H2020 – ICT-13-2018-2019

Machine Learning to Augment Shared Knowledge in Federated Privacy-Preserving Scenarios (MUSKETEER)

Grant No 824988

D7.3 First prototype of the MUSKETEER client connectors

September 2020

Imprint

Contractual Date of Delivery to the EC:	**30 September 2020**
Author(s):	**Davide Profeta (ENG), Susanna Bonura (ENG)**
Participant(s):	**ENG, IBM, IDSA**
Reviewer(s):	**Giacomo Fecondo (FCA-ITEM), Chiara Napione (COMAU)**
Project:	**Machine learning to augment shared knowledge in federated privacy-preserving scenarios (MUSKETEER)**
Work package:	**WP7**
Dissemination level:	**Public**
Version:	**1.0**
Contact:	**Susanna Bonura – susanna.bonura@eng.it**
Website:	**www.MUSKETEER.eu**

Legal disclaimer

The project Machine Learning to Augment Shared Knowledge in Federated Privacy-Preserving Scenarios (MUSKETEER) has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 824988. The sole responsibility for the content of this publication lies with the authors.

Copyright

Executive Summary

The MUSKETEER Client Connector is the component required by a participant to join the MUSKETEER Platform. That software application supports the MUSKETEER platform participants in exchanging the machine learning models and at the same time it prevents the sharing of private data in line with the data sovereignty principles.

This document provides a report that describes the architecture and the configuration of the first prototype of the MUSKETEER Client Connector. In addition to that, a guide to install the client connector and the main functionalities are described. Finally, a detailed description, presents the end-to-end test execution of a use case provided by COMAU that makes use of private datasets to implement one of the possible scenarios, of federated machine learning, enabled by the MUSKETEER Platform thanks to the Client Connector.

The source code of the first prototype version of the MUSKETEER Client Connector is available at the following URLs released as open source under GNU AGPLv3 license:

- https://github.com/Engineering-Research-and-Development/musketeer-client-connector-backend, for the back-end component and
- https://github.com/Engineering-Research-and-Development/musketeer-client-connector-frontend, for the front-end component dedicated to the MUSKETEER project.

Document History

Version	Date	Status	Author	Comment
0.1	03 August 2020	Table of Contents	Susanna Bonura	First Draft
0.2	07 August 2020	First round of contribution to sections	Davide Profeta	Update
0.3	24 August 2020	Second round of contribution to sections	Davide Profeta	Update
0.4	28 August 2020	Third round of contribution to sections	Davide Profeta	Update
0.5	02 September 2020	For internal review	Susanna Bonura	Draft for review
0.6	04 September 2020	Review inputs	Chiara Napione	Update
0.7	07 September 2020	Review inputs	Giacomo Fecondo	Update
0.8	09 September 2020	Updated version addressing comments received during the internal review process	Davide Profeta	Update
0.9	10 September 2020	Finalization	Susanna Bonura	Update
1.0		Clean and submission	Gal Weiss	Final

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

Abbreviation	Definition
API	Application Programming Interface
CA	Consortium Agreement
CC	Client Connector
DP	Differential Privacy
DC	Data Connector
DV	Data Value
FS	Feature Selection
FSM	Finite State Machine
GA	Grant Agreement
IDR	Intermediate Data Representation
IDS	Industrial Data Space
LC	Logistic Classifier
LGFS	Linear Greedy Feature Selection
MK	Master Key
ML	Machine Learning
MLP	Multi-Layer Perceptron
MN	Master Node
OS	Operating System
PERT	Program evaluation and review technique
PK	Public Key
POM	Privacy Operation Mode
PP	Privacy Preserving
PPML	Privacy Preserving Machine Learning
RAM	Reference Architecture Model
ROC	Receiver Operating Characteristics
SQL	Structured Query Language
TA	Task Alignment
UI	User Interface
WN	Worker Node

1 Introduction

1.1 Purpose

The purpose of this document is to provide a report which describes the main user interactions with the first prototype of the MUSKETEER Client Connector (D7.3. - First prototype of the MUSKETEER client connectors).

The client connector is the component required by a participant to join the MUSKETEER Platform. It is the software application supporting MUSKETEER platform participants in the federated ML model exchange, share and process, so to guarantee the data sovereignty principles.

The client-side connectors have to support the set of privacy operation modes made available throughout the project according to the architecture defined in T3.1 and meet the requirements of the federated and privacy-preserving machine learning services designed in WP4 (for more details we refer to D4.1).

Moreover, the client component provides services for locally combining model updates into one consistent, up-to-date model instance. The client component serves as adapter for the integration and industrial validation of the MUSKETEER platform in WP7.

The first version of the MUSKETEER Client Connector prototype together with this report are the first results of the task *T7.2 - Development of client connectors for industrial scenarios*, which aims to assemble and provide the privacy and security machine learning services developed in WP4 and WP5 and providing the main functionalities to communicate with MUSKETEER Federated Machine Learning platform server designed and developed in WP3.

This report provides instructions to install, configure and use the MUSKETEER Client Connector so to interact with the MUSKETEER Cloud Platform.

1.2 Related Documents

As already mentioned, the deliverable D7.3. - First prototype of the MUSKETEER Client Connector, is the first of the task *T7.2 - Development of client connectors for industrial scenarios*.

For the development of the MUSKETEER Client Connector presented in this document, several deliverables were considered as input (see Figure 1) both directly and indirectly linked to the WP7.

The input deliverables are:

D2.1 - Industrial and technical requirements.

D3.2 - Architecture design – Final version.

D3.3 - First prototype of the MUSKETEER platform.

D4.4 - Pre-processing, normalization, data alignment and data value estimation algorithms

D4.4 - Machine Learning Algorithms over Federated Operation Modes algorithms.

D7.1 - Client connectors' architecture design – Initial version.

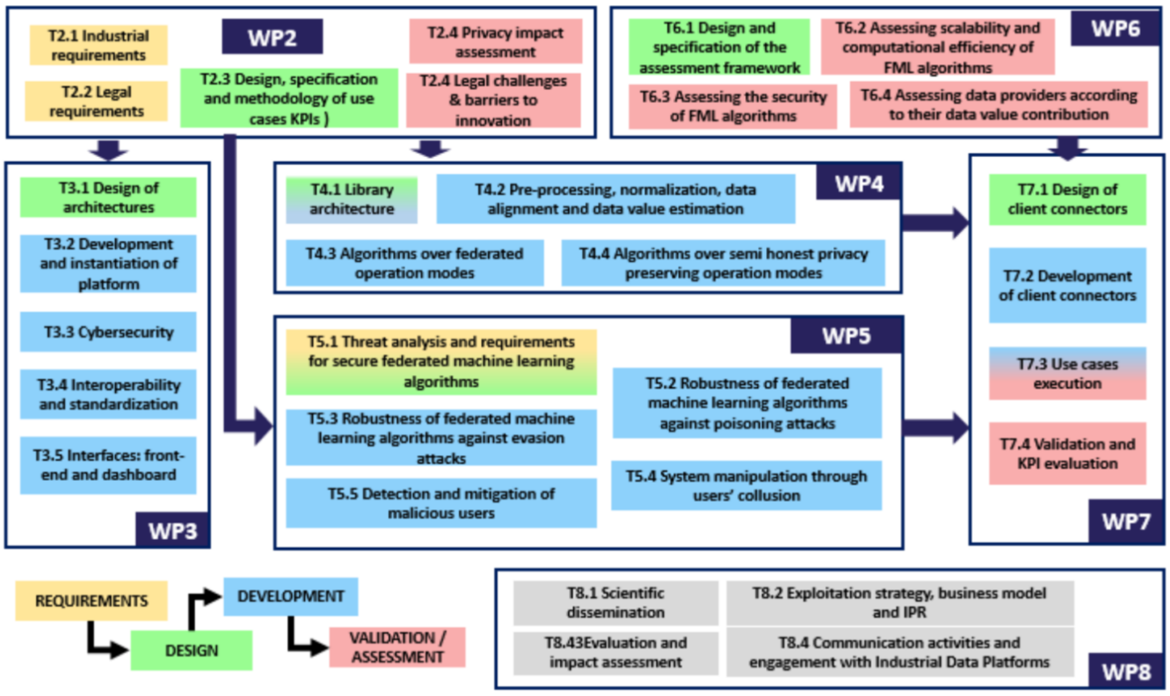

WP2
T2.1 Industrial requirements
T2.2 Legal requirements
T2.3 Design, specification and methodology of use cases KPIs)
T2.4 Privacy impact assessment
T2.4 Legal challenges & barriers to innovation
WP6
T6.1 Design and specification of the assessment framework
T6.2 Assessing scalability and computational efficiency of FML algorithms
T6.3 Assessing the security of FML algorithms
T6.4 Assessing data providers according to their data value contribution
WP3
T3.1 Design of architectures
T3.2 Development and instantiation of platform
T3.3 Cybersecurity
T3.4 Interoperability and standardization
T3.5 Interfaces: front-end and dashboard
WP4
T4.1 Library architecture
T4.2 Pre-processing, normalization, data alignment and data value estimation
T4.3 Algorithms over federated operation modes
T4.4 Algorithms over semi honest privacy preserving operation modes
WP5
T5.1 Threat analysis and requirements for secure federated machine learning algorithms
T5.2 Robustness of federated machine learning algorithms against poisoning attacks
T5.3 Robustness of federated machine learning algorithms against evasion attacks
T5.4 System manipulation through users' collusion
T5.5 Detection and mitigation of malicious users
WP7
T7.1 Design of client connectors
T7.2 Development of client connectors
T7.3 Use cases execution
T7.4 Validation and KPI evaluation
REQUIREMENTS
DESIGN
DEVELOPMENT
VALIDATION / ASSESSMENT
WP8
T8.1 Scientific dissemination
T8.2 Exploitation strategy, business model and IPR
T8.43Evaluation and impact assessment
T8.4 Communication activities and engagement with Industrial Data Platforms

Figure 1 - MUSKETEER's PERT diagram

1.3 Document Structure

In Section 2, the general description of the MUSKTEER Client Connector components is presented.

In Section 3, the instructions to install and use the MUSKETEER Client Connector are presented.

In Section 4, a summary of the MUSKETEER platform architecture is presented, so to have a comprehensive picture before detailing the Client Connector.

Section 5 shows the results of the interactions by means of an end-to-end execution in the context of a smart manufacturing use case provided by COMAU. The problem that COMAU aimed at solving thanks to the MUSKETEER Platform, enabling secure data sharing and federated machine learning approach, consists in a predictive maintenance process applied to a belt and related motor.

Finally, Section 6 concludes the deliverable. It outlines the main findings of the work done which will guide the future efforts in the design and development of the final version of the MUSKETEER Client Connector.

2 MUSKETEER Client Connector components

When defining the scope of the MUSKETEER platform, it is important to keep in mind the distinction between (i) the server-side platform, which enables the creation and execution of data sharing and federated machine learning among geographically distributed participants and (ii) the client connectors, in charge of starting and/or participating to ML training processes.

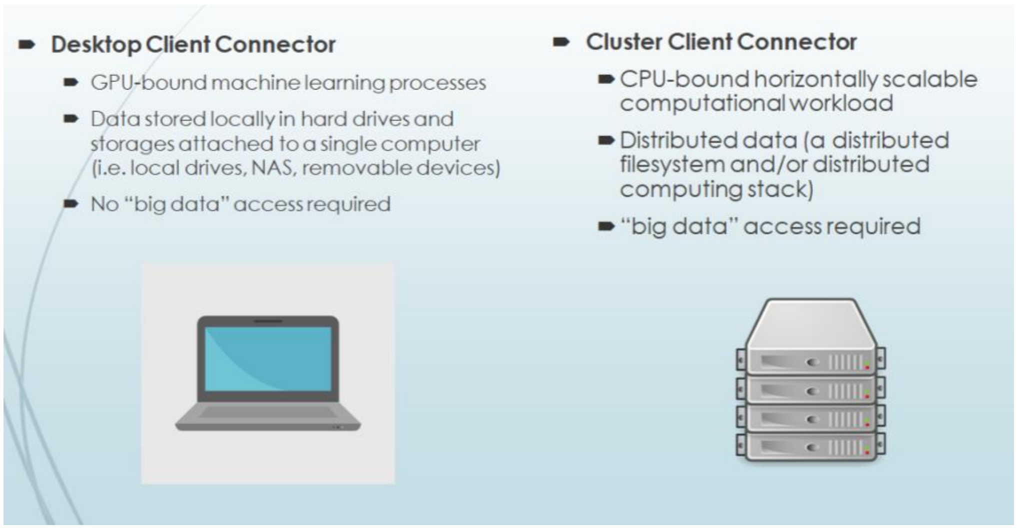
On the server component, detailed information are available in the deliverable D3.2 - Architecture design – Final version. In short, the server is the cloud platform which uses message queues for asynchronous exchange of information required for federated learning, such as the latest version of the central model computed by the aggregator, or model updates computed by the participants on their local data. The platform itself is agnostic to the semantics of this information (generally it will not even be aware whether or not the information is encrypted); it is parsed and interpreted in the context of the federated learning algorithm processes running on the aggregator and participants' sides, respectively.

