MATRYCS

D4.1 | MATRYCS-PROCESSING (1st technology release)

WP4 – Big Data Management & AI Services Layer

August 2021

www.matrycs.eu

Modular Big Data Applications for Holistic Energy Services in Buildings

MATRYCS

Disclaimer

Copyright Message

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

Grant Agreement Number	101000158	Acronym	MATRYCS
Full Title	Modular Big Data Applications for Holistic Energy Services in Buildings		
Topic	LC-SC3-B4E-6-2020 | Big data for buildings		
Funding scheme	H2020- IA: Innovation Action		
Start Date	October 2020	Duration	36
Project URL	www.matrycs.eu		
Project Coordinator	ENG		
Deliverable	MATRYCS-PROCESSING (1st technology release)		
Work Package	WP4 – Big Data Management & AI Services Layer		
Delivery Month (DoA)	M11	Version	1.0
Actual Delivery Date	31/08/2021		
Nature	Other	Dissemination Level	Public
Lead Beneficiary	HOLISTIC		
Authors	Zoi Mylona, Nikolaos Sofias [HOLISTIC], Panagiotis Kapsalis, Haris Doukas, Vangelis Marinakis, Konstantinos Alexakis [NTUA], Leandro Lombardo, Marija Borisov [ENG], Zhiyu Pan [RWTH], Blaz Merela [COMSENSUS], Daniele Antonucci [EURAC]		
Quality Reviewer(s):	Antonucci Daniele [EURAC], Aija Zučika [LEIF]		
Keywords	ML/DL Models, Model Development, ML Suite, Data Feed Module, Evaluation Framework, Model Serving, Serving Framework Deployment, Data Processing, Data Management		

Preface

MATRYCS focuses on addressing emerging challenges in big data management for buildings with an **open holistic solution** for Business-to-Business platforms, able to give a competitive solution to stakeholders operating in building sector and to open new market opportunities. **MATRYCS Modular Toolbox**, will realise a holistic, state-of-the-art AI-empowered framework for decision-support models, data analytics and visualisations for Digital Building Twins and real-life applications aiming to have significant impact on the building sector and its lifecycle, as it will have the ability to be utilised in a wide range of use cases under different perspectives:

- Monitoring and improvement of the energy performance of buildings - **MATRYCS-PERFORMANCE**
- Design facilitation and development of building infrastructure - **MATRYCS-DESIGN**
- Policy making support and policy impact assessment - **MATRYCS-POLICY**
- De-risking of investments in energy efficiency - **MATRYCS-FUND**

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

Who We Are

	Participant Name	Short Name	Country Code	Logo
1	ENGINEERING – INGEGNERIA INFORMATICA SPA	ENG	IT	
2	NATIONAL TECHNICAL UNIVERSITY OF ATHENS	NTUA	GR	
3	FUNDACION CARTIF	CARTIF	ES	
4	RHEINISCH-WESTFAELISCHE TECHNISCHE HOCHSCHULE AACHEN	RWTH	DE	
5	ACCADEMIA EUROPEA DI BOLZANO	EURAC	IT	
6	HOLISTIC IKE	HOLISTIC	GR	
7	COMSENSUS, KOMUNIKACIJE IN SENZORIKA, DOO	COMSENSUS	SL	
8	BLAGOVNO TRGOVINSKI CENTER DD	BTC	SL	
9	PRZEDSIEBIORSTWO ROBOT ELEWACYJNYCHFASADA SP ZOO	FASADA	PL	
10	MIASTO GDYNIA	GDYNIA	PL	
11	COOPERNICO - COOPERATIVA DE DESENVOLVIMENTO SUSTENTAVEL CRL	COOPERNICO	PT	
12	ASM TERNI SPA	ASM	IT	
13	VEOLIA SERVICIOS LECAM SOCIEDAD ANONIMA UNIPERSONAL	VEOLIA	ES	
14	ICLEI EUROPEAN SECRETARIAT GMBH (ICLEI EUROPASEKRETARIAT GMBH)	ICLEI	DE	
15	ENTE PUBLICO REGIONAL DE LA ENERGIA DE CASTILLA Y LEON	EREN	ES	
16	VIDES INVESTICIJU FONDS SIA	LEIF	LV	
17	COMITE EUROPEEN DE COORDINATION DE L'HABITAT SOCIAL AISBL	HOUSING EUROPE	BE	
18	SEVEN, THE ENERGY EFFICIENCY CENTER Z.U.	SEVEN	CZ	

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

Contents

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

Figures

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

Tables

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

Abbreviation and Acronyms

Acronym	Description
AI	Artificial Intelligence
ANN	Artificial Neural Networks
AWS	Amazon Web Services
CSV	Comma-Separated values
DAG	Directed Acyclic Graphs
DB	Database
DL	Deep Learning
ECM	Energy Conservation Measure
EPC	Energy Performance Contracting
IoT	Internet of Things
JSON	JavaScript Object notation
LSP	Large Scale Pilot
ML	Machine Learning
MLOps	Machine Learning Operations
REST	Representational state transfer
SQL	Structured Query Language
VM	Virtual Machine

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

Executive Summary

The D4.1 - *MATRYCS-PROCESSING (1st technology release)* provides a description of the first version of the Big Data Management & Services Layer (MATRYCS-PROCESSING) according with the MATRYCS high level reference architecture defined in the deliverable D2.3 - *MATRYCS Reference Architecture for Buildings Data v1.0*.

The 1st technology release is mainly focused to the preliminary identification and evaluation of the envisaged technologies solutions for the MATRYCS-PROCESSING layer with limited data and in a constraint scenario and reports the activities done until M11 in order to provide a first complete release of the MATRYCS-PROCESSING layer under WP4-*Big Data Management & AI Services laye*r and in particular Task 4.1 - Data Validation and ML / DL Models Definition, Task 4.2 - Models Training, and Task 4.3 - Model Evaluation and Serving Framework.

This document presents an overview of the conceptual architecture of MATRYCS-PROCESSING with a general description of its main components. Furthermore, the connection of MATRYCS-PROCESSING with the MATRYCS-GOVERNANCE layer and the MATRYCS-ANALYTICS layer, and the connection with the MATRYCS Reference Architecture is demonstrated. Finally, the MATRYCS-PROCESSING data flow, components and processes are demonstrated through the LSP1 (BTC) and LSP5 (COOPERNICO) in order to showcase the implementation details and preliminary information on deployment methods of the MATRYCS-PROCESSING modules is provided.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

1 Introduction

1.1 Purpose of this document

The purpose of D4.1-MATRYCS-PROCESSING (1st technology release) is to report the implementation of the first technology release of the Big Data Management & AI Services Layer (MATRYCS-PROCESSING). In this regard, this deliverable reports the activities and outcome carried out in Big Data Management and AI Services Layer (MATRYCS-PROCESSING) focusing on the preliminary evaluation of the envisaged technologies and highlighting the conceptual architecture MATRYCS-PROCESSING layer with the interdependencies of the different components, the technological solutions, the interaction between modules implementation aspects and deployment approaches are analysed and evaluated. Finally, the MATRYCS–PROCESSING components and processes are demonstrated for LSP1 (BTC) and LSP5 (COOPERNICO) datasets.

1.2 Structure of the document

The D4.1-MATRYCS-PROCESSING (1st technology release) is organized as follows:

- In section 1, the purpose of the document and related structure is presented.
- In section 2, the overall MATRYCS-PROCESSING layer architecture and a general description of its main components is demonstrated.

Sections 3, 4, 5, 6 and 7 describe in detail the MATRYCS-PROCESSING main components architecture, development, and their interconnections. Specifically:

- In section 3, the Data Feed Module the output of T4.1-Data Validation and ML/DL Models definition is presented.
- In section 4, the Machine Learning Suite, one of the outputs of T4.2-Models Training is presented.
- In section 5, the Model Development Module one of the outputs of T4.2-Models Training is presented.
- In section 6, the Model Serving Module one of the outputs of T4.3-Model Evaluation and Serving Framework is demonstrated.
- In section 7, the Evaluation Framework is presented, which is one of the outputs of T4.3-Model Evaluation and Serving Framework.

There is a dedicated section for MATRYCS-PROCESSING demonstration:

- In section 8, the MATRYCS-PROCESSING layer and its building blocks are demonstrated for LSP1 (BTC) and LSP5 (COOPERNICO).
- Finally, section 9 outlines the upcoming activities to be undertaken in order to proceed with a more advanced development of the MATRYCS-PROCESSING layer.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

2 MATRYCS-PROCESSING Architecture

MATRYCS-PROCESSING architecture contains all the components that can provide intelligence and quick adaptation over stored data. The main components as illustrated below (Figure 1) and they are the Data Feed Module, the Model Development Module, the Model Evaluation Module, the Models Shared Storage, the ML Suite, and the Model Serving Framework.

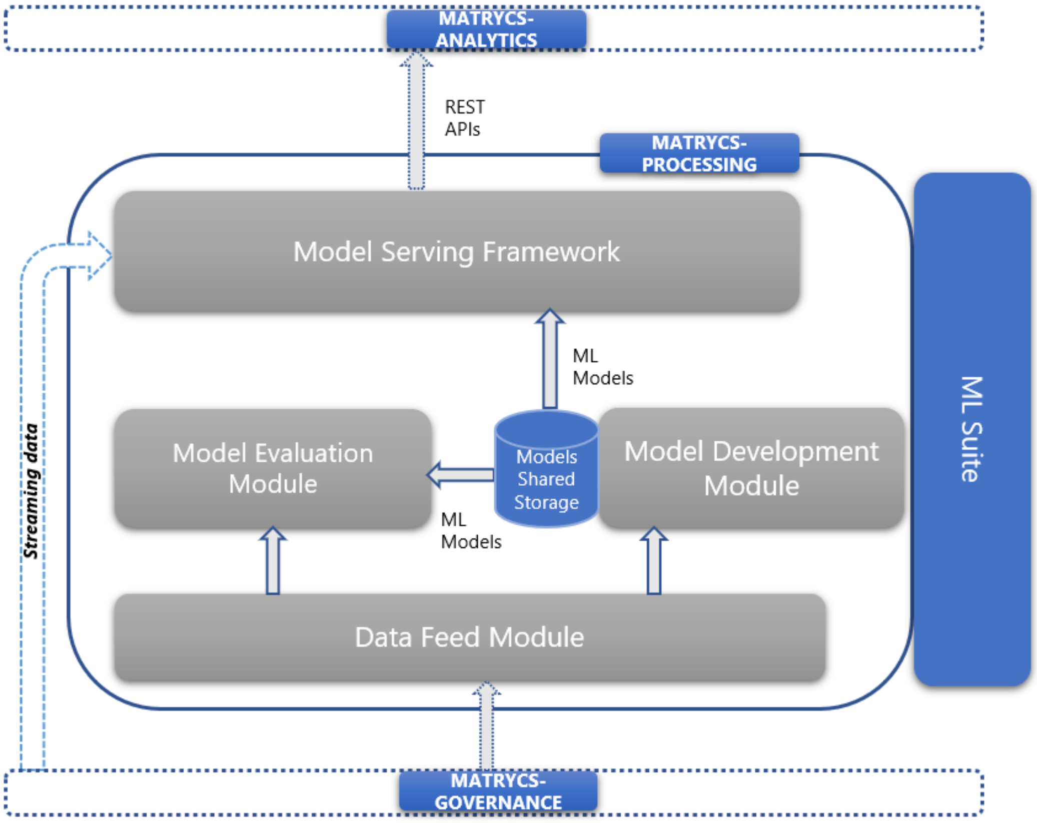


Figure 1: MATRYCS PROCESSING High Level Architecture

2.1 MATRYCS-PROCESSING Definition

The MATRYCS-PROCESSING layer envisions to get all the intelligence components of the MATRYCS solution together and encapsulate them into a stand-alone library of reusable ML and DL models. MATRYCS aims to make these models available, in order to promote and facilitate quick adaptation along different contexts.

The standard-based data modelling and processing methods devised in WP4, will contribute to the definition of highly replicable and usable analytics building services.

MATRYCS-PROCESSING is responsible for a series of critical issues, which focus on:

- retrieving the data from storage
- transforming properly raw data
- the deployment of ML models (ANN, classifiers, knowledge representation and reasoning etc.)
- using state-of-the-art tools and libraries

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

- developing models based on the underlying data,
- training the models according to the LSPs' needs and
- feeding models with both batch and streaming data coming from the Query Engine and the Data Streaming Module respectively.

The MATRYCS-PROCESSING layer is a composition of four modules:

- Data Feed Module,
- ML Suite,
- Model Development Module, and
- Model Serving Module.

In Figure 2 the MATRYCS-PROCESSING low level architecture is demonstrated. Specifically, the data are inserted on MATRYCS-PROCESSING layer through Data Feed Module, which receives them from MATRYCS Staging Area and processes them. In the next step, the Model Development Model receives the transformed data from the Data Feed Module REST Services for starting the training of ML/DL models. After training, the output models are stored to Models Shared Storage where the Model Evaluation Module receives them for evaluation. Finally, the evaluated ML/DL models are exposed to MATRYCS-ANALYTICS layer through Serving Framework.

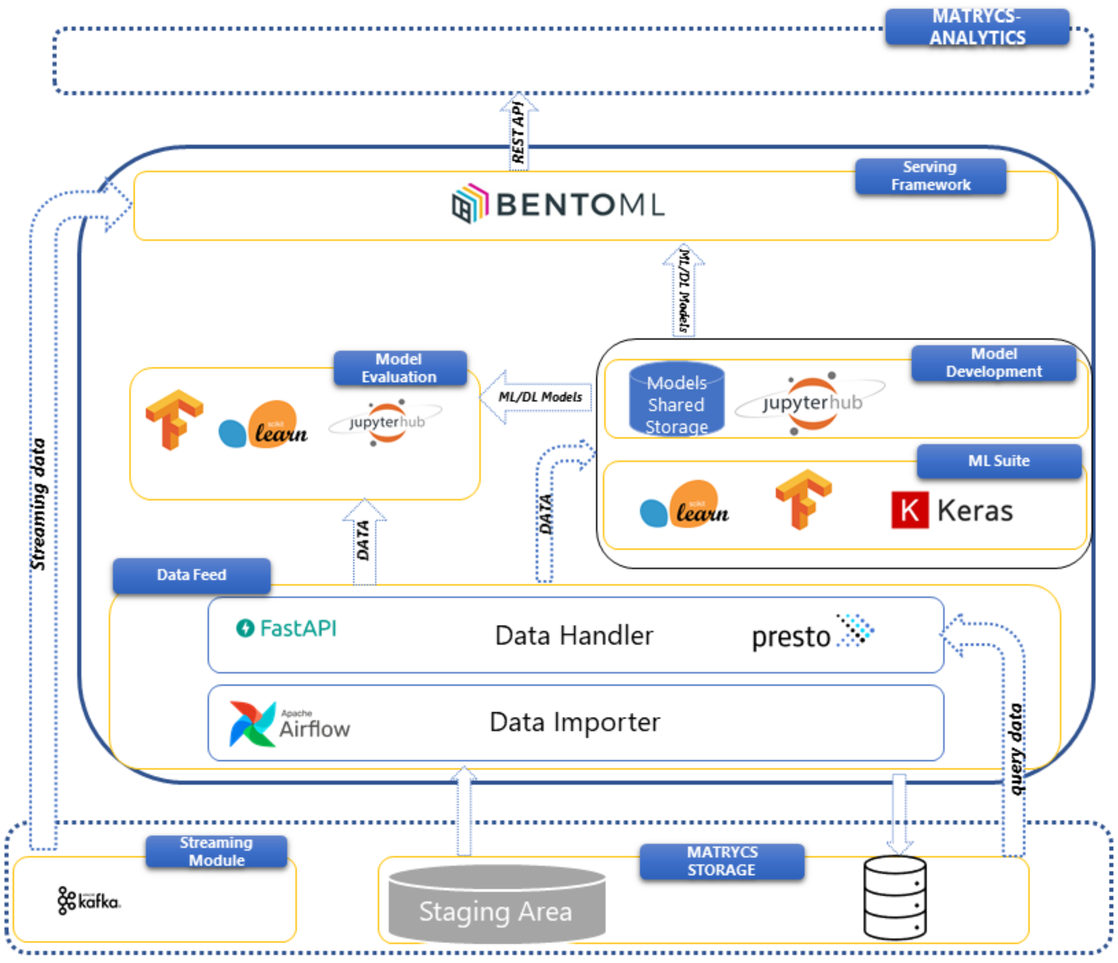


Figure 2: MATRYCS Low Level Architecture

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

2.2 MATRYCS-PROCESSING Layer Components

The MATRYCS-PROCESSING Layer includes components of the MATRYCS solution in order to provide a library of reusable AI-based ML/DL models, which can be quickly adaptable and reusable along different contexts. The included functional components are briefly described below:

- **Data Feed Module**: This module's role is to retrieve the underlying data from the storage, perform the needed transformations and finally pass the properly transformed data to the AI models. The models cannot operate with the raw data, in the format they are stored. Contrariwise, each model requires the data to have a specific format in order to be able to handle them. Consequently, this stage is mandatory. An example of a transformation after interacting with the query execution engine, is the handling of missing values, the normalization of them if needed or the selection of the right features. Once this step is completed, the properly transformed (final) data are passed to the AI models from the ML suite.
- **ML Suite**: ML Suite is a library of state-of-the-art AI data-driven tools and methods, that is used for the development of the MATRYCS AI models. Multiple different technologies and software are exploited for ML (scipy[1], scikit-learn[2], Spark MLib[3]), DL (Keras[4], Pytorch[5], TensorFlow[6], Horovod[7]) and Image Processing (OpenCV[8], scikit-image[9]). The result is to expose a rich and flexible software library in order to define, train and deploy ML models, including ANN classifiers, knowledge representation and reasoning aiming to attach new knowledge and predictions on the existing extreme-scale streams of data.
- **Model Development Module**: This module is about the exploitation of the ML Suite and the usage of the available tools in order to create and train the models based on the existing data. By using well established and stable methods, such as regression analysis, clustering and neural networks, the properly transformed data are fed to the training models.
- **Model Serving Module**: This module constitutes the building blocks of the upper layer and is going to include the developed and trained models that are being produced by the Model Development Module. These models are fed with both batch and streaming data, which are output by the Query Engine and the Data Streaming Module respectively.

