR), defined as the ratio of the portion of the photovoltaic power-voltage (PV) production consumed by the loads over the produced energy ( $E_p$ ) of the PV plant. It ranges between 0 % and 100 %. The optimization objective is to maximize self-consumption.

- Positive reward  $+R_1$  if the self-consumption increases.  $R_1$  could be computed as proportional to the increment.
  - Negative reward  $-R_1$  if the self-consumption decreases.  $R_1$  could be computed as proportional to the decrement.
- Self-sufficiency ration (SSR), defined as the portion of energy produced that has been consumed, out of the total energy consumed by the utility, i.e., the absorbed energy ( $E_a$ ) and the self-consumed energy. It ranges between 0 % and 100 %. The optimization objective is to maximize self-sufficiency.
  - Positive reward  $+R_2$  if the self-sufficiency increases.  $R_2$  could be computed as proportional to the increment.
  - Negative reward  $-R_2$  if the self-sufficiency decreases.  $R_2$  could be computed as proportional to the decrement.
- Power losses, defined as the dissipated energy across the EG. It is computed by summing over the segments of the EG of the dissipated energy by Joule effect. The optimization objective is to minimize power losses.
  - Positive reward  $+R_3$  if the power losses decrease.  $R_3$  could be computed as proportional to the decrement.
  - Negative reward  $-R_3$  if the power losses increase.  $R_3$  could be computed as proportional to the increment.
- Disturbances: EG should not show disturbances. High negative reward is expected if EG shows them. A better analysis of the EG disturbances is required to include this factor in the RL-based optimization control.

The total reward  $R_T$  in a single optimization step, starting from state  $s \in S$ , selecting an action  $a \in A$ , and transitioning to a new state  $s' \in S$  can be computed as a linear function of previously mentioned individual rewards:

$$R_T = w_1 R_1 + w_2 R_2 + w_3 R_3 + f(dist)$$

where  $w_i$  are normalized weights defined as hyperparameters of the RL model, and the function  $f(dist)$  will compute the reward associated with the EG disturbances.

## 4 IoT-NGIN Privacy-Preserving Federated Learning Framework

Federated Learning (FL) has been identified as a reliable technique for distributed training of ML models [22]. Specifically, a set of dispersed nodes may collaborate through a federation in producing a jointly trained ML model without disclosing their data to each other. Instead of this, each node performs model training locally and then shares its model with a 'server' node, usually called "Aggregator" in FL. State of the art FL frameworks are in place to support federated training in real or simulated settings, as described in D3.2 [23].

Although in FL the participant nodes would not share any data, which by design protects privacy to some extent, a malicious actor could still be able to extract private information from the models or even compromise the communication during the model parameters exchange, allowing several attacks in FL, such as data or model poisoning attacks. More details about potential attacks in FL can be found in D3.2 and D5.1 [24].

Privacy preservation in FL is mainly pursued via three techniques, namely differential privacy, homomorphic encryption, and secure multiparty computation, which are analyzed in D3.2.

The IoT-NGIN Privacy-Preserving Federated Learning (PPFL) Framework is not aimed to re-invent the wheel by suggesting just one more FL framework, but rather it is aimed to facilitate access to existing frameworks, enhanced with privacy preserving techniques, allowing the AI developer to select their preferred framework with privacy guarantees.

### 4.1 Description

IoT-NGIN PPFL envisions to enable model training under Federated Learning ideally via any FL framework. In this regard, IoT-NGIN would be able to provide PPFL as a service (PPFLaaS), allowing the instantiation of FL frameworks on demand. The high-level architecture of the PPFLaaS is depicted in Figure 33. As shown in the figure, the FL API plays a crucial role in accessing desired FL frameworks, as it is consumed by external entities wishing to access FL services, such as another IoT-NGIN service, a third party-service or an external user, exposing services such as the instantiation and management of an FL framework. Assuming common specifications for the description of the FL frameworks, the API could allow the deployment of the network of FL nodes and collaborative training among them.

Without loss of generality, IoT-NGIN currently supports three state-of-the-art FL frameworks, namely NVIDIA FLARE [25], Flower [26], which for the sake of privacy preservation has been integrated with PATE (Private Aggregation of Teacher Ensembles) technique, and Tensorflow Federated [27].

In the next subsections, the three frameworks are analyzed, identifying the supported privacy preservation techniques, while in section 4.3 experimental evaluation of the frameworks under those techniques is presented in the scope of the Living Lab use cases.

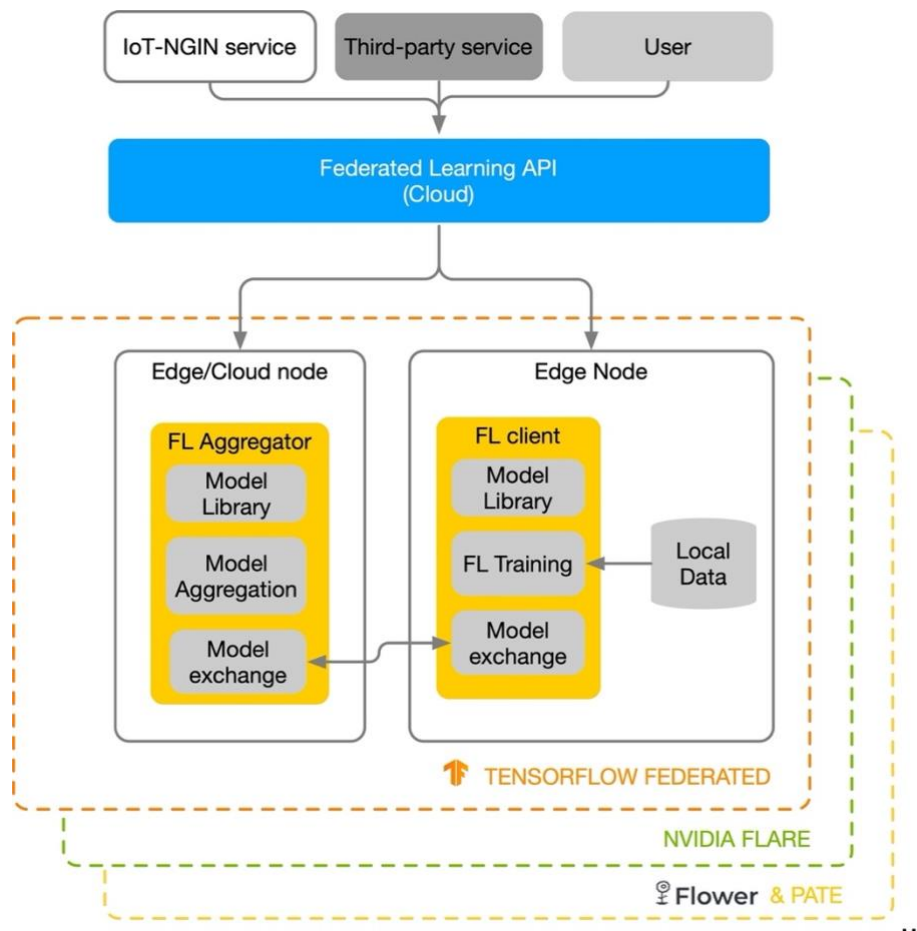


Figure 33: High-level architecture of PPFLaaS.

## 4.2 Technical Design

This section is devoted to technical details of the three different FL frameworks that are integrated and supported by the IoT-NGIN privacy-preserving federated learning framework. Details about the Federated Learning API, which describes their homogenization under a common API, will be included in deliverable D3.4 “ML models sharing and Transfer learning implementation”, due in the first quarter of 2023.

With regards to the FL frameworks, in the previous version of this deliverable, D3.2, a comparative analysis of different FL frameworks has been conducted. The analysis comprises the identification of main advantages and disadvantages for eight FL frameworks, as well as their analysis under criteria related to their FL features and privacy preservation options. Based on the outcomes of D3.2 and continuous analysis of new frameworks, the most prominent candidates for supporting the Living Lab use cases have been identified as:

- NVIDIA FLARE, which was unveiled as open source in late November 2021 and constitutes a remarkable FL framework under the support and community of NVIDIA

- Flower, which is a good candidate for real FL settings and is integrated with PATE by IoT-NGIN to support privacy-preserving FL, named after “FedPATE” by the IoT-NGIN team.
- Tensorflow Federated, which is a popular SoTA FL framework, appropriate for extensive FL simulations, which could support a variety of research tasks.

Although already existing -except for FedPATE, these frameworks deserve comprehensive analysis of their operation in the context of the use cases of IoT-NGIN interest and the application of privacy preserving mechanisms. This is due to the high diversity of potential ML applications, which requires further investigation of the effects of the elements that are involved in the FL model training on the performance under the applications of interest. Moreover, NVIDIA FLARE is rather new and FedPATE is suggested by IoT-NGIN, so our analysis provides further insights on their application and behavior under the use cases of interest.

In the following subsections 4.2.x, a technical introduction to each of these FL frameworks is presented, followed by the privacy preservation enhancements considered in IoT-NGIN.

## 4.2.1 NVIDIA FLARE

NV FLARE (NVIDIA Federated Learning Application Runtime Environment) [25] is an open-source FL framework that allows researchers and data scientists to adjust their workflows implemented in PyTorch, TensorFlow, or even just NumPy, and apply them in a federated setting. The main purpose of NV FLARE is to provide the opportunity to developers to build a secure, privacy-preserving federated system where the nodes can share model information and avoid potential malicious attacks. All participants can join the FL process across different locations and train the model with their own data.

NV FLARE mainly consists of the main node (server) and the federated nodes (clients). There is consecutive communication between them; specifically, the server is responsible for broadcasting tasks to clients (e.g., train, validation). After the clients have executed their tasks, they return the results to the server, where they are aggregated. All tasks can be implemented with filters related to privacy and security mechanisms. As shown in Figure 34, two basic entities manage the federated learning. On one hand, Figure 34 the *Controller* runs on the FL server and controls or coordinates the FL clients to get a job done. On the other hand, the *Worker* runs on FL clients and can perform tasks.



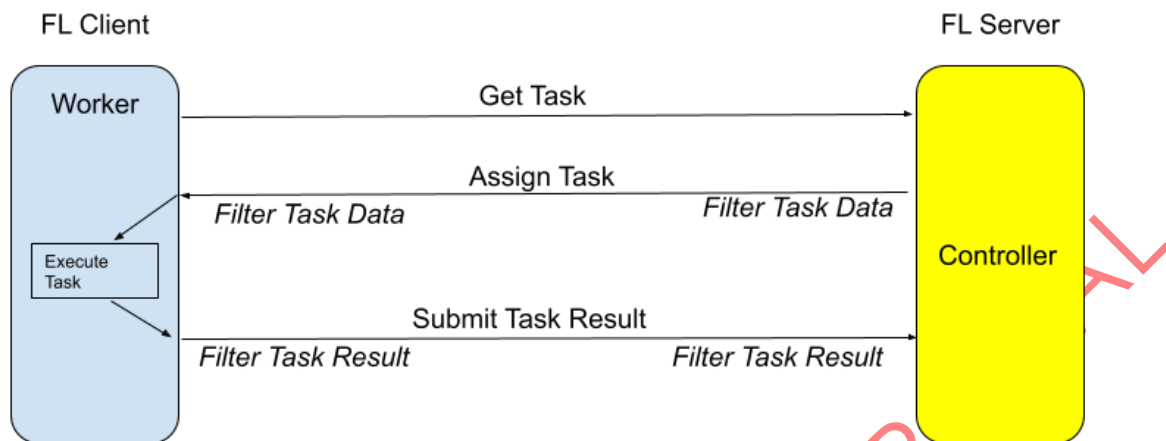


Figure 34: NVIDIA Flare Controller/Worker interactions.

In more technical detail, two configuration JSON files should be created. The first file corresponds to the server (*config\_fed\_server.json*) and contains details about the implementation of the whole process. Particularly, it can specify the model architecture, the way that the weights will be aggregated, or even the number of rounds. The second configuration file corresponds to the clients (*config\_fed\_clientr.json*). This file defines the paths of the execution files for deploying the tasks.

The message between the server and the clients is a *Shareable* object. Technically a *Shareable* object is implemented as a python dictionary. This dictionary carries meta-information about the peer identity name, the task name that the client must execute, ML model weights, etc. Responsible for executing the tasks on the clients is the class *Executor*. The *Executor* retrieves the task name, (e.g., train, submit model) and all the necessary information from the *Shareable* object and carries out the task.



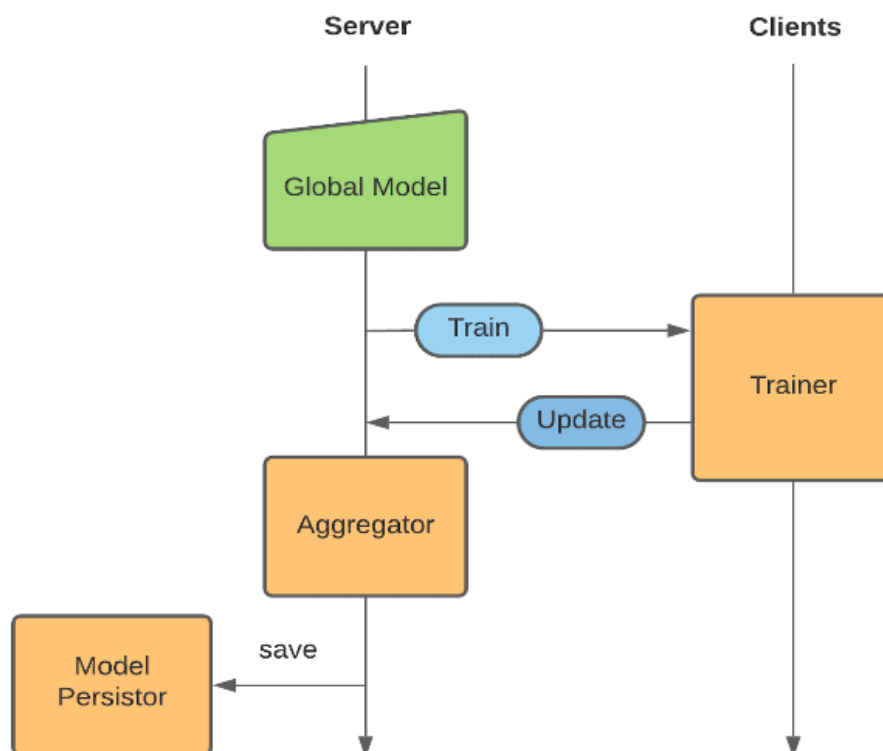


Figure 35: FL Training Workflow.

Figure 35 provides a better understanding of the FL process performed by NVIDIA FLARE. Initially, the server creates the training model architecture and sends it to the clients. Once the clients have retrieved the model, they wait for the server to inform them about the executing task. If the initial task is "training", then every client begins the training with their own data. After completing the training process, each client sends the model weights to the server. The server aggregates the weights of the clients and sends the new aggregated weights back to all clients. This process is repeated for every round. The final round follows a validation process where the server and the clients validate the final model on their data. NV FLARE provides to lead scientists/administrators 'admin' packages, that allow controlling the whole federated process, e.g., submit jobs to deploy applications, check statuses, abort / shutdown training. Specifically, the admin packages contain key and certificate files to connect and authenticate with the server, and the administration can be done through an included command prompt or through a programmatic API, namely FLAdminAPI<sup>16</sup>.

#### 4.2.1.1 Privacy Preserving Mechanisms

NV FLARE provides different types of privacy-preserving mechanisms, such as percentile privacy, homomorphic encryption, etc. The supported privacy-preserving mechanisms can

<sup>16</sup> [https://nvflare.readthedocs.io/en/main/user\\_guide/operation.html?highlight=prompt#admin-command-prompt](https://nvflare.readthedocs.io/en/main/user_guide/operation.html?highlight=prompt#admin-command-prompt)

be applied as filters when the information is sent or received between peers. These mechanisms are described in detail below.

- Exclude Vars

The “Exclude Vars” filter can be applied to the client's configuration file. The behavior of the filter depends on the input. If the input is a list of variable/layer names, only specified variables will be excluded. In the case that the input is a string, it will be converted into a regular expression, and only matched variables will be excluded. Figure 36 shows how the filter is applied in the client's configuration file; in this paradigm, the input is a string.

```

"task_result_filters": [
  {
    "tasks": [
      "train"
    ],
    "filters": [
      { "path": "filter.ExcludeVars",
        "args": {
          "exclude_vars": [
            "flatten"
          ]
        }
      }
    ]
  }
]

```

Figure 36: NVIDIA FLARE filter for exclude/remove variables.

- Percentile Privacy

Another filter supported by NY-FLARE is the one referring to “Percentile Privacy”. This filter is based on the “largest percentile to share” privacy preserving policy which is presented by Shokri and Shmatikov [28]. The main idea is that participants train independently on their own datasets and share small subsets of their models' parameters during training. The number of shared models' parameters depends on the percentile variable of the filter, which acts as a threshold. Using the “Percentile Privacy” filter, the client can control the percentile of parameters desired to be shared. Figure 37 represents the use of the filter with 20% shared model parameters.

```

"task_result_filters": [
{
  "tasks": ["train"],
  "filters": [
    {
      "path": "nvflare.app_common.filters.percentile_privacy.PercentilePrivacy",
      "args": {
        "percentile": 20,
        "gamma": 0.01
      }
    }
  ]
}
]

```

Figure 37: NVIDIA FLARE filter for Percentile Privacy.

- SVT Privacy

A differential privacy method provided by NV FLARE is the Sparse Vector Technique (SVT) [29]. This filter applies a fundamental method for satisfying differential privacy by adding noise to ML model weights. SVT takes a sequence of queries and a certain threshold  $T$  and outputs a vector  $\{\perp, T\}^\ell$ , where  $\ell$  is the number of queries answered,  $T$  specifies that the corresponding query answer is above the threshold, conversely  $\perp$  indicates it is below  $T$  [30]. This algorithm, after identifying the meaningful queries, adds standard differentially private noise from the Laplace distribution. The SVT privacy filter of NV FLARE provides gradient clipping, which is a method that acts as model regularization to prevent overfitting. SVT filter takes five parameters, namely *fraction*, *epsilon*, *noise\_var*, *gamma*, and *tau*, where *fraction* determines the fraction of the model to upload, *noise\_var* the additive noise and *gamma* is the upper bound of sensitivity of method. In the client's configuration file SVT Privacy filter can be defined with all the necessary parameters. Figure 38 shows how the filter can be used.

```

"task_result_filters": [
{
  "tasks": ["train"],
  "filters": [
    {
      "path": "nvflare.app_common.filters.svt_privacy.SVTPrivacy",
      "args": {
        "fraction": 0.9,
        "epsilon": 0.001,
        "noise_var": 1.0,
        "gamma": 1e-4
      }
    }
  ]
}
]

```

Figure 38: NVIDIA FLARE filter for SVT Privacy.

- Homomorphic encryption

Homomorphic encryption (HE) is also available on NV FLARE as a privacy-preserving choice. In HE, the clients receive keys to homomorphically encrypt their model updates before

sending them to the server. The server does not own a key, it only sees the encrypted model updates and can aggregate these encrypted weights. As soon as the weights are aggregated, the server sends the updated model back to the clients, where they can decrypt the model weights because they have the keys. The implementation of HE in NV FLARE has been performed using the TenSEAL library [31]. The way that HE can be used is shown in Figure 39.

```

"task_result_filters": [
{
  "tasks": ["train"],
  "filters": [
    {
      "name": "HEModelEncryptor",
      "args": {
        "aggregation_weights": {
          "site1": 1.0,
          "site2": 1.0
        }
      }
    }
  ]
}
]
]

"task_data_filters": [
{
  "tasks": ["train", "validate"],
  "filters": [
    {
      "name": "HEModelDecryptor",
      "args": {
      }
    }
  ]
}
]
]

```

Figure 39: NVIDIA FLARE filter for Homomorphic Encryption.

#### 4.2.1.2 NVIDIA FLARE Secure FL Infrastructure

NV FLARE performs FL with a trusted setup between remote clients through a *provisioning* tool, which helps the developer to accomplish this setup, by creating a startup kit for each site in an encrypted package. When FL starts the communication channels, it uses shared Secure Sockets Layer (SSL) certificates to create the identities and accomplish the secure communication between the participants. The provisioning tool is developed based on the Open Provision API, while NV Flare provides all the builder modules. The NVIDIA Flare provision tool creates mutual-trusted system-wide configurations for all participants. Specifically, these configurations contain the following information:

- **Network discovery:** domain names, IP addresses. The name of the servers should be in the format of fully qualified domain names. It is possible to use a unique hostname rather than Fully-Qualified Domain Name (FQDN), with the IP mapped to the hostname.

- **Credentials for authentication:** certification of participants, root authority. Creates server, client and admin certificates, and having them signed by the root Certificate Authority (CA) for secure communication.
- **Authorization policy:** roles, rules. It defines the role of each client e.g., admin, training-participant role. Also, it defines the rights of every participant, and how relaxed or restricted they should be.
- **Tamper-proof mechanism:** signatures. It creates signatures for all the files signed with the root CA for the startup kits so that they can be cryptographically verified to ensure any tampering is detected.
- **Convenient commands:** shell script with command line options to help participants. Each participant can easily join the FL process by executing the shell script provided by the coordinator.

The developers have in their possession an Open Provision API, as shown in Figure 40 to control the process and perform all provision tasks using their own requirements. With the API they can freely add/ modify or even remove the aforementioned configurations.

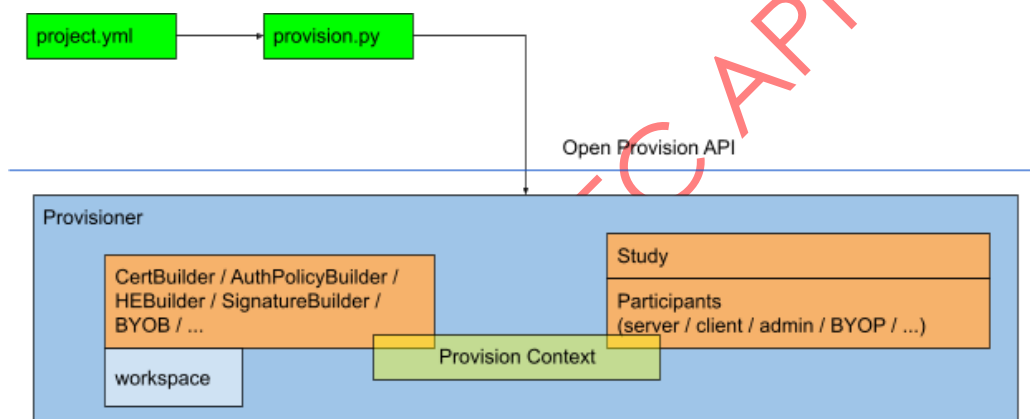


Figure 40: NVIDIA FLARE provisioning Tool Workflow.