Besides the exchange of information for the execution of the actual federated learning tasks, the server side also provides services to manage tasks throughout their lifecycle, such as: creating new tasks, browsing created tasks, aggregating tasks, joining tasks as a participant or deleting tasks. The meta-information that is required for task management is stored in a cloud database.

Concerning the Client Connector, the first official version of the architecture is documented in D7.1. Since then, improvements and updates have been made to the design of the MUSKETEER Client Connector to meet the needs of the end-user partners. The final version of the Client Connector architecture design (D7.1, due at M24) will consider two types of client connectors to meet two different sets of user requirements and needs. Two types of Client Connector architecture have been designed: a Cluster mode, the first version of which has been described in D7.1, and a Desktop mode, whose first prototype is released along with this documentation as deliverable D7.3. The following figure shows the main differences between the desktop and cluster modes.

The Cluster Client Connector supports the storage and the processing of Big Data, through horizontal scalability and workload distribution on multiple nodes of the cluster (more details may be found in D7.1).

The Desktop Client Connector can be used when data is collected in a non-centralized way and there is no need to use a cluster to distribute the workload, both in terms of computing and big data storage. Anyway, the Desktop version could also leverage GPUs for the training process, enabling the processing of a large amount of data in terms of volume. Finally, the Desktop Client Connector can be easily deployed in any environment thanks to the use of Docker (https://www.docker.com/) in order to containerize the Client Connector application. Docker containers ensure us a lightweight, standalone and executable package of the software that includes everything needed to run the Desktop Client Connector: operating system, code, runtime, system tools, libraries and settings. In this way the whole Desktop Client Connector application can be easily deployed in a sandbox to run on the host operating system of the user.



- Desktop Client Connector
  - GPU-bound machine learning processes
  - Data stored locally in hard drives and storages attached to a single computer (i.e. local drives, NAS, removable devices)
  - No "big data" access required
- Cluster Client Connector
  - CPU-bound horizontally scalable computational workload
  - Distributed data (a distributed filesystem and/or distributed computing stack)
  - "big data" access required

Figure 2 – Client Connector Modes

The Desktop Client Connector architecture is shown in Figure 3. The application is mainly composed by 5 components that will be described in detail. There are also two external components that are loaded inside the Client Connector after the application is *up and running*: the communication messenger and the federated machine learning library. This solution produces a modular application with respect to those components, reusable in any context and independent from the central server and federated machine learning library used.

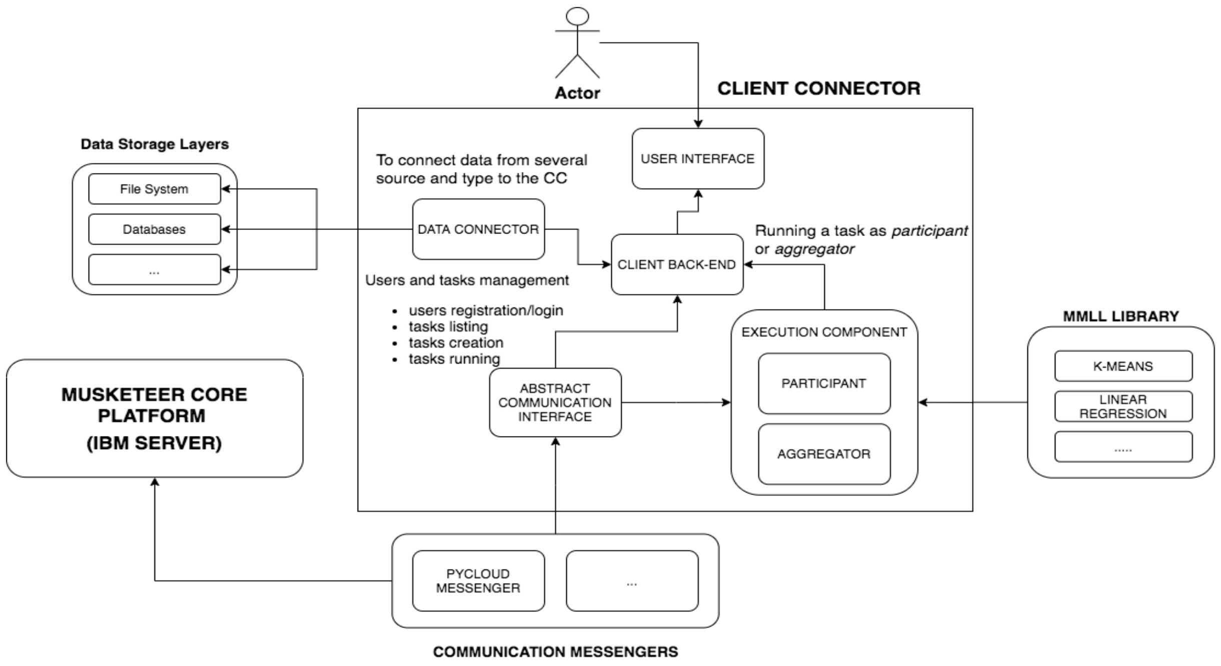


Figure 3 - Desktop Client Connector Architecture

At higher level, the *Actor*, through a *User Interface* component, that is a local web application, performs a set of functionalities that are described on Section 5. These functionalities range from the access to the target server platform, with which the *Communication Messenger* library communicates, to the binding of local data to the Client Connector, up to obtaining the results produced by the completed tasks. The *User Interface* is developed as a web application using Angular CLI version 8.3.8. This component represents the frontend part of the Client Connector, in accordance with the specifications described in D3.3.

The core *Client Back-End* component acts as a RESTful Web Service that handles all user requests, ranging from local operations (e.g. to connect user data to the Client Connector) to server operations (e.g. tasks and users management); these operations need to use a *Communication Messenger* library to communicate toward a target external server.

The *Data Connector* component connects user data, which may come from different sources or storage layers, to the Client Connector. In addition to connecting data from different sources, the component can manage and support different kinds of data: in fact, a user can load a CSV tabular data from the File System, images files, binary data, a table from a database and so on.

The *Abstract Communication Interface* component allows to import and use an implementation of the communication library. In the MUSKETEER project the *Communication Messenger* library used is the *pycloudmessenger* library developed by IBM, and it is available at the following URL: https://github.com/IBM/pycloudmessenger. After the *pycloudmessenger* library is configured and installed (see Section 4), the Client Connector can use the APIs to communicate toward the MUSKETEER core platform. As a result, this component integrates all the user and task management parts: the listing task functionality, login and registration step, task creation and so on. This component is also connected and used by the *Execution* component, since during the training process the weights are sent and received to and from the central server (Musketeer Core Platform) using the *Communication Messenger* (*pycloudmessenger*) API.

On the other hand, the execution of tasks as a participant or aggregator is handled by the *Execution* macro-component. This component instantiates and runs a federated machine learning algorithm according to an interface that has been defined in WP4 by UC3M and TREE; which algorithm to be used and with which parameters are defined in the task definition and stored in the central server during the task execution. As well as the *Communication Messenger* library, the *Federated Machine Learning* library is an external library imported into the Client Connector. The imports of these libraries can be fully performed through *User Interface* in an initial configuration step after the first start of the Desktop Client Connector application.

3 Installation guide and user interactions

In the MUSKETEER project, federated ML is extended to support different Privacy Operation Modes (POMs), which control the amount and type of information that the data owners share during the model training and validation process. In POMs 1-3 (which closely follow conventional federated ML protocols), the model training is coordinated by a user initiator, called aggregator who creates and publishes a task, while the data owners act as participants by joining the task. Model training is typically performed iteratively throughout a number of rounds which is either determined a priori, or dynamically, e.g. by considering a model convergence criterion. In each round, the aggregator dispatches the current central version of the model to all the participants. Then the participants compute updates to that model based on their local data and send the updates back to the aggregator. Model updates can either be in the form of gradients, or in the form of new versions of the model. Upon having received the updates from all participants, the aggregator incorporates them (e.g. by taking an average of all the updates) into the new version of the central model. After the training rounds have completed, the aggregator holds the final version of the model, which can then be centrally stored for later use and/or deployed by the participants in their local production environments. These following sections describe the steps to install and run the Desktop Client Connector according the abovementioned approach.

3.1 Installation

As a requirement, it is necessary to have a Docker engine installed on the host machine to run the Desktop Client Connector application.

The source code of the first prototype version of the Desktop Client Connector is available at the following URLs released as open source under GNU AGPLv3 license:

- https://github.com/Engineering-Research-and-Development/musketeer-client-connector-backend, for the back-end component and
- https://github.com/Engineering-Research-and-Development/musketeer-client-connector-frontend, for the front-end component dedicated to the MUSKETEER project.

As a first step create the Docker image of the backend components. From the project root folder, run the following command through the terminal:

- *docker build -f Dockerfile -t MYBUILDIMAGE*

The same *MYBUILDIMAGE* name chosen must be inserted in the *docker-compose.yml* file.

Before running the *docker-compose.yml* the user must also configure the Docker volumes for the backend component. In particular, it is necessary to specify:

- FS_PATH_DATA: a filesystem path directory where there are the datasets that you want to bind to the Client Connector.
- FS_PATH_LOGS: a filesystem path directory where to store all the logs file generated by the task you run.
- FS_PATH_RESULTS: a filesystem path directory where to store all the results file generated by the task you run and complete.

The *docker-compose.yml* contains both the backend image just created and the frontend component. The frontend Docker image is located on a repository accessible through authentication to our Docker registry. To log in, run the following command:

- *docker login gitlab.alidalab.it:5000/musketeer/ngx-musketeer-client*, followed by USER and PASSWORD that have been provided.

Finally, run the following command to run and up the Desktop Client Connector:

- *docker-compose pull && docker-compose up*

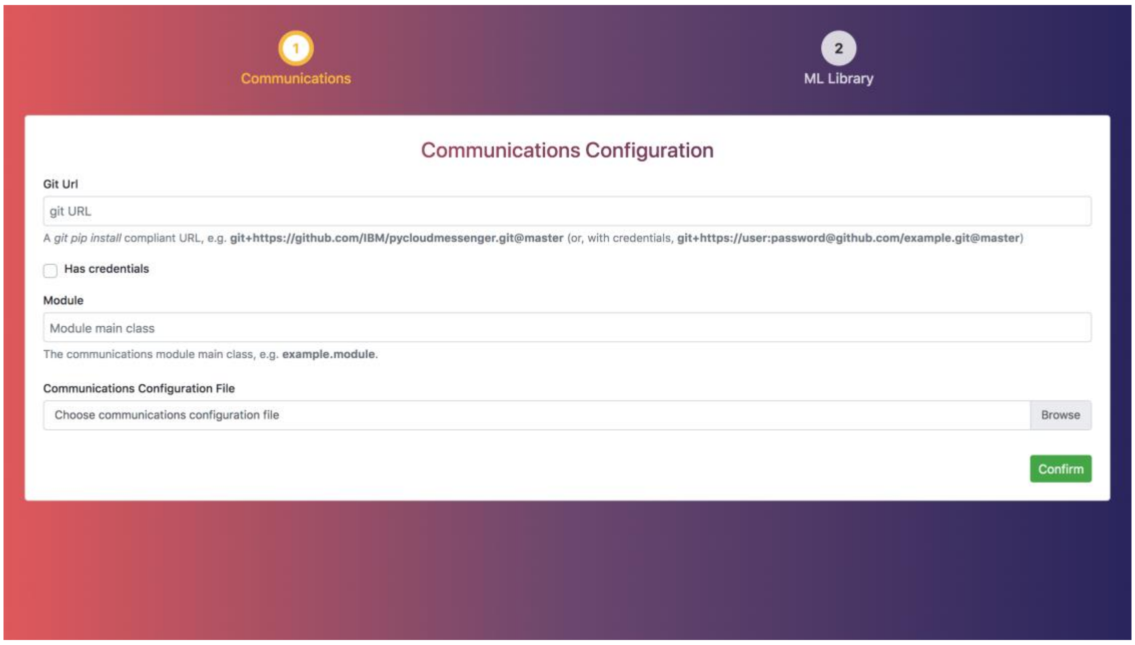
The frontend Docker image will be automatically pulled from the register, if it is not present. It may take some minutes to download all the required dependencies based on your internet connection. Once it is done, the local server will be running at '127.0.0.1:5000', whilst you can use the User Interface by opening a browser and writing the following URL: '127.0.0.1:4500' (or 'localhost:4500').