[1] Scipy, https://www.scipy.org/

[2] Scikit-Learn, https://scikit-learn.org/stable/

[3] SPARK-MMlib, https://spark.apache.org/mllib/

[4] Keras, https://keras.io/

[5] Pytorch, https://pytorch.org/

[6] Tensorflow, https://www.tensorflow.org/

[7] Hovorod, https://horovod.ai/

[8] OpenCV, https://opencv.org/

[9] Scikit-Image, https://scikit-image.org/

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

2.3 Connection with Big Data and IoT Reference Architecture

The goal of MATRYCS is to define and deliver an open Reference Architecture for Buildings Data, which ensures compatibility with existing dataset format across Europe and be compliant with applicable EU standards (e.g. privacy, security, intellectual property). The MATRYCS architecture is following the three-tier architecture MATRYCS-GOVERNANCE, MATRYCS-PROCESSING and MATRYCS-ANALYTICS. According to the Big Data Value Chain of MATRYCS framework (Figure 3), the MATRYCS-PROCESSING will contribute to the Management & Storage and Usage. The technical specifications for MATRYCS-PROCESSING are AI/ML/DL Training, Testing, Simulation and Visualisation.

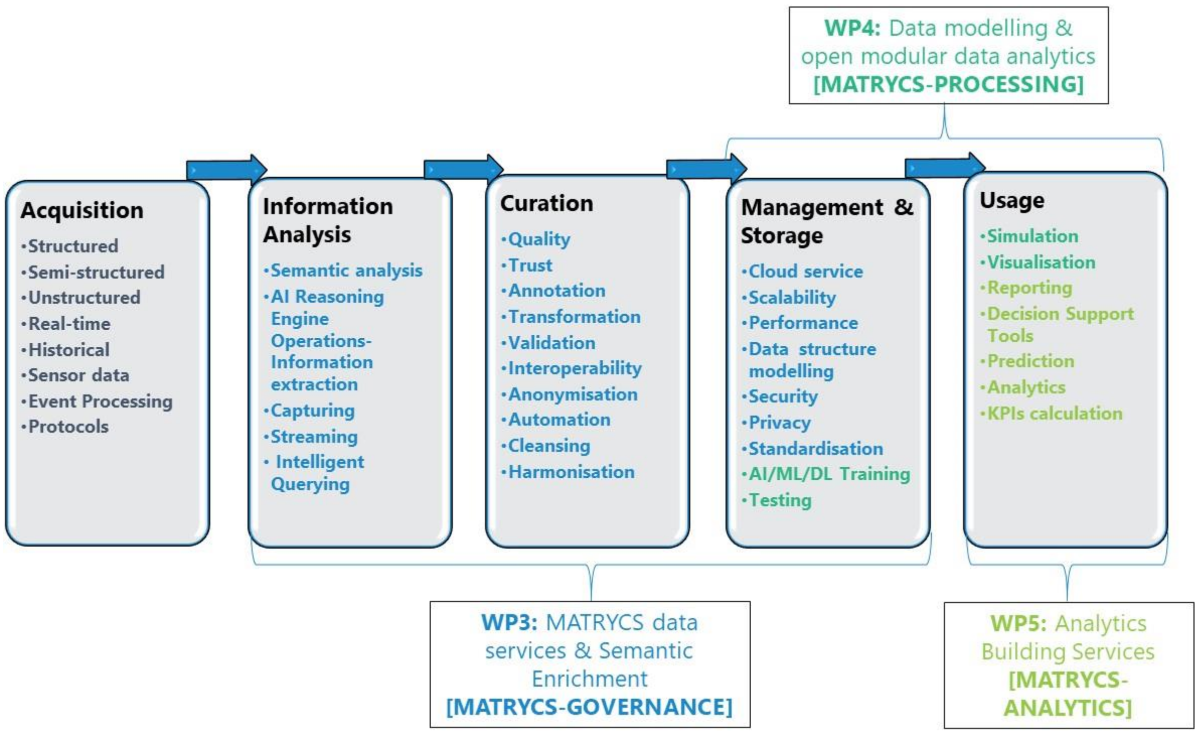


Figure 3: BDVC for MATRYCS Framework

The MATRYCS-PROCESSING is a decentralised architecture, where data are being stored locally at the edge layer and run at the AI layer in centralised cloud. The focus of MATRYCS-PROCESSING is to implement the Big Data management and AI services layer. It is responsible for specific purposes and suitable for generating new information and supporting decision-making. Data analysis includes investigation, transformation, and modelling of data to identify relevant data, compose and extract useful hidden information with high business potential. It is a broad field that combines knowledge from many scientific fields and utilises a range of tools such as data mining, ML, DL, and business intelligence. Specifically, in MATRYCS-PROCESSING there is a library of reusable AI-based ML and DL models that are developed to adapt and reuse the ML models quickly. The AI components are delivered to the services for the Digital Building Twins and real-life applications in MATRYCS-ANALYTICS.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

2.4 Connection with MATRYCS-GOVERNANCE Layer

MATRYCS-PROCESSING Layer is connected with the MATRYCS-GOVERNANCE Layer through Data Feed Module and Serving Framework. More specifically, the Data Feed Module consists of two sub-modules: The Data Importer and the Data Handler.

The Data Importer is the pipeline, based on APACHE Airflow[10] framework that receives data from the MATRYCS distributed storage/staging area, transforms them into a suitable format to be used for the training of the ML/DL models. After the preparation process, the data are stored on MATRYCS MongoDB[11].

The Data Handler is the REST service for distributing the transformed data across MATRYCS-PROCESSING. Furthermore, it allows some ad-hoc operations over stored data. The following schema (Figure 4) demonstrates the data flow between MATRYCS-PROCESSING and MATRYCS-GOVERNANCE layers.

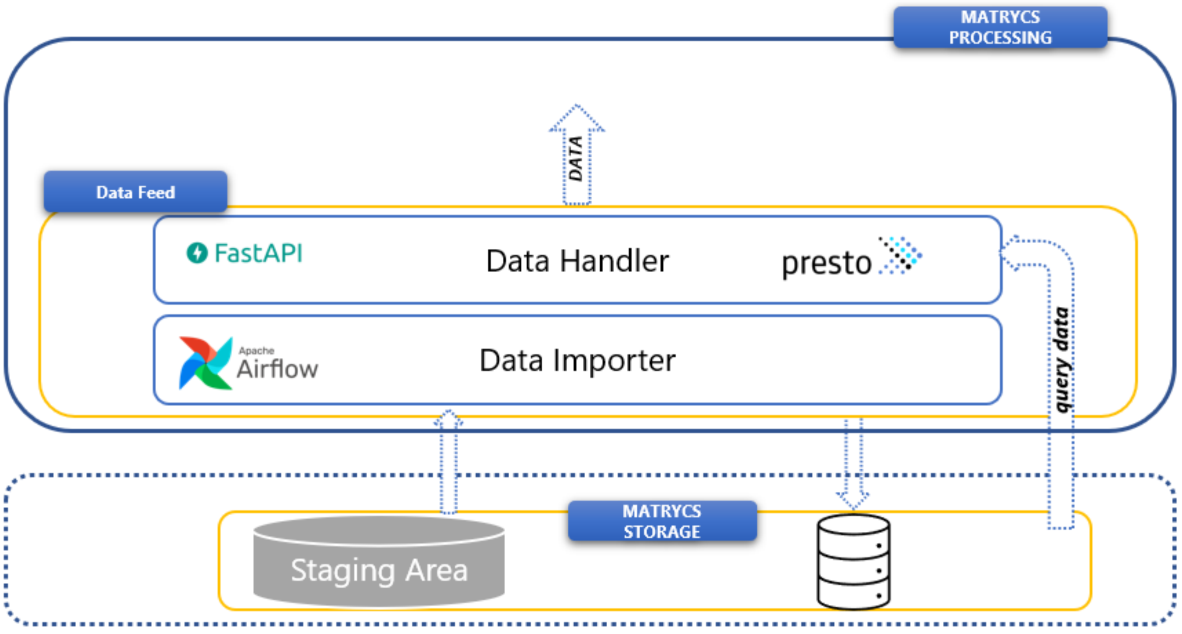


Figure 4: Data Feed Connection with MATRYCS-GOVERNANCE

The second connection point with MATRYCS-GOVERNANCE is the Serving Framework which receives streaming data from MATRYCS-GOVERNANCE and more specifically from the Data Streaming module to make predictions using trained ML/DL models from models shared storage. Figure 5 demonstrates the data connection between Serving Framework and MATRYCS-GOVERNANCE layer.

[10] APACHE Airflow, https://airflow.apache.org/

[11] MongoDB, https://www.mongodb.com/

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

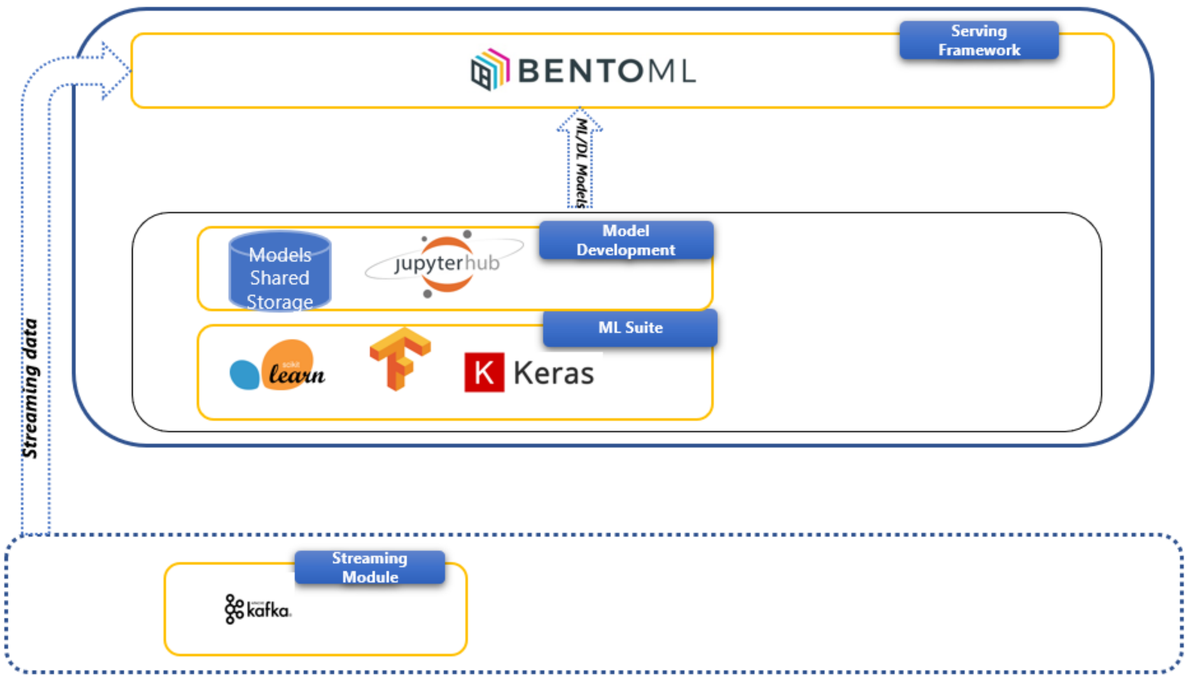


Figure 5: Serving Framework connection with MATRYCS-GOVERNANCE

2.5 Connection with MATRYCS-ANALYTICS Layer

IoT and Big Data technologies are becoming crucial components in the management of different processes in the construction industry. The cross-connection between data and sensor networks generates systems capable of creating intelligent buildings that reduce consumption and improve indoor comfort.

According to A. Daissaoui.[12], an intelligent system in the context of an intelligent building is composed by three levels:

- The infrastructure level of the input data: representing all the data sources generated by the connected objects in the building such as energy consumption, humidity level, indoor and outdoor temperature, CO_2, presence, energy consumption, system controls, etc.
- System infrastructure level: representing the core of the intelligent system since it allows the collection, processing, and merging and storage data. Thus, this allows the use of this data for knowledge extraction through data mining algorithms, automatic learning through artificial intelligence algorithms or simply offering reporting services.
- The level of services: representing the list of services offered by the system to different type of users, such as building managers, residents, and energy suppliers, etc.

In the MATRYCS project, different analytics building services have been designed to accommodate the demands of different pilots, and to be able to cover a very wide range of possible demands from different

[12] IoT and Big Data Analytics for Smart Buildings: A Survey, A. Daissaoui, A. Boulmakoul, L. Karim, A. Lbath, Procedia Computer Science – Elsevier, April 2020

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

types of users. Four main groups of services have been defined:

- MATRYCS – PERFORMANCE: It represents services able to manage all aspects related to energy performance and indoor condition of building. Specifically:
  - Energy Prediction
  - Building Automation Control
  - KPIs calculation
  - Technical Building Management
  - Optimization for network operations.
- MATRYCS – DESIGN: It focalizes in building infrastructure. The main service is the ECM - evaluation with the related technologies catalogue.
- MATRYCS – POLICY: It provides services for policy making and policy impact assessment. The tools are:
  - SECAPs decision-making support.
  - Energy Performance Certification harmonization and checking.
  - National and EU policy impacts assessment and support.
- MATRYCS – FUND: in which analytics for de-risking investments in energy efficiency will be developed.

Together with these groups, two other tools will be developed to cover fundamental aspects of building design and management. The two tools are Building Digital Twins and the geoclustering tool.

More details about each service will be provided in the deliverable D5.1 – *MATRYCS Analytics Building Services V1.0 (October 2021)*.

All tools will be stand-alone software. The possibility to be installed both on premises and on cloud is explored. A key point for the functioning of the services is the access to the data. The latter will be carried out in three ways:

- direct access to building database
- connection to specific frameworks where different ML algorithms can run generating data and analysis to be shown in a dedicated front-end
- using the reasoning engine

As far as the specific frameworks is concerned, Figure 6 demonstrates the schema about the connection of analytics services to data (MATRYCS-GOVERNANCE and MATRYCS-PROCESSING).

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

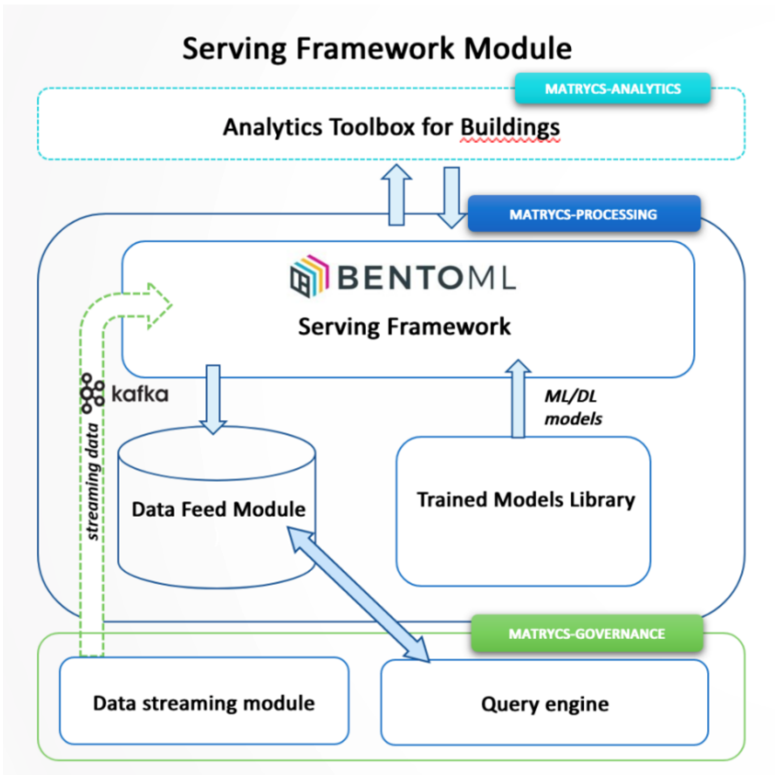


Figure 6: MATRYCS Analytics Connection with Serving Framework

Therefore, the data flows from the MATRYCS-GOVERNANCE to the MATRYCS-PROCESSING in two ways, using a query engine or directly with the data streaming module. MATRYCS services will use BENTOML[13] serving framework to manage all processes in which the use of a ML models is required. BENTOML is a flexible, high-performance framework for serving, managing, and deploying machine learning models. It supports multiple ML frameworks, including Tensorflow, PyTorch, Keras, XGBoost and more, it is Cloud native deployment with Docker, Kubernetes, AWS, Azure,etc. It has a high-performance online API serving and offline batch serving, and it has Web dashboards and APIs for model registry and deployment management.