Figure 40 describes the architecture of the Open Provision API. In the file *project.yml* developers can describe the participants and builders and store information about the implementation of the whole process. For example, the developer can define the version of the API s/he wants to use, s/he can name the project, determine the participants and the builder classes. An example of the *project.yml* file is shown in Figure 41.

```

participants:
# change overseer.example.com to the FQDN of the overseer
- name: overseer.example.com
  type: overseer
  org: nvidia
  protocol: https
  api_root: /api/v1
  port: 8443
# change example.com to the FQDN of the server
- name: example1.com
  type: server
  org: nvidia
  fed_learn_port: 8002
  admin_port: 8003
  # enable byoc loads python codes in app. Default is false.
  enable_byoc: true
  components:
    <<: *svr_comps
- name: example2.com
  type: server
  org: nvidia
  fed_learn_port: 8002
  admin_port: 8003
  # enable byoc loads python codes in app. Default is false.
  enable_byoc: true
  components:
    <<: *svr_comps
- name: site-1
  type: client
  org: nvidia
  enable_byoc: true
  components:
    <<: *cln_comps
- name: site-2
  type: client
  org: nvidia
  enable_byoc: false
  components:
    <<: *cln_comps

# The same methods in all builders are called in their order defined in builders section
builders:
- path: nvflare.lighter.impl.workspace.WorkspaceBuilder
  args:
    template_file: master_template.yml
- path: nvflare.lighter.impl.template.TemplateBuilder
- path: nvflare.lighter.impl.static_file.StaticFileBuilder
  args:
    # config folder can be set to inform NVIDIA FLARE where to get configuration
    config_folder: config

overseer_agent:
  path: nvflare.ha.overseer_agent.HttpOverseerAgent
  # if overseer_exists is true, args here are ignored. Provisioning
  # tool will fill role, name and other local parameters automatically.
  # if overseer_exists is false, args in this section will be used.
  overseer_exists: true
  # args:
  #   sp_end_point: example1.com.8002:8003

snapshot_persistor:
  path: nvflare.app_common.state_persistors.storage_state_persistor.StorageStatePersistor
  args:
    uri_root: /
    storage:
      path: nvflare.app_common.storages.filesystem_storage.FilesystemStorage
      args:
        root_dir: /tmp/nvflare/snapshot-storage
        uri_root: /

- path: nvflare.lighter.impl.auth_policy.AuthPolicyBuilder
  args:
    orgs:
      nvidia:
        - relaxed
    roles:
      super: super user of system
    groups:
      relaxed:
        desc: org group with relaxed policies
        rules:
          allow_byoc: true

```

Figure 41: Provisioning in NVIDIA FLARE by project.yml.

The python file *provision.py* [32] is a sample application to interact with the Open Provision API. The file also loads all the information from *project.yml* and instantiates classes/subclasses defined by Open Provision API. Provisioner is the container class that carries all instances of *Project*, *Workspace*, *Provision Context*, *Builders*, and *Participants*.

For reasons of brevity, the basic classes of the Open Provision API are briefly described in the following. The *Project* class keeps the information about participants, while the *Participant* class has detailed information for participants such as the name and the organization. *Builder* performs a key role in the whole process and consists of eight different classes. The main purpose of the *Builder* is to take the information from the project and the provisioner, and generate data commonly used zip files for a typical NVIDIA FLARE system, such as tensed context files for server and client. For example, *StaticFileBuilder* uses the information from *project.yml* to go through participants and write the contents of each file and replace them with the appropriate values. *HEBuilder* builds Homomorphic related content.

It should also be noted that NV FLARE, to simplify the user's interaction with the *project.yml* file, includes a UI-based provisioning tool. In this UI, as shown in Figure 42, the user can specify information about the entire provisioning process such as the rights of each participant, their names as well as the server IP, and the ports to which they communicate. After the user specifies all needed information, s/he can generate the *project.yml* file.

Figure 42: UI of provisioning tool helper in NV FLARE.

## 4.2.2 IoT-NGIN FedPATE: Flower & PATE

Flower [33] is an open-source FL framework which supports heterogeneous environments ranging from few IoT devices and mobile phones up to thousands of clients. The *Flower* framework assists engineers to integrate workflows from existing ML applications, regardless of the ML/DL framework (PyTorch, TensorFlow, etc.). *Flower* is developed to meet some design goals such as the compatibility with most state-of-the-art ML framework (PyTorch, TensorFlow, Keras, etc.), the interoperability with different operating systems, the programming languages and hardware characteristics (IoT devices, mobile phones, server, etc.) among the FL system's clients and the capability of scaling to a large number of clients. However, *Flower* does not officially provide any privacy preserving mechanisms. In the following sections, IoT-NGIN FedPATE is proposed as a Federated Learning framework which uses Private Aggregation of Teacher Ensembles (PATE) [34] for privacy preservation, integrated with the *Flower* FL framework.

### 4.2.2.1 PATE as Privacy Preserving Mechanism

PATE aims at providing differential privacy to machine learning, on the premise that if two independently trained classifiers agree on a classification decision, then the decision does not reveal information about any single training example [34]. This approach consists of a *Student-Teachers* model architecture in order to ensure the privacy of each entity's data. Multiple *Teacher* models are trained with disjoint data, potentially private such as medical records, while the *Student* model is trained with public data, in which a portion of labels comes from a noisy (Laplacian) voting among all teachers. Then, an aggregated *Teacher* takes the predictions (votes) of every trained *Teacher* on the *Student's* unlabeled data for



each sample and adds random noise, as shown in Figure 43. Then, the most voted predictions are used as labels to train the *student* model.

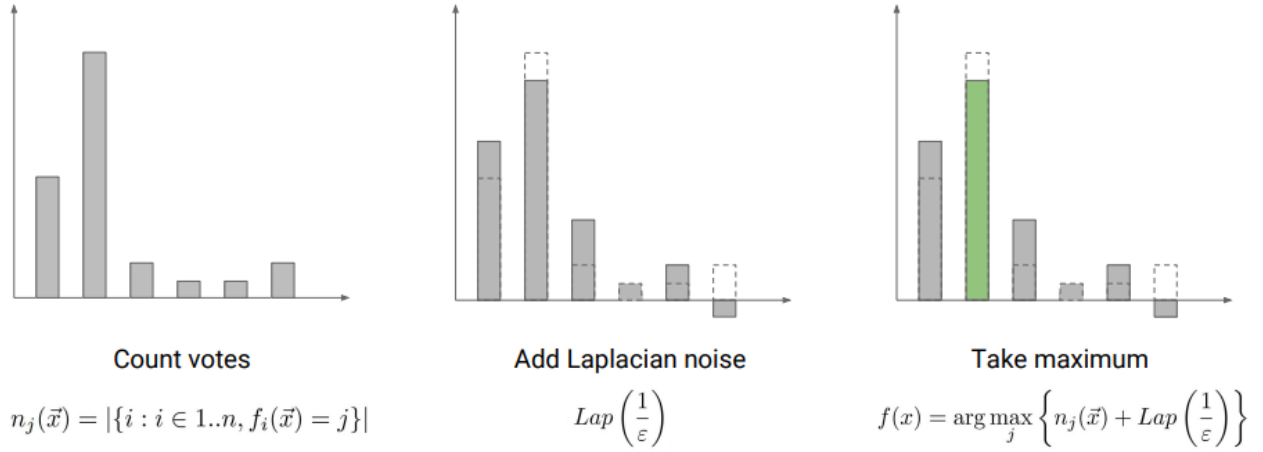


Figure 43: Overview of the noisy aggregation mechanism in PATE [35].

Thus, the *Student* model, which may later be publicly available, incorporates aggregated knowledge out of teachers' private data, based on their votes, while it preserves their privacy. The entire approach of PATE is illustrated in Figure 44.

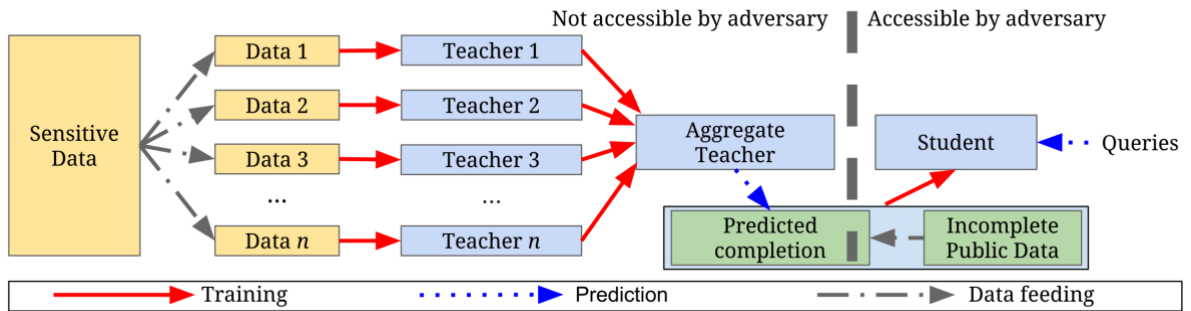


Figure 44: Approach diagram: an ensemble of teachers is trained on disjoint subsets of the private data and a student model is trained on public data labelled using the ensemble.

However, the PATE framework, originally relying on the LNMax (max-of-Laplacian mechanism) aggregator, has been evaluated only on simple benchmark classification tasks like MNIST, while its performance when applied to larger-scale learning tasks and real-world datasets was not clear. Papernot et al. [34] introduce new noisy aggregation mechanisms (e.g., Confident-GNMax, GNMax) that are more selective and add less noise (Gaussian noise) resulting in the improvement of the original PATE on all measures. For a sample  $x$  and classes 1 to  $m$ , let  $f_j(x) \in [m]$  denote the  $j$ -th teacher model's prediction on  $x$  and  $n_i(x)$  denote the vote count for the  $i$ -th class such as  $n_i(x) = |\{j: f_j(x) = i\}|$ . The new aggregation mechanism called GNMax (Gaussian-NoisyMax) is defined as [34]:

$$M_\sigma(x) \triangleq \arg \max_i \{n_i(x) + N(0, \sigma^2)\} \quad (1)$$

where  $N(0, \sigma^2)$  is the Gaussian distribution with mean 0 and variance  $\sigma^2$ .



As an extension to the GNMax aggregator, Papernot et al. [34] also propose the Confident-GNMax aggregator that enables the filtering of the labels that teachers predict. In other words, the most voted label, will be the final training label for the student if and only if the teachers have a sufficiently strong consensus. Figure 45 shows the improvement of the proposed method in contrast with the original LNMax method.

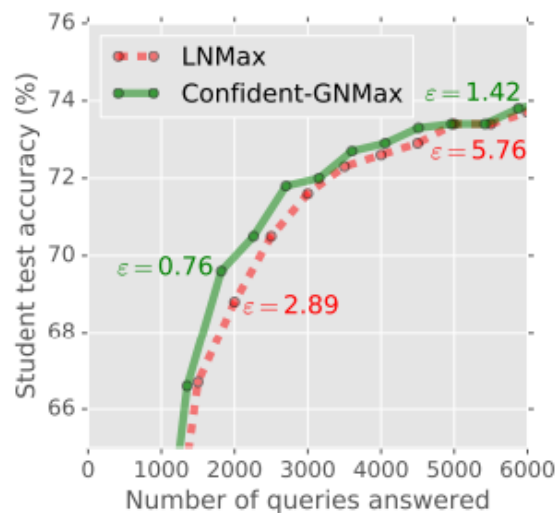


Figure 45: Accuracy is higher throughout training, despite greatly improved privacy [34].

That said, PATE constitutes a transfer learning framework, but does not consider the scenario of federated learning, which would require privacy protection against malicious participants. Yanghe Pan et al. [36] proposed a new FL framework called "FL-PATE", which combines the PATE transfer learning technique in the form of Federated Learning.

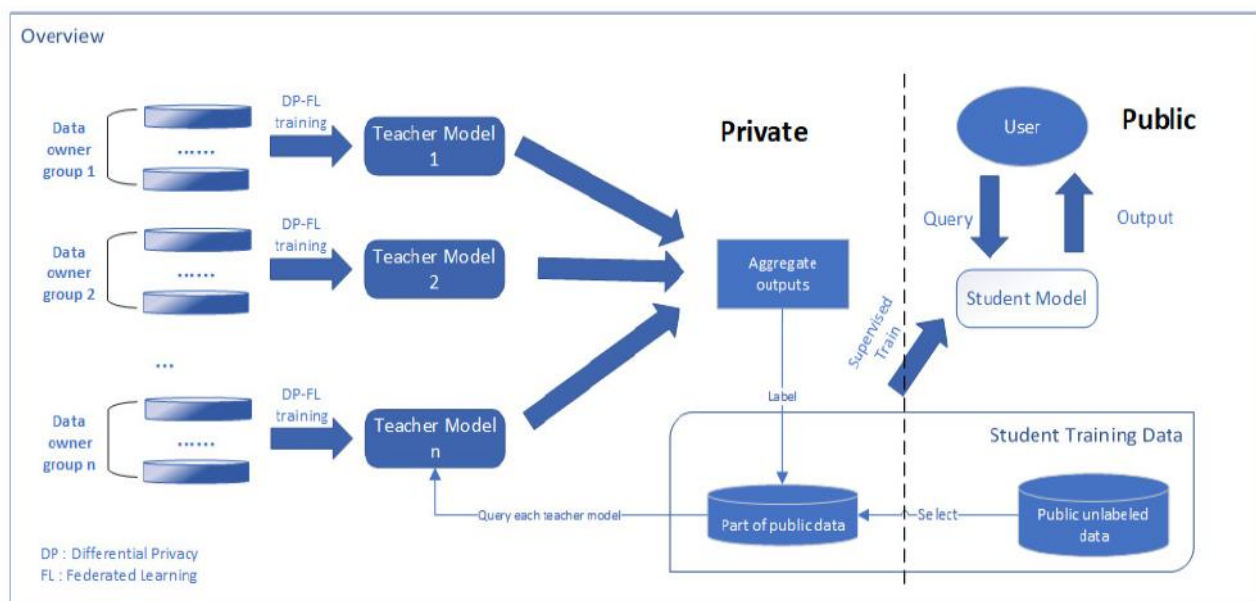
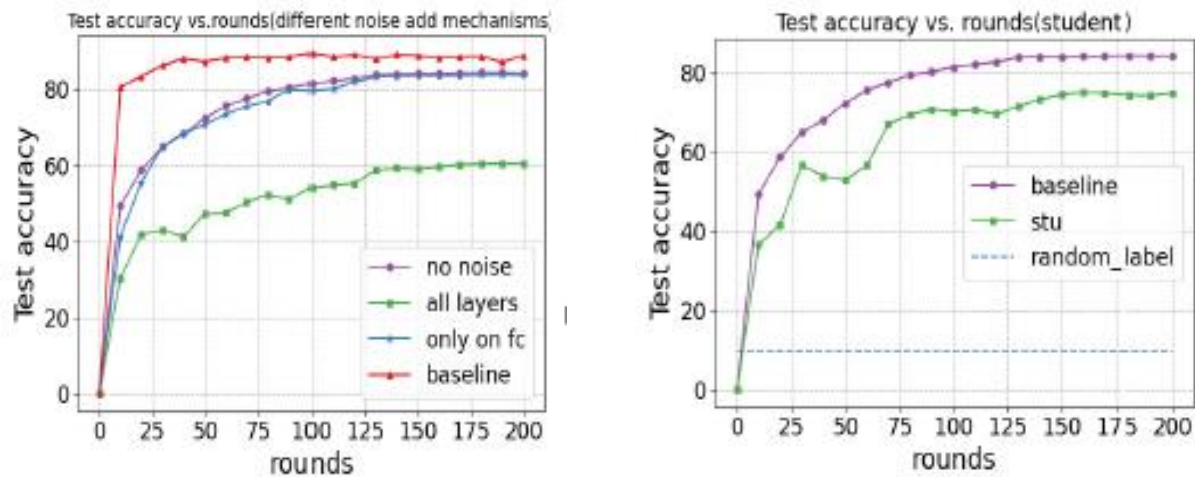


Figure 46: Overview of FL-PATE framework [36].

FL-PATE can be divided into two stages, as shown in Figure 46. In the first stage, *Teacher* models are trained via Federated Learning applied on groups of participants and then applying differential privacy by adding Gaussian noise on a special layer of the *Teacher* models (e.g., the last fully connected layer in ResNet18). In this way, the influence of the noise on the model's accuracy is significantly reduced. Then in the second stage, the *student* model is trained over the public datasets labeled by the aggregation of *teacher* models' outputs.



(a) Impact of noise addition mechanism

(b) Student's accuracy

Figure 47: Teacher's and Student's accuracy [36].

Figure 47(a) shows the impact that different noise addition mechanisms have in *Teacher*'s accuracy, while Figure 47(b) depicts the comparison between the baseline model trained via centralized training, the FL-PATE student model and the model trained by a randomly labeled public dataset in terms of accuracy.

Nevertheless, Papernot originally introduces PATE as a privacy preserving technique but the proposed FL-PATE [36] is using PATE only as a transfer learning method and not as a privacy preserving framework since it uses gaussian noise addition on the aggregated model's weights during the FL training while on the other hand PATE uses Laplacian noise addition on the most voted teachers' predictions. In the scope of IoT-NGIN, we create an FL framework by exploiting PATE as the main architecture in both transfer learning and privacy preserving technique.

#### 4.2.2.2 IoT-NGIN FedPATE

As mentioned above, FL-PATE [36] considers the federated learning scenario, in which it uses a client-level differential privacy mechanism for teacher model training. However, PATE proposes a vote-level differential privacy mechanism. We introduce FedPATE as a federated learning framework based on PATE for knowledge transfer, which also applies differential privacy. FedPATE, like PATE [37], has a teachers-student scheme. The teachers are trained under a federated learning setup. When the FL process finishes, the model of each teacher performs inference on public data of the student. These predictions on public data are aggregated, adding noise, to train the final FL Learning framework.

Our system consists of a *Flower* Server and multiple *Flower* Clients with local, private datasets. The teachers and the student behave as clients in an FL training process [38]. The *Flower* server trains the teacher models and then it is responsible for the noisy aggregation of the teachers' outputs, in order to label the student's public, unlabeled dataset.

#### 4.2.2.2.1 Model Overview

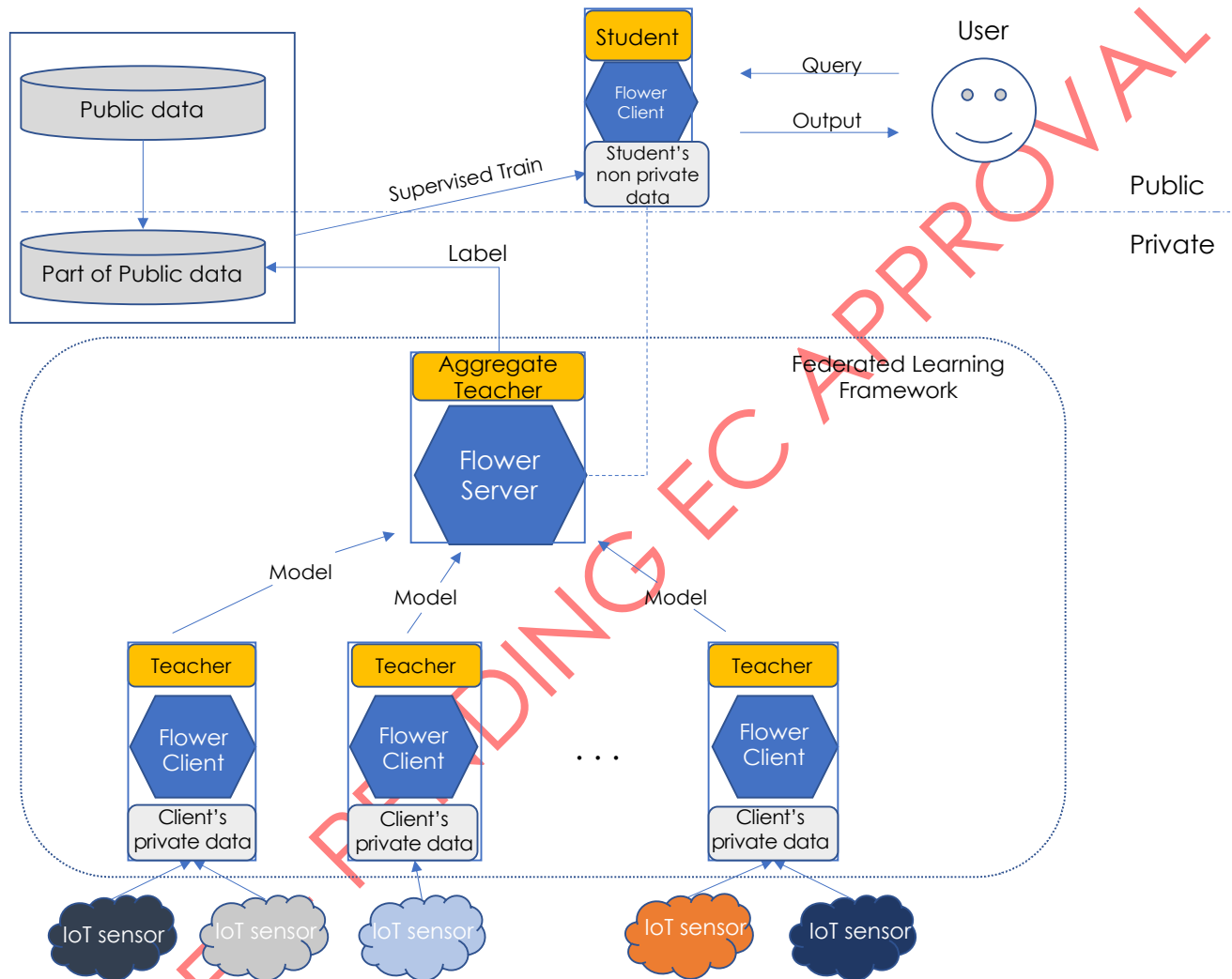


Figure 48: Overview of proposed FedPATE framework.

An overview of the design of FedPATE is illustrated in Figure 48. The goal is to train a model in a privacy-preserving way, which can be also used for complex and real-world tasks. Each teacher's model is trained at its own sensitive data, which are kept private. The final output of the whole procedure is the student model, which can be released to the public. The server trains the teachers under a federated learning setup, while the student is waiting for the teachers to finish their training process. At each round of FL, the teachers download the global model from the *Flower* Server and train it on their local private-sensitive training set by the SGD (Stochastic Gradient Descent) [39] training algorithm. Then, the teachers upload the weights  $\Delta w$  to the *Flower* Server and the global model is updated by the average of  $\Delta w$ . We use *FedAvg* [38] as the aggregation strategy.

At the final round of FL training, the teachers do not upload their model to the server. These models are rather utilized to perform predictions on student's public data. Then, the server uses unlabeled public student's data to query the teachers and performs noisy aggregation to the corresponding predictions of the teachers. Finally, the server uses these predictions to label the student's training set and train the student in the form of supervised learning. By labeling the student's training dataset, teachers' knowledge is transferred to the student. During the whole process, only the student's model is public, and the teachers are inaccessible to users. The final trained student's model is more resistant to membership inference attacks and model inversion attacks [24] [23] since the final training is done at public data and therefore does not disclose directly information about the private data of Teachers. However, it contains knowledge gained out of the private data of each teacher since the labels that participate in its training procedure arise from the predictions of teachers' models on the unlabeled data.

#### 4.2.2.2.2 Student Training

The student's training dataset comes from public unlabeled data and the server labels them by the noisy aggregation on the outputs of the teacher models on that data. FedPATE uses the same noisy aggregation mechanism as PATE [37] and it is described as follows.

Let  $c$  be the number of classes in our task. For a given class  $j \in [c]$  and an input  $x$ , the label count is the number of teachers that assign class  $j$  to input  $x$ :  $n_j(x) = |\{i: i \in [n], f_i(x) = j\}|$ , where  $[n]$  illustrates the indices of the teacher model set. The final output of the aggregated teacher is:

$$noisy\_agg(x) = \arg \max_j \left\{ n_j(x) + Lap\left(\frac{1}{\varepsilon}\right) \right\} \quad (2)$$

where  $n_j(x)$  is the label count,  $\varepsilon$  is a privacy parameter and  $Lap(b)$  the Laplacian distribution with location 0 and scale  $b$ . The parameter  $\varepsilon$  influences the privacy guarantee we can prove. Intuitively, a large  $\varepsilon$  leads to strong privacy guarantee but can degrade the accuracy of the labels.

### 4.2.3 TensorFlow Federated

TensorFlow federated (TFF) [27] is an open-source framework that supports federated computations and methods introduced by Google. It is designed to work only in simulation mode (i.e., it cannot be deployed on real world applications) and thus cannot be used to perform FL on real nodes, but it is only suitable for research and experimentation purposes. Since it only works in simulation mode, this framework is significantly faster and lighter compared to other popular choices. The architecture of this framework is depicted in Figure 49 [40].

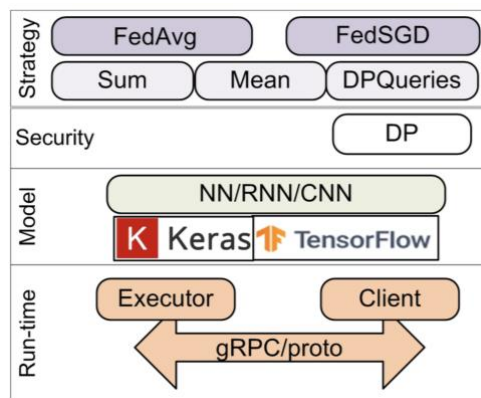


Figure 49: TensorFlow federated architecture [40].