3.2 Configuration

This section describes the configuration steps once the Desktop Client Connector has been started for the first time. These steps consist in the installation and configuration of the two external components presented in the Client Connector architecture (see Section 2): the Communication Messenger and Federated Machine Learning Python-based library.

Once you open the page on localhost:4500 from your browser for the first time, you will be redirected to localhost:4500/configure, where it is possible to configure and install the first Communication Messenger component as shown in Figure 4 below. As shown in the figure, the required information is the following:

- Git Url: a Git URL where the communication library is hosted.
- Module: the communication module main class, in the form *package.module*.
- Communications Configuration File: a Json file containing all the information needed by the communication messenger library to connect towards the core server.



Git Url

git URL

A *git pip install* compliant URL, e.g. **git+https://github.com/IBM/pycloudmessenger.git@master** (or, with credentials, **git+https://user:password@github.com/example.git@master**)

☐ Has credentials

Module

Module main class

The communications module main class, e.g. **example.module.**

Communications Configuration File

Choose communications configuration file

Figure 4 - Communication Configuration step

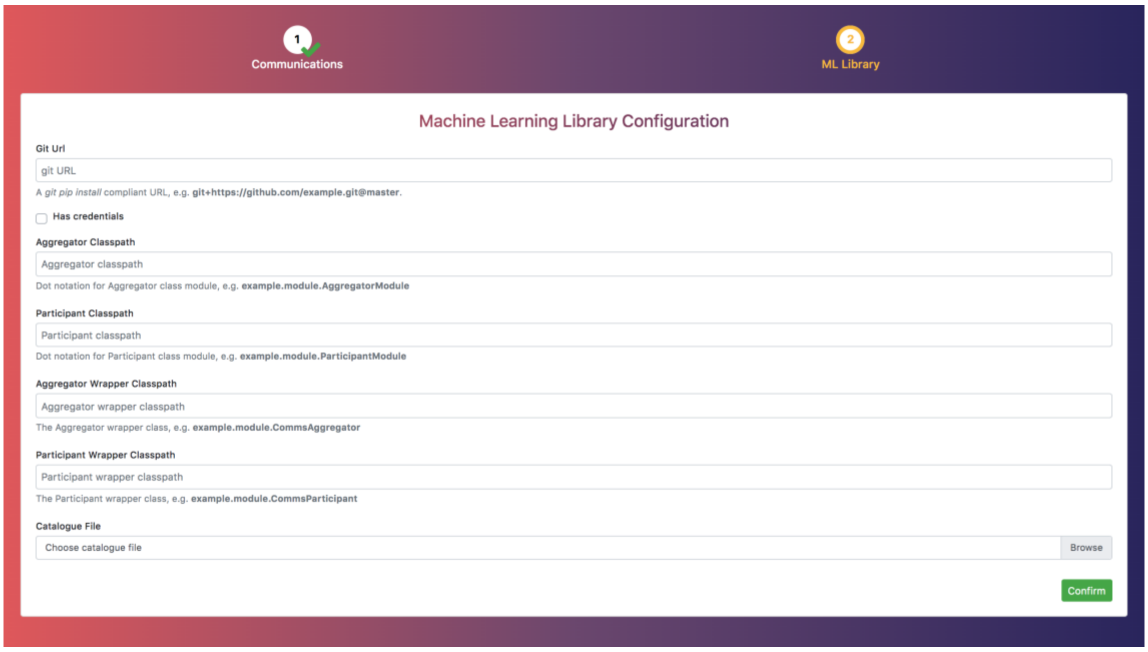
In the MUSKETEER project, the communication messenger used is the *pycloudmessenger* library developed by IBM and available at the following GIT repository: https://github.com/IBM/pycloudmessenger. For this instance, the settings used are the following:

- Git Url: *git+https://github.com/IBM/pycloudmessenger.git@master*
- Module: *pycloudmessenger.ffl.fflapi*
- Communications Configuration File: the Json file provided by IBM.

Once you have entered this information you can confirm clicking on the related button and install the library. If the installation is successful you will proceed to the next step. If something has gone wrong you will be notified with an error message.

The next step, as shown in Figure 4 below, concerns the configuration and installation of the machine learning library. For the Machine Learning library configuration, the required information is:

- Git Url: a Git URL where the machine learning library is hosted.
- Aggregator Classpath: the aggregator class module where are present the main classes to instantiate the objects of the machine learning algorithms related to the role of aggregator.
- Participant Classpath: the participant class module where are present the main classes to instantiate the objects of the machine learning algorithms related to the role of participant.



Machine Learning Library Configuration

Git Url
git URL
A *git pip install* compliant URL, e.g. **git+https://github.com/example.git@master**.

☐ Has credentials

Aggregator Classpath
Aggregator classpath
Dot notation for Aggregator class module, e.g. **example.module.AggregatorModule**

Participant Classpath
Participant classpath
Dot notation for Participant class module, e.g. **example.module.ParticipantModule**

Aggregator Wrapper Classpath
Aggregator wrapper classpath
The Aggregator wrapper class, e.g. **example.module.CommsAggregator**

Participant Wrapper Classpath
Participant wrapper classpath
The Participant wrapper class, e.g. **example.module.CommsParticipant**

Catalogue File
Choose catalogue file

Figure 5 - Machine Learning Library Configuration step

- Aggregator Wrapper Classpath: the aggregator wrapper class module used to wrap the communication messenger library related to the role of aggregator.

- Participant Wrapper Classpath: the participant wrapper class module used to wrap the communication messenger library related to the role of participant.
- Catalogue File: it is a Json file containing the meta-model of the algorithms that are available in the machine learning library imported. In Figure 5 is shown a meta-model example of a single algorithm, related to an Artificial Neural Network algorithm.

```
{
   "id":1,
   "POM":1,
   "type":"classification",
   "name":"NN",
   "label":"ANN (Artificial Neural Network)",
   "description":"Generic machine learning algorithm based on neural networks.",
   "properties":[
      {
         "name":"Nmaxiter",
         "label":"Max number of iterations",
         "defaultValue":100,
         "type":"number",
         "description":"Number of epochs."
      },
      {
         "name":"learning_rate",
         "label":"Learning rate",
         "defaultValue":0.001,
         "type":"number",
         "description":"Learning rate value."
      },
      {
         "name":"model_architecture",
         "label":"Model architecture",
         "defaultValue":null,
         "type":"json",
         "description":"Neural network parameters."
      }
   ]
}
```

This catalogue file defines the available algorithms (specifying the POMs because not all the algorithms can be implemented for all the POMs) collecting the meta-models and all the required information. This is useful in the creation task step of the User Interface, where you choose the algorithm. In fact, it allows you to select among the algorithms defined in this file.

As shown in the Json example of the meta-model, a set of algorithm information is described including: the type of algorithm (between clustering, regression, classification), for which POM it is appointed and a description of the algorithm parameters that can then be valorised by the user during the task creation.

As for the previous step, once you have filled all the information, confirm for the machine learning library installation. Properly installed also this component, you will be redirected to the login/registration page.

3.3 User registration and login

Once you have configured the Desktop Client Connector you will be redirected to the login page, as shown in Figure 6 below.

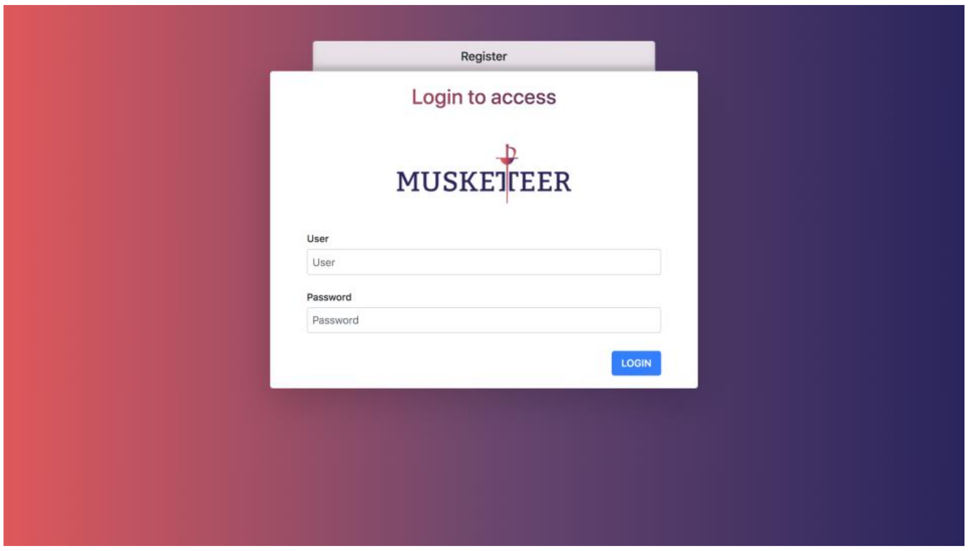


Figure 6 - User login page

If the user is already registered to the target platform, the MUSKETEER platform, it is possible to access with their own credentials. Otherwise, the user can click on the window behind the login window to register a new user. To register a new user, it is necessary to insert the following information, as shown in the Figure 7 below:

- Username.
- Organization name.
- Password.
- Confirm of the password.

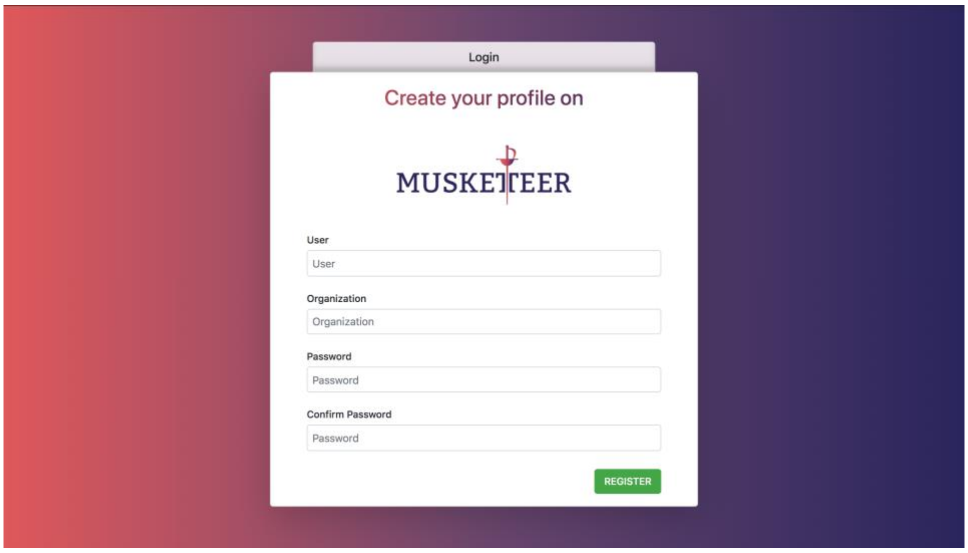
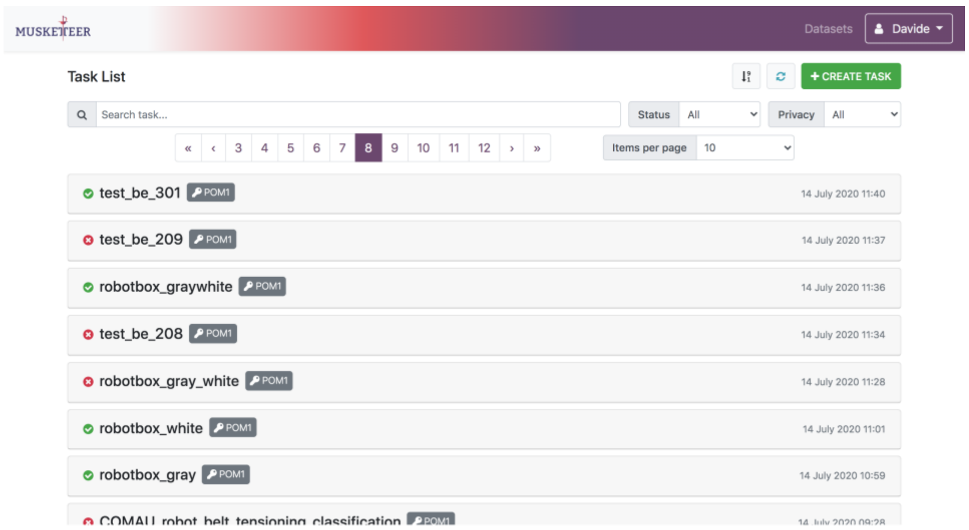


Figure 7 - User registration page

Enter the login credentials and you authenticate to the target MUSKETEER server accessing to the main page. Under the hood is used the Communication Messenger library and the configuration parameters that have been described in Section 4.

3.4 Tasks listing and browsing

From the main page (located on http://localhost:4500/tasks) it is possible to visualize and browse the tasks that are stored through the Musketeer platform. The tasks are listed as showed in the following Figure 8.