Some of the services that will use BENTOML are the Energy Prediction and the Geoclustering tool.

Figure 7 shows the general approach. The analytic services will use the Reasoning engine to obtain the data needed to perform the analytics. This approach is useful for all services that need to have data and metadata processed using a specific data model (es. FIWARE or BRICK schema model) based on standard ontology. The Building DigitalTwin, KPIs calculation and Measurement and Verification to support ECM contracts services will use this schema to get data from pilots.

[13] BentoML, https://www.bentoml.ai/

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

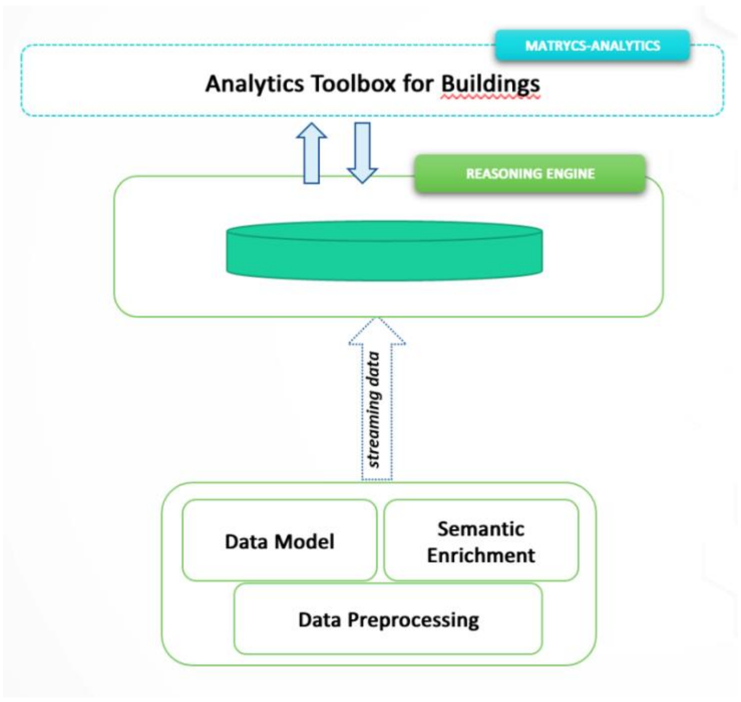


Figure 7: Connection of MATRYCS Analytics to the Reasoning Engine

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

3 Data Feed Module

The Data Feed Module is the component responsible for data transferring between MATRYCS-GOVERNANCE and MATRYCS-PROCESSING. More specifically, it consists of two sub-components: The Data Importer and the Data Handler.

The Data Importer is a data pipeline system that receives data from MATRYCS distributed storage. When the data are on boarded to the Data Importer, a series of basic data preparation steps are conducted over them such as preparation of data to be inserted on MATRYCS MongoDB and deletion of duplicate and null values. The Apache Airflow has been selected for being the basis of data pipelines management on Data Feed Module. It contains the following core components:

- **Web Server**: This is the UI of Airflow that can be used to get an overview of the overall health of different Directed Acyclic Graphs (DAG) and also in visualizing different states of each DAG[14].
- **Scheduler**: This is the most important part of Apache Airflow, as it orchestrates various DAGs taking care of their interdependencies.
- **Executor**: Executors are the components that actually execute tasks. The type of Executor used in production is the CeleryExecutor, based on Celery framework[15], for scaling the execution of tasks/DAGs across computational resources.
- **Meta-data database:** This database stores metadata about DAGs and Apache Airflow configuration details.

The data are inserted to MATRYCS-PROCESSING through Data Importer subcomponent and stored to MongoDB. The Data Handler is the REST service responsible for distributing these stored data across MATRYCS-PROCESSING components. This module leverages Python libraries such as Presto[16] Python Client, Pandas Framework for DataFrames[17] and FastAPI[18] for REST services. The REST APIs are used for enabling data selection, data aggregation, data grouping, dates handling, numerical scaling, categorical encoding (one-hot encoding, label encoding) and converting time series to supervised procedures. The Data Handler is connected with MATRYCS Query Engine (Presto), and all queries are transformed into SQL queries and all the MongoDB collections are transformed into tables in Presto.

In Table 1 and Table 2 some usage examples of these services are listed. The following query is used for receiving records with "timestamp", "value" fields from table "btc_tower" (LSP1) in the period between "2020-12-01 and 2020-12-02". The query that is executed on the backend is presented in Table 1.

[14] DAG, https://en.wikipedia.org/wiki/Directed_acyclic_graph

[15] Celery Framework, https://docs.celeryproject.org

[16] Presto, https://prestodb.io/

[17] Pandas, https://pandas.pydata.org/

[18] FastAPI, https://fastapi.tiangolo.com/

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

Table 1: Data Handler REST API call example (LSP1)

```
 POST /complex/select/query HTTP/1.1
Host: matrycs.epu.ntua.gr:8000
Content-Type: application/json
Content-Length: 158

{
   "table": "btc_tower",
   "columns": [ "timestamp", "value"],
   "where_column": "timestamp",
   "between_values": ["2020-12-01", "2020-12-02"]
}
```

The following query (Table 2) is used for receiving the fields “timestamp”, “produced” from the table “coopernico_solar_plants” (LSP5) where the solar plant name is “27 Adega Palmela” and the records are generated after the data “2020-06-25”.

Table 2: Data Handler REST API call example (LSP5)

```
 POST /complex/select/query HTTP/1.1
Host: matrycs.epu.ntua.gr:8000
Content-Type: application/json
Content-Length: 377

{
   "table": "coopernico_solar_plants",
   "columns": [ "timestamp", "produced"],
   "AND_": [
   {
      "where_symbol": ">",
      "where_column": "timestamp",
      "where_clause_term": "2020-06-25"
      },
   {
      "where_column": "solar_plant",
      "where_clause_term": "27 Adega Palmela"
   }],
   "order_by_column": "timestamp"
}
```

The MATRYCS project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement no.101000158

3.1 High-Level Architecture of Data Feed Module

Figure 8 presents the architecture of Data Feed Module and the data flow from MATRYCS storage along with the interconnection of the different subcomponents.

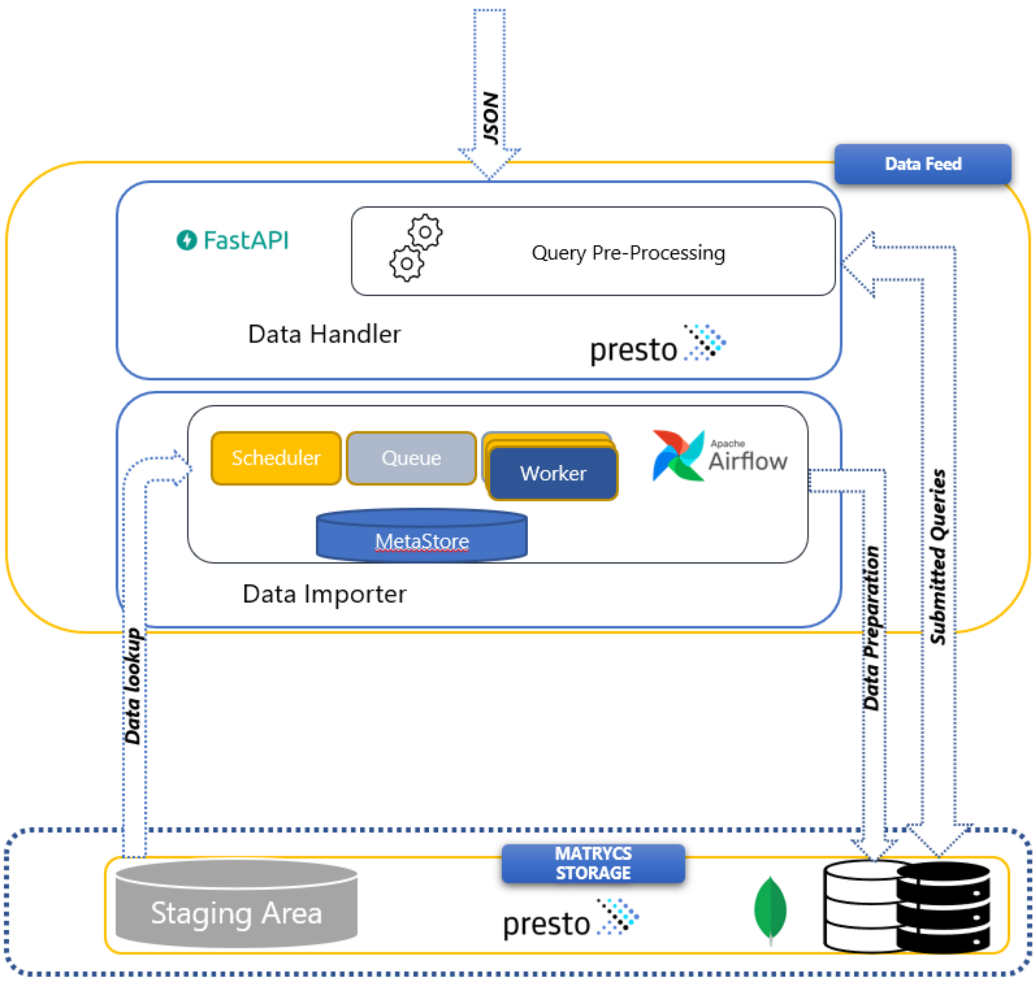


Figure 8: Data Feed Component Architecture

The Data Importer waits for incoming data on MATRYCS storage staging area. The staging area is a distributed file storage where data are placed after the procedures of MATRYCS-GOVERNANCE. The preparation pipelines are Python 3.7 code integrated on Airflow framework and they apply transformation steps, for example dropping duplicates, removing null values, and normalizing dates. These actions are scheduled and executed from Apache Airflow workers, and they are executed when new data are detected. At the end, the transformed data are inserted to MATRYCS Storage for later use.

The Data Handler is the collection of REST services responsible for distributing the data across MATRYCS-PROCESSING. These services receive JSON payloads which are processed from Query pre-processing class of the Data Handler for constructing the Presto-SQL query. These queries are sent to MATRYCS storage for getting the results back as a response. Furthermore, these APIs are used for data selection, data aggregation, data grouping, timeseries transformation and could be the input for multiple MATRYCS components such as Visualization Engine, Serving framework, etc.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

3.2 Migration from ScyllaDB to MongoDB

Since the first months of the project, Scylla DB is used for database. However and during MATRYCS-PROCESSING technology evaluation activities for the 1st technology release, it was observed that columnar databases and in particular ScyllaDB due to their nature cannot support nested data formats and updates on stored data, as ScyllaDB is a schema-depended database. For that reason, databases that are schema-less and support the storage of nested object were investigated. The outcome of that procedure was that the MongoDB database is the ideal solution for keeping transformed and prepared data for ML/DL training. Thus, MongoDB was selected instead of ScyllaDB because it is flexible for document schemas, easily scalable and optimized for querying and analytics. Furthermore, MongoDB is the main data store used in various FIWARE components, that would possibly be integrated with MATRYCS Framework future releases.

Currently, all the components (Visualization Engine, Model Development Module, Serving & Evaluation Framework) that communicate with Data Feed Module receive data from MATRYCS MongoDB instance. However, some components, due to their functionality, must have a direct connection with the database (MATRYCS Workbench). These components use ScyllaDB and have tested the connection with MongoDB. Until the 2nd technology release, all components will receive data from MongoDB.

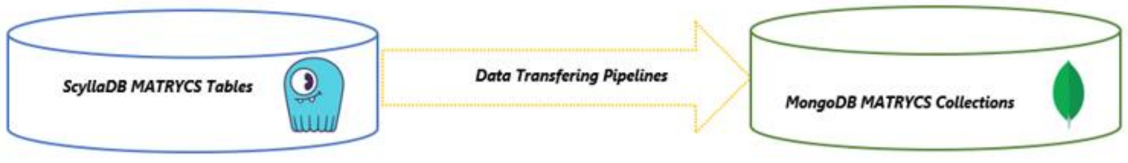


Figure 9: Migration from ScyllaDB to MongoDB

Data Migration was the main challenge to deal with, as it was necessary to transfer the existing data from ScyllaDB to MongoDB. Figure 9 depicts the data migration pipelines. That was accomplished using a series of Python scripts (Table 3) for receiving all the data from ScyllaDB tables and then batch insert them to MongoDB. After data migration the new version of the Data Feed has integrated Python functions for ensuring the connectivity and data exchange between MongoDB and the Data Feed REST services.

Table 3: Python Script for migrating data from ScyllaDB to MongoDB

```
from cassandra.cluster import Cluster
import pandas as pd

# ScyllaDB Connection
cluster = Cluster(['matrycs.epu.ntua.gr'])
session = cluster.connect('matrycs_transformed')

# Retrieve data from ScyllaDB
scylladb_btc_data = pd.DataFrame(session.execute('SELECT * FROM btc_data')).to_dict('records')

# MongoDB Connection
mongo_conn = pymongo.MongoClient('mongodb://${user}:${password}@matrycs.epu.ntua.gr:27017/')
db = mongo_conn['matrycs_transformed']
```

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

```
#Create collection named btc_data and bulk insert the btc data
collection = db['btc_data']
collection.insert_many(scylladb_btc_data)
```

3.3 Data Feed Model Deployment Approach

The Data Feed Module and its components are installed and configured using Docker[19] compose and bash scripts. The following script deploys the Data Importer and the Data Handler modules, and it is displayed below (Table 4).

Table 4: Data Feed Module Docker compose

```
  version: '3.2'
services:
  postgres:
    image: postgres:9.6
    container_name: postgres
    hostname: postgres
    restart: always
    environment:
      - POSTGRES_USER=airflow
      - POSTGRES_PASSWORD=airflow
      - POSTGRES_DB=airflow
    ports:
      - 5432:5432
    volumes:
      - pgdata:/var/lib/postgresql/data
    networks:
      - airflow
  redis:
    container_name: redis
    hostname: redis
    image: redis:5.0.5
    environment:
      REDIS_HOST: redis
      REDIS_PORT: 6379
    ports:
```

[19] Docker, https://www.docker.com/

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

```
    - 6379:6379
   networks:
    - airflow
  webserver:
   container_name: webserver
   hostname: webserver
   build:
    context: '..'
    dockerfile: config/Dockerfile
   env_file:
    - .env
   ports:
    - 8080:8080
   volumes:
    - ../dags:/opt/airflow/dags
    - ../PythonProcessors:/opt/airflow/PythonProcessors
    - ../utils.py:/opt/airflow/utils.py
    - ../MongoDBClient:/opt/airflow/MongoDBClient
    - ../data:/opt/airflow/data
    - ../settings.py:/opt/airflow/settings.py
    - ../models:/opt/airflow/models
   depends_on:
    - postgres
    - redis
    - initdb
   command: webserver
   healthcheck:
    test: [ "CMD-SHELL", "[ -f /opt/airflow/airflow-webserver.pid ]" ]
    interval: 30s
    timeout: 30s
    retries: 3
   networks:
    - airflow
  scheduler:
   container_name: scheduler
   hostname: scheduler
   build:
```

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

```
    context: '..'
    dockerfile: config/Dockerfile
  env_file:
    - .env
  volumes:
    - ../dags:/opt/airflow/dags
    - ../PythonProcessors:/opt/airflow/PythonProcessors
    - ../utils.py:/opt/airflow/utils.py
    - ../MongoDBClient:/opt/airflow/MongoDBClient
    - ../data:/opt/airflow/data
    - ../settings.py:/opt/airflow/settings.py
    - ../models:/opt/airflow/models
  command: scheduler
  depends_on:
    - postgres
    - initdb
    - webserver
  networks:
    - airflow
worker:
  container_name: worker
  hostname: worker
  build:
    context: '..'
    dockerfile: config/Dockerfile
  env_file:
    - .env
  volumes:
    - ../dags:/opt/airflow/dags
    - ../PythonProcessors:/opt/airflow/PythonProcessors
    - ../utils.py:/opt/airflow/utils.py
    - ../MongoDBClient:/opt/airflow/MongoDBClient
    - ../data:/opt/airflow/data
    - ../settings.py:/opt/airflow/settings.py
    - ../models:/opt/airflow/models
  command: celery worker
  depends_on:
```

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

```
      - scheduler
    networks:
      - airflow
```

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

4 Machine Learning Suite

The ML Suite is provided through the use of ML libraries such as numpy[20], scipy, scikit-learn, etc. Libraries from the realm of so called "AutoML" are also provided, such as AutoGluon[21], AutoSklearn[22], etc.