The TFF framework is architecture agnostic since it can compile all code into an abstract representation. With that in mind, the model can be compiled in diverse environments. Moreover, it provides a series of extensions that range from privacy preserving (differential privacy) to compression of ML model variables with large values. It should be mentioned that the implementation of a federated process becomes very easy and fast.

The main components which define TensorFlow Federated are [27]:

- **Federated Learning (FL) API:** This offers a high-level set of functionalities and interfaces which enable the researchers to experiment with federated procedures without detailed knowledge of how things are working under the hood. Examples of such functionalities are differential privacy and aggregation algorithms.
- **Federated Core (FC) API:** This provides a set of lower lever interfaces which enable the ML researcher and ML engineer to express novel federated learning algorithms and methods.

Regarding the aggregation process, TFF provides many built-in aggregation algorithms, which come as functions and accept parameters as inputs. These parameters can be client or server optimizers, privacy-preserving techniques, client weighting, etc. Examples of algorithms available within TFF are *FedProx* algorithm, *federated average* and *Mime lite* [41]. However, using the module *tff.aggregators* custom aggregation of values can be implemented. It is also noted that for all aggregation algorithms, the weighted and unweighted alternatives are both implemented.

TensorFlow federated simulates the federated process with a special type of dataset of type *tff.simulations.dataset.ClientData*. This type is essentially a collection of *tf.data.datasets* each assigned to a different client. A specific number could be simply assigned to a record from the original dataset and distribute it to clients. TFF hosts multiple real-world datasets that are frequently seen in literature. Examples of such datasets are *celeba* [42], *Federated MNIST* [43], *Federated Cifar* [44] etc. Moreover, TFF datasets can be created from a *tf.data.dataset*. Even though datasets are distributed in a federated scenario, this is a very convenient way, since, in many experiments, being able to process data in a centralized manner can have many benefits.

### 4.2.3.1 Privacy Preserving mechanisms

Regarding privacy preserving, TensorFlow Federated has many options related to differential privacy. Some of the options are:

- **dp\_aggregator:** Zeroes out extremely large values for the robustness of data corruption on clients and performs adaptive clipping and addition of Gaussian noise for differentially private learning.
- **ddp\_secure\_aggregator:** Zeroes out extremely large values for the robustness of data corruption on clients, and performs distributed DP (compression, discrete noising, and SecAgg) with adaptive clipping for differentially private learning.
- **Custom:** A custom differential privacy algorithm could be implemented as a subclass of `tensorflow_privacy.DPQuery`.

## 4.3 Implementation for IoT-NGIN LLs

Considering the diversity of the ML applications that may be supported through the selected FL frameworks and the diversity of their performance and privacy requirements, as well as the fact that NVIDIA FLARE is at a nascent age, the FL framework performance is still not clear for every application. Therefore, the three selected frameworks are investigated in the context of representative cases, which could provide insights on the use of the FL frameworks of interest, at least for the -but not limited to- the IoT-NGIN LL UCs, as tabulated in Table 9. This section presents the implementation details of integrating relevant ML models in the FL frameworks and the results of their experimental evaluation.

Table 9: Summary of AI services and features experimented for the 3 FL frameworks.

FL Frameworks	NVIDIA FLARE	IoT-NGIN FedPATE	TensorFlow Federated
AI service	Object detection	Image classification	Tabular data classification
Privacy Preservation technique	Percentile Privacy, Homomorphic Encryption, both	Differential Privacy (PATE)	Differential Privacy
Data heterogeneity	yes	yes	yes

### 4.3.1 NVIDIA FLARE

NVIDIA FLARE ecosystem is used to train and evaluate a ML-based object detector under different scenarios. Specifically, in the context of IoT-NGIN, an object detection system, capable of recognizing obstacles, is implemented. For the experiments reported in this deliverable, the YOLO v5 (You Only Look Once) [45] is used as the ML model for object detection, which it is trained to detect objects of the class person of the COCO dataset [46].

The main goal is to create a federated learning system, in which the training process is implemented by clients. Every client trains the object detection algorithm with its own data



and sends the training results to the server, which concatenates the training weights and creates the final trained model. Moreover, experiments are conducted on various scenarios, and the performance of the federated ML model analyze on different cases. A detailed description is given in the following.

#### 4.3.1.1 Experimental Setup

YOLO [47] is one of the most famous object detection algorithms that divide images into a grid system. Each cell in the grid is responsible for detecting objects within itself. YOLO is renowned for speed and accuracy. There have been many versions of YOLO, and in every version the performance is improved. The experiments of this deliverable are conducted with YOLOv5 [45], which is a robust version of YOLO series released as open-source code.

This version provides various YOLOv5 models, that are compound-scaled variants of the same architecture. Specifically, very small models can give real-time FPS on edge devices, while very large models are more accurate and meant for cloud GPU deployments. The following is a short description of each variation:

- YOLOv5n: The nano model is the smallest of all models, and the model's capacity is around 4 MB. It is ideal for mobile solutions.
- YOLOv5s: The small model with around 7.2 million parameters. This model is more accurate than the nano model and it's ideal for IoT devices.
- YOLOv5m: This is the medium-sized model with 21.2 million parameters. It's a good choice for many applications since it provides a good balance between speed and accuracy.
- YOLOv5l: The large model of the YOLOv5 family with 46.5 million parameters.
- YOLOv5x: It is the largest model among the 5 with 86.7 million parameters. This model can accomplish the highest mAP (Mean Average Precision) among the 5 but is much slower.

All the models mentioned above are trained on the COCO dataset with 80 different classes for 300 epochs. Moreover, all the trained models are available and can be used as pre-trained modes to speed up the training procedure.

The purpose of our object detection algorithm at an early stage was to detect humans. So, the next step was to find the appropriate dataset. The available open-source object detection datasets that include person class are Pascal Visual Object Class (VOC) and COCO.

#### **PASCAL VOC**

PASCAL VOC [48] is a reference dataset in object detection task, consisting of approximately 12 thousand images annotated with object bounding boxes. Specifically, it contains more than 28 thousand annotations for 20 basic classes, including the class "Person". The first version of this dataset was introduced in 2005, containing only 4 categories. The fixed final number of 20 classes was established in 2007, while the last and complete version of this dataset was presented in 2012. It should be mentioned that the PASCAL VOC dataset does not exhibit to systematic bias, such as images with centered objects, good illumination, non-occluded objects, etc.

This dataset provides challenging images and high-quality annotations for all classes. Regarding the class “Person”, PASCAL VOC dataset contains images that person is involved in a wide range of activities, such as walking, riding bikes, sitting on buses, etc. Figure 50 illustrates images with the bounding boxes of objects belonging to class person.

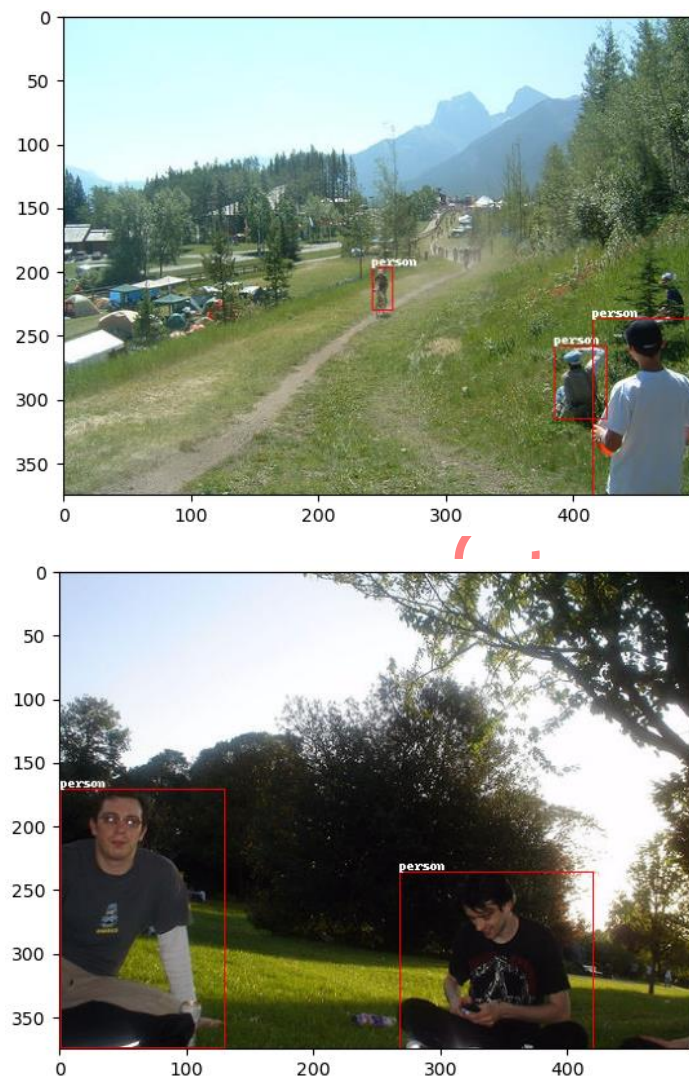


Figure 50: Annotations of PASCAL VOC dataset

### COCO

COCO is a large-scale object detection dataset that approaches core analysis problems such as detecting non-iconic scenes of objects, contextual reasoning within objects, and accurate 2D localization of objects. The dataset consists of 1.5 million object instances, 80 object classes, object segmentation, and more. Additionally, it should be mentioned that YOLOv5 is already trained on this dataset with satisfactory performance.

The annotation of people is done in detail. Specifically, bounding boxes are annotated for each individual person of the image, even in the case of many people. Figure 51 shows images with the bounding boxes of objects belonging to class person. The left image is the annotation of a single person, while the right one is the annotation of a group of people.



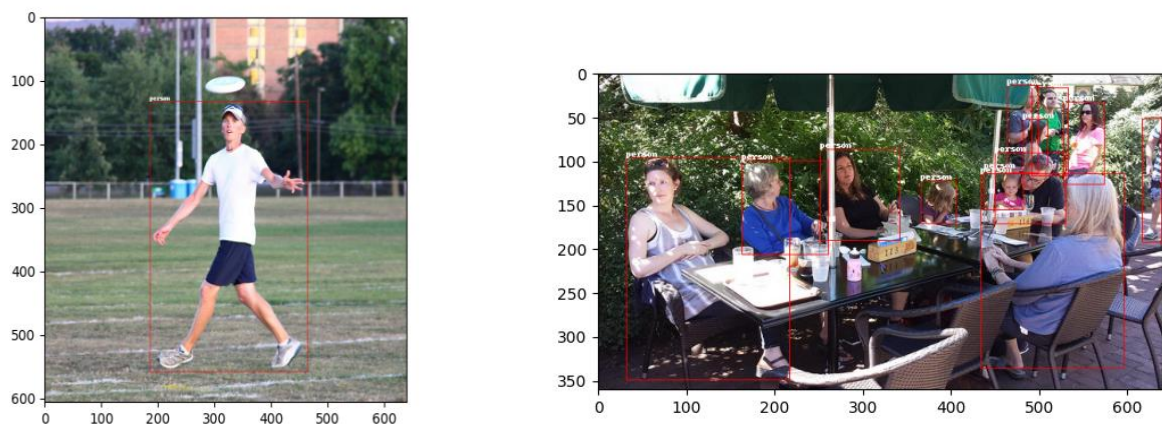


Figure 51: Annotations of the MS Coco dataset.

In the scope of IoT-NGIN, the **YOLOv5s** model is selected from the YOLOv5 series since it has a small architecture, that can easily be trained on IoT devices. This model is based on pre-trained model on the **COCO** dataset and further FL training and evaluation on appropriate test and evaluation datasets based on the VOC dataset. To achieve this, VOC images containing people should be selected, while their corresponding files should be converted to the suitable format.



Figure 52: YOLOv5 annotation format.

YOLOv5 has a specific ground-truth annotation format, which is visualized at Figure 52. Particularly, YOLOv5 requires for every image one txt file, which includes one row for each object. The object is described with the format of class  $[x\_center \ y\_center \ width \ height]$ , where the fields are space-delimited, and the coordinates are normalized from 0 to 1.

However, the annotations of the VOC dataset are described in different format. The objects of each image have four coordinates  $[x\_min, y\_min, x\_max, y\_max]$ , describing the two edges of bounding box. Thus, the annotations of VOC are converted to YOLOv5 format. The open-source tool, introduced at [49], is used to download images containing the object of

the class person, as well as the corresponding ground truth files with the format that YOLOv5 requires.

The most popular metric to evaluate the performance of the object detection models, such as YOLOv5, is the mean Average Precision (mAP). The mAP is calculated by the combination of the Intersection over Union (IoU) and Precision-Recall curve. Initially, the IoU is used to determine the deviation of the predicted bounding boxes with the ground truth and the calculation of the Precision and Recall. Then, it follows the plotting of the Precision-Recall curve, where the Average Precision (AP) is calculated by taking the area under the curve by segmenting the recalls. So, the mAP is the average of the AP calculated for all the classes and is defined:

$$MAP = \frac{\sum_q^Q AveP(q)}{Q} \quad (3)$$

Where Q is the number of queries in the set and the AveP(q) is the average precision (AP) for a given query, q.

The training of **YOLOv5s** model is performed at the NVIDIA FLARE framework. Each client has its own data from the VOC dataset. The pre-trained weights that YOLOv5s provides are used to accomplish better results in fewer epochs. NVIDIA Flare provides two modes for ML model training, which are the POC (Proof Of Concept) and Secure FL workspace using the provisioning tool. Initially, part of experiments done in POC mode. POC is a FL Simulator, coupled to a set of tools used to deploy and manage production workflows. The benefit that this mode provides is that the developers can test the FL locally and evaluate the performance of the process. All the experiments were performed with 3 clients in total to test how each one reacts to the FL process. Also, a part of the experiment was implemented with Provisioning. Provisioning provides a secure FL infrastructure which generates mutual-trusted system-wide configurations for all participants so all can join the NVIDIA FLARE system across different locations as described in Section 4.2.1.2. At Secure FL workspace mode, we tested a combination of privacy-preserving mechanisms in order to analyze their integrity.

#### 4.3.1.2 Experimental Results

The purpose of the performance evaluation is, firstly, to ensure that the integration of the YOLO v5 model with the federated learning framework run properly and, secondly, to evaluate the performance of the FL trained model (i.e., the mAP) under different scenarios, such as various privacy preserving mechanisms and data heterogeneity. Regarding the privacy preserving techniques, the main purpose of experimenting these scenarios is to explore the possibilities of each technique as well as the trade-off between privacy level and model performance. Additionally, the best parameters for every privacy-preserving technique are determined in order to minimize the trade-off between the performance of the model and privacy. Concerning the data heterogeneity, three different splitting schemes, such as mild, moderate and intense, are examined to define the effect of unequal data splitting among clients.

##### Centralized Learning

Initially, in the first scenario, the YOLOv5s model is trained and evaluated, simulating a centralized training baseline. To accomplish this, only one node is used, in which all the training samples are located. The training is done for one federated round and 20 epochs.

The performance of the centralized model is used as a comparison measure and the performance of each subsequent experiment is compared with it.

The second scenario regards the training of the ML model in a federated setup. Specifically, 3 clients are established to train the YOLOv5s model based on private data. The training is performed at 4 federated rounds of 5 epochs each. Table 10 lists information about those experiments.

Table 10: Scenarios of centralized and FL training.

Run ID	Description	Number of clients	Rounds	Epochs
Centralized	Without FL	1	1	20
FL	With FL	3	4	5

Table 11 shows information about the size of the dataset and batch size in centralized and FL training. The batch size is decreased from 64 to 16 in FL training, as it was not possible to train 3 clients with more batch size due to limitations in processing resources. In the centralized mode, the client contains the whole dataset, while in the FL setup, the dataset is split equally among each client.

Table 11: Information about batch size and split of dataset.

Epochs	Batch size	Total size	Training dataset	Testing dataset
Centralized	64	3900	3.500	450
FL	16	1166	1000	166

In Figure 53 visualizes the performance of the centralized model (left image) as well as the performance of FL training with 3 clients (right image). It is observed that the performance from the first epoch is high enough because of pre-trained weights. The final mAP reached 86.0% in both cases, and it is deduced that the FL training does not affect the performance of the ML model.

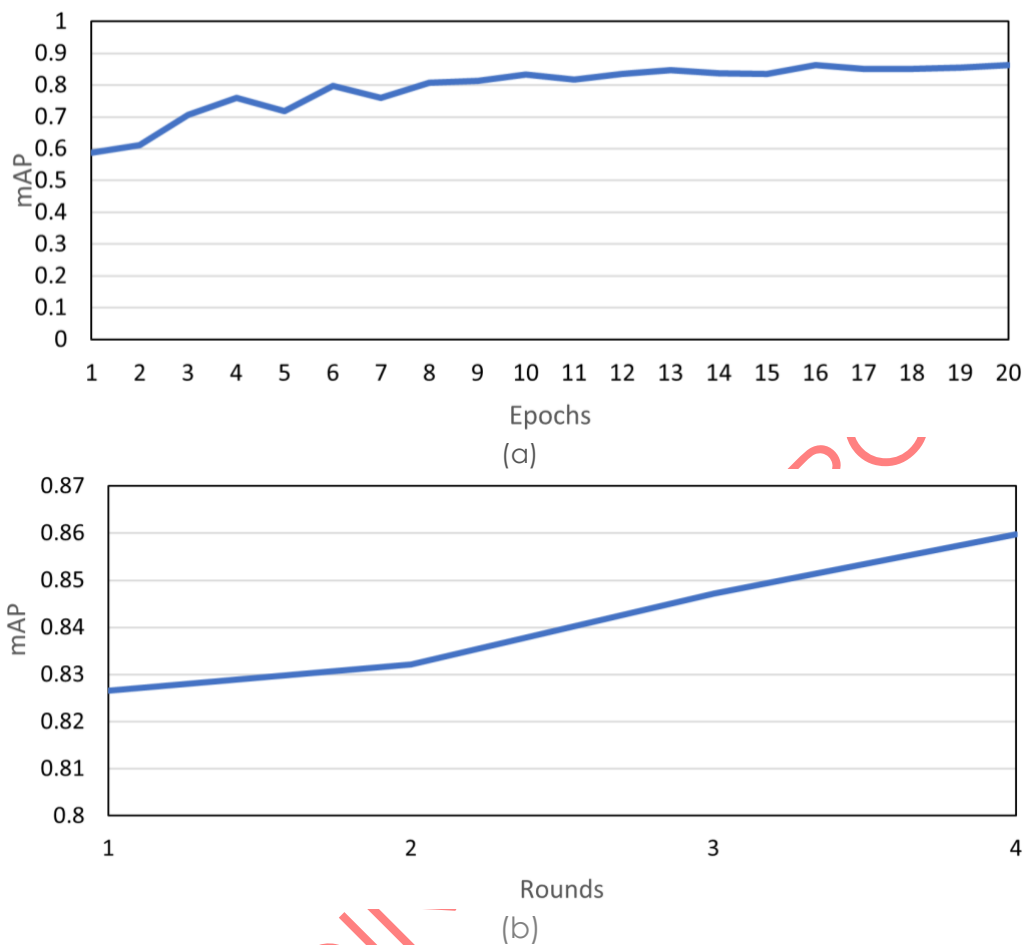


Figure 53: (a) Centralized Learning performance (b) FL training performance.

### Data Heterogeneity

Another scenario concerns the training of the object detection model with heterogeneous data splits. Specifically, the performance of the model is evaluated using different numbers of training samples for each client.

Figure 54 shows the 3 different data splits. In the first case, we tested a data split with no big heterogeneity between the clients. Client 1 owned 50% of the dataset while the other two owned 50% (25% for each client). In the second case, the data is split 60%-10%-30% for each client respectively. The third and the last case has the larger heterogeneity among the clients since the first client owns the 85% of the whole dataset while the other 2 clients the 15% and the 5%, respectively.

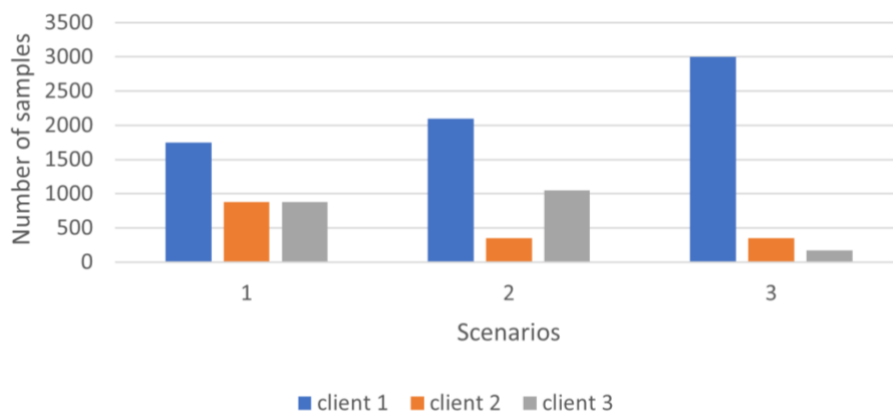


Figure 54: Number of samples with different data Heterogeneity cases.

Based on Figure 55, it is concluded that the performance of the model decreases when the heterogeneity is bigger among all the clients. In case 1, the mAP decreased by 6%, while for the second by 14% and for the last one by 36%, compared to the original model. This behavior is reasonable because the clients with fewer data achieve lower mAP and, therefore, influence the aggregated model in a negative way.

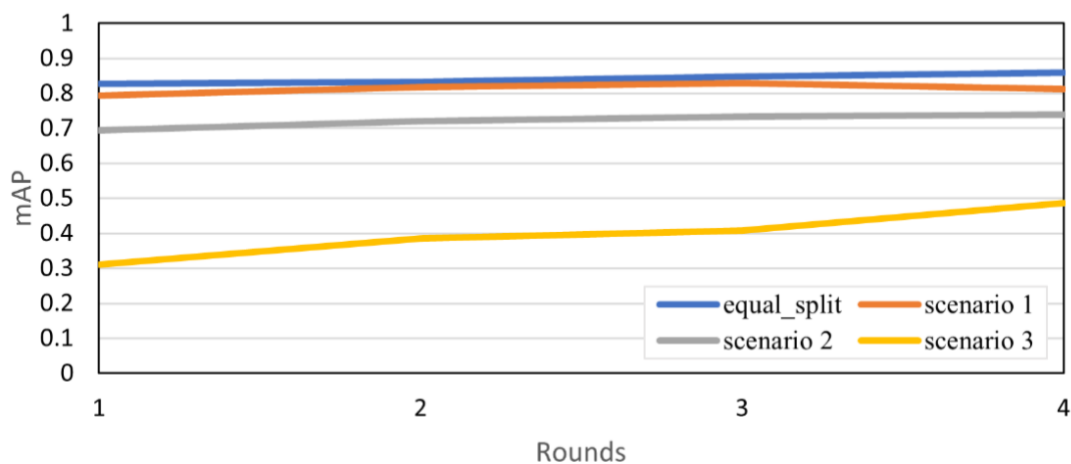


Figure 55: Performance of the model with data heterogeneity.

### FL experiments with Percentile Privacy mechanism

The effect of the Percentile Privacy, as a privacy preserving mechanism, on the model performance is evaluated in different experimental scenarios, as described in Table 12.

Table 12: Scenarios of FL training with Percentile privacy.

Run ID	Parameters	Number of clients	Rounds	Epochs
Percentile_10	Percentile: 10 Gamma: 0.1	3	4	5
Percentile_50	Percentile: 50 Gamma: 0.1	3	4	5
Percentile_70	Percentile: 70 Gamma: 0.1	3	4	5
Percentile_70_0.01	Percentile: 70 Gamma: 0.05	3	4	5

A high value of percentile indicates that a large fraction of parameters will be shared, while a low percentile value means that few parameters will be sent to the server. Figure 56 shows the performance of the model trained on FL setup without any privacy as well as the performance of each model trained with Percentile Privacy. It is observed that in the first two cases the performance of the model is not affected. However, it must be reported that the model with percentile privacy needs more federated rounds to reach the performance of the federated model without privacy. When the value of percentile is equal to 70, a large decrease in the model's performance is noticed. At this point, we decreased the gamma value to check how this affect the performance of the model, but we did not notice a remarkable difference.