✓	test_be_301 POM1	14 July 2020 11:40
✗	test_be_209 POM1	14 July 2020 11:37
✓	robotbox_graywhite POM1	14 July 2020 11:36
✗	test_be_208 POM1	14 July 2020 11:34
✗	robotbox_gray_white POM1	14 July 2020 11:28
✓	robotbox_white POM1	14 July 2020 11:01
✓	robotbox_gray POM1	14 July 2020 10:59
✗	COMAU_robot_belt_tensioning_classification POM1	14 July 2020 09:28

Figure 8 - Main page

Tasks can be filtered in several ways:

- by name.
- by status: created, started, completed, failed.
- by privacy level (POM).

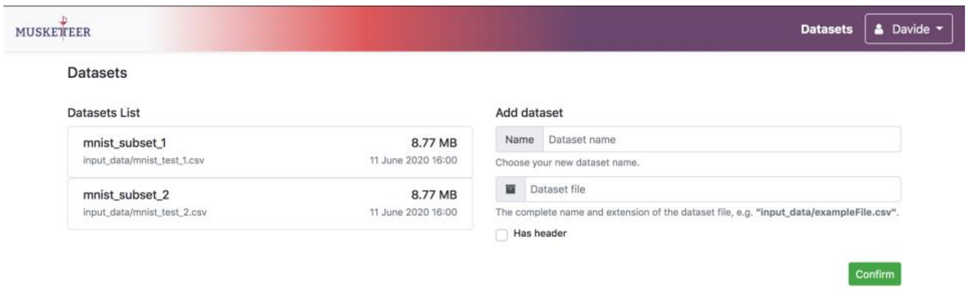
In addition, a client-side task pagination has been added in order to more easily browsing the tasks and lighten the whole page.

3.5 Data connection

From the main page, clicking on "Datasets" in the top right bar you will be redirected to the data connection page (http://localhost:4500/worker/datasets), where the user can connect new datasets to the Client Connector, and check the ones already inserted. The first prototype of the Desktop Client Connector supports the binding of datasets in CSV format from the user's File System to the Client Connector, being able to specify whether or not the dataset has a header.

The following Figure 9 shows the data connection page. On the left side of the page the already inserted datasets are shown with the following information: name of the dataset, path, size of the dataset and date of insertion.

On the other side a form for the binding of a new dataset that have to contain: name (as label) of the dataset, relative path in the form "input_data/FILENAME.csv" that is inside the dataset folder set by the user during the Client Connector installation (see Section 3), and a checkbox to indicate whether or not the dataset contains a header.



Datasets

Datasets List

mnist_subset_1 input_data/mnist_test_1.csv	8.77 MB 11 June 2020 16:00
mnist_subset_2 input_data/mnist_test_2.csv	8.77 MB 11 June 2020 16:00

Add dataset

Name: Dataset name

Choose your new dataset name.

Dataset file

The complete name and extension of the dataset file, e.g. "**input_data/exampleFile.csv**".

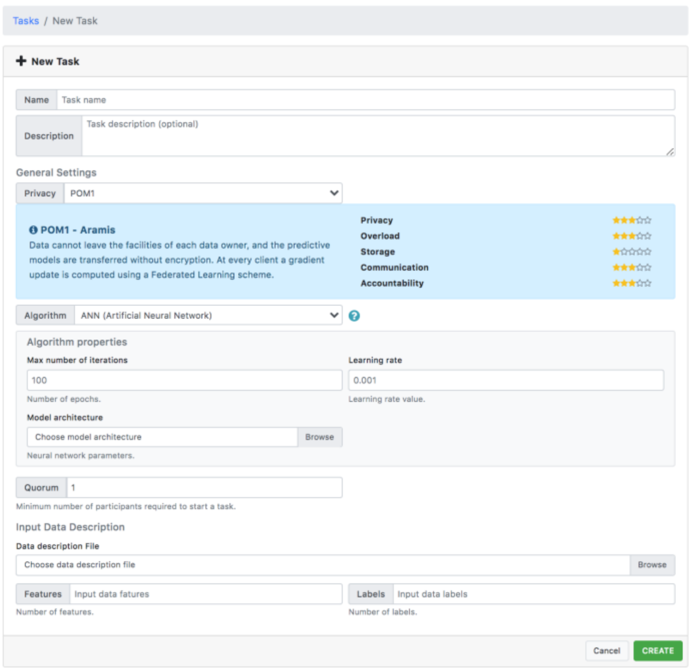
☐ Has header

Figure 9 - Data connection page

By clicking on the "Confirm" button the new dataset will be added and shown in the list of datasets; if something went wrong an error message will be displayed.

3.6 Tasks creation

From the main page you can access the tasks creation page (located on "http://localhost:4500/tasks/create") by clicking on the "Create task" button. Figure 10 below shows the details of all the information that can be set while creating a new task by the user.



Name: Task name

Description: Task description (optional)

General Settings

Privacy: POM1

POM1 - Aramis
Data cannot leave the facilities of each data owner, and the predictive models are transferred without encryption. At every client a gradient update is computed using a Federated Learning scheme.

Privacy	★★★☆☆
Overload	★★★☆☆
Storage	★☆☆☆☆
Communication	★★★☆☆
Accountability	★★★☆☆

Algorithm: ANN (Artificial Neural Network)

Algorithm properties

Max number of iterations: 100
Number of epochs.

Learning rate: 0.001
Learning rate value.

Model architecture: Choose model architecture
Neural network parameters.

Quorum: 1
Minimum number of participants required to start a task.

Input Data Description

Data description File: Choose data description file

Features: Input data fatures
Number of features.

Labels: Input data labels
Number of labels.

Figure 10 - Task creation page detail

As shown in the previous figure, starting from the top, the user can set the following information:

- Name (required): a task name.
- Description (optional): a task description.
- Privacy (required): a select box to choose the level of privacy (POM) the user wants to apply. Each POM is described by a description and a set of characteristics in comparison with the other POMs.
- Algorithm (required): a select box to choose an algorithm, according to the POM selected, that the user wants to apply for its task. Once an algorithm has been selected, the properties of the algorithm that can be set by the user will be shown.
- Quorum (required): minimum number of participants required to start the task.

Regarding the description of the input dataset, for the first Desktop Client Connector release that supports CSV datasets for the training, the information needed are:

- Data description file (optional): a file containing all the information describing the input dataset required for the execution of the task and therefore for the training of the resulting machine learning model.
- Features (required): number of the input dataset features.
- Labels (required): number of the input dataset labels.

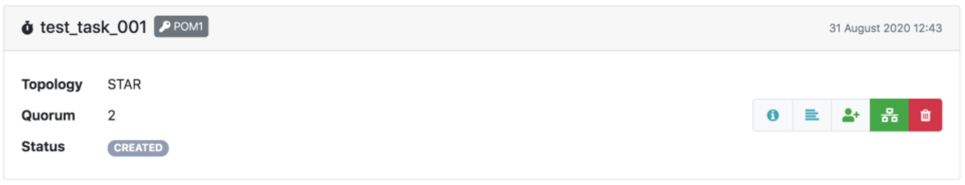
After filling in all the required information you can create the new task by clicking on the "Create" button. The new task just created will now be present in the list of tasks, where you can look again all the parameters that have been set during the task creation.

3.7 Tasks execution

A task can be executed from the main page by selecting one of the available tasks.

The task creator can execute the task itself either as a participant, having a dataset available for the training or as an aggregator. The aggregator collects the weights of the models received from each participant and aggregates them to obtain an aggregated machine learning model.

Figure 11 below shows the possible actions of an "open" task, i.e. not yet executed.

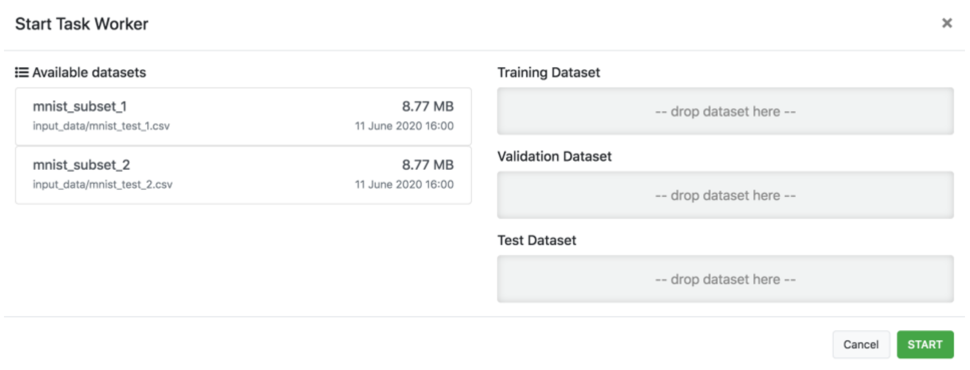

test_task_001 POM1
31 August 2020 12:43

Topology STAR
Quorum 2
Status CREATED

Figure 11 - An open task card

The task card displays basic information such as task name, privacy level (POM), and minimum number of participants required (quorum). On the right side of the card there are the following buttons (from the left to the right):

- Detail button: display on a modal all the task information.
- Logs button: display on a modal a stream log of the task execution.
- Join button: execute the task as participant. As shown in the next Figure 12, it opens a modal where the user can drag-and-drop their dataset and start the task. Only the training dataset is required to execute a task as a participant.


Start Task Worker

Available datasets

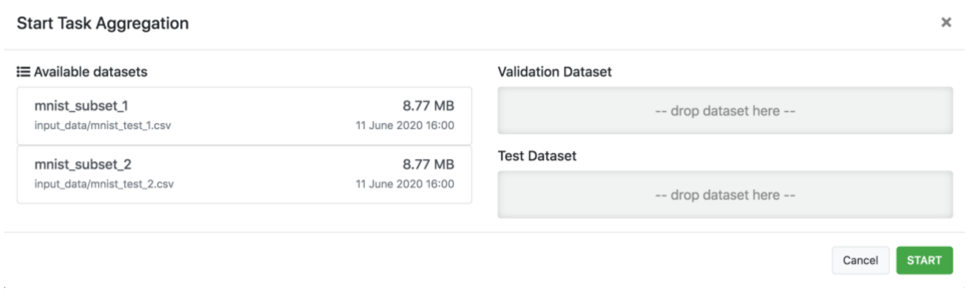
mnist_subset_1 8.77 MB
input_data/mnist_test_1.csv 11 June 2020 16:00

mnist_subset_2 8.77 MB
input_data/mnist_test_2.csv 11 June 2020 16:00

Training Dataset
-- drop dataset here --
Validation Dataset
-- drop dataset here --
Test Dataset
-- drop dataset here --

Figure 12 - Task worker modal

- Aggregate button: execute the task as aggregator. The following Figure 13 shows the task aggregator modal clicking on the aggregate button. For the aggregator only validation and test datasets are required. On the test data, at the end of the task execution, a resulting chart will be generated depending on the algorithm type. For clustering algorithms, for example, a 2D scatter-plot of the first two PCA components, coloured per cluster, is generated.
- Delete button: delete your own tasks.



Start Task Aggregation

Available datasets

mnist_subset_1 input_data/mnist_test_1.csv	8.77 MB 11 June 2020 16:00
mnist_subset_2 input_data/mnist_test_2.csv	8.77 MB 11 June 2020 16:00

Validation Dataset

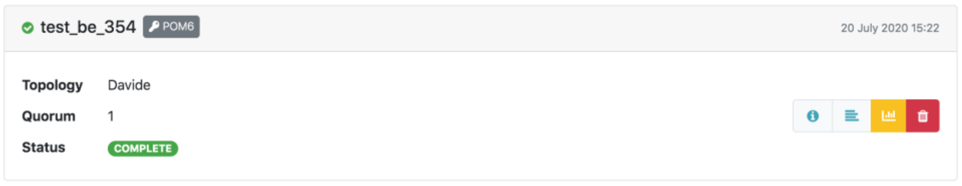
-- drop dataset here --

Test Dataset

-- drop dataset here --

Figure 13 - Task aggregator modal

Once the quorum of participants is reached, the task will change from "open" (or created) status to "running" status. Once completed, the task appears as shown in Figure 14.