The ML Suite is automatically provisioned, using Infrastructure-as-code tools. The highlighted part of the following figure (Figure 10) visually demonstrates the dynamic role that a requirements.txt file plays at the time of provisioning, as triggered by an 'Admin'. Specifically, the Admin user can trigger the installation of new libraries to be exposed from Machine Learning Suite

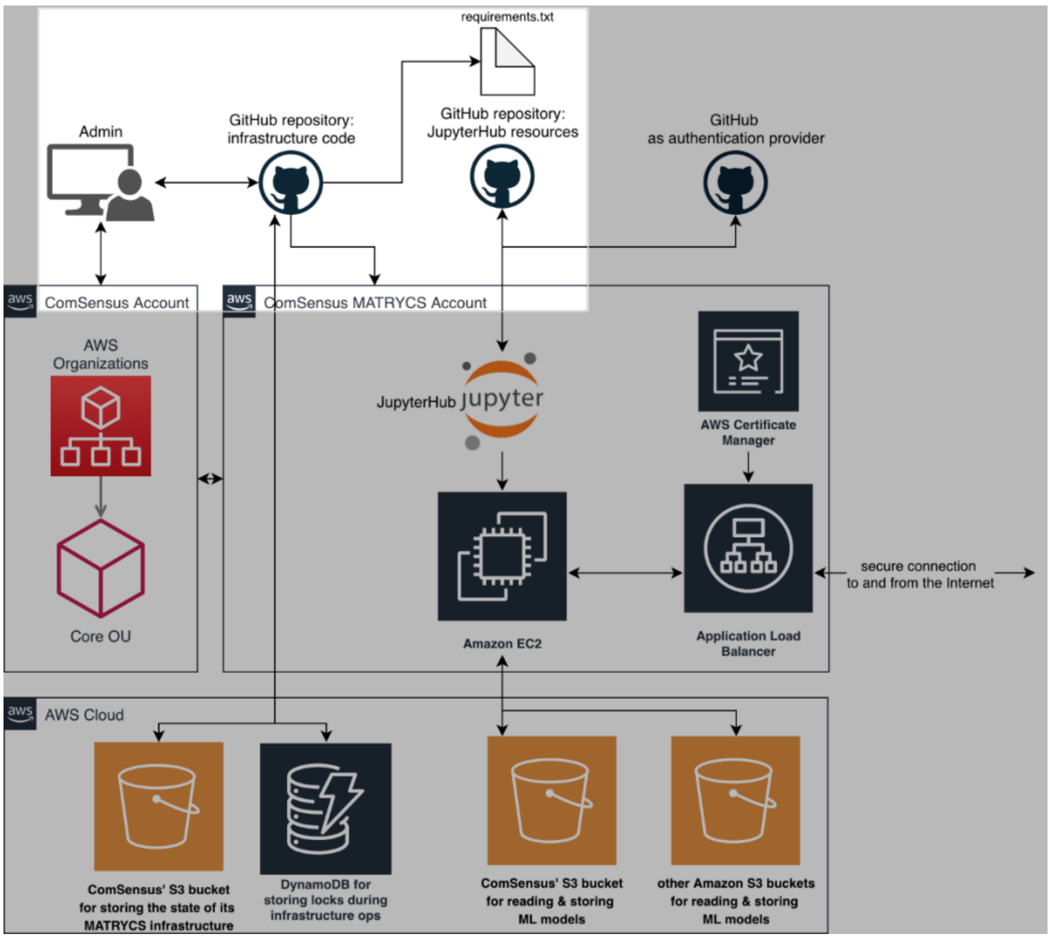

requirements.txt
Admin
GitHub repository: infrastructure code
GitHub repository: JupyterHub resources
GitHub as authentication provider
ComSensus Account
ComSensus MATRYCS Account
AWS Organizations
Core OU
JupyterHub jupyter
AWS Certificate Manager
Amazon EC2
Application Load Balancer
secure connection to and from the Internet
AWS Cloud
ComSensus' S3 bucket for storing the state of its MATRYCS infrastructure
DynamoDB for storing locks during infrastructure ops
ComSensus' S3 bucket for reading & storing ML models
other Amazon S3 buckets for reading & storing ML models

Figure 10: ML Suite Starting Phase

The requirements.txt file contains pinned versions of all the open-source libraries that together present the ML Suite, which is then available to every user of the Model Development Module. The entries of the requirements.txt file contain such lines, with the name of the library to the left of the equals sign, and its version to the right:

- auto-sklearn==0.12.6
- tensorflow==2.5.0
- torch==1.6.0

[20] Numpy, https://numpy.org/

[21] https://github.com/awslabs/autogluon

[22] https://automl.github.io/auto-sklearn/master/

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

The ML Suite can be enriched with additional libraries post-deployment by the Admin user – the newly installed libraries will be available to all users. Additionally, every user may install libraries if fitting in own local workspace by executing the following command in his Notebook:

- !pip install --user <library_name>

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

5 Model Development Module

The Model Development Module leverages the ML Suite module, as described in the previous section, and provides both a personal workspace to every ML/DL developer as well as shared, pre-defined pipelines for classification, regression and timeseries forecasting tasks. The following figure demonstrates the Model Development Module functionality.

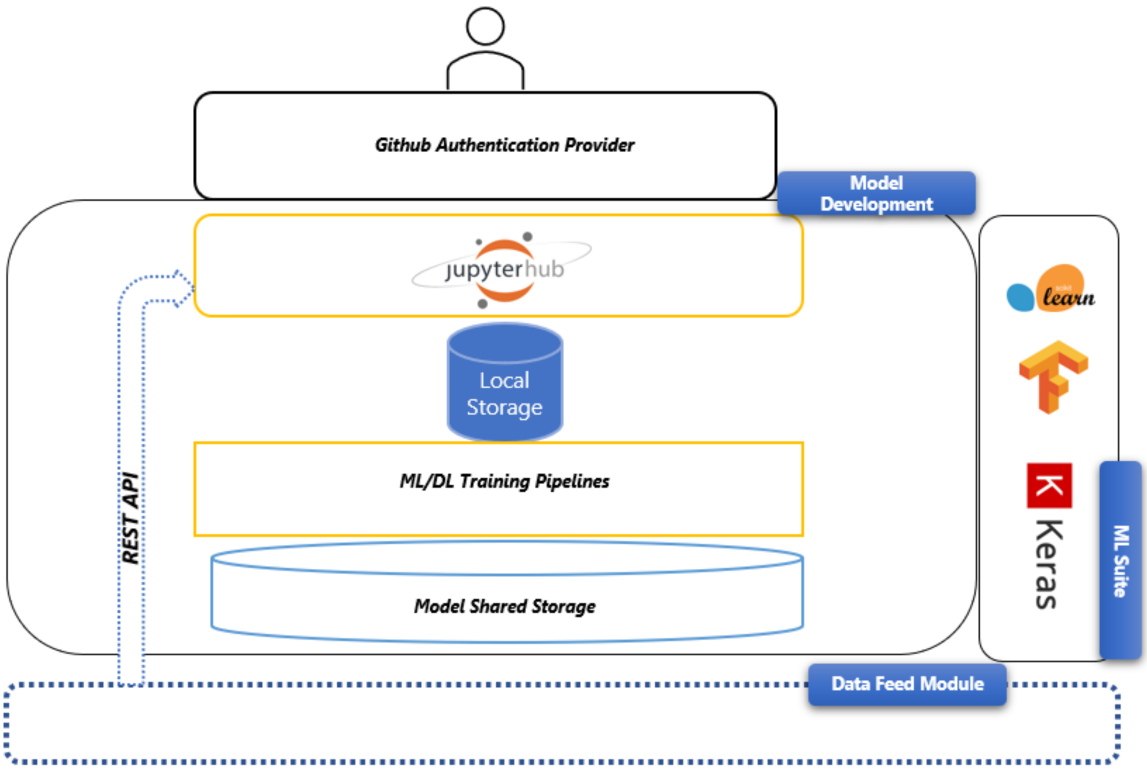


Figure 11: Model Development Module high level Architecture

As demonstrated in Figure 11, a ML/DL developer can interact with Model development module using GitHub Authentication provider. The GitHub Authentication provider will be replaced from End-to-End security framework on MATRYCS PROCESSING second technology release. After authentication the user can start training using processed data provided from Data Feed REST APIs. ML Suite provides the ML/DL libraries for developing ML/DL models. The code provided from end users triggers the ML/DL pipelines. The training metadata (e.g. metrics, data) are stored to Model Development local storage. The output of the training pipelines are trained models that are stored on Model Shared Storage.

Model development interacts with Data Feed module for receiving processed data through REST API calls, also interacts with ML Suite module for leveraging all the available ML/DL libraries and finally with the Evaluation and Serving Framework for evaluating and serving the models after training through Model Shared Storage.

Model Development Deployment on AWS

All the major cloud providers currently offer managed and proprietary services that could serve as the Model Development Module. Examples of such services are AWS SageMaker[23] and Google Cloud

[23] https://aws.amazon.com/sagemaker/

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

Datalab[24]. As both the Model Development Module and the Shared Models Storage should be kept as flexible as possible. It was decided to provide the Model Development Module using the JupyterHub[25] open-source project. Specifically, it is provided via The Littlest JupyterHub[26], which serves as a wrapper around JupyterHub and is meant for a single server deployment. Figure 12 demonstrates the workflow needed to be followed from Model Development Module for configuring AWS infrastructure

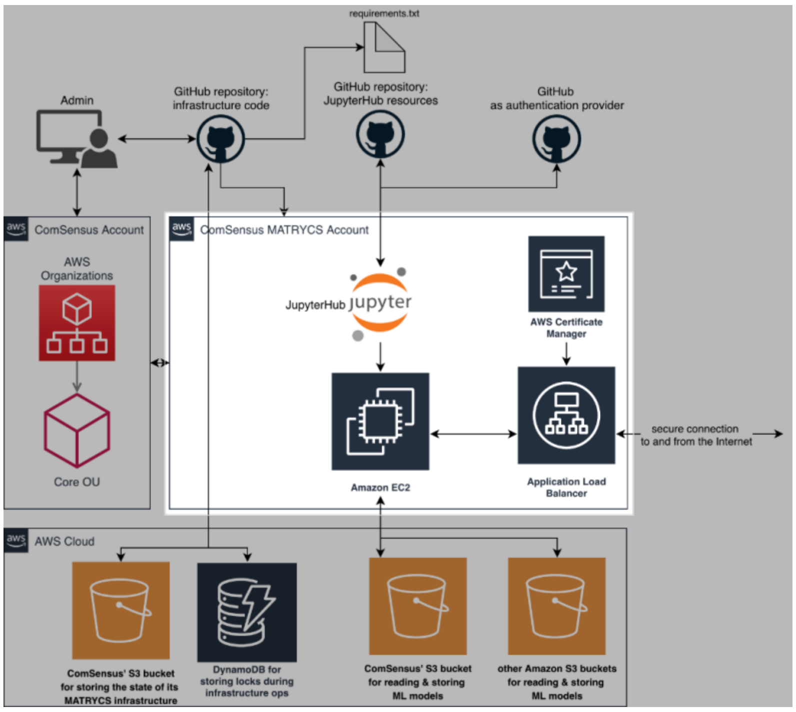

requirements.txt
Admin
GitHub repository: infrastructure code
GitHub repository: JupyterHub resources
GitHub as authentication provider
ComSensus Account
AWS Organizations
Core OU
ComSensus MATRYCS Account
JupyterHub jupyter
AWS Certificate Manager
Amazon EC2
Application Load Balancer
secure connection to and from the Internet
AWS Cloud
ComSensus' S3 bucket for storing the state of its MATRYCS infrastructure
DynamoDB for storing locks during infrastructure ops
ComSensus' S3 bucket for reading & storing ML models
other Amazon S3 buckets for reading & storing ML models

Figure 12: Separate AWS Account containing Module's components

The Model Development Module is deployed on an AWS virtual machine[27]. The workspace is also set up in a way so that every user receives a sample Jupyter Notebook and a guiding README on how to use the existing pipelines[28]. Via the same infrastructure code, the Shared Models Storage is also provisioned as an S3 bucket. During the provisioning, the virtual machine that hosts JupyterHub is set up with all the required read and write rights towards the Model Shared Storage (S3 bucket). Thus, the users of the Model Development Module will not interfere with external storages or authentication as the Model Development Module can upload the trained models to the Shared Models storage transparently. The whole infrastructure is done through the Terraform, an Infrastructure-as-code tool[29]. The provisioning is triggered by an admin user who controls the initiation of Models Shared storage. The provisioning is triggered with the following command (Table 5) from admin user:

[24] https://cloud.google.com/datalab/docs

[25] https://jupyter.org/hub

[26] https://tljh.jupyter.org/

[27] Available at https://matrycs.comsensus.eu

[28] https://matrycs.comsensus.eu/hub?next=%2Fuser-redirect%2Fgit-pull?repo%3Dhttps%253A%252F%252Fgithub.com%252FComSensus%252Fjupyterhub-resources%26branch%3Dmain%26urlpath%3Dlab%252Ftree%252Fjupyterhub-resources%252Fsample.ipynb%253Fautodecode

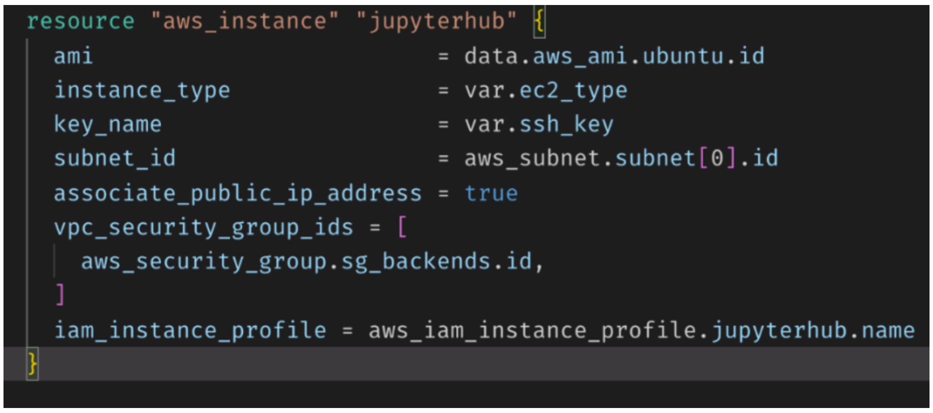
[29] https://www.terraform.io/

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

Table 5: Provisioning Command

terraform apply -var-file="production.tfvars"

An abbreviated, incomplete code to provision the server, annotated as "Amazon EC2" on the images, is the following (Figure 13):



```
resource "aws_instance" "jupyterhub" {
  ami                         = data.aws_ami.ubuntu.id
  instance_type               = var.ec2_type
  key_name                    = var.ssh_key
  subnet_id                   = aws_subnet.subnet[0].id
  associate_public_ip_address = true
  vpc_security_group_ids = [
    aws_security_group.sg_backends.id,
  ]
  iam_instance_profile = aws_iam_instance_profile.jupyterhub.name
}
```

Figure 13: Terraform provisioning code of EC2

The secure connection to the Model Development Module at https://matrycs.comsensus.eu is served using an AWS load balancer together with a certificate provided by AWS Certificate Manager[30]. The authentication is done with GitHub serving as an OAuth2 provider. Once the user is authenticated, the initial JupyterHub screen appears (Figure 14). The workspace can also be set up so that every user receives a sample Jupyter Notebook along with a guiding README on how to use the provided pipelines. All libraries from the ML Suite are seamlessly provided to the user.