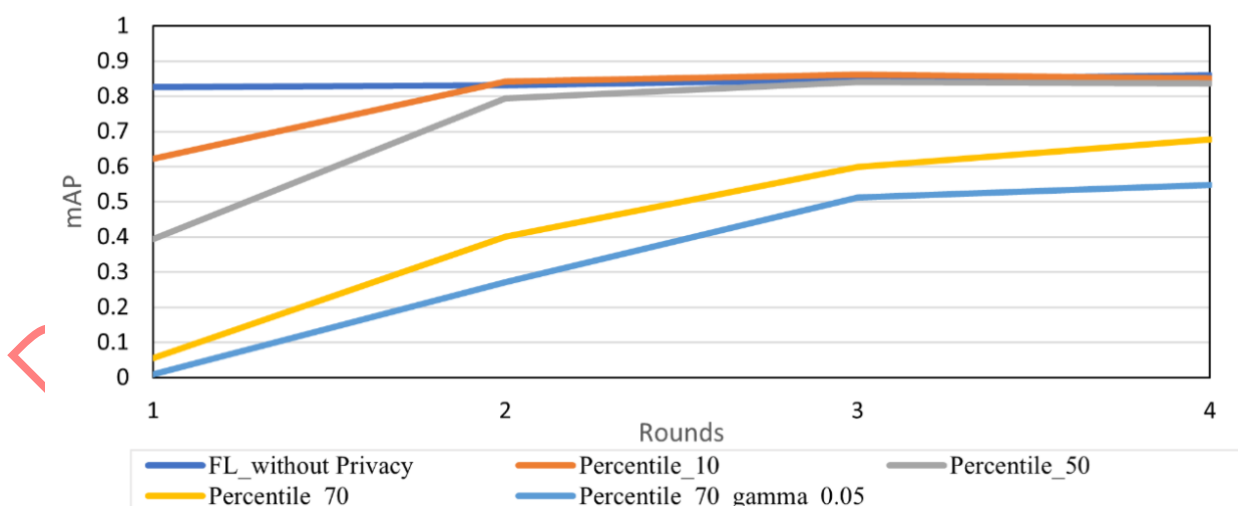


Figure 56: FL with Percentile privacy, the comparison of mAP with various protection levels.

### FL experiments with SVT

The following privacy-preserving mechanism that was tested at different privacy levels is the SVT privacy method. All the experimental scenarios that were examined to evaluate the performance are shown in Table 13.

Table 13: Scenarios of FL training with SVT privacy

Run ID	Parameters	Number of clients	Rounds	Epochs
Noise Var: 1	Fraction: 0.9 Epsilon: 0.1 Noise Var: 1	3	4	5
Noise Var: 0.7	Fraction: 0.9 Epsilon: 0.1 Noise Var: 0.7	3	4	5
Noise Var: 0.5	Fraction: 0.9 Epsilon: 0.1 Noise Var: 0.5	3	4	5
Noise Var: 0.5_0.01	Fraction: 0.9 Epsilon: 0.01 Noise Var: 0.5	3	4	5

At this point, it should be mentioned that the implementation of SVT from NV FLARE is based on the [50], which presents a privacy-preserving federated method in the case of brain tumor segmentation on images. In this work, except for the Laplace noise inserted to weights, also, gradient clipping is applied and is treated as a regularization method to avoid over-fitting. However, in the original paper [30], which introduced the method, gradient clipping was not applied. Initially, some experiments were carried out with the original source code of the SVT that NV Flare provides. In those experiments, the performance of the model was unstable and inadequate. Maybe, the gradient clipping in cases that the model overfits will be appropriate but in our case decreases the performance largely. Therefore, we decided not to apply gradient clipping.

Figure 57 demonstrates the performance of the aggregated model. The main goal of the experiments is to observe the behavior of the model by adding noise gradually. As we can see by adding noise to the model parameters the global mAP decreases. For the two first scenarios with value noise 1 and 0.7, the performance of the model decreases only by 5 percent compared with the baseline model without SVT. For value noise 0.5 (stronger differential privacy) noticed that the mAP reached 76%. Additionally, to check if the performance of the model with strong differential privacy can get better, the  $\epsilon$  variable increased to 0.1 but the performance was impaired. At the SVT privacy observed a instability to the model, specifically, the performance of the aggregated model going upside-down.



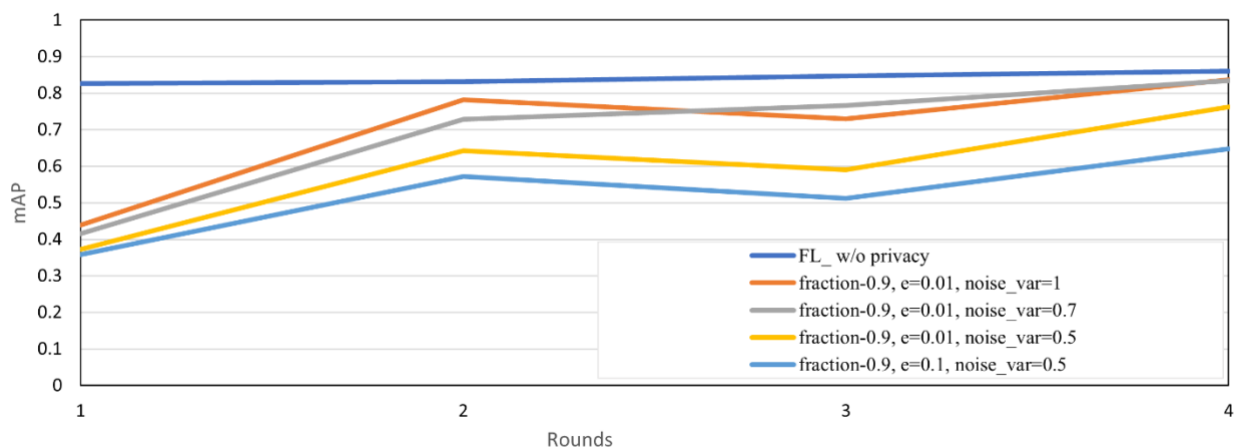


Figure 57: FL with SVT privacy, the comparison of mAP with various noise insertion levels.

### FL experiments with HE

As mentioned in section 4.2.1.1, the HE algorithm allows participants to encrypt the information transferred to the aggregator server. When the server receives the encrypted information, it can operate on encrypted data by using HE and after the aggregation process returns them to the clients. It should be mentioned that HE can be applied only with the provisioning tool, which is described in section 4.2.1.2. Therefore, the HE mechanism is applied in a FL setup with 3 remote clients. Figure 58 presents the global mAP of aggregated model for each federated round with and without HE. As expected, HE is a mechanism that does not decline FL model performance, as both the encrypted and non-encrypted models have the same results.

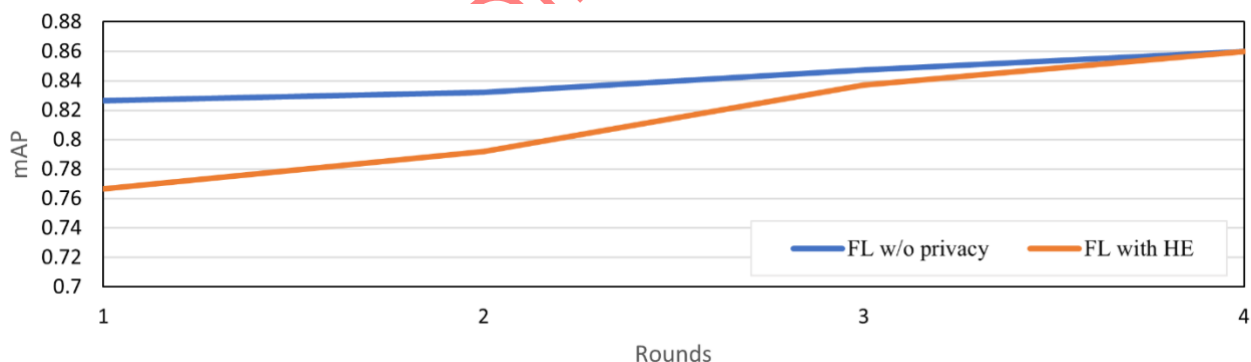


Figure 58: FL with and without HE.

### FL experiments with Percentile Privacy and HE

The last experiment with the NV Flare framework combines privacy-preserving mechanisms to achieve the best possible privacy in the system. For this reason, first, we applied Percentile Privacy to our data, and then we added HE. Figure 59 illustrates the global mAP of the federated model with and without privacy-preserving techniques trained in a server with 3 actual nodes as clients. Our experiments compute the performance of the model trained in an FL with HE, Percentile Privacy as well as a combination of HE and Percentile Privacy. As we expected the combination of these two privacy-preserving techniques does not decline the FL model's performance.



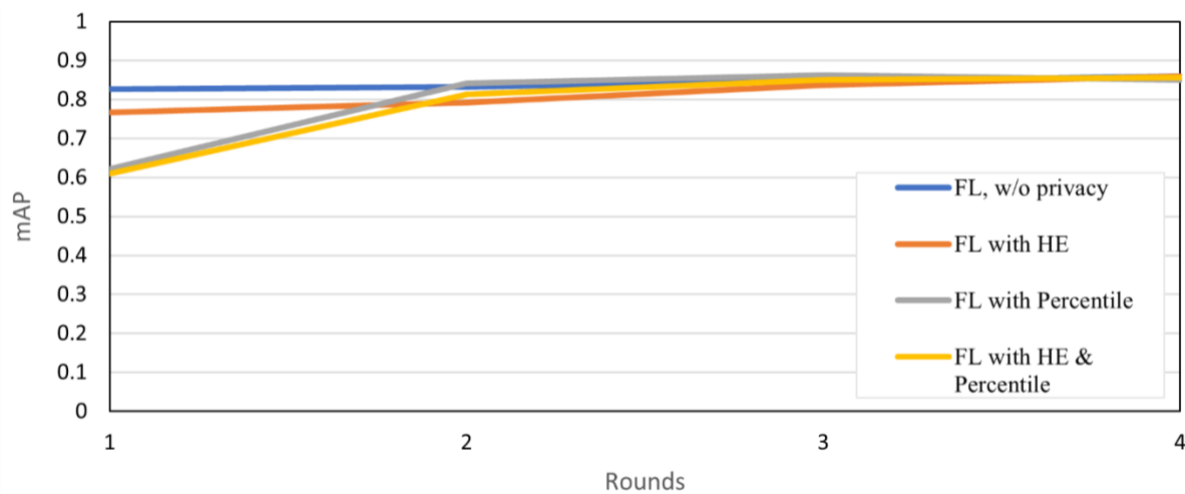


Figure 59: FL with Percentile Privacy with HE.

## 4.3.2 IoT-NGIN FedPATE

### 4.3.2.1 Experimental Setup

For the experiments we used the CIFAR-10 dataset [51], which is a collection of images that are commonly used to train machine learning and computer vision algorithms. It is one of the most widely used datasets for machine learning research. CIFAR-10 consists of 60000 32x32 color (RGB) images in 10 different classes. Table 14 shows the total size of dataset as well as the training and testing sets. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships and trucks. There are 6000 images of each class.

Table 14: Number of images in the CIFAR-10 dataset.

Dataset	Total Images	Training set size	Test set size
CIFAR-10	60000	50000	10000

To train a ML model in a Federated Learning setup among clients, partitions of the dataset are required to ensure that each Teacher of the FL System is trained with different data. We partition the CIFAR-10 training dataset in  $n$  disjoint sets and train a Teacher model separately on each set, where  $n$  is the number of Teachers. The CIFAR-10 test dataset represents the public unlabeled data for the student training, for which a portion of labels are extracted from the most voted predictions of the Teachers on that data. Particularly, the 90% of the CIFAR-10 test dataset is used for the training procedure of the Student and the rest is used to evaluate the whole training procedure as shown in Table 15.

Table 15: CIFAR10 dataset partitions for student's training.

Dataset	Total Images	Teachers' training set size	Student's training set size	Student's test set size
CIFAR-10	60000	50000	9000	1000

The Deep Learning model, that is trained in FedPATE, is a Convolutional Neural Network (CNN), which has 3 convolutional layers, each consisting of two 3x3 layers, one Max-Pooling layer and one Batch Normalization layer, followed by 3 fully connected layers. The performance of the ML model is described by the accuracy metric, which is equal to the correct number of predictions divided by the total number of predictions.

#### 4.3.2.2 Experimental results

The baseline model is trained in a Federated Learning scenario, without any privacy preserving mechanisms, with 1 client (centralized case) for 20 epochs on the same train and test dataset with the student and achieves 79% accuracy. This performance is the upper bound of the FL algorithms and works as a baseline for the next experiments. Figure 60 shows the accuracy of the baseline model for each epoch, which is trained in a centralized manner.

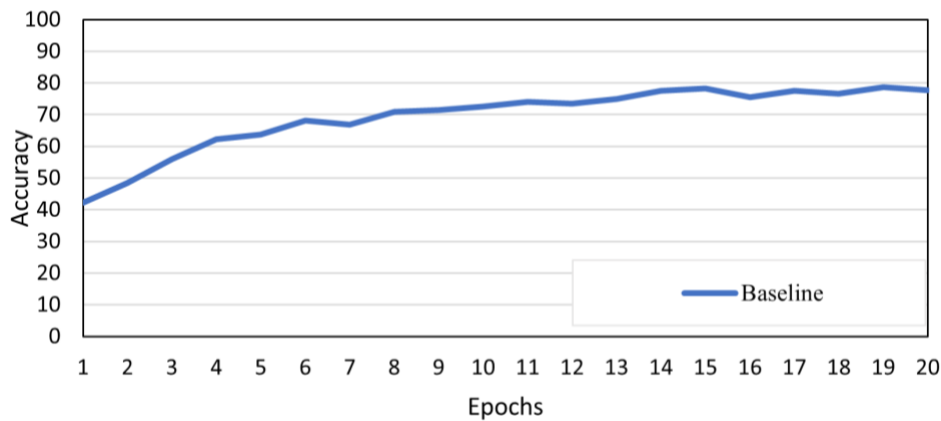


Figure 60: Performance of the student baseline model of FedPATE.

As mentioned in section 4.2.2.2, at the first stage of FedPATE, the Flower server trains the teachers in the form of federated learning while the student is waiting for the teachers to finish their training process. We consider two ensembles of 10 and 14 teachers and therefore we split the dataset into 10 and 14 equally-length, disjoint partitions, each containing 5000 and 3571 images respectively. Each teacher is trained for 10 federated rounds and 10 epochs each round. Figure 61 illustrates the test accuracy of the aggregated model after each federated round during the teachers' training for the two ensembles. At the final round, it performs 89.5% and 89.2% in accuracy for 10 and 14 clients respectively.

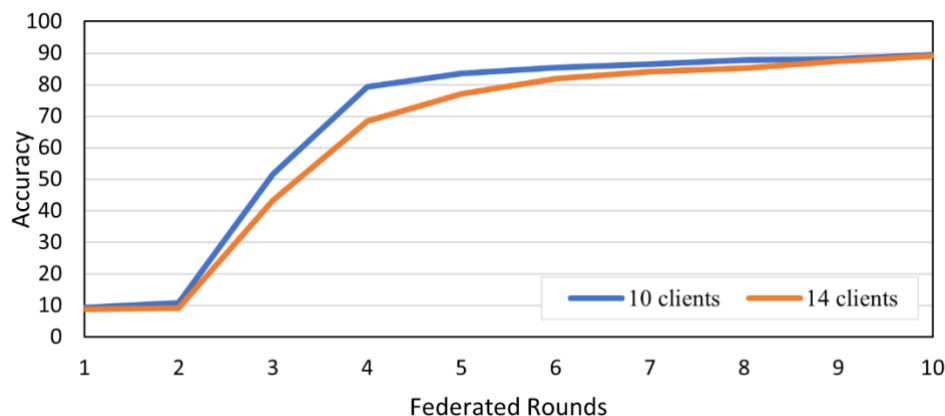


Figure 61: Performance of the aggregated model for each federated round.

At the second stage of FedPATE, the flower server queries the predictions of teachers' model on student's training data and performs a noisy aggregation of their votes (predictions) to label a portion of that data and trains the student. As mentioned above, we perturb the vote counts with Laplacian noise of inversed scale  $\varepsilon$  ranging with the values 1, 0.2, 0.1, 0.001. As the value of  $\varepsilon$  is decreased the more random noise is injected to each query. The student has access to 9000 samples of the CIFAR10 dataset, which a subset of either 2000 or 9000 is labeled by using the noisy aggregation mechanism for each value of  $\varepsilon$ . Figure 62 illustrates the baseline model's accuracy compared with the student model, trained with 10 and 14 teachers without noise injection. We observe that each model performs almost the same with accuracy ~79%. This is to be expected as the average test accuracy of individual teachers is ~86% and ~84% for 10 and 14 teachers respectively, resulting in accurate predictions to the student's public, unlabeled data.

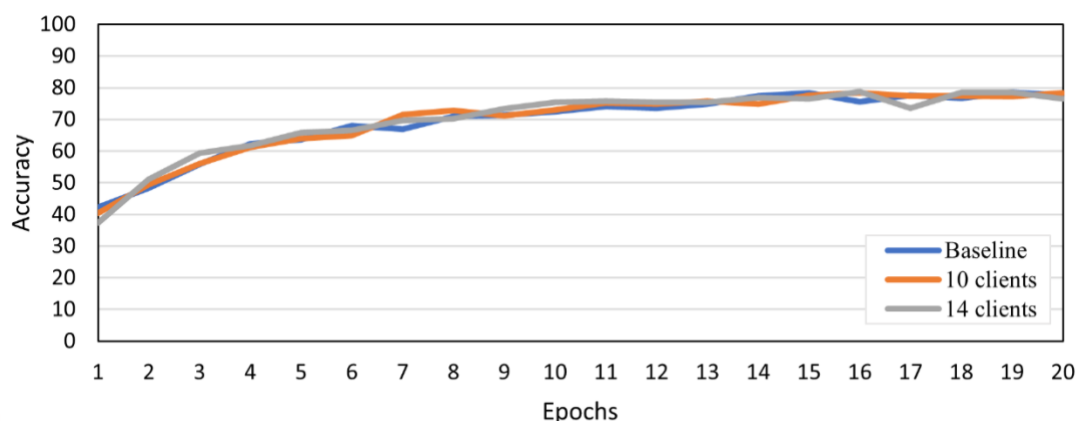
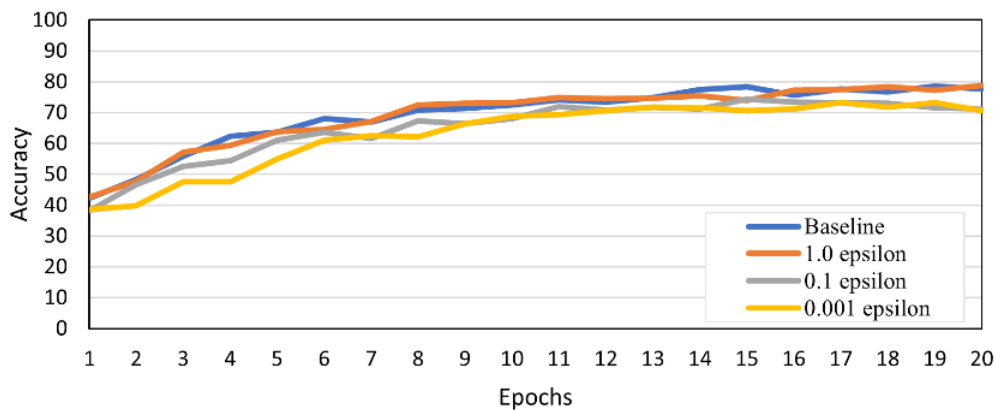


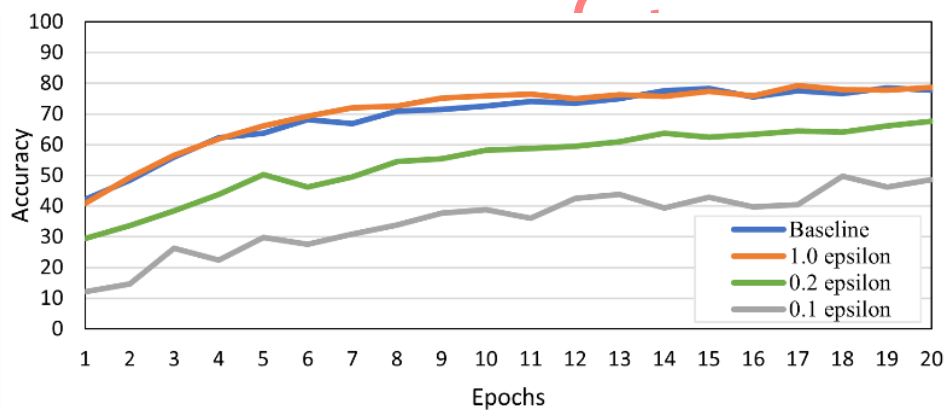
Figure 62: Student's test accuracy for 9000 queries, without noise injection.

Figure 63 shows the student's test accuracy when the flower server performs 2000 and 9000 queries from the 2 different ensembles of teachers with several values of the noise parameter  $\varepsilon$ . As expected, in both ensembles, the student's accuracy decreases while the parameter  $\varepsilon$  is decreased too. In the case of 9000 queries is meaningless to experiment with values lower than 0.1 for  $\varepsilon$  as it adds too much noise to the student's labels resulting in pointless training. We found that when Laplacian noise with  $\varepsilon = 0.2$  is added to the teachers' queries, we achieve a significant performance of 70.1% accuracy on the student model (~7% lower compared to the baseline) while we preserve high privacy concerning each teacher's

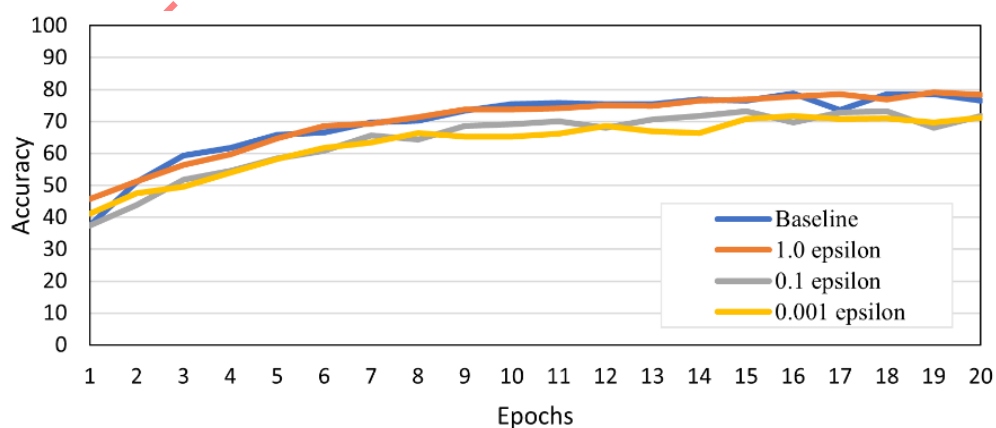
sensitive data. In addition, we observe that the more the number of teachers is increased (i.e., the clients of the FL system), the less impact has the noise injection on the model's accuracy. Particularly, as Figure 63(b) and Figure 63(d) show, the student model achieves 50% accuracy with  $\epsilon = 0.1$  and 10 teachers, while it achieves almost 60% accuracy with 14 clients and noise parameter  $\epsilon = 0.1$ .