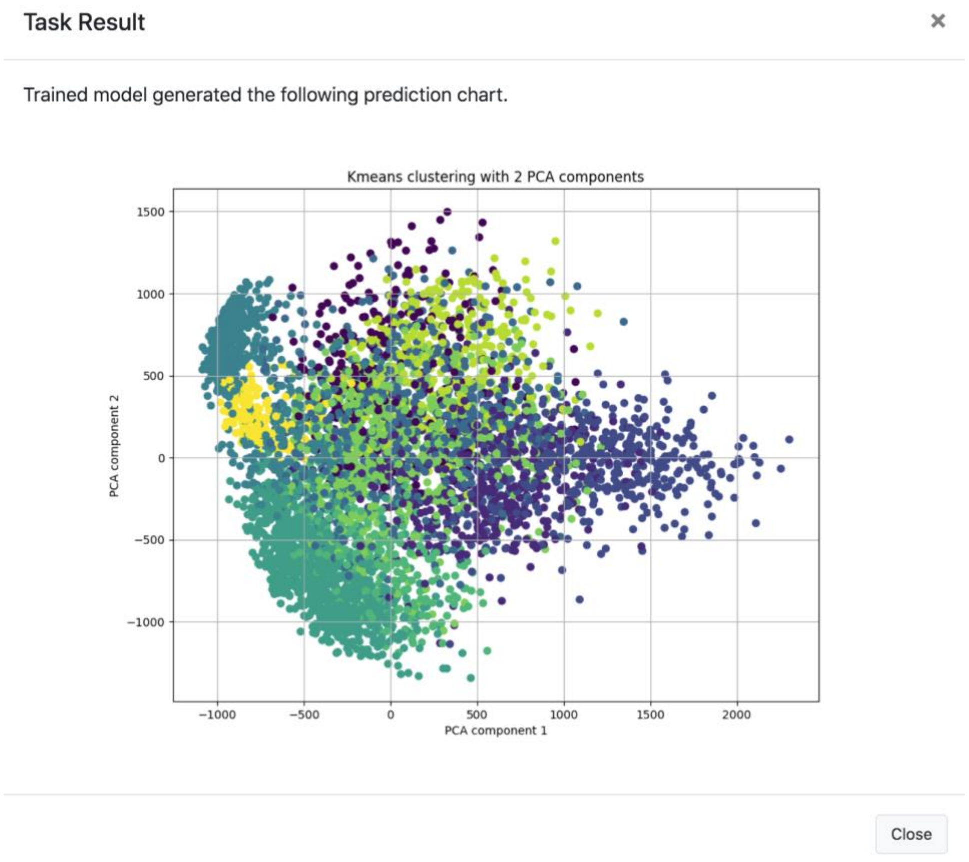


test_be_354 POM6 20 July 2020 15:22

Topology	Davide
Quorum	1
Status	COMPLETE

Figure 14 - A completed task card

A new yellow button appears for completed tasks, as shown in the previously figure, that opens a modal showing the resulting chart as shown in Figure 15.



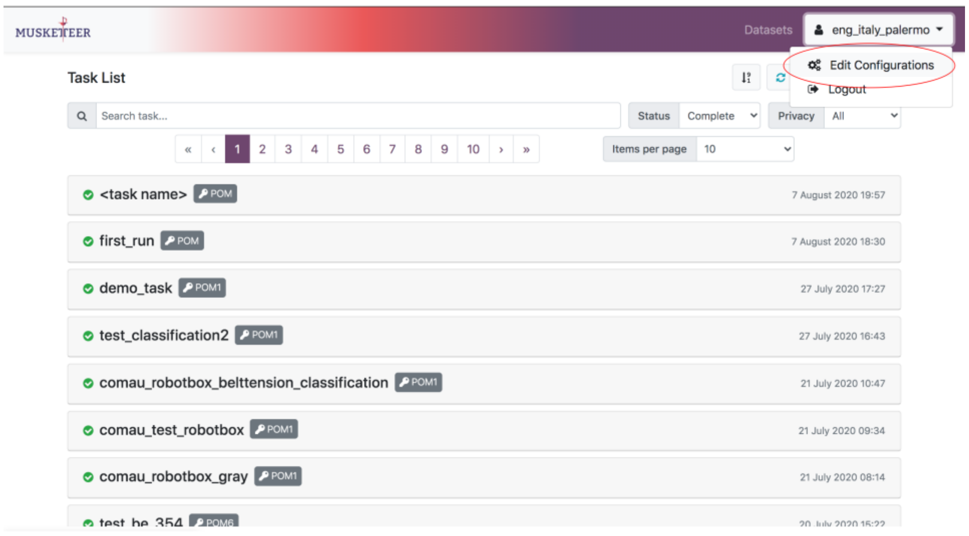
Task Result

Trained model generated the following prediction chart.

Figure 15 - K-Means Clustering example result

3.8 Client Connector configuration settings

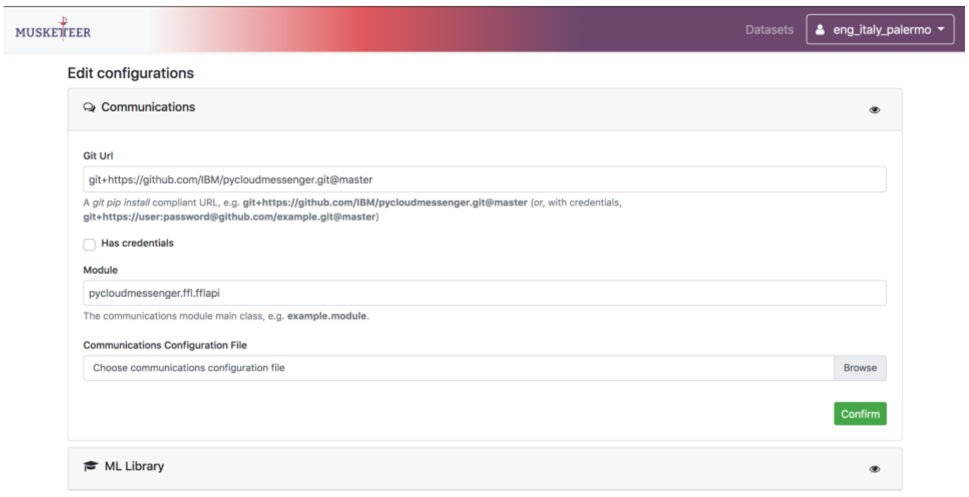
The Client Connector configurations, once set as explained in Section 4, can be updated by going to: "http://localhost:4500/settings/edit-configurations" or from the main page as shown in Figure 16 below.



<task name> POM 7 August 2020 19:57
first_run POM 7 August 2020 18:30
demo_task POM1 27 July 2020 17:27
test_classification2 POM1 27 July 2020 16:43
comau_robotbox_belttension_classification POM1 21 July 2020 10:47
comau_test_robotbox POM1 21 July 2020 09:34
comau_robotbox_gray POM1 21 July 2020 08:14
test_be_354 POM6 20 July 2020 15:22

Figure 16 - Main page to Edit Configurations page

The edit configurations page is presented as shown in Figure 17. From this page it is possible to modify the two libraries already imported, the one for communication to the target server and the one for federated machine learning, separately. Thus, you can always update the libraries already set or change them to include new ones.



Git Url
git+https://github.com/IBM/pycloudmessenger.git@master
A *git pip install* compliant URL, e.g. **git+https://github.com/IBM/pycloudmessenger.git@master** (or, with credentials, **git+https://user:password@github.com/example.git@master**)

☐ Has credentials

Module
pycloudmessenger.ffl.fflapi
The communications module main class, e.g. **example.module**.

Communications Configuration File
Choose communications configuration file

Figure 17 - Edit configurations page

4 End-to-end test execution

In this section we present an execution instance to verify the client connectors features implemented by means of a use case of data sharing and federated learning in the smart manufacturing domain.

4.1 Smart manufacturing use case: problem statement

COMAU is an automation provider and its robots are installed in dozens of plants, in automotive context but not only. This implies that COMAU customers are competitors to each other and so in the majority of the cases, they don't want to share their data.

The problem that COMAU can solve thanks to the MUSKETEER platform, enabling secure data sharing and federated machine learning approach, consists in the belt tensioning maintenance planning. The joints of the robot contain a belt that naturally loses its elasticity over time changing its tensioning. Actually, in order to prevent failures caused by a wrong tension, operators have to regularly check the belt status dismantling the entire robot axes. These operations require a lot of time, effort and eventually a production stop. Moreover, these are expensive and maybe useless if the belt status is still good. MUSKETEER platform enables the privacy preserving data sharing among COMAU and its customers and so the possibility to use that data to build a classification model based on federated machine learning to understand when the belts require maintenance.

To test the platform and in particular the client connector, two identical testbeds called RobotBox have been built. For this validation we have imagined that the data coming from the first RobotBox, from now on called gray RobotBox, were located in COMAU headquarter in Turin while the data coming from the second one, the white RobotBox, were in ENGINEERING facility in Palermo. This simulated two different customers plants (Figure 18).

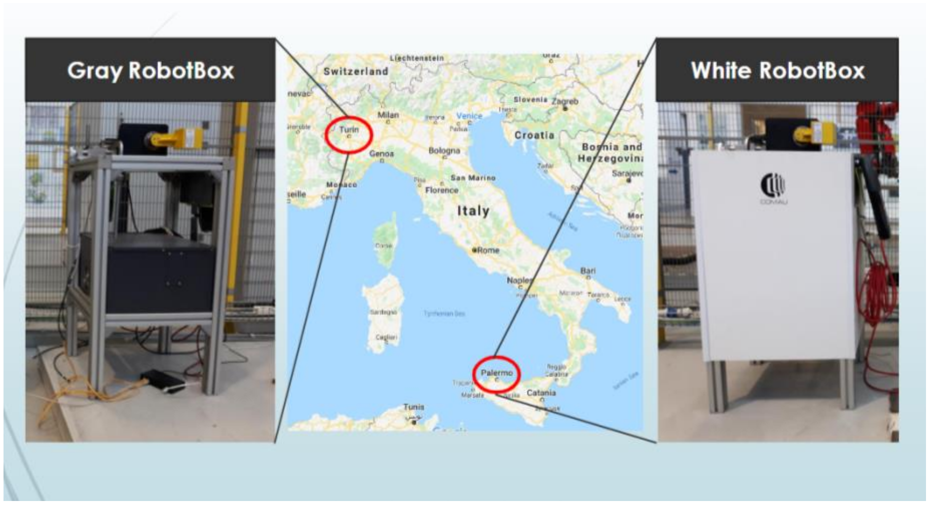


Figure 18 – Simulation of two different customers plants: in Palermo and Turin

Going into the details of the testbed we underline that it replicates an axis of COMAU robot and is composed of a motor, a belt, a gearbox reducer and 5 kilos bulk. In order to collect data at different belt tensioning levels, we had to distance the motor and gearbox each other. To do this we have installed a slicer to move the motor and a dial gauge to measure the distance between the two. We have decided to study 3 different belt tensioning levels. Let's see now which data we have used to create the dataset. The RobotBox always performs the same movement, called cycle. After each 24 seconds of cycle, we have collected robot data of motor position and absorbed current. From these two signals we have calculated 141 features at each cycle. For example, some of them are the mean of the current, the maximum value, minimum, the rms, the skewness, the integral and many others. The features have been chosen with the help of the robotics department, who better know how the belt tensioning influences the two signals.

For each level, we have performed around 6000 cycles for a total of 18000 samples for each RobotBox considering the 3 different belt tension levels obtained setting the distance between motor and gearbox as in Table 1 summarizes. For this scenario, we have chosen to train a classification model, in particular an artificial neural network.

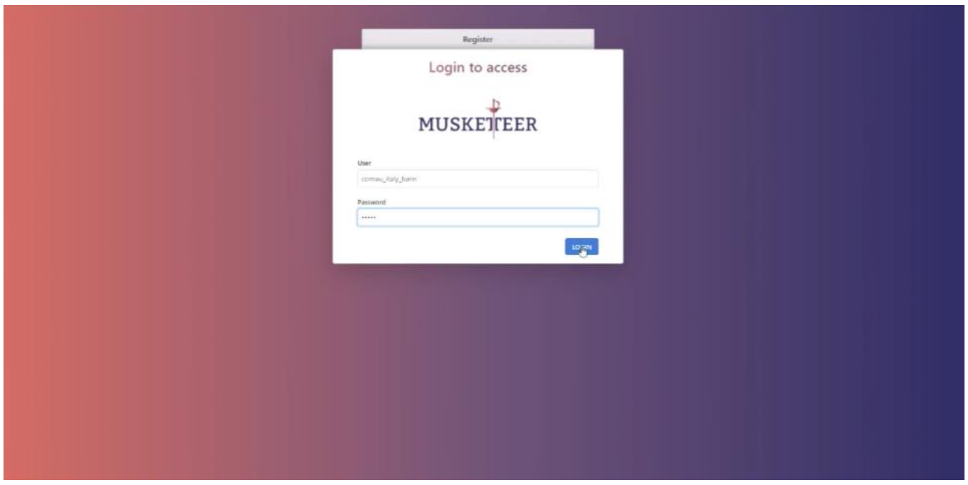
Table 1 - Data collection summary

Distance between motor and gearbox	Belt tensioning levels – labels	Samples for RobotBox white	Samples for RobotBox gray
Distance 1	Label 0	5980	5988
Distance 2	Label 1	5975	5993
Distance 3	Label 2	5981	5990

4.2 Use case execution

COMAU, as robot manufacturer, has played the role of the aggregator of the federated machine learning task and for this use case also the role of the first participant. ENGINEERING instead has played the role of the second participant. First of all, COMAU installed and configured the Desktop Client Connector by importing IBM communication and UC3M and TREE machine learning libraries as described in Section 3.2.