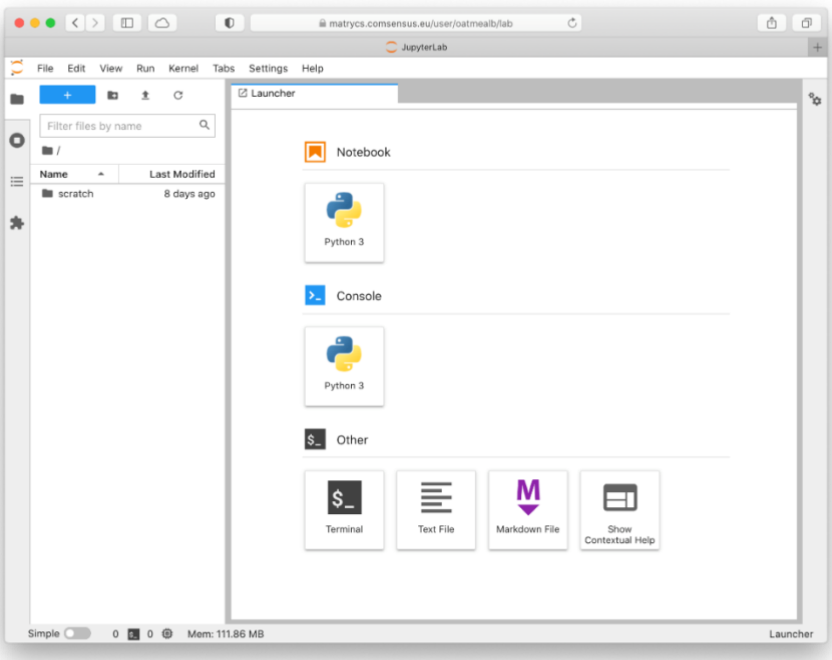
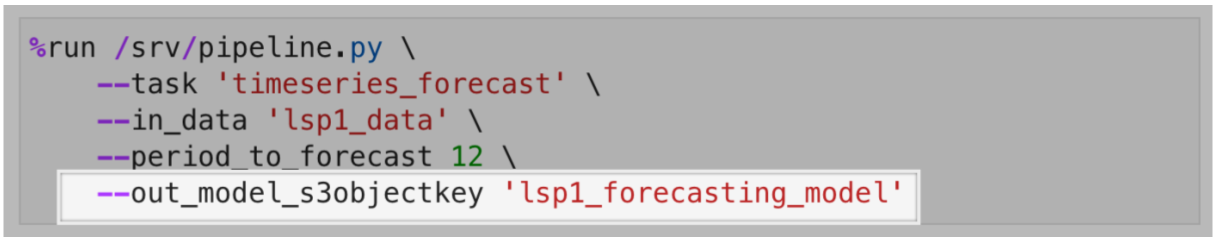


Figure 14: Model Development Module JupyterHub

[30] https://aws.amazon.com/acm/

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

The ML/DL developer that uses the Model development module can receive data processed from Data Feed Module or by using the MATRYCS Query Engine to receive data from other sources. The output of training pipeline are the trained models where they are uploaded directly to Model Shared Storage or stored locally to Model Development Module workspace. An example usage of the provided pipeline is shown in the following snippet, with a highlighted usage of the Shared Models Storage (Figure 15).



```
%run /srv/pipeline.py \
    --task 'timeseries_forecast' \
    --in_data 'lsp1_data' \
    --period_to_forecast 12 \
    --out_model_s3objectkey 'lsp1_forecasting_model'
```

Figure 15: Pipeline option that enables usage of the Shared Models Storage

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

6 Model Serving Module

The MATRYCS project foresees that a certain number of Machine Learning models will be developed during the Model Development phase. The ML models will be the building blocks of the MATRYCS Analytics layer and will be developed to be capable of fulfilling the needs expressed by the pilots during the preliminary project phase in which the different Pilot's requirements are collected. The development of ML models and their automatic learning is certainly a critical part of the project and requires steps to be taken before they can be considered reliable and valid and used for their intended purpose.

In particular, two parties take part in these activities: the developer of the models and the end-user who, in order to use them in his own services, may invoke them through a serving system or incorporate them directly into his own code. This differentiation turns a light on what similar to the concept of DevOps is defined as MLOps [31]and that is the practice adopted for the MATRYCS project. MLOps comes into play to unify the two processes of developing ML models (Dev) and putting them into operation (Ops) that is the goal of this serving module. In particular, the Model Serving phase is the process that is downstream of a series of preparatory steps ranging from data acquisition, model training and model evaluation with subsequent refinement cycles. In summary, the following image (Figure 16) shows this process being managed manually, starting with the data manager, continuing with the model developers and concluding with the serving of the models.

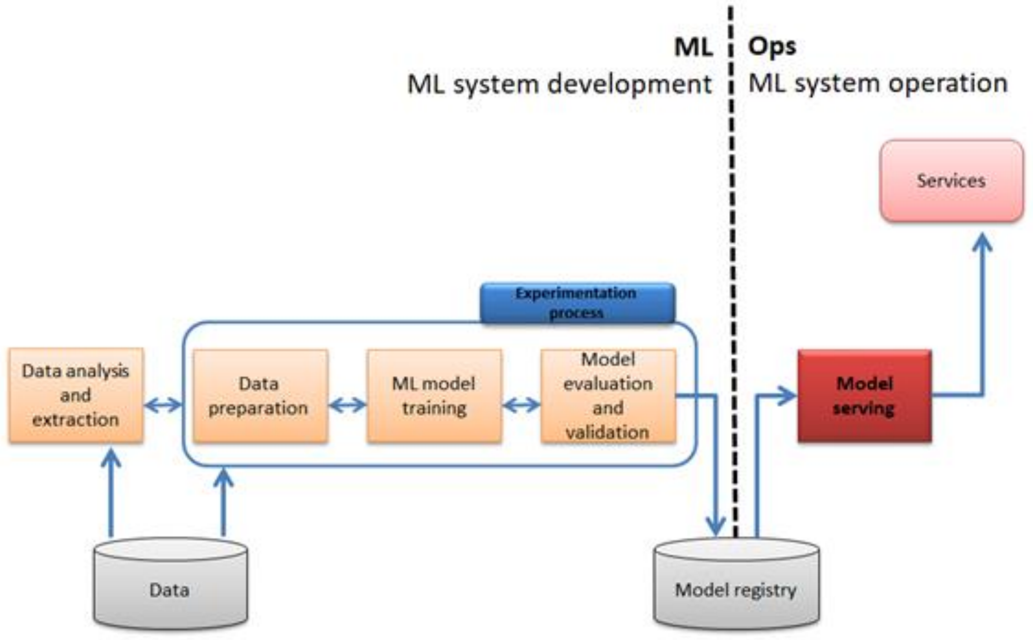


Figure 16: MLOps process – Model serving

It is clear that the development of models and their use is potentially the responsibility of different working groups/project partners and that they do not necessarily use the same software to develop and run the ML models and do not always have the possibility of using development and execution environments that are compatible with the technologies and libraries used by one group rather than the other. To accomplish this task, the concept of "model serving" came into play. Its purpose is to serve models that have been previously trained and evaluated, and which is capable of executing ML models

[31] MLOps, https://en.wikipedia.org/wiki/MLOps

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

by having a runtime environment capable of executing as many different software libraries as required, thus allowing an almost agnostic interaction between the developer of the models and the consumer.

Within the MATRYCS project, a series of analytical requirements have been defined for specific use cases, to be solved using specific services by certain user groups. Although at this point of the project not all these ML models have yet been developed and the libraries necessary for their development are not known, it can be assumed that libraries and technologies among the most commonly used by the scientific community will be used and that the Serving Framework must be a multi-framework environment able to support models developed with technologies/libraries such as:

- Scikit-learn32,
- Pytorch 33
- Pytorch Lightning 34
- Tensorflow 35
- FastAI 36v1
- Keras 37(Tensorflow 2.0 as the backend)
- fastText 38
- CoreML 39
- Spacy 40
- Transformers 41

At the same time, the intention is to provide those who will consume the ML models with as much information as possible about the available ML models through a WEB user interface showing the relevant catalogue of available models and serving these through high performance APIs for consumers without the need for lower-level web server development. To this purpose, a study was made on the most popular ML model serving techniques and their existing frameworks and how much they can be potentially customised to meet the needs of the MATRYCS project. Frameworks such as TensorFlow Serving[42], ML-FLOW[43], AWS SageMaker[44], BentoML[45] and so on were analysed and at the end of this study, the BentoML was our choice for different aspects, both technical and operational that fits with the project needs.

[32] Scikit-Learn, https://scikit-learn.org/stable/
[33] Pytorch, https://pytorch.org/
[34] Pythorch Lightening, https://www.pytorchlightning.ai/
[35] Tensorflow, https://www.tensorflow.org/
[36] FastAI, https://www.fast.ai/
[37] Keras, https://keras.io/
[38] Fasttext, https://fasttext.cc/
[39] CoreML, https://developer.apple.com/documentation/coreml
[40] Spacy, https://spacy.io/
[41] Transformers, https://huggingface.co/transformers/
[42] TensorFlow Serving, https://www.tensorflow.org/tfx/tutorials/serving/rest_simple
[43] MI-FLOW, https://mlflow.org/
[44] AWS SageMaker,https://aws.amazon.com/it/sagemaker/
[45] BentoML, https://www.bentoml.ai/

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

6.1 Model Serving Architecture

The Model Serving is placed within the project conceptual architecture in the MATRYCS-PROCESSING Layer and represents the contact point between this layer and the underlying MATRYCS-GOVERNANCE Layer and the MATRYCS-ANALYTICS Layer. It will serve the ML models saved under the Trained Models library and already trained by the ML developers and subsequently evaluated and refined thanks to the Model Evaluation & Validation module.

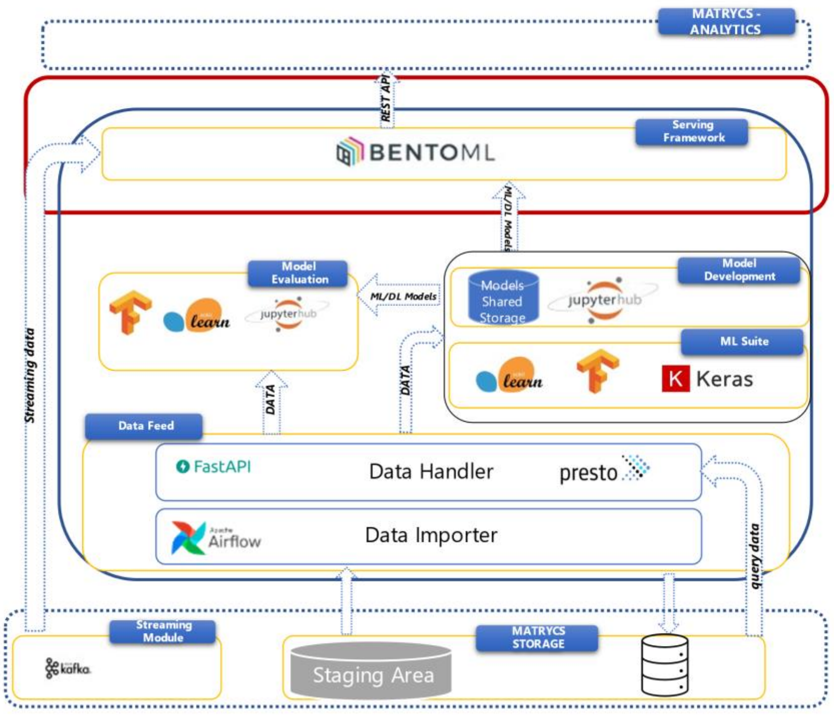


Figure 17: MATRYCS conceptual architecture - Serving Module

As mentioned earlier in the conceptual architecture (Figure 17), the service module is the tool capable of providing the results of ML model processing to the MATRYCS-ANALYTICS Layer, thus feeding the various services of the related Toolbox. The next figure (Figure 18) demonstrates the architecture of the Serving Framework and the interaction between the different modules and layers. The data will be read in two different ways: either through the Data Feed Module, which has the task to prepare the data provided by the Query engine for the different needs of the ML models, or through the Data Streaming Module (see deliverable D3.1 - *MATRYCS-GOVERNANCE (1st Technology release)* – Section 3.4) which sends the data through Kafka streaming under specific topics).

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

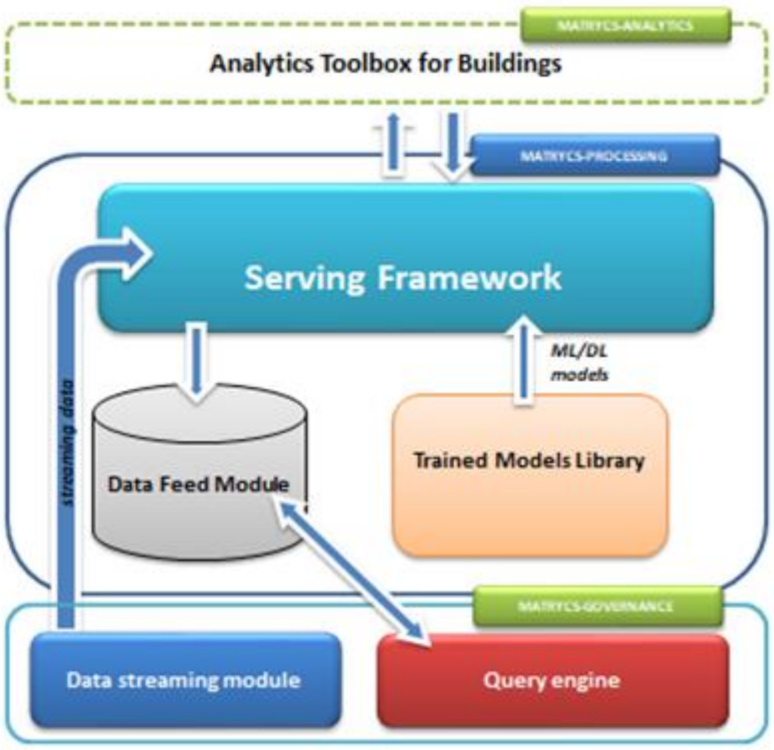


Figure 18: Serving module conceptual architecture

6.2 Model Serving technological components

Serving framework core component

The technological choice took into account several characteristics that are intended to be given to the Serving framework. Ease to use even for users who are not necessarily ML experts, the presence of APIs that can be invoked directly by the services to be developed or through a graphical interface, the presence of an easily accessible repository for loading and managing models via specific APIs, whether external to the project or developed directly in the project, simplicity in the extension of the ML libraries adding new ones, and being ready for deployment in cloud platforms, which must be as free as possible from architectural constraints. For all the above, the BentoML framework has been adopted as the engine of the MATRYCS serving framework.

Below are the main characteristics of the Serving framework solution:

- The Serving Framework provides a Web UI to send prediction requests, and related APIs for model registry and deployment management. Yatai[46], model management component of BentoML, is available via Web UI, CLI, Python API.
- Production-ready online API serving and adaptive offline batch serving.

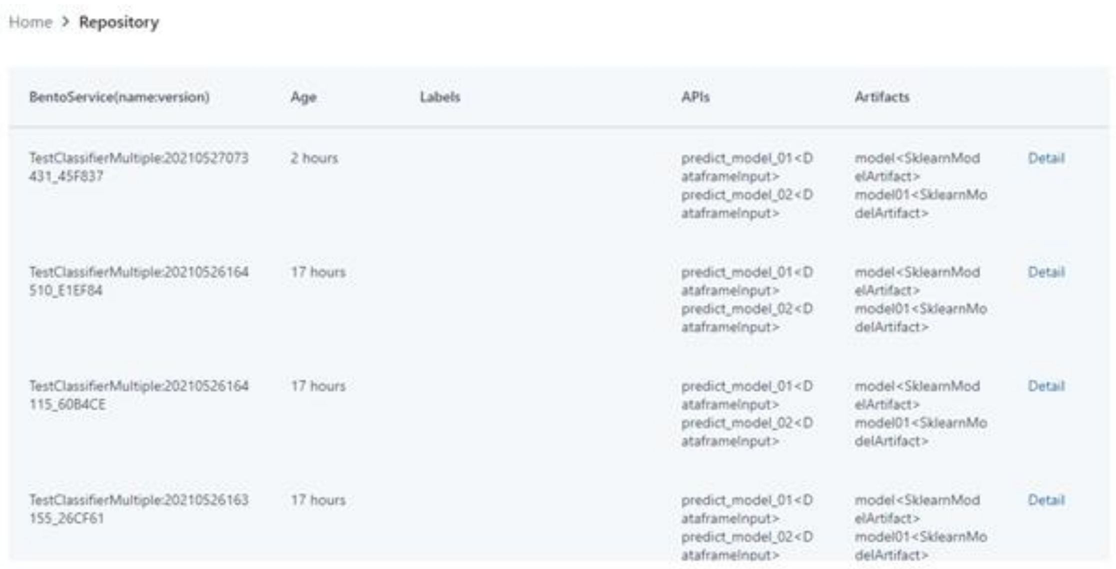
[46] Yatai, https://docs.bentoml.org/en/latest/api/yatai_client.html

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

- Various ML frameworks, including Tensorflow[47], PyTorch[48], Keras[49], XGBoost[50], FastText[51], CoreML[52], Spacy[53] etc. are supported.
- Supports deployment with open-source platforms like Docker[54], Kubernetes[55], Kubeflow[56], etc. as well as one-click deployment with AWS Lambda[57], Azure Functions [58]and others, and manual cloud deployment with AWS ECS[59], Google Cloud Run[60], Heroku[61].
- Possibility to serve sets of multiple models.

Serving framework repository

At this stage of the project a specific repository has been configured to create a staging area to save the ML models after the training and evaluation process and is accessible from the external. This repository will be replaced with the official project ML models library. At the same time, the Serving Module repository has been appropriately configured and is accessible from the outside to upload ready-to-use models. A direct access has been tested and configured to allow the delivery of models directly from the evaluation framework environment at the end of the evaluation and refinement models' process.