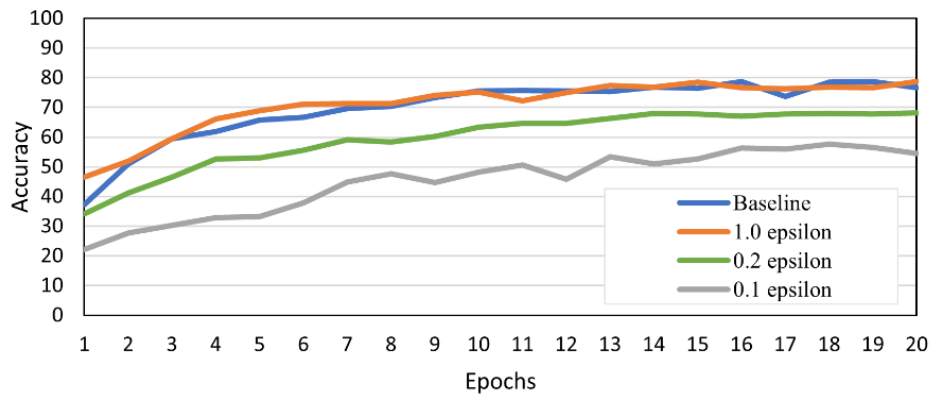
a) Test accuracy for 2000 queries, 10 clients.



b) Test accuracy for 9000 queries, 10 clients.



c) Test accuracy for 2000 queries, 14 clients.



d) Test accuracy for 9000 queries, 14 clients.

Figure 63: Student's test accuracy.

### Data Heterogeneity

To examine how our model performs in real-world scenarios, we investigate the scenario of data heterogeneity. In real life applications, clients of a Federated Learning system may own a dissimilar amount of data compared to each other, resulting in effecting the entire model's performance. We examine the impact that data heterogeneity has in our FL system model's accuracy in 2 different scenarios. Figure 64 illustrates the number of samples that each client contains in the scenario of Soft Heterogeneity, where the difference of each client's data size is small (~150 samples in average), and the scenario of Hard Heterogeneity, where the difference of each client's data size is higher (~4000 samples in average).

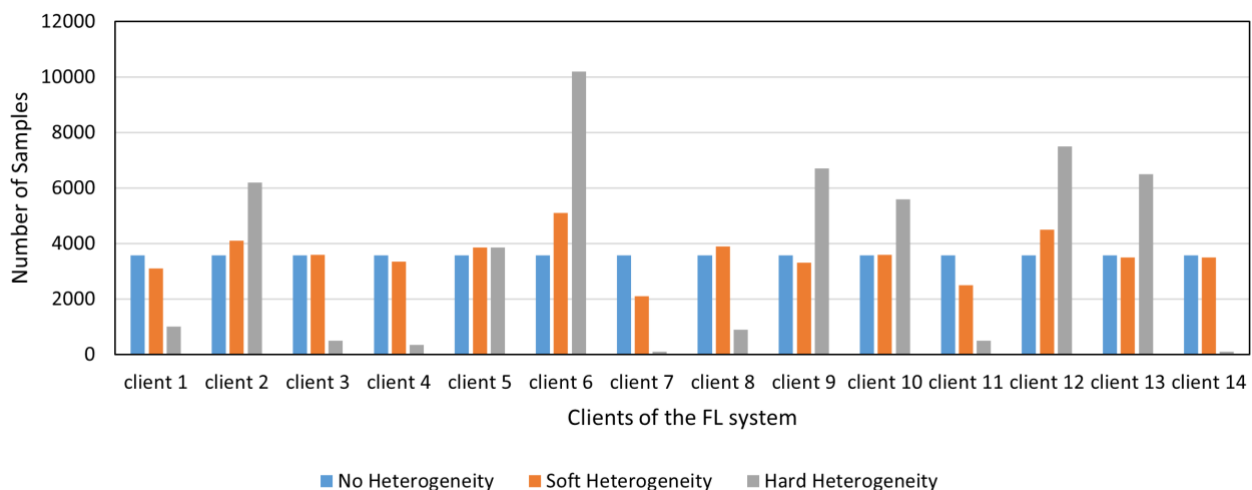


Figure 64: Data heterogeneity scenarios.

That said, Figure 65 shows the accuracy of the FL system when applied the 2 scenarios of data heterogeneity discussed above. We observe that in the cases of No Heterogeneity and Soft Heterogeneity the accuracy has almost the same value and the scenario of Hard Heterogeneity performs slightly better. This is to be expected, since the CIFAR10 dataset is an easy to learn dataset. The images contained in the dataset are 32x32 pixels and our model can successfully be trained on them even with a small number of images. However, as Figure

64 shows, some clients have a large number of images, and they manage to achieve high results in the image classification. This results in better quality of knowledge transfer when the FedAvg algorithm is applied to the client model's trained weights during the FL training and therefore the aggregated model after each round performs better.

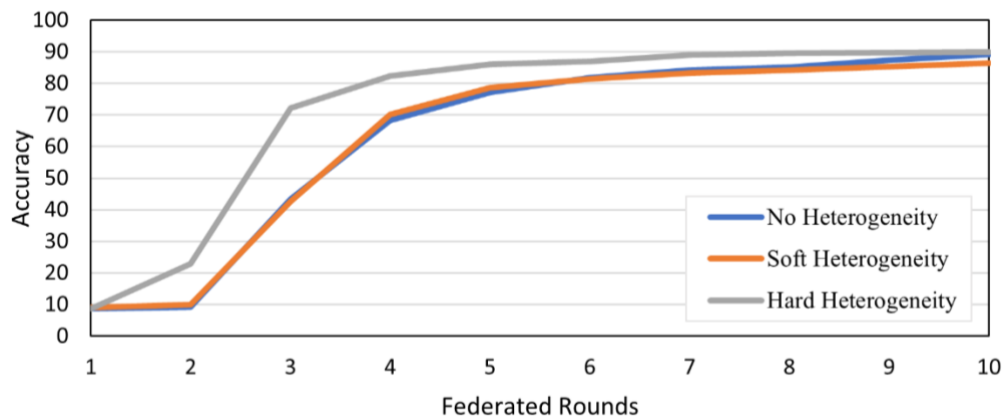


Figure 65: Accuracy of the FL system when applied data heterogeneity.

### 4.3.3 TensorFlow Federated

In this section, experiments of federated learning scenarios using TensorFlow Federated are described. More specifically, we implemented a network Intrusion Detection System (IDS) which is trained and deployed in a federated manner. For training and testing purposes we use the NSL-KDD dataset [52] which consists of packet logs with 6 different types of attacks and normal records. Results show that the approach is robust and can detect malicious packages in advance thus enhancing cybersecurity in IoT Systems.

#### 4.3.3.1 Experimental setup

For the training of the federated model, we use the NSL-KDD dataset for our experiments. For its features, it is widely used for the testing and evaluation of intrusion detection systems. The dataset is composed of 257673 cyber-attacks labeled in 5 different classes as Dos, Probe, U2R, R2L (attack types) and Normal (no attack). We modify the dataset in a way that we have 2 classes, converting our problem to binary classification problem, where the labels are "No DoS" and "DoS". That said, "No DoS" encapsulates normal and other kinds of attack samples. Consequently, our goal is only to detect Denial of Service attacks.

In the scope of IoT-NGIN, we introduce an FL-based, privacy preserving intrusion detection system (IDS) for the detection of DoS attacks in IoT networks. The developed model consists of 2 fully connected Dense layers, an input layer and an output layer. The output layer is responsible for the binary classification task and is selected to be activated with the SoftMax activation function. In addition, we use the Stochastic Gradient Descent (SGD) as our optimizer with 0.001 learning rate and the loss function is the Binary Cross-Entropy (BCE) since our goal is binary classification ("DoS" or "Not DoS" attacks). Moreover, the metric accuracy describes the model's performance for the various experiments similar to the case of IoT-NGIN FedPATE.

The proposed model architecture follows the standard Federated Learning training process using the TFF framework. A central Server loads the initial, generic global model. For each federated round, the Server sends the model to its clients and each individual client trains the model in their corresponding local, sensitive dataset. After the training procedure of all clients is finished, each client transforms its model's weights with the Local Differential Privacy mechanism by adding Gaussian noise. Finally, the clients send back to the central Server the noisy-aggregated model updates, where the Server updates the global model by averaging the updated weights using the FedAvg method. After the model's training process is finished, the model can successfully classify if there is a "DoS" attack or not and can be applied to IoT systems to detect anomaly attacks in networks as shown in Figure 66.

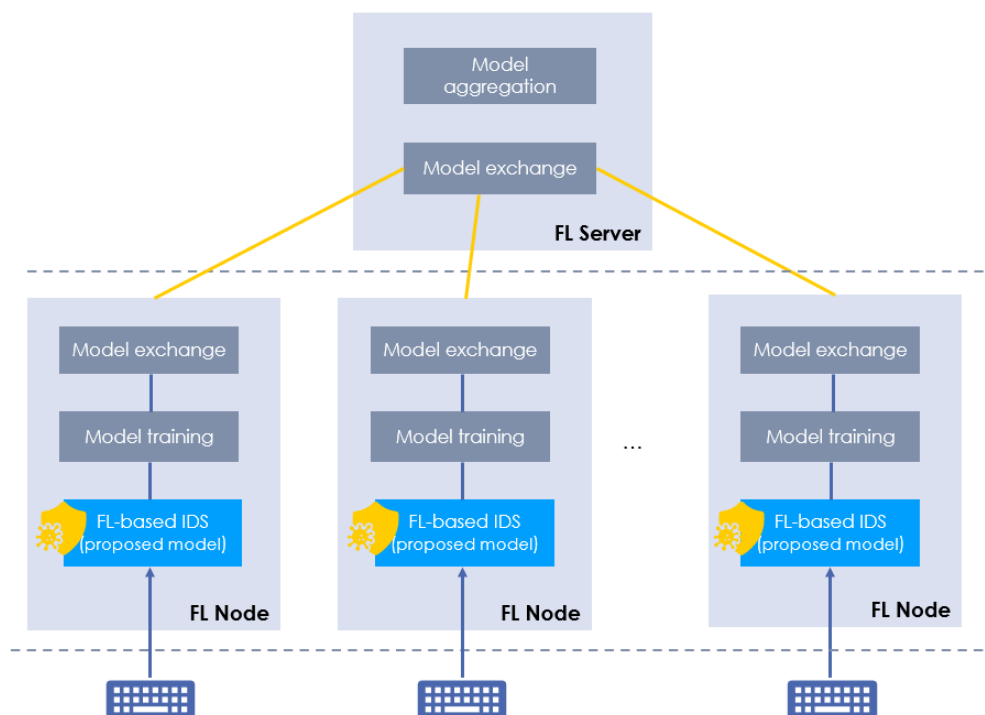


Figure 66: Proposed architecture of FL in an intrusion detection system.

#### 4.3.3.2 Experimental Results

We implement a variety of experiments to test the robustness of the FL intrusion detection model regarding many real-world scenarios. The main parameters that are being tested are imbalance ratio, number of clients and application of differential privacy. These scenarios are first tested separately and subsequently jointly in an attempt to model realistic conditions.

##### Number of clients

In this scenario the effect of the number of clients on the model's performance is examined. As the total number of clients in our IoT system increases, the security of each individual IoT device (client) is enhanced because each client transfers its knowledge to the others while they are training in the form of federated learning.

Regarding this scenario, we experimented with the values 1 (baseline model), 10, 25 and 50 for the number of clients. Figure 67 illustrates the impact that this number has on the model's

accuracy. The total accuracy achieved for the testing phase is 79.06%, 78.95%, 77.88% and 75.54% respectively. We observe that a higher number of clients results in accuracy degradation. This is to be expected since splitting the dataset in smaller sets eliminates certain relationships that define the data. The difference can be as high as 3.52%.

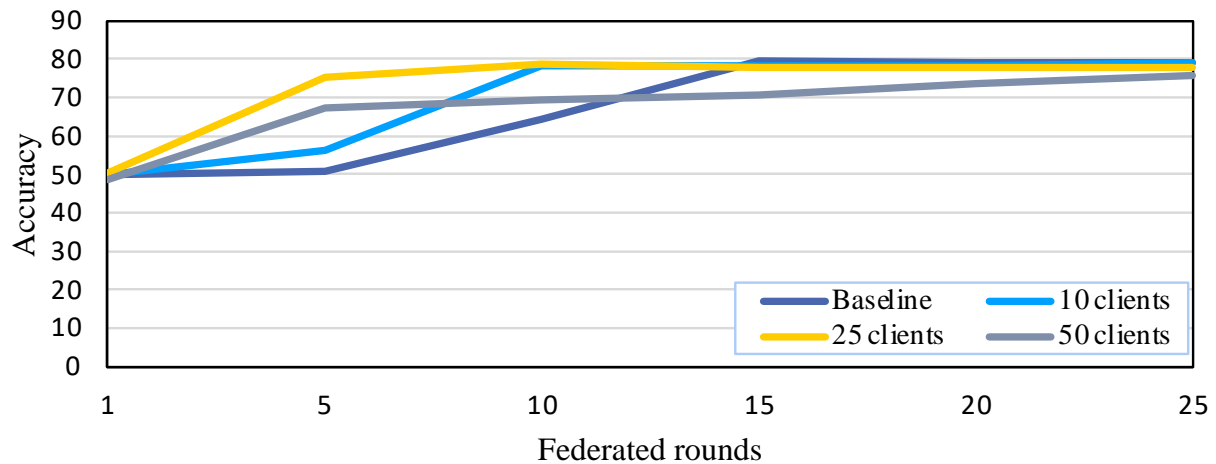


Figure 67: Testing accuracy for different number of clients.

### Differential privacy

The main component of our proposed architecture is the Differential Privacy (DP). Adding noise to our model can lead to higher data privacy resulting in keeping the data of each IoT device secure. Therefore, there is a trade-off between applying differential privacy and achieving a high level of model accuracy. In this scenario, the effect of the noise addition on model's accuracy is investigated.

In the second scenario, we experiment with the differential privacy that our model preserves. The parameter which controls the quantity of noise addition is called noise multiplier: the ratio of the noise standard deviation to the clipping norm. We experiment with the values 0, 0.5, 1, 1.5 and 2 where the value 0 refers to a model trained with no DP, a classical federated learning model as introduced to its original paper [1]. As the value of noise multiplier is increased, the more Gaussian noise is added to the model and so more privacy is preserved. In Figure 68, we observe that adding more noise has a negative impact on the model's accuracy and adding too much may cause the model to collapse. In particular, our model achieves 79.06%, 75.85%, 73.49%, 63.33% and 50.58% test accuracy with respect to the values of the *noise multiplier* mentioned above. There is a trade-off between privacy preservation and accuracy, and this is something that must be fine-tuned for each specific situation. For small portions of noise, the difference in accuracy is between 3.21-15.73 % bounds, whereas for bigger ones can be as high as 28.48%.



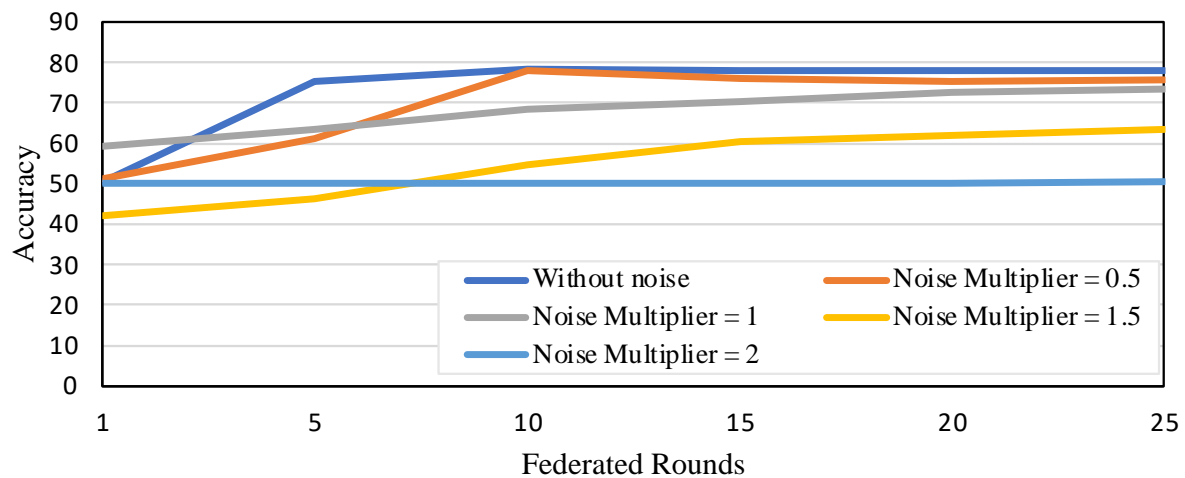
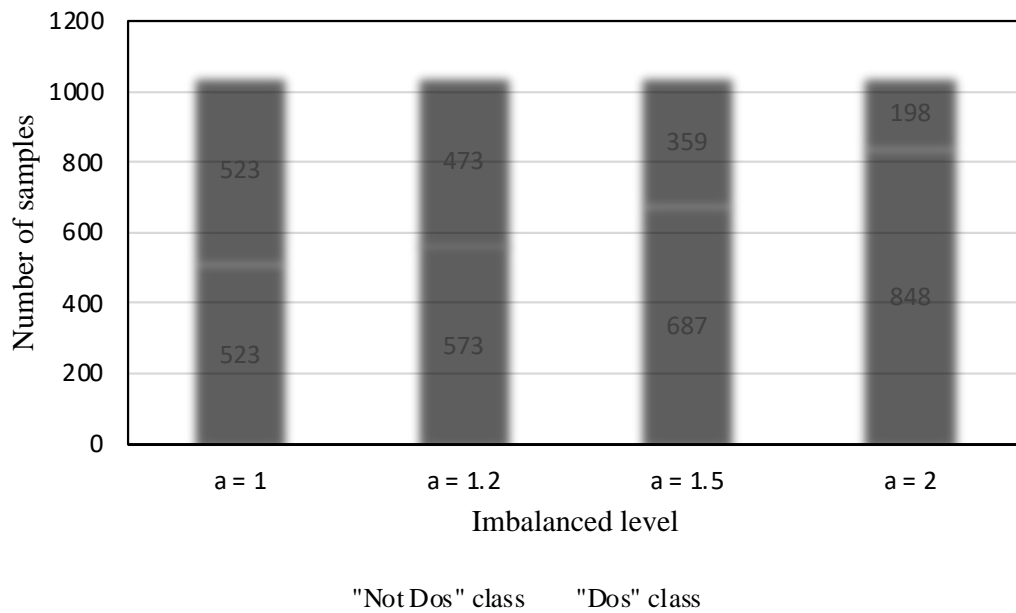
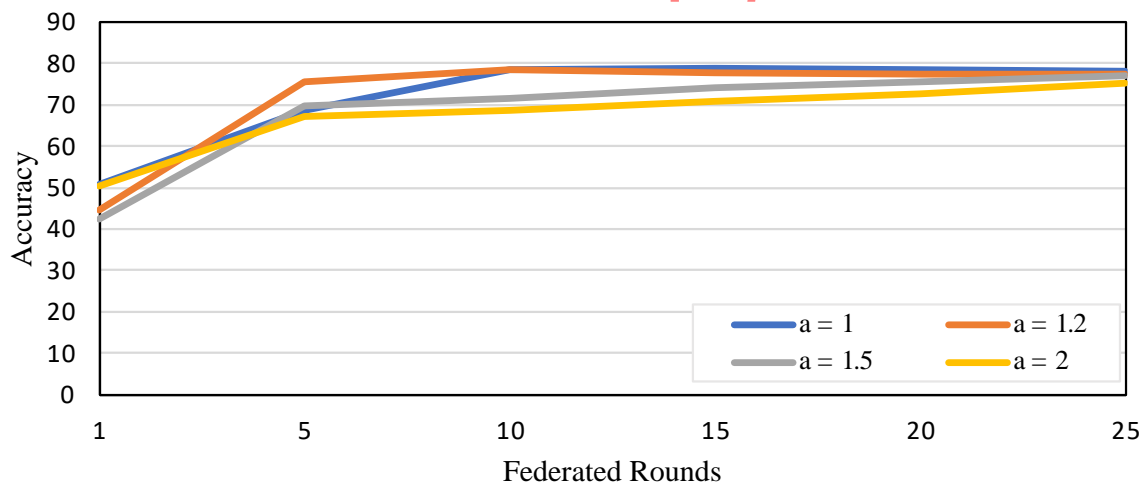


Figure 68: Impact of noise on model's accuracy using 25 clients.

### Imbalance ratio

In real-world scenarios, the local dataset size of each IoT device may differ from each other affecting the entire IoT systems' performance. To define the imbalance level of each client's dataset we introduce a new parameter  $\alpha$ , a factor that defines how imbalanced each client's dataset will be. A value of 1 means no imbalance, whereas higher values than 1 assign datasets to clients with higher imbalance.

For this scenario, we experiment with the values of 1 (no imbalance), 1.2, 1.5 and 2 for the parameter  $\alpha$ . The quantity and quality of samples that each client has for these values of the parameter  $\alpha$  during their training procedure are shown in Figure 69, while Figure 70 reflects the impact of the parameter  $\alpha$  (imbalanced level) on model's accuracy. In particular, our model achieves 79.06%, 77.42%, 76.98% and 75.11% with respect to the values for the parameter  $\alpha$  mentioned above. As expected, while  $\alpha$  is being increased, the model's accuracy becomes worse. It can be seen that higher imbalance (higher  $\alpha$ ) leads to worse results. Specifically, the range of accuracy degradation is 1.42-3.95%.

Figure 69: Number of samples per client for different values of  $a$ .Figure 70: Impact of the component  $a$  on model's accuracy.

### Realistic scenario

Considering all the scenarios mentioned above, we display an FL-based IDS which performs accurately under real world settings. We assume that the scenario of  $a = 2$  (imbalance ratio  $\sim 2.5:1$ ) represents the real-world circumstances since malicious packets appear less frequently in network traffic logs. Furthermore, a value of *noise multiplier* = 1.5 is sufficient to provide both security guarantees and viable performance since experiments show that higher values cause model failure. This combination achieves high privacy concerning data as well as minimizes the risk of sacrificing model's performance. Thus, the experiment described below reflects realistic applications of IDS in IoT Systems. Figure 71 plots the proposed model and the baseline model. Our introduced model consists of 25 clients. In such a case, we can see that the model performs decently compared to the baseline and is a

viable solution for real world applications. The resulting model after 25 rounds achieves accuracy that is only 7% lower compared to the baseline.

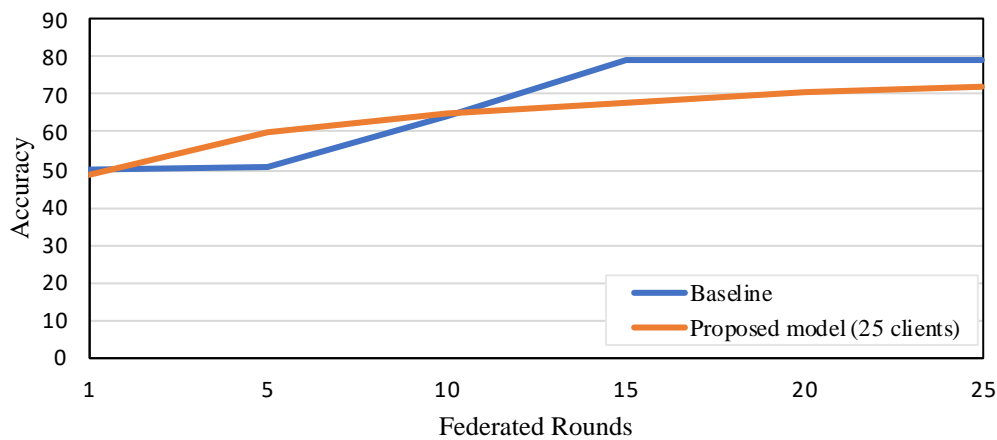


Figure 71: Testing accuracy for the realistic scenario with 25 clients.

## 4.4 Discussion

In section 4.3 experimental analysis of the three FL frameworks considered in IoT-NGIN has been conducted, exploring the impact of each FL framework or privacy preservation parameters on the accuracy of the investigated applications' model.