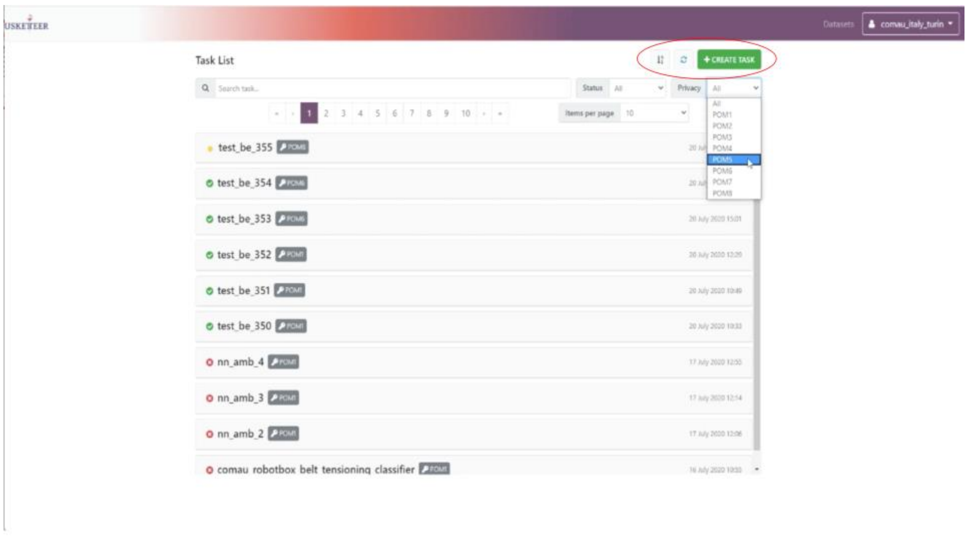
COMAU, from Turin, logged in to the Musketeer login through the login page of the Client Connector as shown in the following Figure 18.



User: comau_italy_turin
Password: *****

Figure 19 - COMAU login to Musketeer platform

Once entered in the platform, you can view and browse the created tasks. Figure 20 shows the possible filtering options for searching for a specific one: filtering by name, status and privacy level (POM). In addition, three buttons are highlighted in the image where you can change the sorting of the tasks by date: from the most recent created to the least recent, and vice versa; a button to update the task list and the button to create a new task.

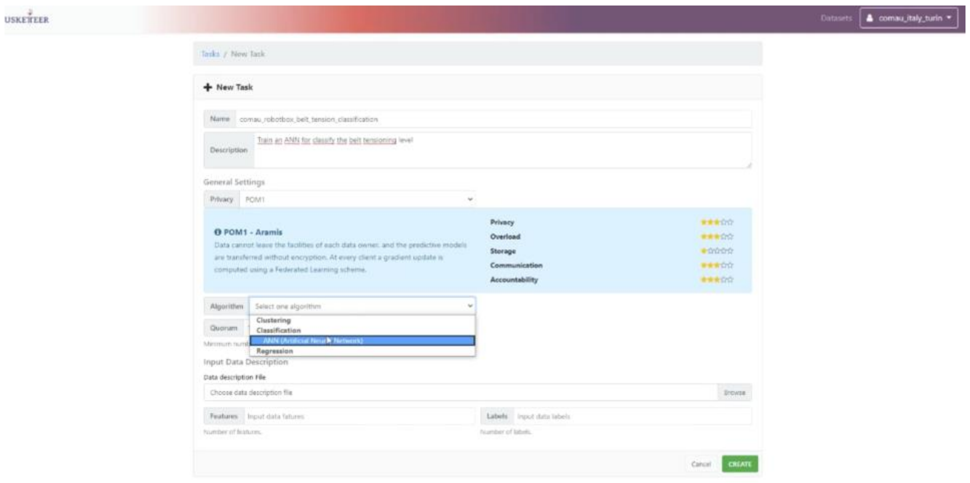


Status: All
Privacy: All
Items per page: 10

test_be_355	POM8	20 Jul
test_be_354	POM6	20 Jul
test_be_353	POM6	20 July 2020 15:01
test_be_352	POM1	20 July 2020 12:20
test_be_351	POM1	20 July 2020 10:49
test_be_350	POM1	20 July 2020 10:33
nn_amb_4	POM1	17 July 2020 12:50
nn_amb_3	POM1	17 July 2020 12:14
nn_amb_2	POM1	17 July 2020 12:06
comau robotbox belt tensioning classifier	POM1	16 July 2020 10:33

Figure 20 - COMAU task browsing

So COMAU, created the task choosing the POM1 (privacy operation mode 1) where data cannot leave the facilities of each data owner and the predictive models are transferred without encryption, and the artificial neural network algorithm developed by UC3M and TREE as algorithm. As shown in Figure 21 on the task creation page, the algorithm chosen from COMAU was the artificial neural network.



Tasks / New Task

New Task

Name: comau_robotbox_belt_tension_classification

Description: Train an ANN for classify the belt tensioning level

General Settings

Privacy: POM1

POM1 - Aramis
Data cannot leave the facilities of each data owner, and the predictive models are transferred without encryption. At every client a gradient update is computed using a Federated Learning scheme.

Privacy	★★★☆☆
Overload	★★★☆☆
Storage	★☆☆☆☆
Communication	★★★☆☆
Accountability	★★★☆☆

Algorithm: Select one algorithm
- Clustering
- Classification
  - ANN (Artificial Neural Network)
- Regression

Input Data Description

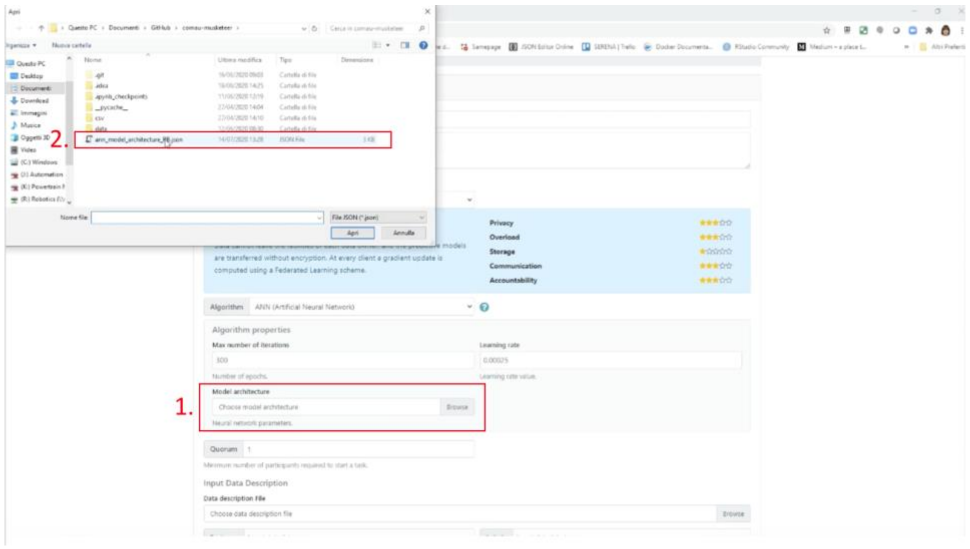
Data description File: Choose data description file — Browse

Features: Input data features — Number of features.
Labels: Input data labels — Number of labels.

Cancel — CREATE

Figure 21 - COMAU task creation

After several parameters tunings tests the following ones have been set. The maximum number of iterations was set to 300, the learning rate to 0.00015. For the model architecture, that is essentially the shape of the neural network, a 3 layers neural network with 48, 16 and 3 units was chosen. The neural network architecture, defined as a Json, was inserted by importing a Json file from the COMAU desktop file system as shown in Figure 22.



Apri — Questo PC › Documenti › GitHub › comau-musketeer

Nome	Ultima modifica	Tipo	Dimensione
.git	16/06/2020 09:03	Cartella di file	
.idea	16/06/2020 14:25	Cartella di file	
.ipynb_checkpoints	11/06/2020 12:19	Cartella di file	
__pycache__	27/04/2020 14:04	Cartella di file	
csv	27/04/2020 14:10	Cartella di file	
data	12/06/2020 08:30	Cartella di file	
ann_model_architecture_[illegible].json	14/07/2020 13:28	JSON File	1 KB

Nome file: — File JSON (*.json) — Apri — Annulla

Algorithm: ANN (Artificial Neural Network)

Algorithm properties

Max number of iterations: 300 — Number of epochs.
Learning rate: 0.00015 — Learning rate value.

Model architecture: Choose model architecture — Browse — Neural network parameters.

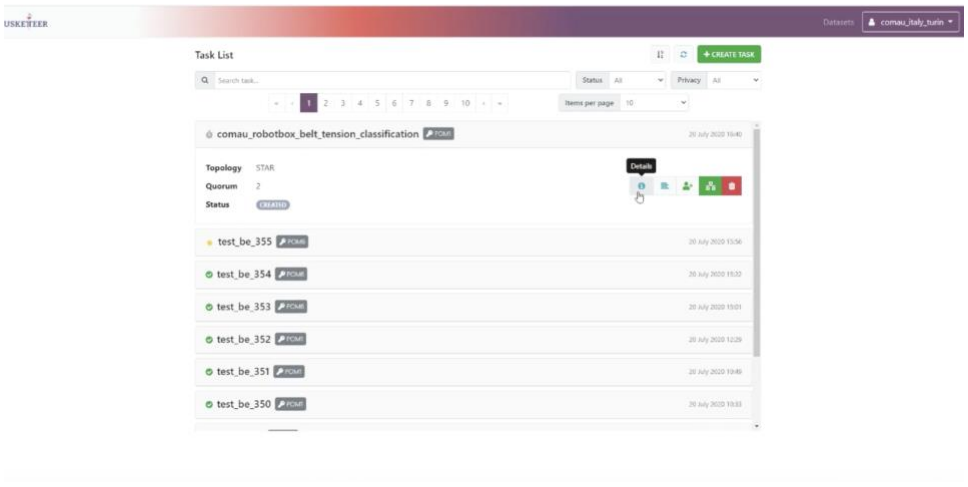
Quorum: 1 — Minimum number of participants required to start a task.

Input Data Description

Data description File: Choose data description file — Browse

Figure 22 - COMAU model architecture inserting

Once created the task it will be displayed among the available tasks. From the task card, you can also view a recap of all parameters set by clicking on the “Details” button as shown in Figure 23.



Task List

Search task... | Status: All | Privacy: All | Items per page: 10

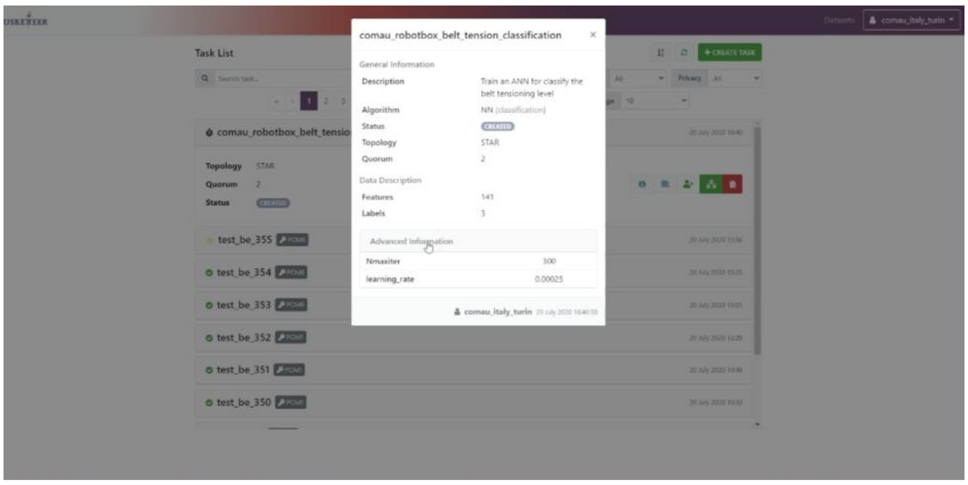
comau_robotbox_belt_tension_classification POM1 — 20 July 2020 16:40

Topology: STAR
Quorum: 2
Status: CREATED

test_be_355 POM6 — 20 July 2020 15:56
test_be_354 POM6 — 20 July 2020 15:32
test_be_353 POM6 — 20 July 2020 15:01
test_be_352 POM1 — 20 July 2020 12:29
test_be_351 POM1 — 20 July 2020 10:49
test_be_350 POM1 — 20 July 2020 10:33

Figure 23 - COMAU task card: details button

The recap of the task is then displayed in a modal, as shown in Figure 24.