Home > Repository

BentoService(name:version)	Age	Labels	APIs	Artifacts	
TestClassifierMultiple:20210527073431_45F837	2 hours		predict_model_01<DataframeInput> predict_model_02<DataframeInput>	model<SklearnModelArtifact> model01<SklearnModelArtifact>	Detail
TestClassifierMultiple:20210526164510_E1EF84	17 hours		predict_model_01<DataframeInput> predict_model_02<DataframeInput>	model<SklearnModelArtifact> model01<SklearnModelArtifact>	Detail
TestClassifierMultiple:20210526164115_60B4CE	17 hours		predict_model_01<DataframeInput> predict_model_02<DataframeInput>	model<SklearnModelArtifact> model01<SklearnModelArtifact>	Detail
TestClassifierMultiple:20210526163155_26CF61	17 hours		predict_model_01<DataframeInput> predict_model_02<DataframeInput>	model<SklearnModelArtifact> model01<SklearnModelArtifact>	Detail

Figure 19: Serving framework ML models repository (Yatai Web UI)

[47] Tensorflow, https://www.tensorflow.org/
[48] Pytorch, https://pytorch.org/
[49] Keras, https://keras.io/
[50] XGBoost, https://xgboost.readthedocs.io/en/latest/
[51] FastText, https://fasttext.cc/
[52] CoreML, https://developer.apple.com/documentation/coreml
[53] Spacy, https://spacy.io/
[54] Docker, https://www.docker.com/
[55] Kubernetes, https://kubernetes.io/
[56] Kubeflow, https://www.kubeflow.org/
[57] AWS Lambda, https://en.wikipedia.org/wiki/AWS_Lambda
[58] Azure functions, https://azure.microsoft.com/en-us/services/functions/
[59] AWS ECS, https://aws.amazon.com/ecs/?nc1=h_ls&whats-new-cards.sort-by=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc&ecs-blogs.sort-by=item.additionalFields.createdDate&ecs-blogs.sort-order=desc
[60] Google cloud run, https://cloud.google.com/run
[61] Heroku, https://www.heroku.com/

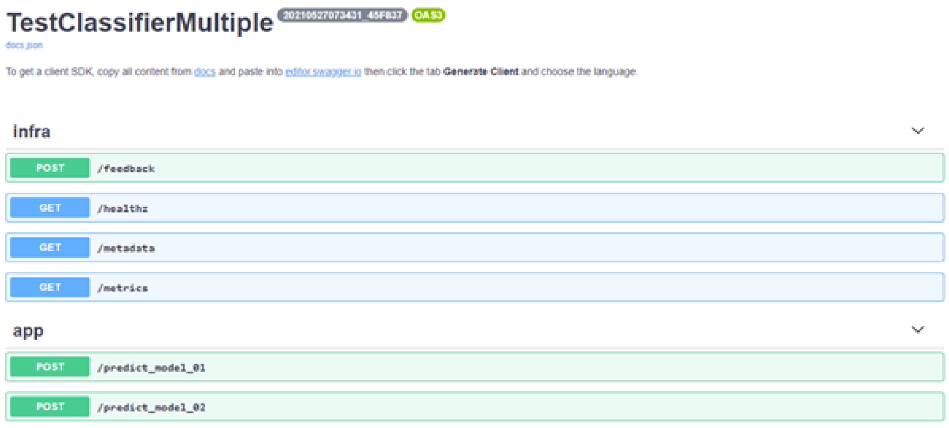
The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

Serving framework ML libraries

An analysis of the main used ML libraries has been done and for the 1st technology release, a subset of these ML libraries has been installed and tested into the serving framework environment to permit the serving of the first test ML models. New libraries will be installed and tested based on the different models that will be developed during the project ML development phase.

Serving framework APIs

Basic APIs have been created to test simple already available ML models and are available both via batch commands and via a Swagger user interface for a fast access without code writing.


TestClassifierMultiple 20210527073431_45F837 OAS3

docs.json

To get a client SDK, copy all content from docs and paste into editor.swagger.io then click the tab **Generate Client** and choose the language.

infra

POST /feedback

GET /healthz

GET /metadata

GET /metrics

app

POST /predict_model_01

POST /predict_model_02

Figure 20: Swagger REST API

6.3 Serving framework deployment approach

In the context of MATRYCS project, Serving Framework services have been deployed using Docker technologies. As it is previously stated, there is a number of possible deployment ways. The choice of the type of installation fell on Docker for its versatility, given the possibility of deploying it on different environments, also in light of the fact that at this stage of the project the final infrastructure has not been identified, for its ability to manage failure and recovery problems and the ease of creating snapshots of the environment in use to be redeployed in any other environments, because it allows the simultaneous use of different versions of the same application during the periods of testing of new functions without interrupting what is already in place. Docker [62]enables fast, easy, and transferable application development, and deploying the applications in Docker containers. Docker Compose [63]is a tool for creating and running multiple Docker containers with Docker applications.

The process of utilizing Docker Compose is the following:

- If needed, create a Dockerfile in which the application's environment and additional configuration of the base image are set.
- Then, services based on the applications are defined in the YAML file, so they run together, preferably in the same network.

[62] Docker, https://www.docker.com/
[63] Docker compose, https://docs.docker.com/compose/

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

- In the end, the "`docker-compose up -d`" command is issued to start running the applications in Docker Containers.

Table 6: Example of Docker Compose YAML file for BentoML services

```
version: "3.1"
services:
 bentoml:
  image: bentoml/model-server:latest
  command: bash -c "pip install wheel && pip install pandas && pip install sklearn && bentoml serve-guncorn /root/bentoml --enable swagger --yatai-url 217.172.12.158:8891"
  ports:
   - "8882:3000"
   - "8883:50051"
   - "8884:5000"
   - "8885:33513"
   - "8886:5000"
  volumes:
   - /root/bentoml/repository/TestClassifierMultiple/20210512100649_D1B6F7:/root/bentoml
  networks:
   - yatai
 yatai-service:
  restart: unless-stopped
  image: bentoml/yatai-service:latest
  command: " --db-url=postgresql://postgres@yatai-db:5432/bentomldb --repo-base-url=/bentoml/repository"
  volumes:
   - /root/bentoml:/bentoml
  environment:
   - BENTOML_HOME=/bentoml
   - REPOSITORY_BASE_URL=/bentoml/repository
  depends_on:
   - yatai-db
  environment:
   - REPO_BASE_URL=/bentoml/repository
  ports:
   - "8890:3000"
   - "8891:50051"
```

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

```
      networks:
        - yatai
    yatai-db:
      image: library/postgres:9.6.20
      environment:
        - LC_ALL=C.UTF-8
        - POSTGRES_DB=bentomldb
        - POSTGRES_USER=postgres
        - POSTGRES_PASSWORD=
        - POSTGRES_HOST_AUTH_METHOD=trust
      volumes:
         - /dati/bentoml/testdb:/var/lib/postgresql/data
      ports:
        - "8999:5432"
      networks:
        - yatai
  networks:
    yatai: {}
```

From the Docker Compose YAML file, it can be seen that:

- All images used are latest images from the official BentoML's Docker HUB site.
- Volumes are created that enable persisting multiple ML models that will be served, the whole repository of all ML models, and the data from the PostgreSQL database.
- All necessary ports are opened and mapped, which facilitates seamless connection between the containers as well as exposing APIs to the users.
- Environment variables necessary for running the containers are set.

For serving ML models, a Docker container is created from the image `bentoml/model-server:latest`. The commands that run in the container enable the production version of the tool, and Swagger API, which facilitates easy queries about metadata, and what is most important, calls to the prediction service of each deployed model.

The official Docker images for BentoML can be found on the official docker hub[64]. It should be pointed out that, on the host machine, using the BentoML library in Python environment, the bundles of ML models are created and prepared to be served in one of the proposed ways. The MATRYCS Serving framework is created with the aim to be easily integrated at the end of the ML model training workflow.

Besides using Docker technologies, there is another way to deploy BentoML to serve multiple ML models at once, which is also leveraged in our environment. In this additional way, BentoML is called from the Python environment to create a production-ready service that will have the same capabilities as

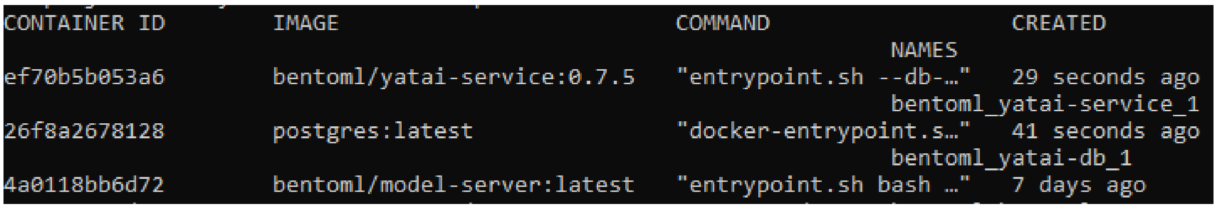
[64] BentoML docker hub official page https://hub.docker.com/u/bentoml

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

dockerized version. Both approaches to deployment have been analysed and assessed.

Another important tool that has been identified, installed and configured for the Serving Framework environment is a BentoML[65]'s model management tool called Yatai[66]. It is used to store all ML models to be served and run as a deployment automation component also. Yatai provides WEB UI, CLI and Python API to the bundles of ML models that are created within the Serving Framework. Two databases can be used, SQLite[67] or PostgreSQL[68]. For the production mode, PostgreSQL database deployed in the Docker container is an advisable solution. This approach is taken in the MATRYCS project too.

Figure 21 demonstrates the running Docker containers of a couple of Serving Framework tools, namely Yatai and PostgreSQL components.



```
CONTAINER ID        IMAGE                          COMMAND                   CREATED
                                                                    NAMES
ef70b5b053a6        bentoml/yatai-service:0.7.5    "entrypoint.sh --db-…"    29 seconds ago
                                                                    bentoml_yatai-service_1
26f8a2678128        postgres:latest                "docker-entrypoint.s…"    41 seconds ago
                                                                    bentoml_yatai-db_1
4a0118bb6d72        bentoml/model-server:latest    "entrypoint.sh bash …"    7 days ago
```

Figure 21: Docker running containers

[65] BentoML, https://www.bentoml.ai/
[66] Yatai, https://docs.bentoml.org/en/latest/api/yatai_client.html
[67] SQLite, https://www.sqlite.org/index.html
[68] PostgreSQL, https://www.postgresql.org/

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

7 Evaluation Framework

The MATRYCS project envisages that during the Model Development phase a certain number of ML models are developed able to satisfy the needs expressed by the end-users as defined from the use cases developed in the first months of the project and constitute the building blocks of the upper layers of MATRYCS ANALYTICS Layer. These ML models after the development and training need a process of evaluation and refinement through appropriate techniques that determine for example the accuracy, performance, and error level. before the models can be served.

To define the cycle described above, two different aspects make up this activity: the development of models and their testing and the final use of the models for example through specific functions and services. This differentiation sheds light on what, similarly to the concept of DevOps, is called MLOps and which is the practice adopted for the MATRYCS project. MLOps comes into play to unify the two processes of ML model development (Dev) which is the objective of this service module and commissioning (Ops) which is better specified in the section dedicated to the Model Serving Module. In particular, the model testing phase is the process that lies halfway between model development and training and the subsequent serving of the models. In summary, Figure 22 shows this process starting with the data manager, moving to the model developers and their evaluation, and concluding with the model serving.

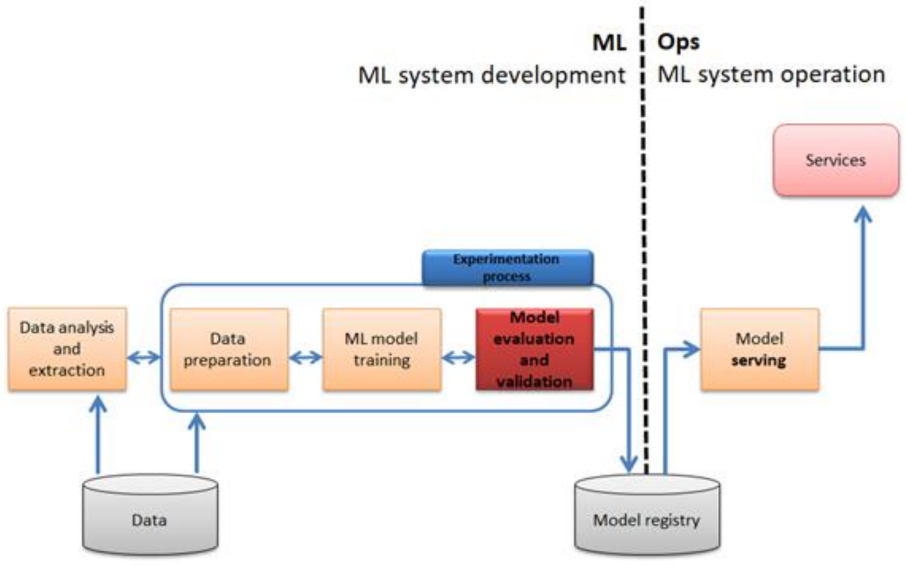


Figure 22: MLOps process – Model evaluation

Within the MATRYCS project, a series of analytical requirements have been defined for specific use cases, to be solved using specific services by certain user groups, such as EPC calculations rather than the calculation of expected production from photovoltaic panels rather than energy efficiency mechanisms, these services need certain ML models to provide data for their calculations that need to be evaluated and refined before to be served via the Serving framework module and used inside specific services. During this first phase of the project, it was decided to make available common tools for model developers that could facilitate the evaluation process of the developed models after their training phase. In particular, the main features identified are:

- A common repository where to save the developed models and keep track of the different versions created during the finishing phase.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

- A common access point to the different data sets of the project whether they come from assets of the different pilots or from sources outside the project.
- A common set of tools for testing and evaluating models and refining them if necessary.
- A common set of tools to keep track of the logs related to the processing results and related metrics that can also be easily analysed through a graphical interface.
- An environment capable of providing developers with the most common ML libraries and that can be easily updated with new ones if needed.

7.1 Evaluation framework Architecture

The evaluation framework is placed into the project conceptual architecture within the MATRYCS-PROCESSING Layer (Figure 23) and is part of the Model Development phase. It will evaluate the ML models already trained by the ML developers to be ready to be served to the upper MATRYCS-Processing Layer and used by the different services that will be developed within the MATRYCS project. The evaluation framework will use the data coming via the Data Feed module.

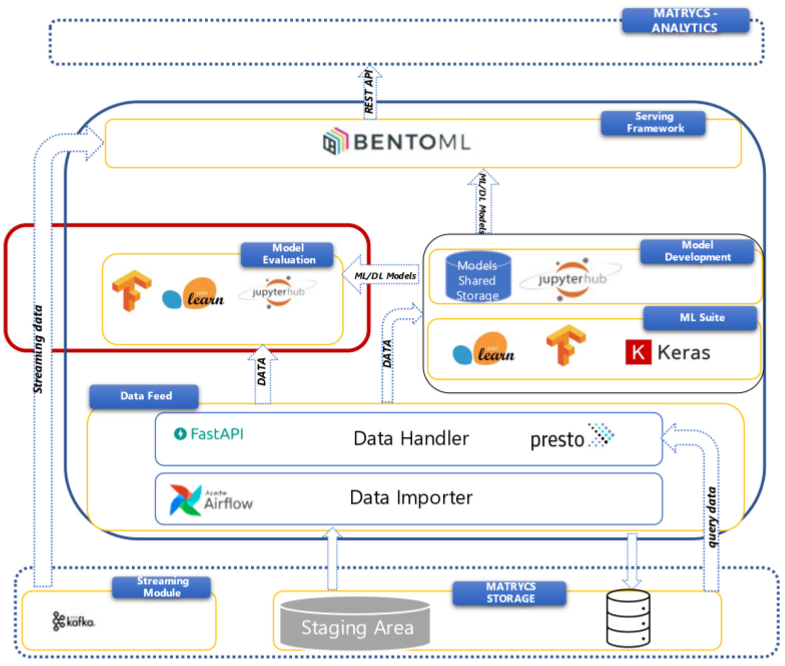


Figure 23: MATRYCS conceptual architecture - Evaluation framework

Figure 24 demonstrates the architecture of the Evaluation Framework and the interaction between the different modules and layers.

Model Evaluation Module provides the already configured environment and relative tools for the ML models developer and tester. It provides a dedicated repository to store the models and related versioning. This module is connected to the Data Feed Module using the instruments provided by the module that retrieve data by the Query Engine module.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

The Model Training Library will share the already trained model sending these directly to the Model Evaluation Module shared repository.