Considering the outcomes of this analysis, Table 16 highlights the basic lessons learnt through the hands-on experience with the three FL frameworks, complementing the desk research and comparison of existing FL frameworks included in D3.2, thus assessing the main benefits for each framework.

Table 16: Lessons learnt from FL frameworks' experimental assessment.

FL Framework	Main Benefits
NVIDIA FLARE	<ul style="list-style-type: none"> <li>It can be used to train an ML model in real life scenarios between one server and many remote clients. Each remote client has its own private data, while the server is used for model aggregation and does not possess data.</li> <li>Provides privacy preserving methods, such as Percentile Privacy, Homomorphic Encryption and MPC, which can also be combined.</li> <li>It is customizable, supporting the integration of ML models implemented via state-of-the-art ML frameworks, such as TensorFlow and PyTorch.</li> </ul>
IoT-NGIN FedPATE	<ul style="list-style-type: none"> <li>It can be used to train an ML model in real life scenarios between one server and many remote clients. Each remote client (Teacher)</li> </ul>

TensorFlow Federated	<p>has its own private data, while one of them (Student) desires to perform predictions on public data.</p> <ul style="list-style-type: none"> <li>• The public data of the student can be partially or completely unlabeled.</li> <li>• It provides increased privacy levels, since the final trained model (Student's model) is not directly trained on the sensitive data of each client (Teacher).</li> <li>• The final trained model is resistant to membership inference attacks and model inversion attacks.</li> <li>• ML algorithms implemented with PyTorch can be directly integrated.</li> <li>• It is primarily opted for research purposes. ML researchers can use this FL Framework to perform experiments to investigate the performance of the FL process under various scenarios (such as different number of clients, privacy level, heterogeneity of data).</li> <li>• Differential privacy is supported as privacy preserving mechanism.</li> <li>• ML models implemented at TensorFlow can be directly integrated.</li> </ul>
----------------------	---

Next, Table 17 elaborates on how FL could be applied in the IoT-NGIN use cases and provides recommendations on the use of each of the three frameworks analyzed within those use cases. The table is not exhaustive but is meant to identify potential FL applications in real-life use cases. The cases in which Living Lab validation is foreseen within the IoT-NGIN timeline are indicated in bold.

Table 17: Recommended uses of FL in the LL use cases.

LL	Use Case	NVIDIA FLARE	IoT-NGIN FedPATE	TensorFlow Federated
Smart City	Traffic Flow & Parking Prediction	<b>train the visual recognition of vehicles across a set of edge nodes within a smart city</b>	train the visual recognition of vehicles over data shareable within an administrative domain (e.g., smart city)	research on vehicles' recognition across hundreds of nodes
	Crowd Management	<b>train the visual recognition of people across a set of edge nodes within a smart city</b>	train the visual recognition of people over data shareable within an administrative domain (e.g., smart city)	research on people's recognition across hundreds of nodes

## D3.3 - ENHANCED IOT FEDERATED DEEP LEARNING/ REINFORCEMENT ML

	Co-commuting solutions based on social networks	train ML model to help better match and organize the co-commuters with available shared rides over social networks' data across users' devices	train this model over public data collected by smart city platform	research in large scale scenarios
Smart Agriculture	Crop diseases prediction & irrigation precision	<b>train the crop disease predictions, using private smart farm data</b>	train the crop disease predictions, using private smart farm data and use them to train over public datasets e.g., collected/shared at district/national/regional level	research on crop disease prediction across hundreds of nodes; research on the effect of model and data poisoning attacks on FL system based on GAN-based data generator
	Sensor aided crop harvesting	train the object detection model across edge nodes	train the object detection model across edge nodes and then train over public unlabeled data e.g., collected/shared at district/national/regional level	research in large scale scenarios
Industry 4.0	Human-centered safety in a self-aware indoor factory	train the model for object detection among edge nodes	train the object detection model among edge nodes and then train over public unlabeled data (e.g., across factories of the same administrative domain)	research in large scale scenarios

## D3.3 - ENHANCED IOT FEDERATED DEEP LEARNING/ REINFORCEMENT ML

Industry 4.0	Digital powertrain and condition monitoring	to train pattern recognition in sensor data among edge nodes	train the model among edge nodes and then train over public unlabeled data (e.g., across factories of the same administrative domain)	research in large scale scenarios
Smart Energy	Move from reacting to acting in smart grid monitoring and control	train pattern recognition in sensor data among edge nodes	to train pattern recognition in sensor data among edge nodes and then train over public unlabeled data (e.g., in smart city context)	research in large scale scenarios
	Driver-friendly dispatchable EV charging	train pattern recognition in EV chargers' data among edge nodes	train pattern recognition in EV chargers' data among edge nodes and then train over public unlabeled data (e.g., in smart city context)	research in large scale scenarios

## 5 Installation and User Guide

### 5.1 MLaaS Framework

#### 5.1.1 Installation principles

This section describes the principles used for the installation of the MLaaS platform. One of the goals is to have a consistent and reproducible installation, thereby a GitOps and Infrastructure as Code (IaC) approach is used. The IaC is used to fully describe the platform using IaC files. This is the single source of truth and ensure that any installation will have the same setup. Then a GitOps approach is used to synchronize the Kubernetes resources with a Git repository. The principles for the installation are:

- IaC manifests are centralized in a Git repository
- ArgoCD application manifests referencing the Git repository are used to trigger installation of the applications
- The GitOps tool reads the IaC manifests from the Git repository, creates and synchronize the associated Kubernetes resources

In order to avoid exposing sensitive data, some resources such as Kubernetes secrets used by some components or certain apps are kept outside the Git repository and are deployed manually via kubectl. Possible future work will look for a more automated approach for that part. Figure 72 illustrates the principles for the installation of the Framework.

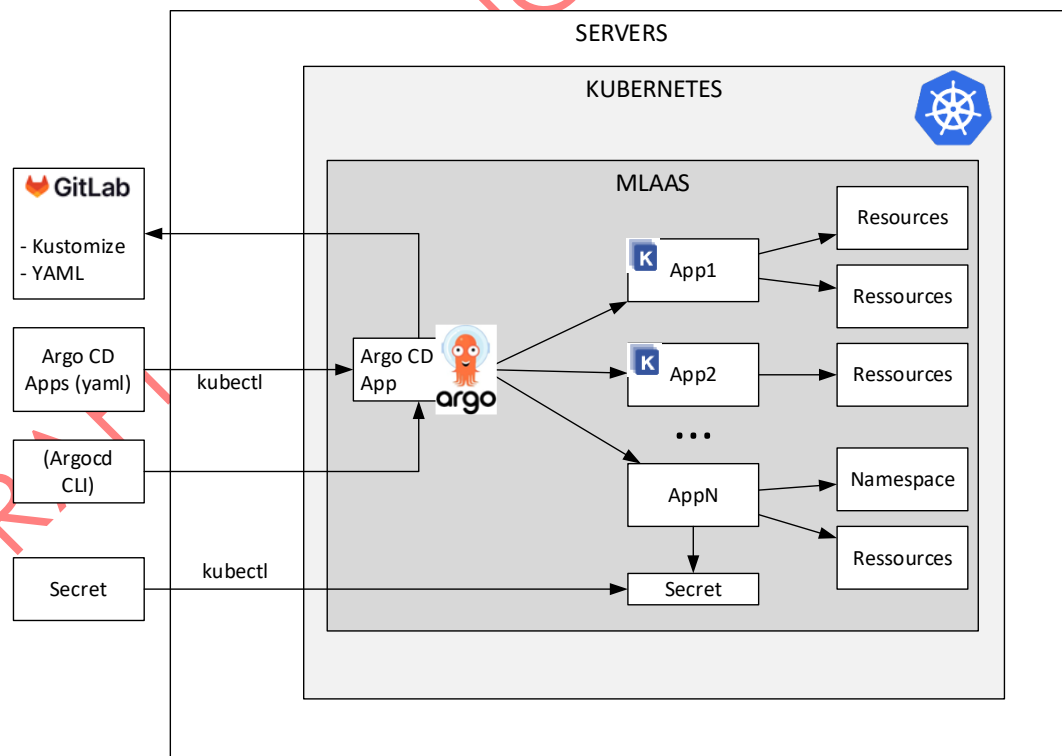


Figure 72: MLaaS installation principles.

The GitOps approach is used:

1. To keep all configuration files in a single repository.
2. To ensure that the platform is always synchronized with a single source of truth. If a change is made on the IaC manifests of a resource in the Git repository, the corresponding Kubernetes resource in the cluster is synchronized by being automatically modified. Similarly, if a Kubernetes resource is changed directly on the cluster (via kubectl for example), it is automatically changed back to the IaC manifests version on the Git repository.
3. To easily deploy changes to the platform by simply committing the changes in the Git repository
4. Allow to easily check status of an application and view its associated components with their health in a user-friendly graphical interface.

Yaml and Helm files are used for the IaC part. For the IoT-NGIN installation, GitLab is used for the repository of the IaC files. The Git structure is based on:

1. A "base" folder that contains the standard IaC files. For Kubeflow, for example, these are the files from the Kubeflow Manifest repository.
2. A "mlaas" folder that contains the IaC files specific to the IoT-NGIN MLaaS platform. This can be files replacing components from the Kubeflow manifest or installation of other components for other applications.

The structure is illustrated in Listing 3.



```

+---base
|   \---kubeflow
|       +---apps
|           +---admission-webhook
|           +---centraldashboard
|           +---jupyter
|           +---katib
|           +---pipeline
|           +---profiles
|           +---tensorboard
|           +---training-operator
|           \---volumes-web-app
|       +---common
|           +---cert-manager
|           +---dex
|           +---istio-1-14
|           +---knative
|           +---kubeflow-namespace
|           +---kubeflow-roles
|           +---oidc-authservice
|           \---user-namespace
|       \---contrib
|           \---kserve
\---mlaas
    +---apps
    |   +---argocd-apps
    |   \---kserve-test
    +---kubeflow
    |   +---argocd-apps
    |   +---istio-install-mlaas
    |   +---knative-eventing-mlaas
    |   +---knative-serving-mlaas
    |   +---kserve
    |   +---kubeflow-istio-resources-mlaas
    |   \---nginx-ingress-mlaas
    \---tools
        +---argocd
        +---argocd-apps
        \---metrics-server

```

Listing 3: MLaaS GitOps structure.

ArgoCD [53] tool is used for the GitOps part. ArgoCD is a declarative, GitOps continuous delivery tool for Kubernetes. It pulls updated code from Git repositories and deploys the associated resources directly to the Kubernetes cluster. At the time of this writing, the platform is organized around three ArgoCD projects. One is for the Kubeflow components, another one for the monitoring and managing tools and the last one for the other apps. Figure 73 shows the projects in the ArgoCD console.

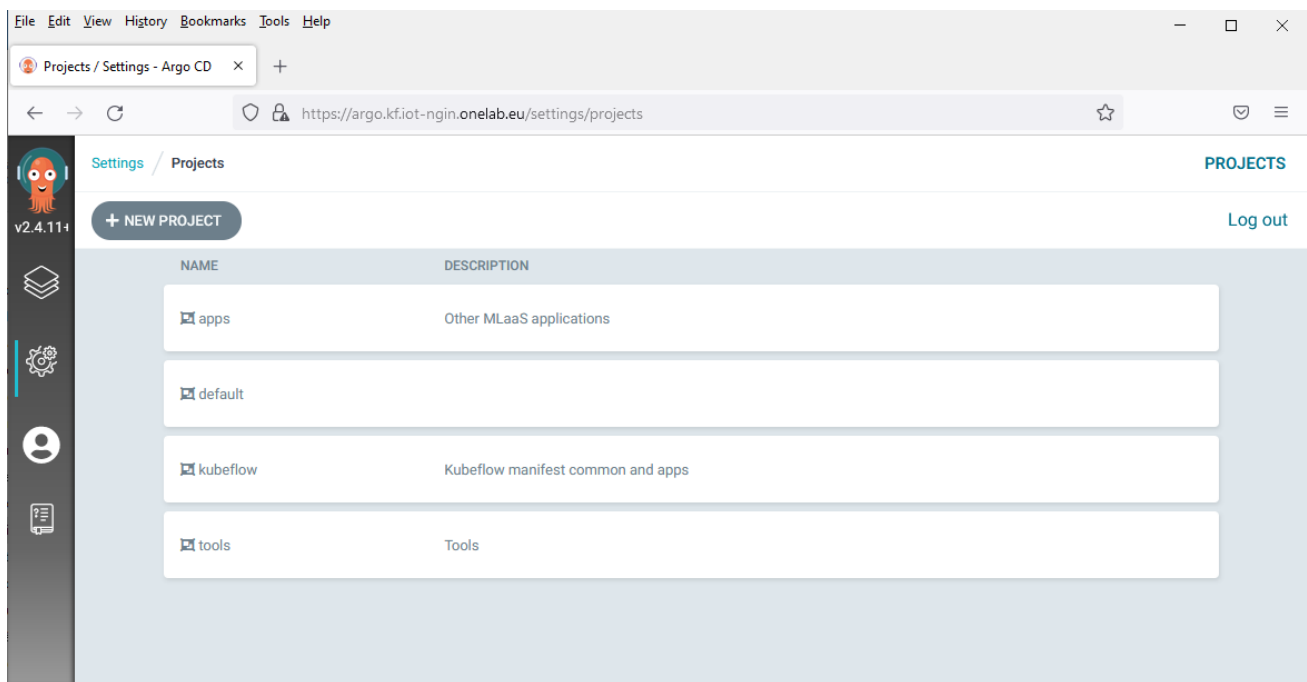


Figure 73: ArgoCD projects.

The components for the deployment are grouped into ArgoCD applications. For Kubeflow the grouping follows the structure of the Kubeflow manifest repository. This choice has been made to simplify synchronization with the official Kubeflow Manifest repository [5] and easily identify the possible failing part of Kubeflow. The “mlaas” folder contains the ArgoCD application files in folders called “argocd-apps”.

The installation includes first a manual bootstrap phase to install the tools required for GitOps, which is used for the rest of the installation and then, in a second phase, the installation of the various components by ArgoCD.

For the bootstrap phase:

1. Install ArgoCD. This component is used to deploy the various Kubernetes components. Optionally, the ArgoCD CLI can be installed on a management station.
2. Copy the IaC files to the GitOps repository
3. Change the address of the GitOps repository in the files
4. Add GitOps directory to ArgoCD configuration
5. Create projects in ArgoCD
6. Copy the files or sensitive apps with the secrets to a local machine
7. Copy the ArgoCD application files to a local machine

Once the bootstrap is complete, the installation of the platform is executed by creating the ArgoCD applications from the various application files. ArgoCD then fetches the GitOps repository corresponding to the application and creates all the Kubernetes resources defined in the associated yaml files. The installation relies heavily on Kustomize [54] to simplify creation of the yaml files. For applications relying on secrets outside the GitOps mechanism, these secrets must be created manually.

Note that at the time of this writing, work is still on-going on the installation process, especially to make it as simple as possible.

## 5.1.2 Application installation

To install the platform, all the ArgoCD applications must be created. This is done by launching the yaml files located into the "argocd-apps" directories.

For example, the following command in Listing 4 is used to install the Kubeflow central dashboard:

```
\mlaas\kubeflow\argocd-apps>kubectl create -n argocd -f centraldashboard.yaml
application.argoproj.io/centraldashboard created
```

Listing 4: ArgoCD application installation.

After a few seconds the application appears into the ArgoCD dashboard as shown in Figure 74.

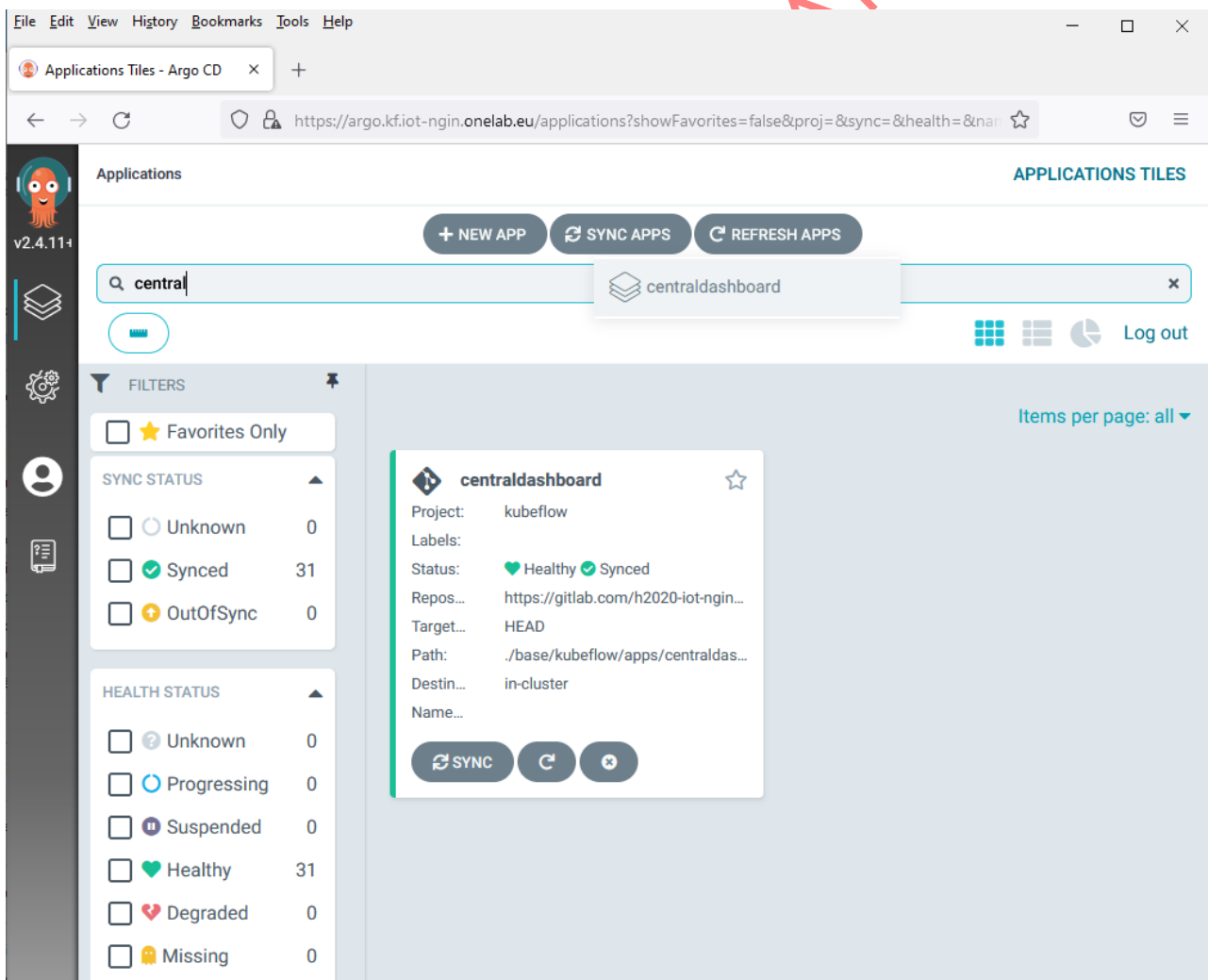


Figure 74: ArgoCD application example.

As shown in Figure 75, details about the Kubernetes resource linked to the application can be seen by clicking on the application.

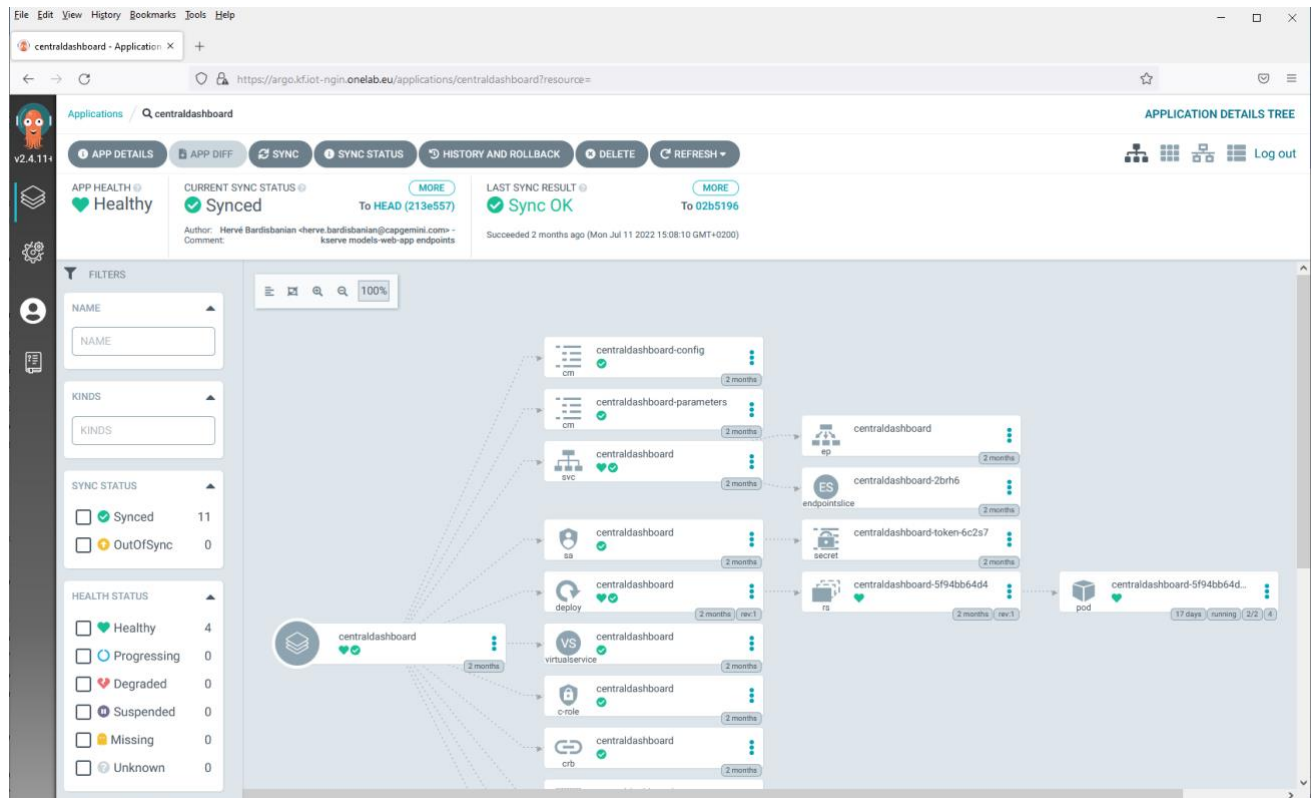


Figure 75: ArgoCD application example.

Figure 76 shows the ArgoCD console with all the applications from the IoT-NGIN MLaaS test system installed. We can easily check that all applications are synchronized and healthy.

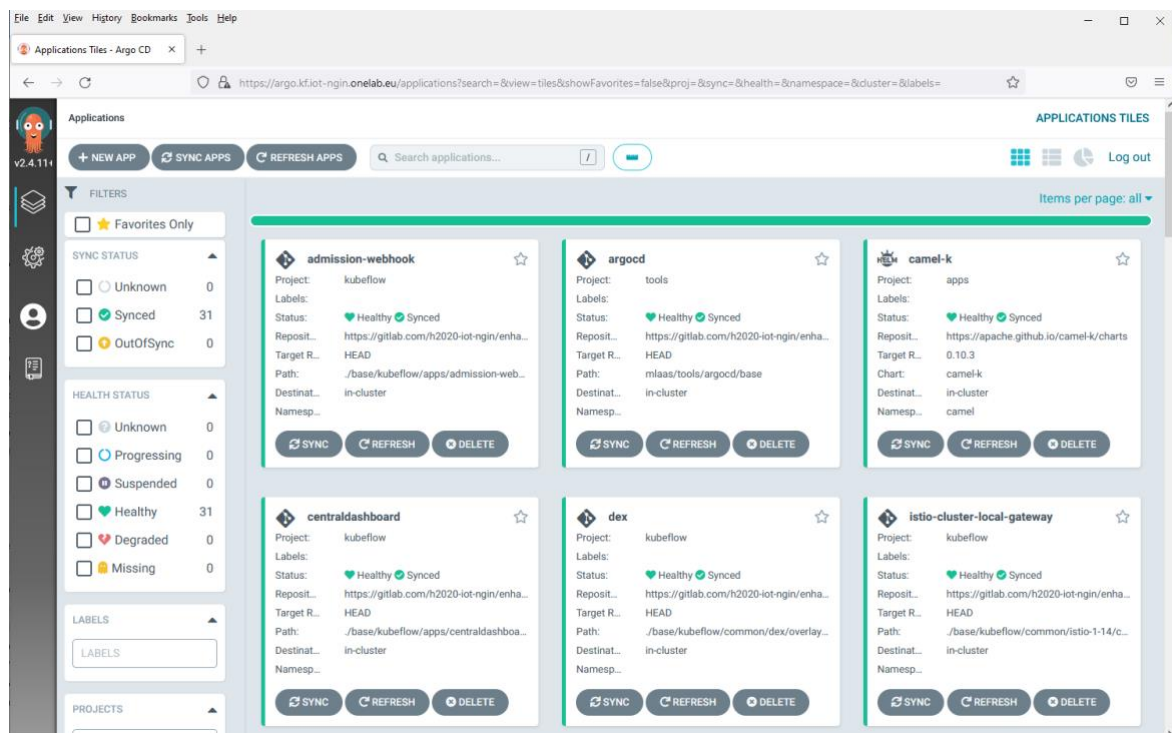


Figure 76: IoT-NGIN MLaaS test applications.