comau_robotbox_belt_tension_classification

General Information

Description	Train an ANN for classify the belt tensioning level
Algorithm	NN (classification)
Status	CREATED
Topology	STAR
Quorum	2

Data Description

Features	141
Labels	3

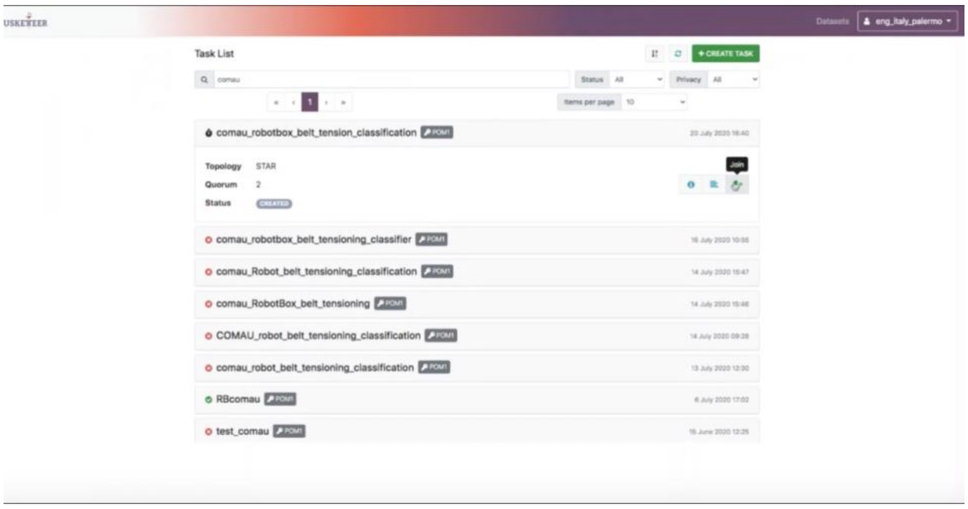
Advanced Information

Nmaxiter	300
learning_rate	0.00025

comau_italy_turin 20 July 2020 16:40:30

Figure 24 - COMAU task recap

After the task creation, COMAU aggregated it by clicking on the “Aggregate” button of the related task as shown in Figure 25.



Task List

comau | Status: All | Privacy: All | Items per page: 10

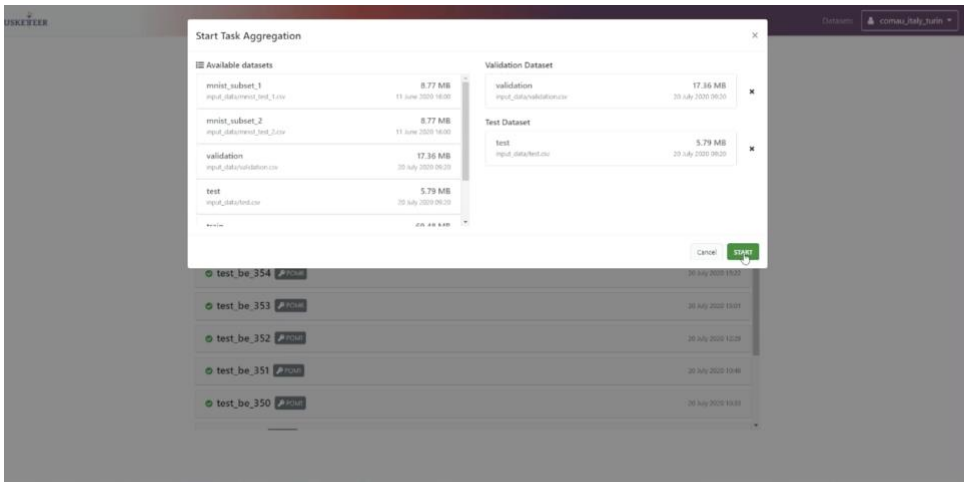
comau_robotbox_belt_tension_classification POM1 — 20 July 2020 16:40

Topology: STAR
Quorum: 2
Status: CREATED

comau_robotbox_belt_tensioning_classifier POM1 — 16 July 2020 10:56
comau_Robot_belt_tensioning_classification POM1 — 14 July 2020 15:47
comau_RobotBox_belt_tensioning POM1 — 14 July 2020 15:46
COMAU_robot_belt_tensioning_classification POM1 — 14 July 2020 09:28
comau_robot_belt_tensioning_classification POM1 — 13 July 2020 12:30
RBcomau POM1 — 6 July 2020 17:02
test_comau POM1 — 16 June 2020 12:26

Figure 25 - COMAU task card: aggregation button

Then in a modal, as shown in Figure 26, COMAU drag-and-dropped their validation and test data and started the task as an aggregator; in the same way COMAU also joined the task as participant using the gray RobotBox data for the training process.


Start Task Aggregation

Available datasets

mnist_subset_1 — 8.77 MB — input_data/mnist_test_1.csv — 11 June 2020 16:00
mnist_subset_2 — 8.77 MB — input_data/mnist_test_2.csv — 11 June 2020 16:00
validation — 17.36 MB — input_data/validation.csv — 20 July 2020 09:20
test — 5.79 MB — input_data/test.csv — 20 July 2020 09:20

Validation Dataset

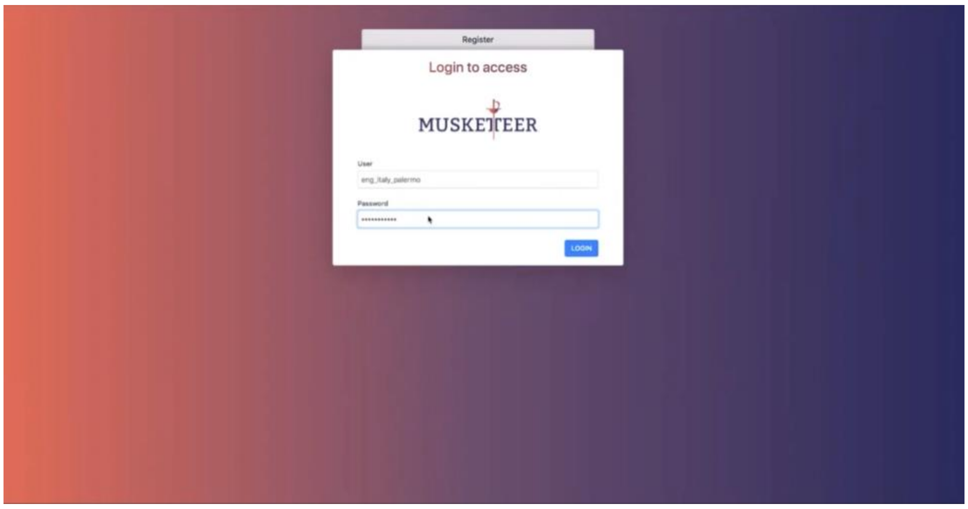
validation — 17.36 MB — input_data/validation.csv — 20 July 2020 09:20

Test Dataset

test — 5.79 MB — input_data/test.csv — 20 July 2020 09:20

Figure 26 - COMAU task aggregation

Then ENGINEERING also logged into the platform, as shown in Figure 27.

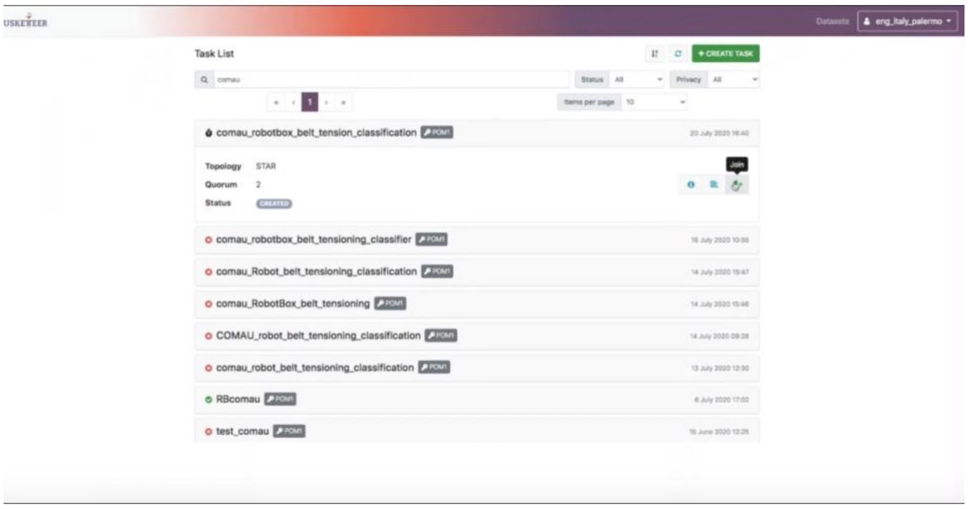

Login to access

User: eng_italy_palermo

Password: ••••••••••••

Figure 27 - ENGINEERING login to Musketeer platform

Entered the platform, ENGINEERING searched the task created by COMAU through the filtering options and joined it as it is shown in Figure 28.



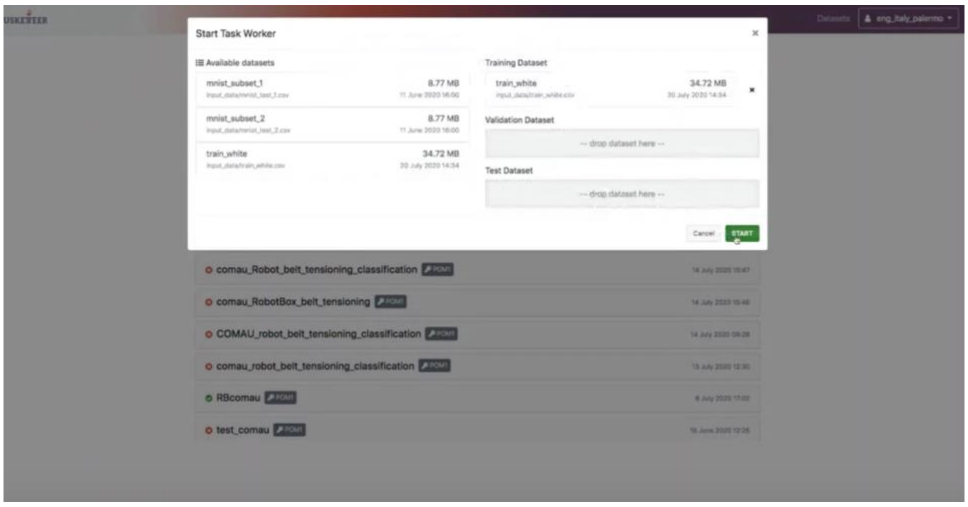
comau_robotbox_belt_tension_classification POM1 20 July 2020 16:40

Topology STAR
Quorum 2
Status CREATED

comau_robotbox_belt_tensioning_classifier POM1 16 July 2020 10:56
comau_Robot_belt_tensioning_classification POM1 14 July 2020 15:47
comau_RobotBox_belt_tensioning POM1 14 July 2020 15:46
COMAU_robot_belt_tensioning_classification POM1 14 July 2020 09:28
comau_robot_belt_tensioning_classification POM1 13 July 2020 12:30
RBcomau POM1 6 July 2020 17:02
test_comau POM1 16 June 2020 12:26

Figure 28 - ENGINEERING task browsing

Finally, ENGINEERING started the task as participant with the training data of the white RobotBox that was already bonded to the Client Connector.



Start Task Worker

Available datasets

mnist_subset_1 8.77 MB
input_data/mnist_test_1.csv 11 June 2020 16:00

mnist_subset_2 8.77 MB
input_data/mnist_test_2.csv 11 June 2020 16:00

train_white 34.72 MB
input_data/train_white.csv 20 July 2020 14:34

Training Dataset
train_white 34.72 MB
input_data/train_white.csv 20 July 2020 14:34

Validation Dataset
-- drop dataset here --

Test Dataset
-- drop dataset here --

comau_Robot_belt_tensioning_classification POM1 14 July 2020 15:47
comau_RobotBox_belt_tensioning POM1 14 July 2020 15:46
COMAU_robot_belt_tensioning_classification POM1 14 July 2020 09:28
comau_robot_belt_tensioning_classification POM1 13 July 2020 12:30
RBcomau POM1 6 July 2020 17:02
test_comau POM1 16 June 2020 12:26

Figure 29 - ENGINEERING task join

The quorum is then reached, and the task can turn to the "running" status. It is then possible from COMAU to visualize the logs of the task execution, showing the progress of the task until its completion, as shown in the following figure.