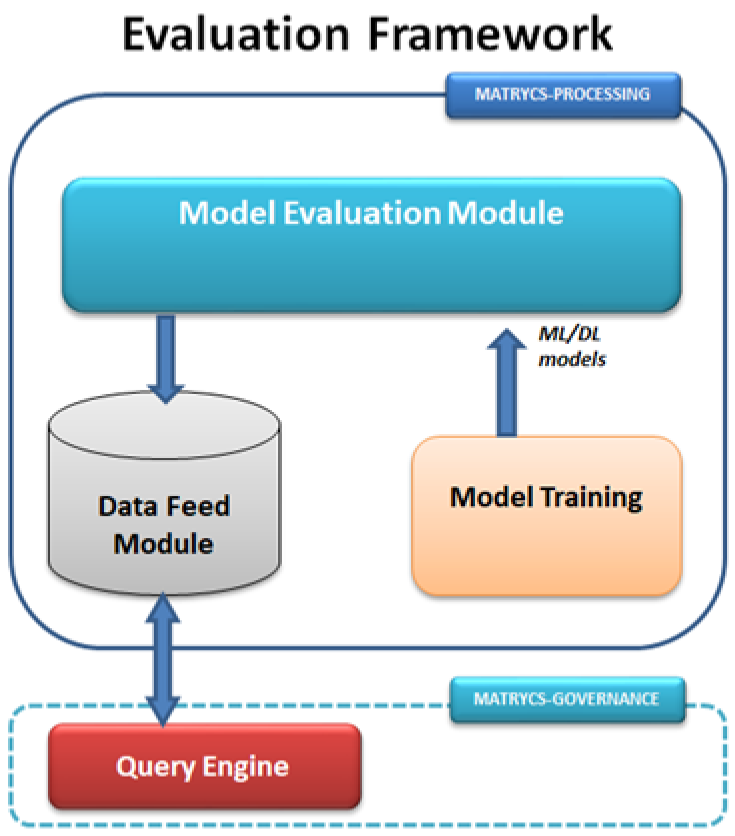


Figure 24: Evaluation framework architecture

7.2 Evaluation framework technological components

Evaluation framework core component

The technology choice for this first phase of the project considered several features that are intended to give the assessment framework to provide a ready-to-use environment that can be extended. At this stage of the project, the common tools to test and evaluate the ML models are provided thanks to the installation of the Anaconda Data Science Platform[69] that contains a set of tools able to provide to the data engineers and scientist the instrument for their development and analysis.

More precisely, the following tools have been identified, appropriately installed, and configured for the user needs:

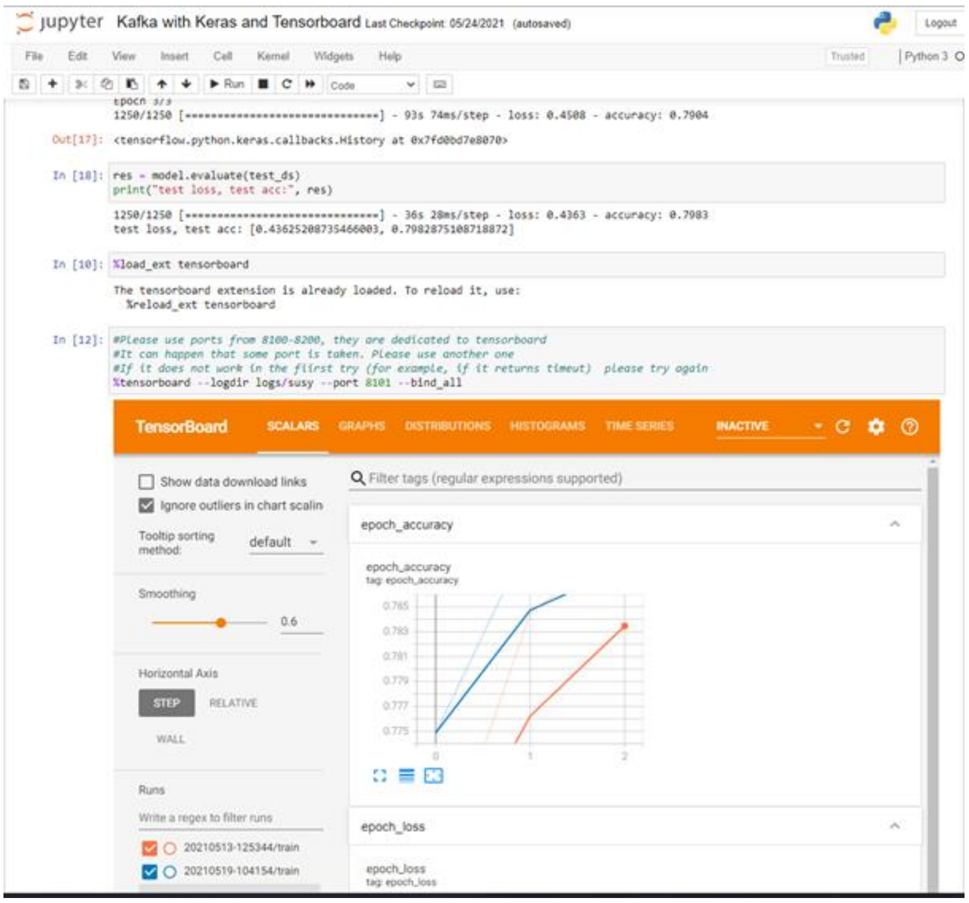
- Anaconda[70] environment with Jupyter Notebook[71]

[69] https://www.anaconda.com/
[70] Anaconda, https://www.anaconda.com/
[71] Jupyter, https://jupyter.org/

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

- Specific libraries that enable straightforward usage of trained ML models and related evaluation, such as Numpy[72], Pandas[73], TensorFlow[74], Keras[75], TensorBoard[76], Scikit-learn[77], Pytorch[78], Matplotlib[79], Seaborn[80], etc.
- Different Python libraries that enable connection to other modules and APIs such as Kafka Stream[81], Databases[82], Cloogy[83] API, etc.

Below, Figure 25 presents the aforementioned environment:


jupyter Kafka with Keras and Tensorboard Last Checkpoint: 05/24/2021 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
TensorBoard
SCALARS
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
TIME SERIES
INACTIVE
Filter tags (regular expressions supported)
epoch_accuracy
tag: epoch_accuracy
epoch_loss
tag: epoch_loss
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs

```
Epoch 3/3
1250/1250 [==============================] - 93s 74ms/step - loss: 0.4508 - accuracy: 0.7904
Out[17]: <tensorflow.python.keras.callbacks.History at 0x7fd0bd7e8070>

In [18]: res = model.evaluate(test_ds)
         print("test loss, test acc:", res)

1250/1250 [==============================] - 36s 28ms/step - loss: 0.4363 - accuracy: 0.7983
test loss, test acc: [0.43625208735466003, 0.7982875108718872]

In [10]: %load_ext tensorboard

The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard

In [12]: #Please use ports from 8100-8200, they are dedicated to tensorboard
         #It can happen that some port is taken. Please use another one
         #If it does not work in the fiirst try (for example, if it returns timeut)  please try again
         %tensorboard --logdir logs/susy --port 8101 --bind_all
```

☐ Show data download links
☑ Ignore outliers in chart scalin
Tooltip sorting method: default
Smoothing: 0.6
☑ 20210513-125344/train
☑ 20210519-104154/train

Figure 25: Jupyter Notebook with TensorFlow, Keras, TensorBoard

An analysis of the main used ML libraries has been done and for the purposes of the 1st technology release, a subset of these ML libraries has been installed and tested into the evaluation framework environment to permit the evaluation of the first test ML models. New libraries will be installed and

[72] Numpy, https://numpy.org/
[73] Pandas, https://pandas.pydata.org/
[74] Tensorflow, https://www.tensorflow.org/
[75] Keras, https://keras.io/
[76] TensorBoard, https://www.tensorflow.org/tensorboard
[77] Scikit-Learn, https://scikit-learn.org/stable/
[78] Pytorch, https://pytorch.org/
[79] MatplotLib, https://matplotlib.org/
[80] Seaborn, https://seaborn.pydata.org/
[81] Kafka Stream, https://docs.confluent.io/platform/current/streams/index.html
[82] Databases, https://pypi.org/project/databases/
[83] Cloogy, https://pypi.org/project/cloogy/

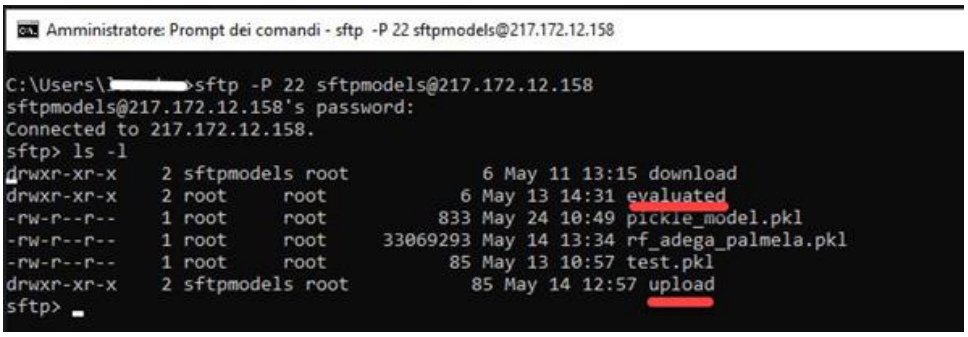
The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

tested based on the different models that will be developed during the project ML development phase.

Evaluation framework Common Repository

A specific repository has been configured to create a staging area to save the ML models coming from the ML trained library. These models will be evaluated and saved to be ready for the Serving phase via the official project model library.

To easy sharing and transferring ML models between different modules in the MATRYCS project, a SFTP server is configured on-premises. Dedicated users are created. Folders for uploading and downloading are set, and credentials are shared among partners.



```
Amministratore: Prompt dei comandi - sftp -P 22 sftpmodels@217.172.12.158

C:\Users\[redacted]>sftp -P 22 sftpmodels@217.172.12.158
sftpmodels@217.172.12.158's password:
Connected to 217.172.12.158.
sftp> ls -l
drwxr-xr-x    2 sftpmodels root            6 May 11 13:15 download
drwxr-xr-x    2 root       root            6 May 13 14:31 evaluated
-rw-r--r--    1 root       root          833 May 24 10:49 pickle_model.pkl
-rw-r--r--    1 root       root     33069293 May 14 13:34 rf_adega_palmela.pkl
-rw-r--r--    1 root       root           85 May 13 10:57 test.pkl
drwxr-xr-x    2 sftpmodels root           85 May 14 12:57 upload
sftp>
```

Figure 26: ML models staging area

7.3 Evaluation framework deployment approach

There are several ways in which the environment for ML models evaluation can be deployed. Docker is one of the most obvious choices, but because of effortless interaction with other modules of the MATRYCS project, the use of a traditional approach was decided. Anaconda package manager and all necessary libraries have been installed on Centos 8 operating system. The instructions for the range of Anaconda package manager installations are available at the official online documentation[84].

Steps for Anaconda installation include:

- Prerequisites are having `sudo` privileges, enough memory, processor power, and storage on the VMs, and access to the terminal window.
- Latest Anaconda Version can be downloaded with `curl` tool and saved in the appropriate folder.
- Run the Anaconda Installer script using `bash`. Go through configuration steps as installation progresses.
- Set and load the path to the `conda` command.
- Verify the installation.
- To run the Jupyter package, issue the command: `jupyter notebook --notebook-dir=/opt/notebooks --ip='0.0.0.0' --port=8888 --no-browser --allow root &.` Note that all notebooks, files, and data for Evaluation framework will be saved in the folder `/opt/notebooks`.

[84] Anaconda online documentation https://docs.anaconda.com/anaconda/install

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

- Keep the token received upon the start of Jupyter Notebook, as this token is access token that needs to be shared to everyone that need access.

The chosen deployment approach is easily transferable because all the Jupyter notebooks, data, and ML models are saved in folders that are regularly backed up and effortlessly portable.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

8 MATRYCS-PROCESSING Integration on M11

The scope of this section is to demonstrate the MATRYCS–PROCESSING components and processes through the flow of two LSPs and in particular LSP1 (BTC) and LSP5 (COOPERNICO) datasets. The code set for the MATRYCS PROCESSING 1st technology release could be found in this link https://github.com/Matrycs. The repository is private now because the code contains information such as usernames and passwords but in the 2nd technology release will be public and confidential information will be removed.

8.1 Connection with MATRYCS-GOVERNANCE Layer

The Data Importer detects LSP1 and LSP5 datasets on MATRYCS staging area, which are processed from the MATRYCS-GOVERNANCE layer. After the detection, these files are inserted on MATRYCS's MongoDB through Data Feed Module, and in this way the aforementioned data are checked into the MATRYCS-PROCESSING layer. Data Feed Module REST Services are responsible for exposing the LSP data to MATRYCS-PROCESSING components. In this layer of the architecture all MATRYCS-PROCESSING components receive data stored in the MongoDB.

8.2 Data Feed Module

As mentioned on Section 3.1, the Data Feed module consists of two submodules: the Data Importer and the Data Handler. In this section the usage of Data Feed components over LSP1 (BTC) and LSP5 (COOPERNICO) datasets is demonstrated. The LSP1 dataset contains hourly electricity consumption data from BTC tower and the LSP5 dataset contains production data from six COOPERNICO solar plants.

8.2.1 LSP1

The Data Importer searches for files on BTC directory and, by leveraging the Airflow File[85] and SFTP[86] sensors, it detects new files on BTC distributed directory. These files constitute the output of MATRYCS-GOVERNANCE, on MATRYCS distributed file storage. Consequently, these files are loaded and queued for processing (date normalization and numerical scaling).

After performing base processing, the data are stored to MATRYCS MongoDB. The actions carried out are:

- dates handling,
- scaling of numerical values,
- dropping null values,
- removing duplicates

[85]AirflowFileSensor, https://airflow.apache.org/docs/apacheairflow/1.10.11/_modules/airflow/contrib/sensors/file_sensor.html
[86]AirflowSFTPSensor, http://airflow.apache.org/docs/apacheairflow/1.10.12/_modules/airflow/contrib/sensors/sftp_sensor.html

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

Figure 27 showcases the set of base processing activities undertaken by the Data Importer for LSP1.

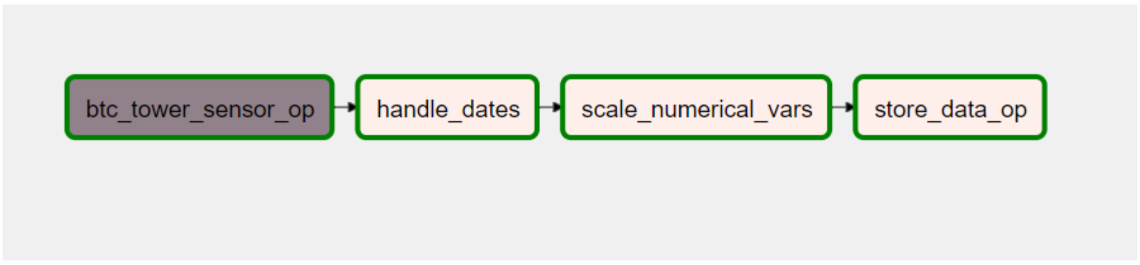


Figure 27: Data Importer procedures for BTC Tower of LSP1

The Data Handler is responsible for distributing the data across the MATRYCS-PROCESSING layer, via REST API calls. Specifically, it is an extra layer which applies the necessary transformations over the data that are stored in MATRYCS's MongoDB.

These extra transformations are:

- Data grouping
- Data aggregations (MIN, MAX, COUNT, AVG, SUM)
- Time series preparation
- Categorical encoding (label encoding, one hot encoding)
- Moving average smoothing

Examples of Data Handler's REST API calls for LSP1

The REST API calls below demonstrate the usage of the Data Handler for transforming the BTC data.

The following (Table 7) REST API call is used for receiving the summary of production value per hour using the Data Handler's *group query* to find the sum of value field on hour groups.

Table 7: GroupBy query REST API call for LSP1

```
POST /complex/group/query HTTP/1.1
Host: matrycs.epu.ntua.gr:8000
Content-Type: application/json
{
"aggregation_metric": "SUM",
"aggregation_column": "value",
"grouping_columns": ["hour"],
"table": "btc_tower",
"aggregation_metric_alias": "total"
}
```

The following (Table 8) REST API is used for enabling extra transformation over stored BTC tower data. More specifically the "value" field is normalized using *min-max scaling*. The fields "unit_of_measure", "energy_source" and "interval" are categorical variables. By using the transformation actions, which are a feature of the Data handler, they are *encoded* and at the end these data are *indexed* using the "timestamp" field. Below is the HTTP request used.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

Table 8: Feature selection REST API call for LSP1 data

```
POST /feature/selection HTTP/1.1

Host: matrycs.epu.ntua.gr:8000

Content-Type: application/json

{

   "table": "btc_tower",

   "all": "true",

   "columns_to_normalize": ["value"],

   "columns_to_encode": ["unit_of_measure", "energy_source", "interval"],

   "date_column": "timestamp"

}
```

8.2.2 LSP5

For LSP5, the Data Importer looks for files on COOPERNICO directory and the Airflow sensors detect the new files for processing using Pandas DataFrames. The following processing steps are for:

- handling null values,
- data normalization,
- scaling numerical values and
- dropping duplicates.

By leveraging the parallelism of Apache Airflow CeleryExecutor, all these procedures are applied on parallel for all solar plants data detected on MATRYCS distributed file storage. At the end, the processed data are stored to MATRYCS MongoDB. The following figure (Figure 28) demonstrates the processing tasks for 6 COOPERNICO solar plants detected on COOPERNICO distributed directory.