Note that ArgoCD is added as an ArgoCD application after the bootstrap as shown in Figure 77. It allows ArgoCD to also benefit from the GitOps mechanisms by monitoring its own installation repository.

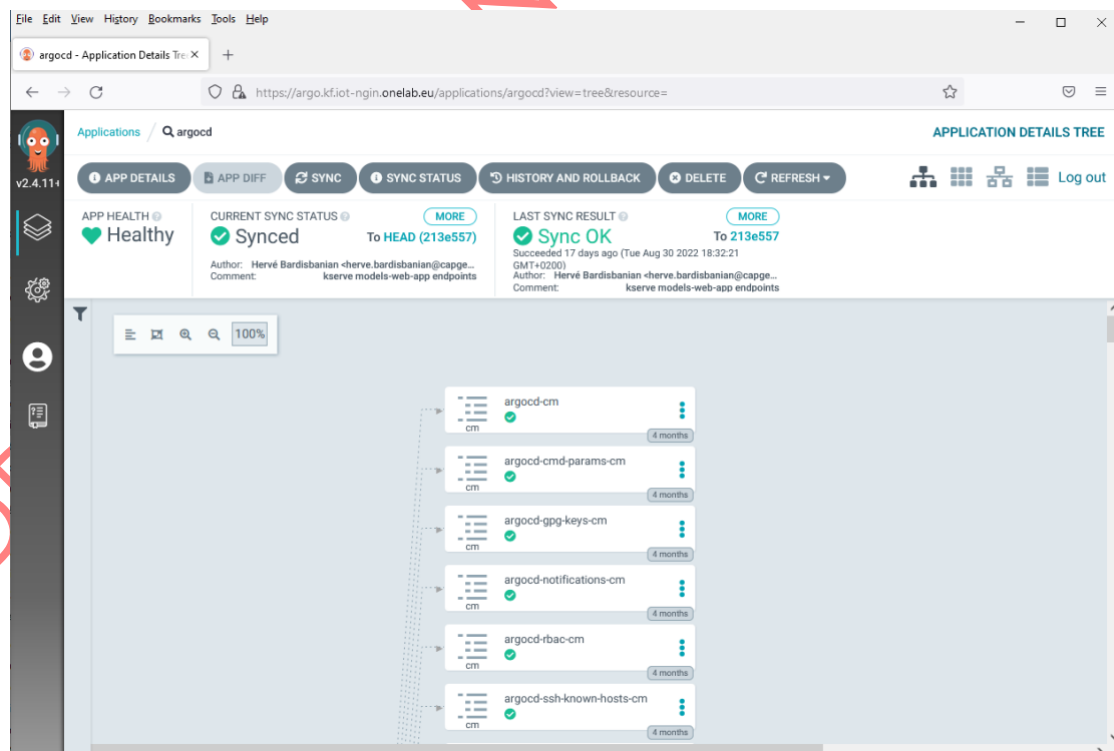
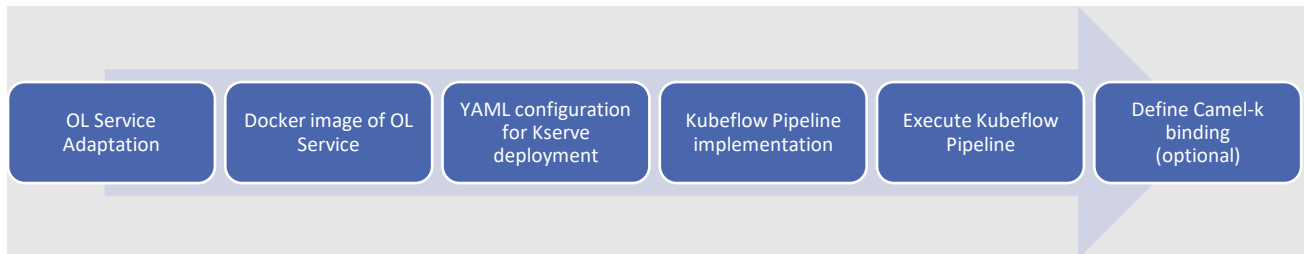


Figure 77: ArgoCD application for ArgoCD.

## 5.2 Online Learning Framework

This section describes the process to create an Online Learning service. The following figure shows the workflow.



Before starting to deploy the Online Learning service, it is required to have a ML/DL model implemented. The model creation is out of the scope of OL framework and must be provided by the ML engineers that request its training with the Online Learning service. It is important to specify the Python library used to implement the model because it will be important in further steps. Once the model is defined, it must be saved in MinIO (see Figure 78), or on another cloud object storage such as S3 or GCP are also supported.

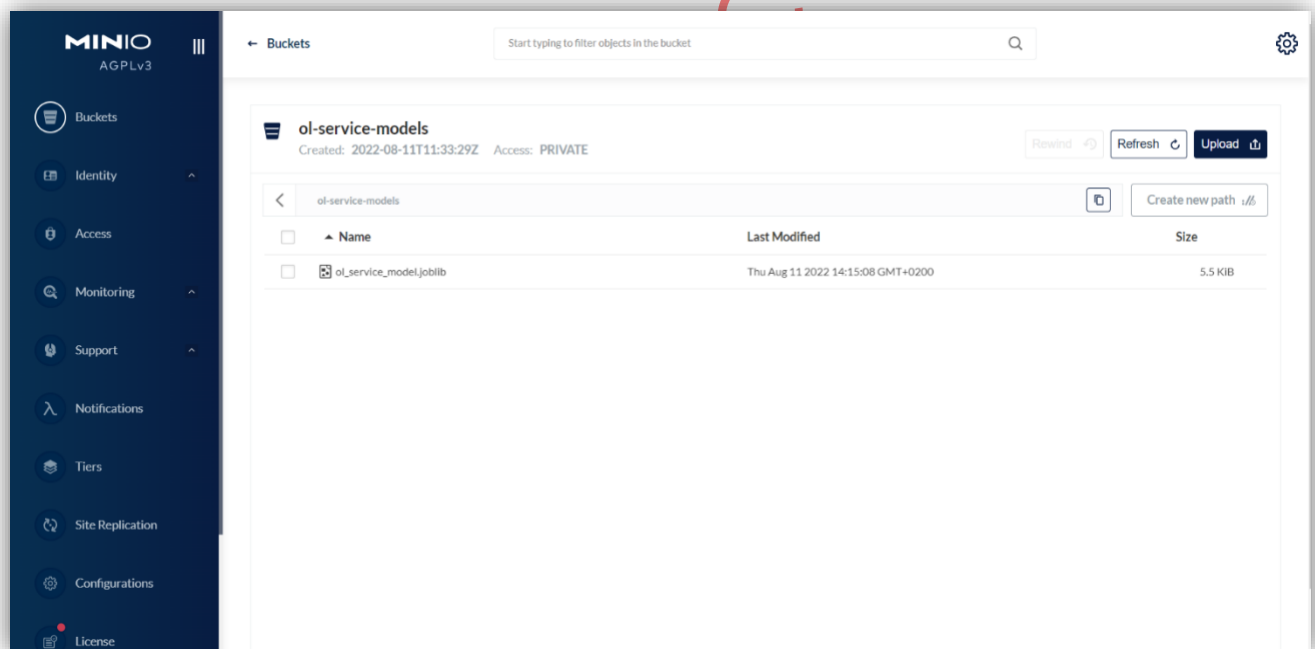


Figure 78: Baseline OL model in MinIO storage.

The initial step consists of configuring the OL service in order to establish different parameters that are specific to each service such as the ML model location in MinIO (host and bucket), the name of the model, the framework used to implement the model (backend), the OL service name, etc.

Once all the configuration parameters are defined and the OL service is adapted to the use case, the next step is to encapsulate it in a Docker image and upload the Docker image into a Docker Registry. Thus, Kubeflow can include the image into the pipeline. An example of

the *Dockerfile* that is used to generate the Docker image of the OL service is shown in Listing 5. The configuration parameters defined in previous steps are included in this file. The Docker image creation can be done with the *docker CLI*<sup>17</sup> tool by means of *docker build* command. Once the Docker image is created, it is possible to upload the image by using *docker push* command. It is important to note that the name of the Docker image must be *<Registry\_Name>/<Image\_Name>:Version* so that the image can be uploaded to the Docker registry. The Dockerfile can be reached at GitLab [https://gitlab.com/h2020-iot-ngin/enhancing\\_iot\\_intelligence/t3\\_2/online\\_learning/-/blob/main/Dockerfile\\_Kubeflow](https://gitlab.com/h2020-iot-ngin/enhancing_iot_intelligence/t3_2/online_learning/-/blob/main/Dockerfile_Kubeflow).

```
# For more information, please refer to https://aka.ms/vscode-docker-python
FROM python:3.8

# Keeps Python from generating .pyc files in the container
ENV PYTHONDONTWRITEBYTECODE=1

# Turns off buffering for easier container logging
ENV PYTHONUNBUFFERED=1

WORKDIR /online_learning

# Install pip requirements
COPY requirements.txt .
RUN python -m pip install -r requirements.txt

# Copy project folders
COPY . /online_learning

ENTRYPOINT ["python", "create_onlinelearning_kubeflow.py", "--arg", "value"]
```

Listing 5: Dockerfile for OL service container.

At this point, the OL service is containerized and its image available in a Docker repository. Before deploying it, by using Kubeflow and Kserve, it is required to define a YAML manifest. An example of manifest is available in GitLab at [https://gitlab.com/h2020-iot-ngin/enhancing\\_iot\\_intelligence/t3\\_2/online\\_learning/-/blob/main/kubeflow/kserve\\_isvc.yaml](https://gitlab.com/h2020-iot-ngin/enhancing_iot_intelligence/t3_2/online_learning/-/blob/main/kubeflow/kserve_isvc.yaml). This file specifies aspects of the service such as the Docker image location, the inference service name, the number of replicas, etc. Listing 6 code snippet shows this configuration.

<sup>17</sup> <https://docs.docker.com/engine/reference/commandline/cli/>



```

apiVersion: serving.kserve.io/v1beta1
kind: InferenceService
metadata:
  labels:
    controller-tools.k8s.io: "1.0"
  name: <Inference_Service_Name>
  namespace: <Namespace>
  annotations:
    sidecar.istio.io/inject: "false"
spec:
  predictor:
    containers:
      - name: kserve-container
        image: <OL_Service_Image>
        command: ["<Arguments>"]

```

Listing 6: YAML manifest for OL Service in KubeFlow/KServe.

Kubeflow allows the creation of Jupyter notebooks for defining a pipeline to deploy an inference service like the Online Learning service. Figure 79 and Figure 80 show the steps to take in Kubeflow to create a Jupyter notebook.

Kubeflow offers an SDK API to deal with Kubeflow pipelines named *KFP*<sup>18</sup>. The pipeline is a description of an ML workflow, including all the components the workflow consists of and the interactions between them. Figure 81 shows the notebook that declares the Kubeflow pipeline for the deployment of the Online Learning service, consisting of 2 different code cells. The first cell imports all required Python libraries whereas the second cell defines a function. This function creates a Kubernetes resource from previous the previously created Kserve YAML manifest. Then, the function is executed, and a pipeline descriptor is created. Next, this pipeline descriptor is uploaded into Kubeflow. An example of Jupyter notebook for pipeline creation is available at: [https://gitlab.com/h2020-iot-ngin/enhancing\\_iot\\_intelligence/t3\\_2/online\\_learning/-/tree/main/kubeflow](https://gitlab.com/h2020-iot-ngin/enhancing_iot_intelligence/t3_2/online_learning/-/tree/main/kubeflow).

<sup>18</sup> <https://kubeflow-pipelines.readthedocs.io/en/stable/index.html>

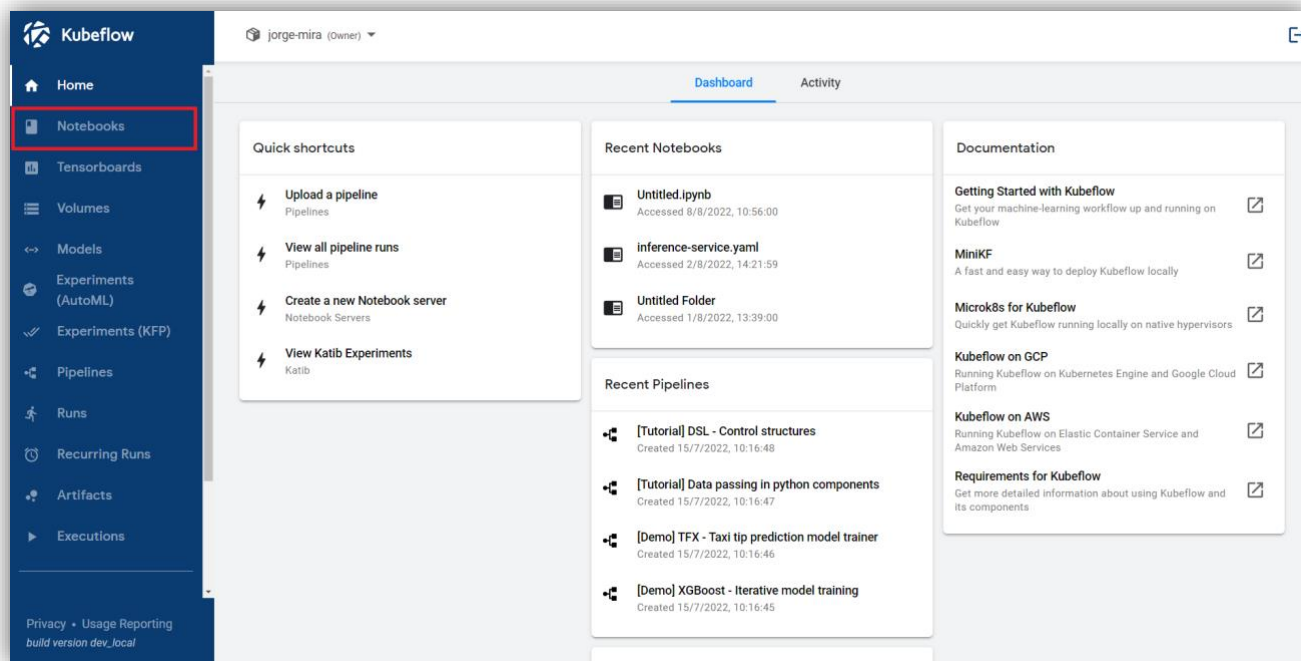


Figure 79: Kubeflow dashboard.

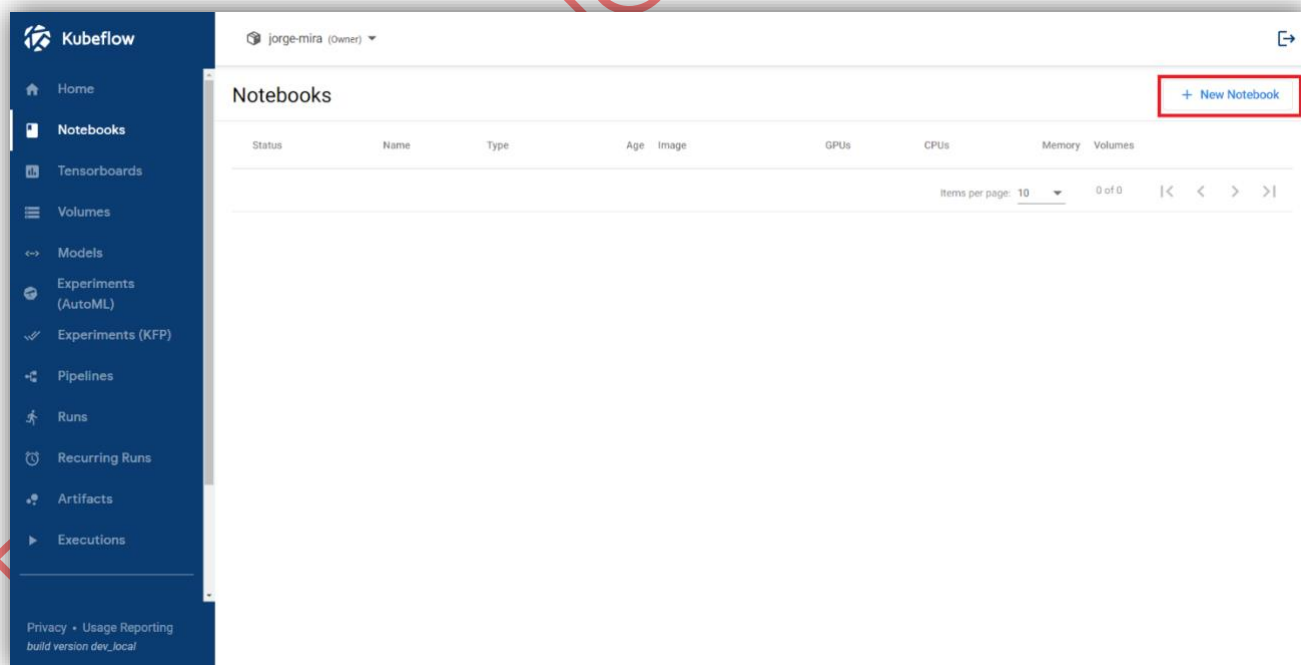
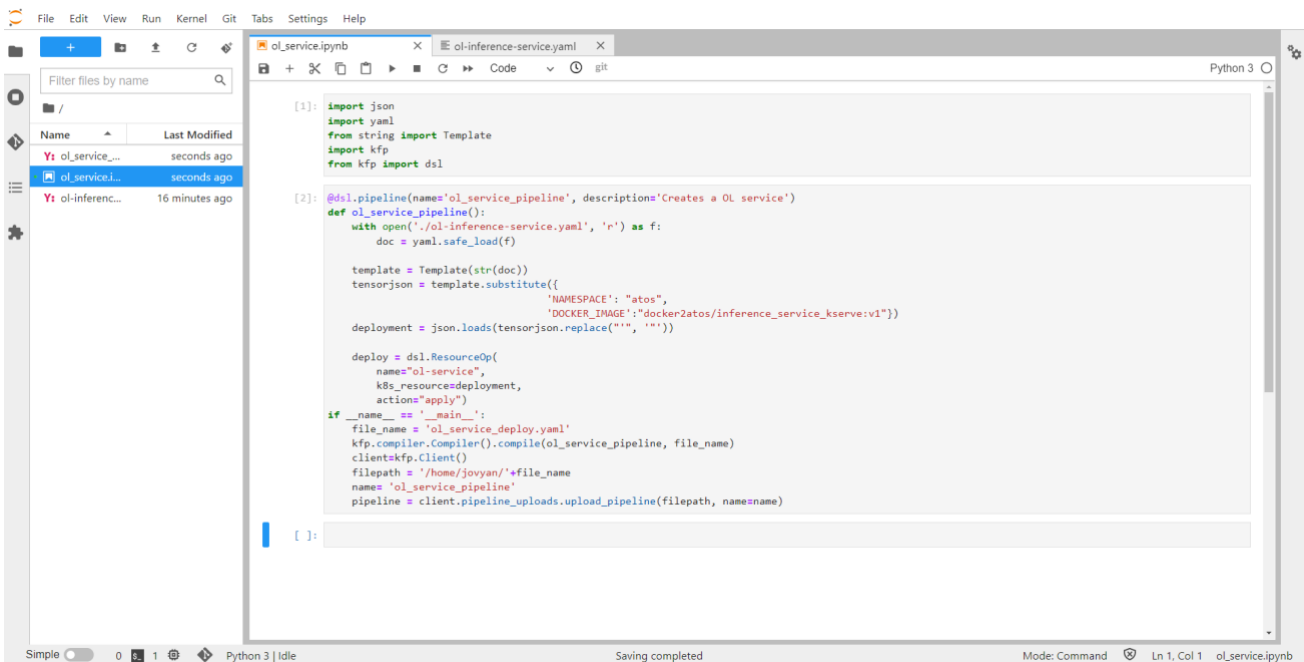


Figure 80: Creation of Jupyter's notebooks in Kubeflow interface.

## D3.3 - ENHANCED IOT FEDERATED DEEP LEARNING/ REINFORCEMENT ML



```

[1]: import json
import yaml
from string import Template
import kfp
from kfp import dsl

[2]: @dsl.pipeline(name='ol_service_pipeline', description='Creates a OL service')
def ol_service_pipeline():
    with open('./ol-inference-service.yaml', 'r') as f:
        doc = yaml.safe_load(f)

        template = Template(str(doc))
        tensorjson = template.substitute({
            'NAMESPACE': 'atos',
            'DOCKER_IMAGE': 'docker2atos/inference_service_kserve:v1'})
        deployment = json.loads(tensorjson.replace("'", ''))

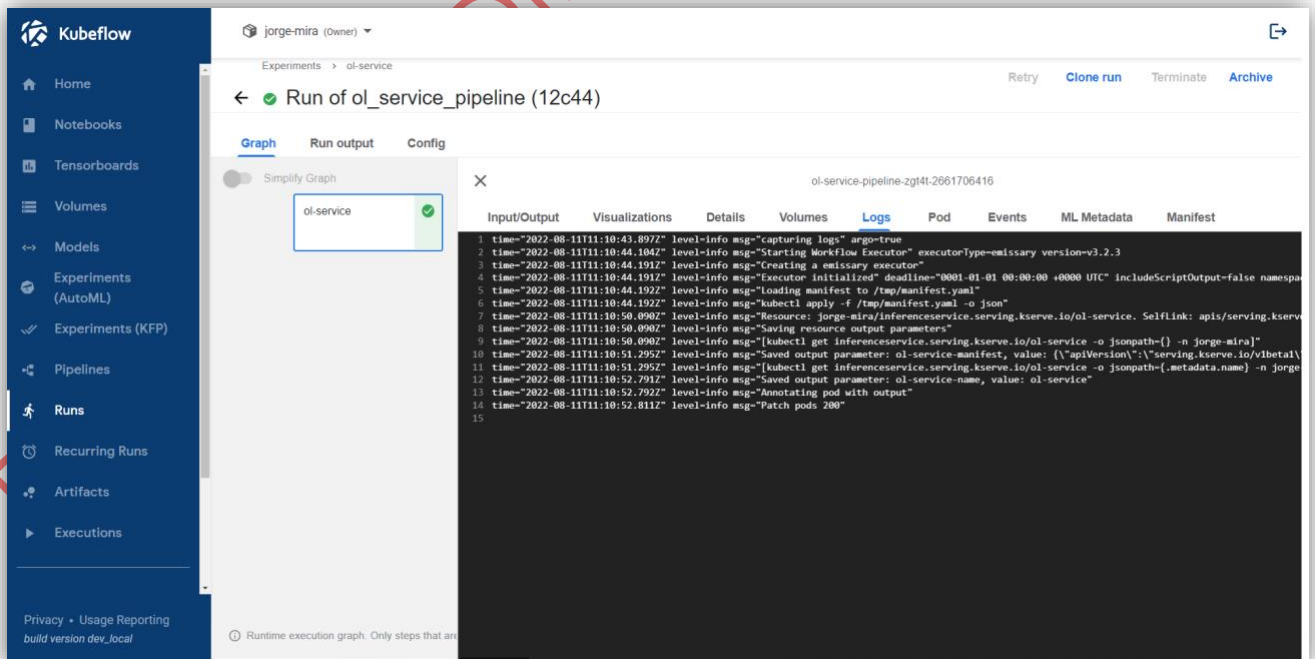
        deploy = dsl.ResourceOp(
            name='ol-service',
            k8s_resource=deployment,
            actions='apply')
    if __name__ == '__main__':
        file_name = 'ol_service_deploy.yaml'
        kfp.compiler.Compiler().compile(ol_service_pipeline, file_name)
        client=kfp.Client()
        filepath = '/home/jovyan/' + file_name
        name= 'ol_service_pipeline'
        pipeline = client.pipeline_uploader.upload_pipeline(filepath, name=name)

[ ]:

```

Figure 81: Jupyter notebook defining a Kubeflow pipeline for deploying the OL service.