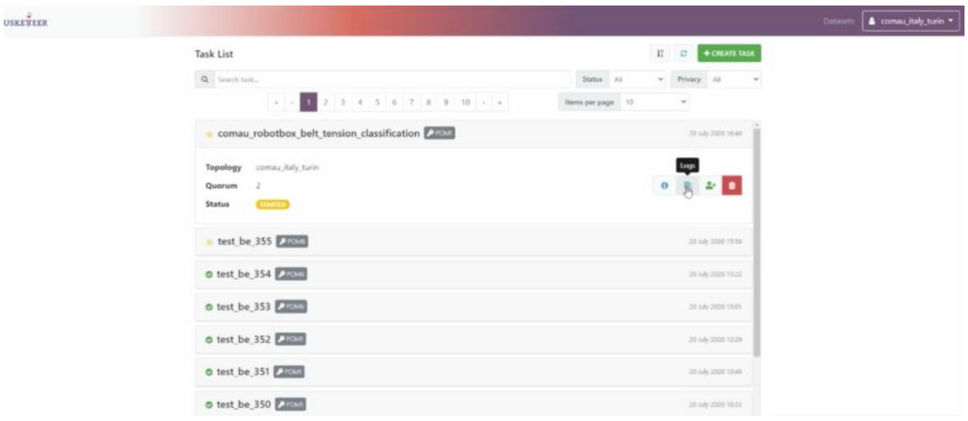
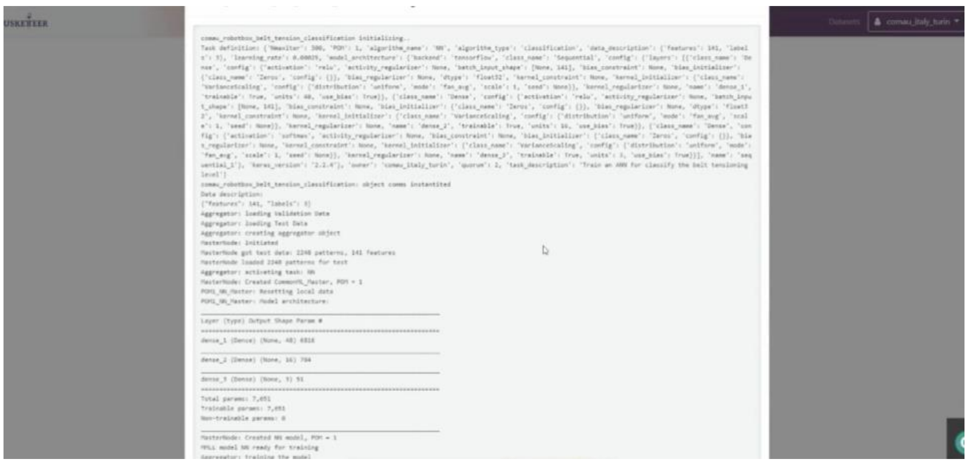


Figure 30 - COMAU task card: logs button

The logs of the use case task execution are showed in the next Figure 31.



```
comau_robotbox_belt_tension_classification initializing..
Task definition: {'Nmaxiter': 300, 'POM': 1, 'algorithm_name': 'NN', 'algorithm_type': 'classification', 'data_description': {'features': 141, 'label
s': 3}, 'learning_rate': 0.00025, 'model_architecture': {'backend': 'tensorflow', 'class_name': 'Sequential', 'config': {'layers': [{'class_name': 'De
nse', 'config': {'activation': 'relu', 'activity_regularizer': None, 'batch_input_shape': [None, 141], 'bias_constraint': None, 'bias_initializer':
{'class_name': 'Zeros', 'config': {}}, 'bias_regularizer': None, 'dtype': 'float32', 'kernel_constraint': None, 'kernel_initializer': {'class_name':
'VarianceScaling', 'config': {'distribution': 'uniform', 'mode': 'fan_avg', 'scale': 1, 'seed': None}}, 'kernel_regularizer': None, 'name': 'dense_1',
'trainable': True, 'units': 48, 'use_bias': True}}, {'class_name': 'Dense', 'config': {'activation': 'relu', 'activity_regularizer': None, 'batch_inpu
t_shape': [None, 141], 'bias_constraint': None, 'bias_initializer': {'class_name': 'Zeros', 'config': {}}, 'bias_regularizer': None, 'dtype': 'float3
2', 'kernel_constraint': None, 'kernel_initializer': {'class_name': 'VarianceScaling', 'config': {'distribution': 'uniform', 'mode': 'fan_avg', 'scal
e': 1, 'seed': None}}, 'kernel_regularizer': None, 'name': 'dense_2', 'trainable': True, 'units': 16, 'use_bias': True}}, {'class_name': 'Dense', 'con
fig': {'activation': 'softmax', 'activity_regularizer': None, 'bias_constraint': None, 'bias_initializer': {'class_name': 'Zeros', 'config': {}}, 'bia
s_regularizer': None, 'kernel_constraint': None, 'kernel_initializer': {'class_name': 'VarianceScaling', 'config': {'distribution': 'uniform', 'mode':
'fan_avg', 'scale': 1, 'seed': None}}, 'kernel_regularizer': None, 'name': 'dense_3', 'trainable': True, 'units': 3, 'use_bias': True}}], 'name': 'seq
uential_1'}, 'keras_version': '2.2.4'}, 'owner': 'comau_italy_turin', 'quorum': 2, 'task_description': 'Train an ANN for classify the belt tensioning
level'}
comau_robotbox_belt_tension_classification: object comms instantited
Data description:
{"features": 141, "labels": 3}
Aggregator: loading Validation Data
Aggregator: loading Test Data
Aggregator: creating aggregator object
MasterNode: Initiated
MasterNode got test data: 2248 patterns, 141 features
MasterNode loaded 2248 patterns for test
Aggregator: activating task: NN
MasterNode: Created CommonML_Master, POM = 1
POM1_NN_Master: Resetting local data
POM1_NN_Master: Model architecture:
_________________________________________________________________
Layer (type) Output Shape Param #
=================================================================
dense_1 (Dense) (None, 48) 6816
_________________________________________________________________
dense_2 (Dense) (None, 16) 784
_________________________________________________________________
dense_3 (Dense) (None, 3) 51
=================================================================
Total params: 7,651
Trainable params: 7,651
Non-trainable params: 0
_________________________________________________________________
MasterNode: Created NN model, POM = 1
MPLL model NN ready for training
Aggregator: training the model
```

Figure 31 - COMAU task aggregator log

Finally, once the task is completed, it is then got a confusion matrix chart applied to the test data, showing the accuracy achieved by the model for each label. The confusion matrix chart is available by clicking on the “Result” button of the completed task by COMAU, as shown in below.

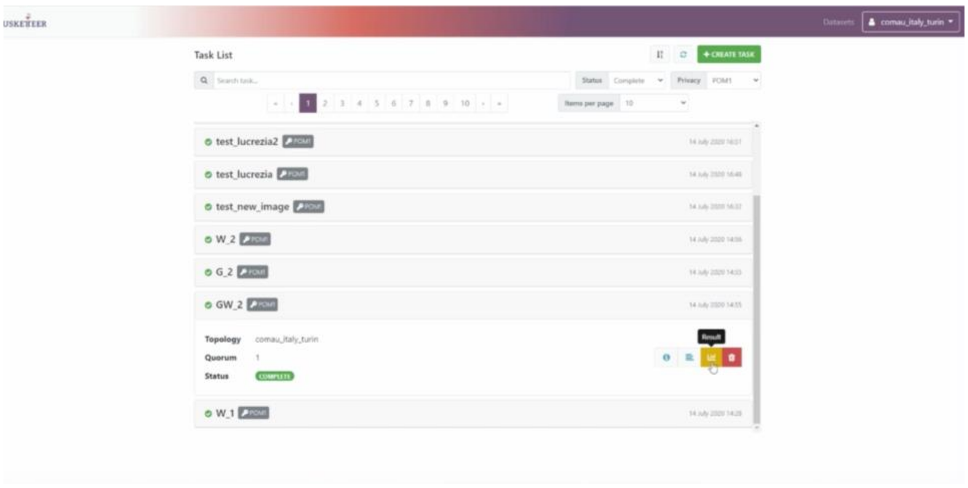
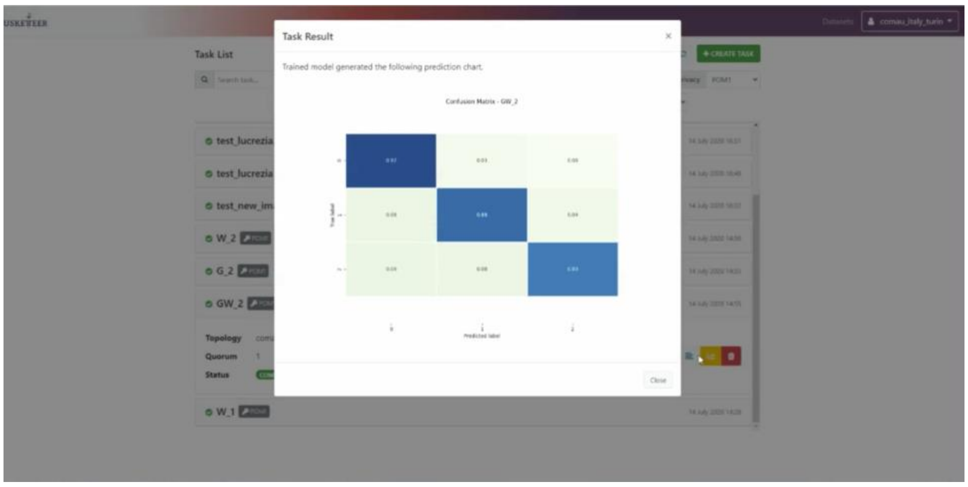


Figure 32 - COMAU task card: result button

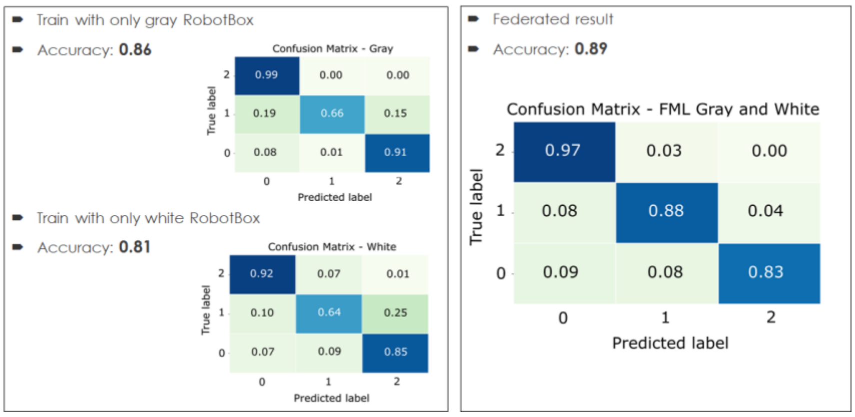
Among the various tests performed, the best result given by the resulting confusion matrix is shown in Figure 33.


Task Result

Trained model generated the following prediction chart.

Figure 33 - COMAU confusion matrix result

4.3 Results



Confusion Matrix - Gray

True label \ Predicted label	0	1	2
2	0.99	0.00	0.00
1	0.19	0.66	0.15
0	0.08	0.01	0.91

Confusion Matrix - White

True label \ Predicted label	0	1	2
2	0.92	0.07	0.01
1	0.10	0.64	0.25
0	0.07	0.09	0.85

Confusion Matrix - FML Gray and White

True label \ Predicted label	0	1	2
2	0.97	0.03	0.00
1	0.08	0.88	0.04
0	0.09	0.08	0.83

Figure 34 - Use case results comparison

The results were very promising and in particular if compared with the results of a non-federated approach. As in Figure 34 on the right, the overall accuracy reached training the model with both RobotBox is 89% and the related confusion matrix has a very populated diagonal. On the other hand, in Figure 34 on the left, the accuracy of a model trained only with the data of the grey RobotBox or only with the data of white RobotBox is less, respectively 86% and 81%.

5 Conclusion

The purpose of this document *D7.3 – First prototype of the MUSKETEER Client connectors*, is to explain the key components and the main user interactions with the Client Connector to exploit the MUSKETEER Platform functionalities.

The instructions for setting up the client connector are provided together with detailed walk-throughs of the demonstration on data from the Smart Manufacturing use case provided by COMAU.

The source code of the first prototype version of the MUSKETEER Client Connector is released as open source under GNU AGPLv3 license [1][2].

However, as the project development activities evolve, this initial version of the described services composing the client connector will receive the necessary updates and optimisations in order to encapsulate all the project's advancements, as well as the new technical requirements that will be extracted from the feedback that will be collected from the platform's evaluation.

Furthermore, a part of the effort within the task T7.2 will be dedicated to the development of the CC in Cluster mode, the requirements of which are desired in all those cases where the storage and the processing of Big Data have to be supported, through horizontal scalability and workload distribution on multiple nodes of the cluster.

Hence, the forthcoming versions of this deliverable will incorporate all the updates that are necessary to be introduced.

6 References

[1] https://github.com/Engineering-Research-and-Development/musketeer-client-connector-backend

[2] https://github.com/Engineering-Research-and-Development/musketeer-client-connector-frontend