The execution of the pipeline will deploy the Online Learning service in Kserve as an inference service and expose it to an HTTP endpoint. To execute the pipeline, an experiment must be created first (if it is not created yet) and then a run. These steps are performed by using the Kubeflow user interface. Figure 82 shows the results of executing the pipeline.



The screenshot shows the Kubeflow user interface. On the left is a navigation sidebar with options like Home, Notebooks, Tensorboards, Volumes, Models, Experiments (AutoML), Experiments (KFP), Pipelines, Runs, Recurring Runs, Artifacts, and Executions. The main panel displays the 'Run of ol\_service\_pipeline (12c44)' under the 'Experiments > ol-service' section. It includes buttons for 'Retry', 'Clone run', 'Terminate', and 'Archive'. Below these are tabs for 'Graph', 'Run output', and 'Config'. The 'Graph' tab shows a simplified graph with a single node 'ol-service'. The 'Run output' tab is active, showing a log of execution steps with timestamps and messages, such as 'Starting Workflow Executor', 'Creating a emissary executor', 'Loading manifest to /tmp/manifest.yaml', and 'Saving resource output parameters'.

Figure 82: Kubeflow pipeline execution for deploying the OL service.

Once the execution has finished, the inference service is deployed. The deployment presents an HTTP endpoint which is waiting for new data to train the ML model. Usually the endpoint looks like:

<https://<InferenceServiceName>.<Domain>.<Extension>/v1/models/<ModelName>:predict>

The power generation forecasting OL service is deployed on the Kubernetes cluster of the project. All the services require an access token to ask for model update or inferences. This service presents the following endpoint:

<http://ol-smart-grid-power-consumption.jmira.kserve.kf.iot-ngin.onelab.eu/v1/models/<ModelName>:predict>

There is an additional step which is only required when data comes from real time protocols (e.g., Kafka and MQTT) since the inference service created through Kserve only supports HTTP communications. To solve this situation, a Camel-K binding that acts as message forwarding is required. This binding is created from a YAML manifest, where data source (i.e., the MQTT/Kafka broker and the specific topic) is indicated along with the destination (in this case the deployed inference service). Thus, the inference service receives the new data and can either update the model or perform a prediction. Kubernetes cluster is needed to apply the YAML file and create the Camel-k binding. Listing 7 shows an example of the YAML manifest.

```
apiVersion: camel.apache.org/v1alpha1
kind: KameletBinding
metadata:
  name: <BindingName>
  namespace: <Namespace>
spec:
  source:
    ref:
      kind: Kamelet
      apiVersion: camel.apache.org/v1alpha1
      name: <SourceName>
  properties:
    brokerUrl: <BrokerURL>
    topic: <Topic>
    clientId: <ClientID>
    password: <Pass>
    username: <User>
  sink:
    uri: <ISVC_URL>
```

Listing 7: YAML manifest for Camel-k binding.

## 5.3 Privacy-preserving Framework

## Federated

## Learning

### 5.3.1 NVIDIA FLARE

This section contains the necessary steps to run the integration of YOLOv5, with NV FLARE.

#### Installation

```
1. git clone https://github.com/NVIDIA/NVFlare.git
2. cd NVFlare/examples
3. git clone https://gitlab.com/h2020-iot-ngin/enhancing_iot_intelligence/privacy-preserving-federated-learning/privacy-preserving-fl-frameworks/nvflare-for-object-detection/nvidia-flare-yolov5-integration.git
4. cd nvidia-flare-yolov5-integration
5. conda create --name <flare-yolo> python=3.8
6. conda activate <flare-yolo>
7. pip install -r requirements.txt
```

#### Change basic parameters

1. Edit the NVFlare/examples/nvidia-flare-yolov5-integration/yolov5/custom/data/myfilename.yaml file by adding your dataset directory path, number, and names of training classes.

```
path: home/user/datasets/mydataset # dataset root dir

train: images/train # train images (relative to 'path') 128 images

val: images/val # val images (relative to 'path') 128 images

test: images/test

# Classes

nc: 2 # number of classes

names: [ 'person', 'car' ] # class names`
```

2. Go back to the NVFlare/examples/nvidia-flare-yolov5-integration/yolov5/config/ and edit the config\_fed\_client.json and config\_fed\_server.json files. On the client's json file you can insert:

- **"imgsz"**: image size of your data
- **"epochs"**: number of epochs to train for every round
- **"batch\_size"**: batch size
- **"num\_clients"**: the number of clients
- **"pretrained"**: use pretrained weights (Yes or No)

On the server's json file you can insert:

- **"min\_clients"**: minimum number of clients
- **"num\_round"**: the number of rounds

Also, on server's json file you can choose if you want a pretrained model with pretrained weights or a model with randomized weights (begin the whole training process from scratch). In the json file insert:

- *For pretrained network:*  
**"path"**: yolonetwork.YoloPretrainedNetwork
- *For no-pretrained network:*  
**"path"**: yolonetwork.YoloNetwork

### How to run

Go back to the root folder and begin the federating learning process in POC (Proof of Concept) mode. The example below is for 2 clients.

```
1. poc -n 2
2. mkdir -p poc/admin/transfer
3. cp -rf NVFlare/examples/* poc/admin/transfer
```

Once you are ready to start the FL system, you can run the following commands to start the server and client systems.

- The first step is starting the FL server:  
./poc/server/startup/start.sh
- Once the server is running, open a new terminal and start the first client:  
./poc/site-1/startup/start.sh
- Open another terminal and start the second client:  
./poc/site-2/startup/start.sh
- In one last terminal, start the admin client:  
./poc/admin/startup/fl\_admin.sh localhost

With the admin client that started from `fl_admin.sh` file, you can control the whole process. You can check the status of the clients and server, you can start the process, stop the process etc. With the admin client command prompt successfully connected and logged in automatically, enter the command.

```
1. submit_job nvidia_flare-yolov5-integration/yolov5
```

In case you want to abort the process enter the command.

```
1. abort_job job_id
```

Where `job_id` is the id that the admin gave you when you started the process. When the whole process ends enter the commands below to shut down the clients and the server. The password is `admin`. For further information about the federated learning process you can read NV FLARE documentation [25].

1. shutdown client
2. shutdown server

### 5.3.2 IoT-NGIN FedPATE

The code and the installation procedures are available on the project Gitlab repository, accessible at:

[https://gitlab.com/h2020-iot-ngin/enhancing\\_iot\\_intelligence/privacy-preserving-federated-learning/privacy-preserving-fl-frameworks/fedpate](https://gitlab.com/h2020-iot-ngin/enhancing_iot_intelligence/privacy-preserving-federated-learning/privacy-preserving-fl-frameworks/fedpate)

We provide two ways to run the FedPATE:

1. With Conda environment, version  $\geq 4.11.0$ .
2. With Docker, version  $\geq 19.03.8$ .

A description of both ways is given in the following.

#### 5.3.2.1 Conda Environments

a) First, install the project's dependencies:

1. `git clone https://gitlab.com/h2020-iot-ngin/enhancing_iot_intelligence/privacy-preserving-federated-learning/privacy-preserving-fl-frameworks/fedpate.git`
2. `conda create -name fedpate_env --file requirements.txt`
3. `conda activate fedpate_env`
4. `./dev/bootstrap.sh`

b) Install PyTorch from <https://pytorch.org/> depending on your system's requirements.

c) Run the server of the FL system:

1. `python3 src/py/server.py --net_name=test_net --num_rounds=10 --num_teachers=14 --epochs=10 --epsilon=0.2 --num_queries=9000 --noise_data=9000`

d) Open as many terminals as the number of teachers + 1 are (Teachers + Student), and run the teachers and the student with the command:

1. `python3 src/py/PATE_pytorch_client_2.py --teacher_id=0 --net_name=test_net --num_rounds=10 --num_teachers=14 --teacher_epochs=10 --student_epochs=20 --epsilon=0.2 --num_queries=9000 --noise_data=9000`

where:

- `--net_name`: str, name of your model.
- `--num_rounds`: int, number of rounds for FL.
- `--num_teachers`: int, the number of teachers/clients.
- `--noise`: str, type of noise for the aggregation mechanism ('laplacian' or 'gaussian', default='laplacian').
- `--teacher_epochs`: int, number of epochs to train the teachers.
- `--student_epochs`: int, number of epochs to train the student.
- `--epsilon`: float, epsilon for laplacian distribution (if `--noise='laplacian'`).



- `--sigma`: float, standard deviation for gaussian-normal distribution (if `--noise='gaussian'`).
- `--num_queries`: int, number of queries to the student.
- `--noise_data`: int, the number of data to add noise (must be less or equal than `num_queries`).

`--teacher_id`: int, the ID of the current teacher (must be in range 0 - `num_teachers`, where `teacher_id = num_teachers` is the Student)

### 5.3.2.2 Docker

a) Set up the docker container.

1. `git clone https://gitlab.com/h2020-iot-ngin/enhancing\_iot\_intelligence/privacy-preserving-federated-learning/privacy-preserving-fl-frameworks/fedpate.git`
2. `docker run -p 8888:8888 -v ~/your_path_to_flower-pate-integration/src/py:/flower -it synelixis/flower-pate:v1.8`

- b) Open your browser and go to the `127.0.0.1:8888` address and enter the token that the terminal gave you. You should see a *JupyterLab* running.
- c) Run the flwr server of the FL system by opening a new terminal inside the JupyterLab and running:

1. `python3 /flower/server.py --net_name=test_net --num_rounds=10 --num_teachers=14 --epochs=10 --epsilon=0.2 --num_queries=9000 --noise_data=9000`

d) Open as many terminals as the number of teachers + 1 are (Teachers + Student), inside the JupyterLab and run the teachers and the student with the command:

1. `python3 /flower/PATE_pytorch_client_2.py --teacher_id=0 --net_name=test_net --num_rounds=10 --num_teachers=14 --teacher_epochs=10 --student_epochs=20 --epsilon=0.2 --num_queries=9000 --noise_data=9000`

### 5.3.3 TensorFlow Federated

The code and instructions for replicating the results we demonstrated can be found at:

[https://gitlab.com/h2020-iot-ngin/enhancing\\_iot\\_intelligence/privacy-preserving-federated-learning/privacy-preserving-fl-frameworks/tff-for-ids](https://gitlab.com/h2020-iot-ngin/enhancing_iot_intelligence/privacy-preserving-federated-learning/privacy-preserving-fl-frameworks/tff-for-ids)

#### 5.3.3.1 Set-up

To correctly set up this project, one needs to have the right environment of library versions, python version etc. To achieve this the following steps must be followed:

- Create a virtual environment
- Install requirements from requirements.txt

- Have python 3.8.10

In our case we used conda environment and pip as the package manager, so an example of this procedure is (after navigating to the project's folder):

- `conda create -n venv python==3.8.10`
- `conda activate venv`
- `pip install -r requirements.txt`

### 5.3.3.2 Model Running

To run the code, one needs to run the following command which enables argument parsing. The arguments customize the learning process and allow multiple and different experiments to be conducted by the user:

```
1. python3 src/tff.py -train_path -test_path --imb_parameter --dp_parameter --n_clients
```

where:

- `--train_path`: path to train dataset, e.g. `.../UNSW_NB15_training-set.csv`
- `--test_path`: path to train dataset, e.g. `.../UNSW_NB15_testing-set.csv`
- `--imb_parameter`: parameter that defines class imbalance. 1 means no imbalance and higher than one means higher data imbalance distribution.
- `--dp_parameter`: parameter that defines application of differential privacy. 0 means no DP and higher than 0 means stronger application of DP.
- `--n_clients`: the number of FL clients.

## 6 Conclusions

The present report has outlined the IoT-NGIN tools for enhancing IoT intelligence. Initially, the MLaaS platform has been presented, following the MLOps paradigm for managing the complete lifecycle of ML tasks. Apart from building and deploying ML services, the MLaaS platform supports next-generation IoT-related technological schemes, such as AI in edge/fog/cloud computing, integration with Digital Twins and Distributed Ledgers. The fact that it builds on state-of-the-art open source tools, ensures the long-term updates of its components, not only from functional, but also from a security perspective. With its MLaaS platform, IoT-NGIN goes beyond existing integrated open-source MLOps, by integrating additional services, such as ML model storage, data storage, data acquisition, access management and CI/CD support.

In addition, three ML techniques are incorporated in the MLaaS platform, namely the Online Learning, the Reinforcement Learning and the Privacy-Preserving Federated Learning framework. Although the concepts are not new in the ML industry, IoT-NGIN provides open-source design and implementation and integrates them in its open-source MLaaS platform. OL is already available through the MLaaS platform, while for RL the technical specification and concrete usage for Smart Energy LL use cases has been defined. Federated Learning will be also integrated in the next release, expected as part of deliverable D3.4. In the present report, Federated Learning has been proposed following the as-a-service model for three state-of-the-art FL frameworks. The focus has been on enhancing and investigating those three frameworks, namely NVIDIA FLARE, Flower and Tensorflow Federated (TFF) with privacy preservation. Although FLARE and TFF include such techniques, it is not clear how different models behave with each of them. To that end, the present document presents the experimental evaluation of the model performance when enhanced with varying privacy preservation levels. In addition, since Flower is less mature in such techniques, it has been integrated with PATE, yielding the FedPATE framework in the context of IoT-NGIN, providing in this way an easy to develop, scalable, ML framework-agnostic FL framework with privacy guarantees.

The software implementation of the components presented in this deliverable are provided as open source on the project's page on the public Gitlab repository, at <https://gitlab.com/h2020-iot-ngin/enhancing-iot-intelligence/>. Details about installation and usage are also provided in this report for convenience of interested audience.

## 7 References

- [1] BDVA, "Big Data Value (BDV) Strategic Research and Innovation Agenda (SRIA)," 2017. [Online]. Available: [https://bdva.eu/sites/default/files/BDVA\\_SRIA\\_v4\\_Ed1.1.pdf](https://bdva.eu/sites/default/files/BDVA_SRIA_v4_Ed1.1.pdf).
- [2] IoT-NGIN, D3.1 - Enhancing deep learning and reinforcement learning, H2020-957246 IoT-NGIN Deliverable Report, 2021.
- [3] Kubeflow, "The Machine Learning Toolkit for Kubernetes," [Online]. Available: <https://www.kubeflow.org/>. [Accessed 2022].
- [4] MINIO, "MinIO for Kubernetes," [Online]. Available: <https://min.io/product/kubernetes>. [Accessed 2022].
- [5] Kubeflow, "Kubeflow Manifests," Github, [Online]. Available: <https://github.com/kubeflow/manifests>. [Accessed 2022].
- [6] KServe, "End to end inference service example with Minio and Kafka," [Online]. Available: <https://kserve.github.io/website/0.9/modelserving/kafka/kafka/>. [Accessed 2022].
- [7] Kubeflow, "An overview of Kubeflow's architecture," 2022. [Online]. Available: <https://v1-5-branch.kubeflow.org/docs/started/architecture/>. [Accessed 2022].
- [8] Kubeflow, "Kubeflow Documentation - KServe," [Online]. Available: <https://www.kubeflow.org/docs/external-add-ons/kserve/kserve/>. [Accessed 2022].
- [9] KServe, "KServe Documentation - Data Plane," [Online]. Available: [https://kserve.github.io/website/0.9/modelserving/data\\_plane/](https://kserve.github.io/website/0.9/modelserving/data_plane/). [Accessed 2022].
- [10] National Security Agency, "Avoid Dangers of Wildcard TLS Certificates and the ALPACA Technique," Cybersecurity Information Sheet, 2021. [Online]. Available: [https://media.defense.gov/2021/Oct/07/2002869955/-1/-1/0/CSI\\_AVOID%20DANGERS%20OF%20WILDCARD%20TLS%20CERTIFICATES%20AND%20THE%20ALPACA%20TECHNIQUE\\_211007.PDF](https://media.defense.gov/2021/Oct/07/2002869955/-1/-1/0/CSI_AVOID%20DANGERS%20OF%20WILDCARD%20TLS%20CERTIFICATES%20AND%20THE%20ALPACA%20TECHNIQUE_211007.PDF). [Accessed 2022].
- [11] D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning internal representations by error propagation," San Diego, California: Institute for Cognitive Science, University of California., 1985.
- [12] T. M. Y. B. Razvan Pascanu, "On the difficulty of training Recurrent Neural Networks," 2012.
- [13] R. Dey and F. M. Salem, "Gate-Variants of Gated Recurrent Unit (GRU) Neural," Department of Electrical and Computer Engineering, 2017.
- [14] R. B. Z. Bharath Ramsundar, TensorFlow for Deep Learning, O'Reilly Media, Inc., 2018.

- [15] D. P. Kingma and J. Lei Ba, "ADAM: A method for stochastic optimization," 2014.
- [16] S. S. Shapiro and M. B. Wilk, "An Analysis of Variance Test for Normality," 1965.
- [17] T. W. Anderson and D. A. Darling, "Asymptotic Theory of Certain "Goodness of Fit" Criteria Based on Stochastic Processes," 1952.
- [18] R. B. D'Agostino and E. S. Pearson, "Tests for Departure from Normality. Empirical Results for the Distributions of  $b_2$  and  $\sqrt{b_1}$ ," in *Biometrika*, 1973, p. 613–622.
- [19] Y. Li, "Deep Reinforcement Learning: An Overview," *arXiv*, 2017.
- [20] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," 2013.
- [21] V. Mnih, A. Puigdomènech Badia, M. Mirza, A. Graves, T. Harley, T. . P. Lillicrap, D. Silver and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," 2016.
- [22] Forrester, "Federated Learning: A Distributed Machine Learning Technique Without Data Aggregation," 2022. [Online]. Available: <https://www.forrester.com/report/federated-learning-a-distributed-machine-learning-technique-without-data-aggregation/RES177214>.
- [23] IoT-NGIN, "D3.2 - Enhancing Confidentiality preserving federated ML," H2020 - 957246 - IoT-NGIN Deliverable Report, 2021.
- [24] IoT-NGIN, "D5.1 - Enhancing IoT Cybersecurity," H2020 - 957246 - IoT-NGIN Deliverable Report, 2021.
- [25] NVIDIA DEVELOPER, "NVIDIA FLARE," [Online]. Available: <https://developer.nvidia.com/flare>.
- [26] Adap GmbH, "Flower," open source, [Online]. Available: <https://flower.dev>. [Accessed 2022].
- [27] TensorFlow, "TensorFlow Federated: Machine Learning on Decentralized Data," [Online]. Available: <https://www.tensorflow.org/federated>. [Accessed 2022].
- [28] R. Shokri, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, 2015.
- [29] C. Dwork, M. Naor, O. Reingold, G. N. Rothblum and S. Vadhan, "On the complexity of differentially private data release: efficient algorithms and hardness results," *STOC*, p. 381–390, 2009.
- [30] M. Lyu, D. Su and L. Ninghui, "Understanding the sparse vector technique for differential privacy," *arXiv preprint arXiv:1603.01699*, 2016.

- [31] Benaissa and Ayoub, "TenSEAL: A library for encrypted tensor operations using homomorphic encryption," *arXiv preprint arXiv:2104.03152*, 2021.
- [32] NVIDIA FLARE, "provision.py," 2022. [Online]. Available: <https://github.com/NVIDIA/NVFlare/blob/b799c15ed82da5df32c3bff00aa92c15f366fa83/nvflare/lighter/provision.py>.
- [33] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. Porto Buarque de Gusmao and N. D. Lane, "FLOWER: A friendly federated learning framework," 2022.
- [34] N. Papernot, S. Song, I. Mironov, A. Raghunathan, K. Talwar and Ú. Erlingsson, "SCALABLE PRIVATE LEARNING WITH PATE," in *ICLR*, 2018.
- [35] N. Papernot, M. Abadi, Ú. Erlingsson, I. Goodfellow and K. Talwar, "iclr.cc," 2017. [Online]. Available: [https://iclr.cc/archive/www/lib/exe/fetch.php%3Fmedia=iclr2017:paperno\\_iclr2017.pdf](https://iclr.cc/archive/www/lib/exe/fetch.php%3Fmedia=iclr2017:paperno_iclr2017.pdf).
- [36] Y. Pan, J. Ni and Z. Su, "FL-PATE: Differentially Private Federated Learning with Knowledge Transfer," in *IEEE*, 2021.
- [37] N. Papernot, M. Abadi, U. Erlingsson, I. Goodfellow and K. Talwar, "Semi-supervised knowledge transfer for deep learning from private training data," in *ICLR*, 2017.
- [38] H. B. McMahan, E. Moore, D. Ramage, S. Hampson and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," 2017.
- [39] J. Kiefer and J. Wolfowitz, "Stochastic estimation of the maximum of a regression function," *The Annals of Mathematical Statistics*, pp. 462-466, 1952.
- [40] I. Kholod, E. Yanaki, D. Fomichev, E. Shalugin, E. Novikova, E. Filippov and M. Nordlund, "Open-Source Federated Learning Frameworks for IoT: A Comparative Review and Analysis," *Sensors*, vol. 21, no. 167, 2021.
- [41] S. P. Karimireddy, M. Jaggi, S. Kale, M. Mohri, S. J. Reddi, S. U. Stich and A. T. Suresh, "Mime: Mimicking centralized stochastic algorithms in federated learning," *arXiv preprint arXiv:2008.03606*, 2020.
- [42] L. Ziwei, L. Ping, W. Xiaogang and T. Xiaoou, "Deep Learning Face Attributes in the Wild," 2015.
- [43] C. Gregory, A. Saeed, T. Jonathan and v. S. Andre, "EMNIST: an extension of MNIST to handwritten letters," *arXiv preprint arXiv:1702.05373*, 2017.
- [44] A. Krizhevsky, V. Nair and G. Hinton, "CIFAR-10 (Canadian Institute for Advanced Research)," 2009.
- [45] G. Jocher, "YOLOv5," <https://github.com/ultralytics/yolov5>, 2020.

- [46] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*, Springer, 2014, pp. 740-755.
- [47] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779-788.
- [48] M. Everingham, L. Van Gool, C. K. Williams, J. Winn and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303-338, 2010.
- [49] Maldivien, "Coco-to-yolo-downloader," <https://github.com/maldivien/Coco-to-yolo-downloader>, 2021.
- [50] W. Li, F. Milletari, D. Xu, N. Rieke, J. Hancox, W. Zhu, M. Baust, Y. Cheng, S. Ourselin, M. J. Cardoso and A. Feng, "Privacy-preserving federated brain tumour segmentation," *Springer*, pp. 133-141, 2019.
- [51] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," 2009.
- [52] "NSL-KDD dataset," [Online]. Available: <https://www.unb.ca/cic/datasets/nsl.html>.
- [53] Argo CD, "Argo CD - Declarative GitOps CD for Kubernetes," [Online]. Available: <https://argo-cd.readthedocs.io/en/stable/>. [Accessed 2022].
- [54] Kustomize.io, "Kubernetes native configuration management," [Online]. Available: <https://kustomize.io>. [Accessed 2022].