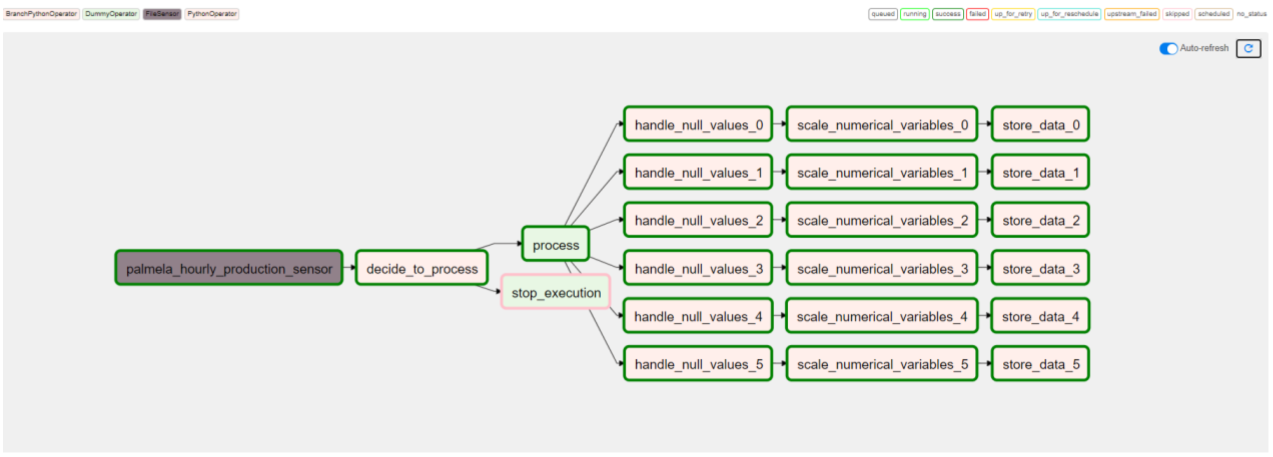


Figure 28: Data Importer procedures for LSP5 Solar plants

Once the MATRYCS MongoDB is populated with the transformed LSP5 solar plants' data, the Data Handler can be used for applying extra steps of transformation such as feature selection, data aggregation, data grouping, categorical encoding, numerical scaling, timeseries transformations and moving averages.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

Examples of Data Handler's REST API calls for LSP5

The following REST API call (Table 9) is used for selecting *average production value* from solar plant "27 Adega Palmela" where timestamp is higher than "2020-06-25". More specifically it uses the endpoint for submitting complex queries to Data Handler, as follows (Table 9):

Table 9: Selection query REST API call for LSP5

```
POST /complex/select/query HTTP/1.1
Host: matrycs.epu.ntua.gr:8000
Content-Type: application/json
{
   "table": "coopernico_solar_plants",
   "aggregation_metric": "AVG",
   "aggregation_column": "produced",
   "AND_": [
   {
      "where_symbol": ">",
      "where_column" : "timestamp",
      "where_clause_term": "2020-06-25"
      },
   {
      "where_column" : "solar_plant",
      "where_clause_term": "27 Adega Palmela"
   }]
```

The following REST API call is used for calculating the *average production per month per year* for the solar plant "27 Adega Palmela" and for dates after the "2020-06-25". The endpoint for submitting complex group queries to MATRYCS MongoDB follows (Table 10).

Table 10: GroupBy query REST API call for LSP5

```
POST /complex/group/query HTTP/1.1
Host: matrycs.epu.ntua.gr:8000
Content-Type: application/json
{
"aggregation_metric": "AVG",
"aggregation_column": "produced",
"grouping_columns": ["month", "year"],
"table": "coopernico_solar_plants",
"aggregation_metric_alias": "avg_produced",
"AND_": [
{
```

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

```
            "where_symbol": ">",
            "where_column" : "timestamp",
            "where_clause_term": "2020-06-25"
            },
        {
            "where_column" : "solar_plant",
            "where_clause_term": "27 Adega Palmela"
        }]
        }
```

The feature selection REST service is used for enabling transformations over stored data. These transformations are controlled through REST service input and by tuning specific payload parameters. The following REST API call is used for *normalizing numerical variables* ("produced", "specific" and "avoided_co2"), indexing the data using the timestamp field and *calculating the moving average* for the "produced" field and finally to prepare the final dataset for timeseries forecasting using the field "lag" as equal to seven (7). This means the seven previous production values to be used for the production value prediction of the next hour. An example of the REST API call for LSP5 follows (Table 11):

Table 11: Feature selection REST API call for LSP5

```
        POST /feature/selection HTTP/1.1
        Host: matrycs.epu.ntua.gr:8000
        Content-Type: application/json
        {
            "table": "coopernico_solar_plants",
            "all": "true",
            "columns_to_normalize": ["produced", "specific", "avoided_co2"],
            "date_column": "timestamp",
            "window": 7,
            "columns_to_move": ["produced"],
                "AND_": [
            {
                "where_symbol": ">",
                "where_column" : "timestamp",
                "where_clause_term": "2020-06-25"
                },
            {
                "where_column" : "solar_plant",
                "where_clause_term": "27 Adega Palmela"
            }],
            "lag": 7,
```

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

```
            "feature_columns": ["produced"]
        }
```

8.3 Model Development / ML Suite / Models Shared Storage

The Model Development module is connected to the Data Feed module by consuming its sub-components' REST API calls for receiving processed data from the Data Feed. One of the Model Development Module's components is a virtual machine as an AWS EC2 instance which is running a JupyterHub service, an open source and self-hosted service. Once logged in through a GitHub account, the user can open a Jupyter Notebook and load the data from the Data Feed Module.

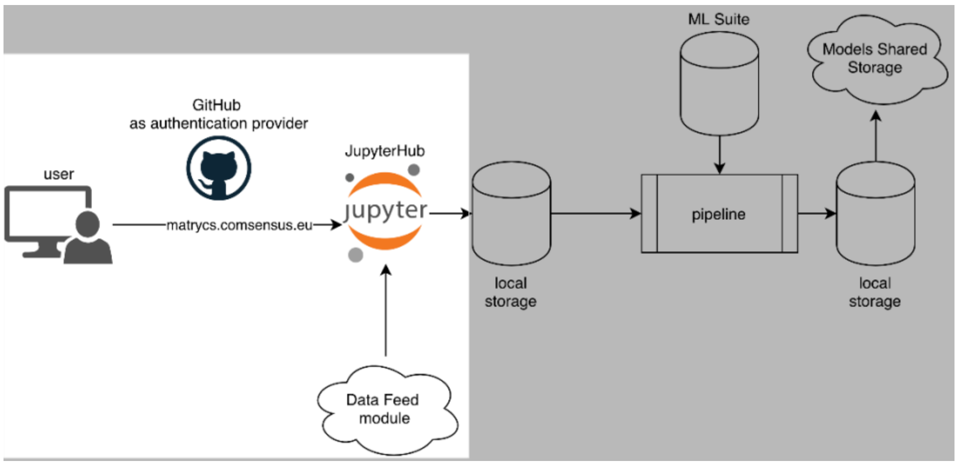


Figure 29: User's initial interaction with Module

In the example following (Table 12), the LSP1 data from the Data Feed Module is loaded into a Pandas DataFrame.

Table 12: Loading LSP1 data from the Data Feed Module

```
DATA_HANDLER_URL = 'http://matrycs.epu.ntua.gr:8000/feature/selection'

feature_selection_payload = {

            "table": "btc_tower",

            "all": "true",

            "columns_to_normalize": ["value"],

            "date_column": "timestamp",

            "lag": 5,

            "feature_columns": ["value"]

      }

 response=requests.post(response=requests.post (url=DATA_HANDLER_URL,
headers=headers,data=json.dumps(features_payload_))
```

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

The data is then transformed into a NumPy array of a nx1 shape, which is the expected input of the timeseries forecasting pipeline. This is shown in Table 13:

Table 13: Data Transformation for the provided pipeline

```
data = (df.loc['value'].to_numpy().reshape((-1,1)).astype(float))
```

After receiving the data from Data Handler's REST Services, the pipeline for training ML/DL models initiates by leveraging ML Suite's libraries. Metadata from the training module are stored to Model Development local storage and the output models are stored in Models Shared Storage for evaluation and serving. The following figure demonstrates Model development module pipeline flow (Figure 30).

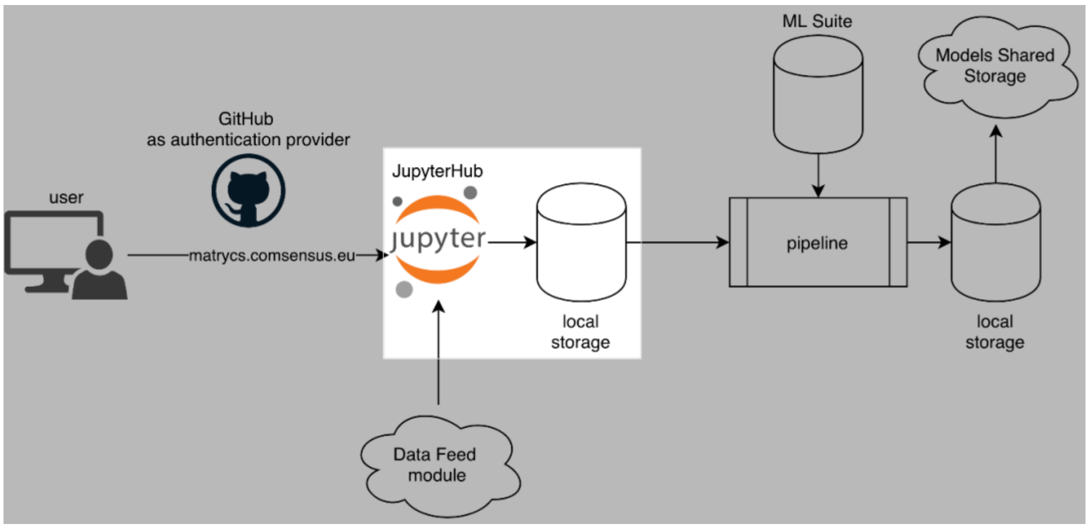


Figure 30: Handing off the data to the local storage

LSP1 data are used for training ML models for energy forecasting, in order to initiate the LSP1 training flow it is required to execute the following command (Table 14). After training the LSP1 forecasting model is stored to Models Shared storage having name "LSP1_forecasting_model".

Table 14: Running the pipeline for LSP1 timeseries analysis

```
%run /srv/pipeline.py –task 'timeseries_forecast' –in_data 'data' –period_to_forecast 12 –out_model 'LSP1_forecasting_model'
```

The following table (Table 15) demonstrates the metadata produced (training and evaluation loss) after LSP1 training.

Table 15: LSP1 training metadata

```
grad_step = 002400, tr_loss = 0.001086, te_loss = 0.040966
grad_step = 002430, tr_loss = 0.001044, te_loss = 0.037749
grad_step = 002460, tr_loss = 0.001010, te_loss = 0.037749
grad_step = 002490, tr_loss = 0.000980, te_loss = 0.037749
Model saved to S3 at comsensus-jupyterhub/lsp1_forecasting_model
```

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

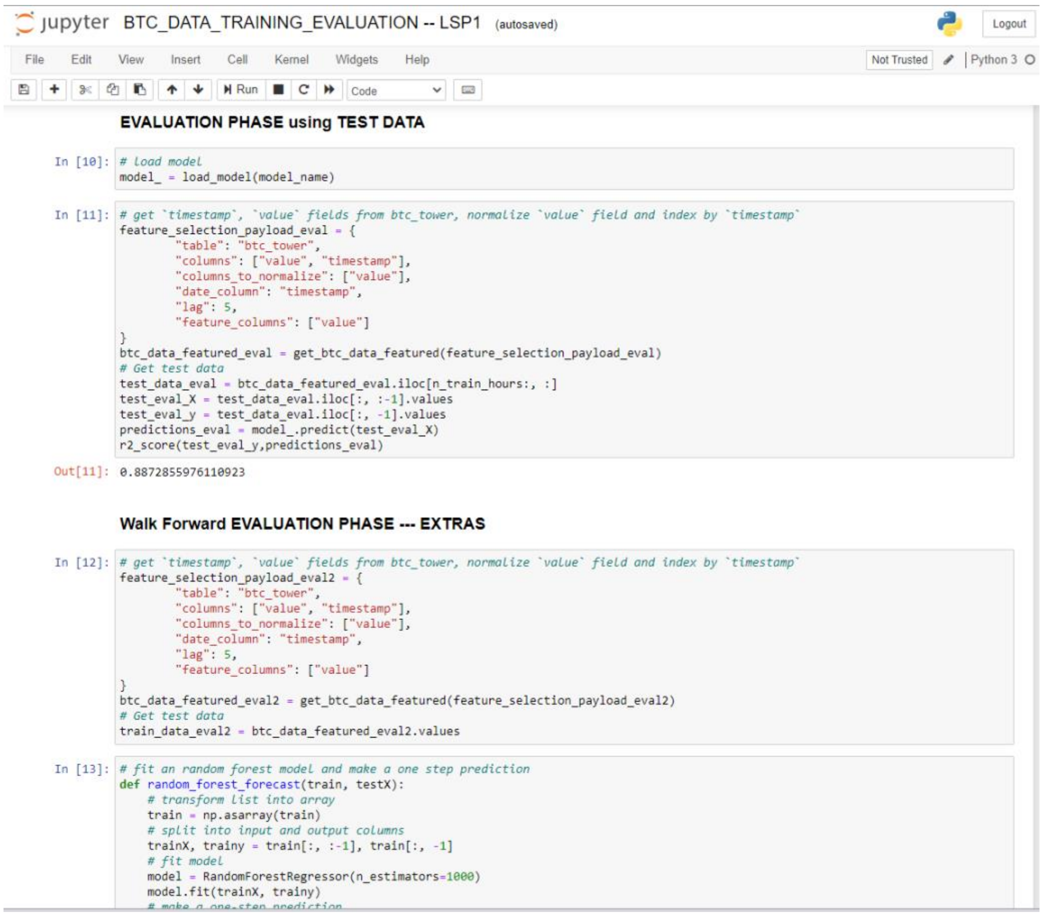
8.4 Evaluation Module

The Anaconda environment is an Python environment appropriate for all ML models developed for the LSPs in the MATRYCS project. Moreover, an SFTP server is created for straightforward integration of trained ML models with the Evaluation module. The credentials and details of the SFTP server are shared between involved partners.

Regarding the Anaconda environment, Jupyter notebooks are available for interactive work with datasets and ML models. The source code, computations, comments with the explanations, and graphical content are combined to make the Evaluation of the ML models easy to perform with high quality.

8.4.1 LSP1

The trained model for energy prediction for BTC Tower is evaluated by the Jupyter notebook instance exposed from Anaconda environment that presents the whole flow is included into the Evaluation module. Libraries like Requests[87], JSON[88], Joblib[89] for lightweight pipelining, Pandas[90], and NumPy[91] are installed and used in the notebook (Figure 31).


jupyter BTC_DATA_TRAINING_EVALUATION -- LSP1 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Code

EVALUATION PHASE using TEST DATA

```
In [10]: # load model
         model_ = load_model(model_name)

In [11]: # get `timestamp`, `value` fields from btc_tower, normalize `value` field and index by `timestamp`
         feature_selection_payload_eval = {
                 "table": "btc_tower",
                 "columns": ["value", "timestamp"],
                 "columns_to_normalize": ["value"],
                 "date_column": "timestamp",
                 "lag": 5,
                 "feature_columns": ["value"]
         }
         btc_data_featured_eval = get_btc_data_featured(feature_selection_payload_eval)
         # Get test data
         test_data_eval = btc_data_featured_eval.iloc[n_train_hours:, :]
         test_eval_X = test_data_eval.iloc[:, :-1].values
         test_eval_y = test_data_eval.iloc[:, -1].values
         predictions_eval = model_.predict(test_eval_X)
         r2_score(test_eval_y,predictions_eval)

Out[11]: 0.8872855976110923
```

Walk Forward EVALUATION PHASE --- EXTRAS

```
In [12]: # get `timestamp`, `value` fields from btc_tower, normalize `value` field and index by `timestamp`
         feature_selection_payload_eval2 = {
                 "table": "btc_tower",
                 "columns": ["value", "timestamp"],
                 "columns_to_normalize": ["value"],
                 "date_column": "timestamp",
                 "lag": 5,
                 "feature_columns": ["value"]
         }
         btc_data_featured_eval2 = get_btc_data_featured(feature_selection_payload_eval2)
         # Get test data
         train_data_eval2 = btc_data_featured_eval2.values

In [13]: # fit an random forest model and make a one step prediction
         def random_forest_forecast(train, testX):
             # transform list into array
             train = np.asarray(train)
             # split into input and output columns
             trainX, trainy = train[:, :-1], train[:, -1]
             # fit model
             model = RandomForestRegressor(n_estimators=1000)
             model.fit(trainX, trainy)
```

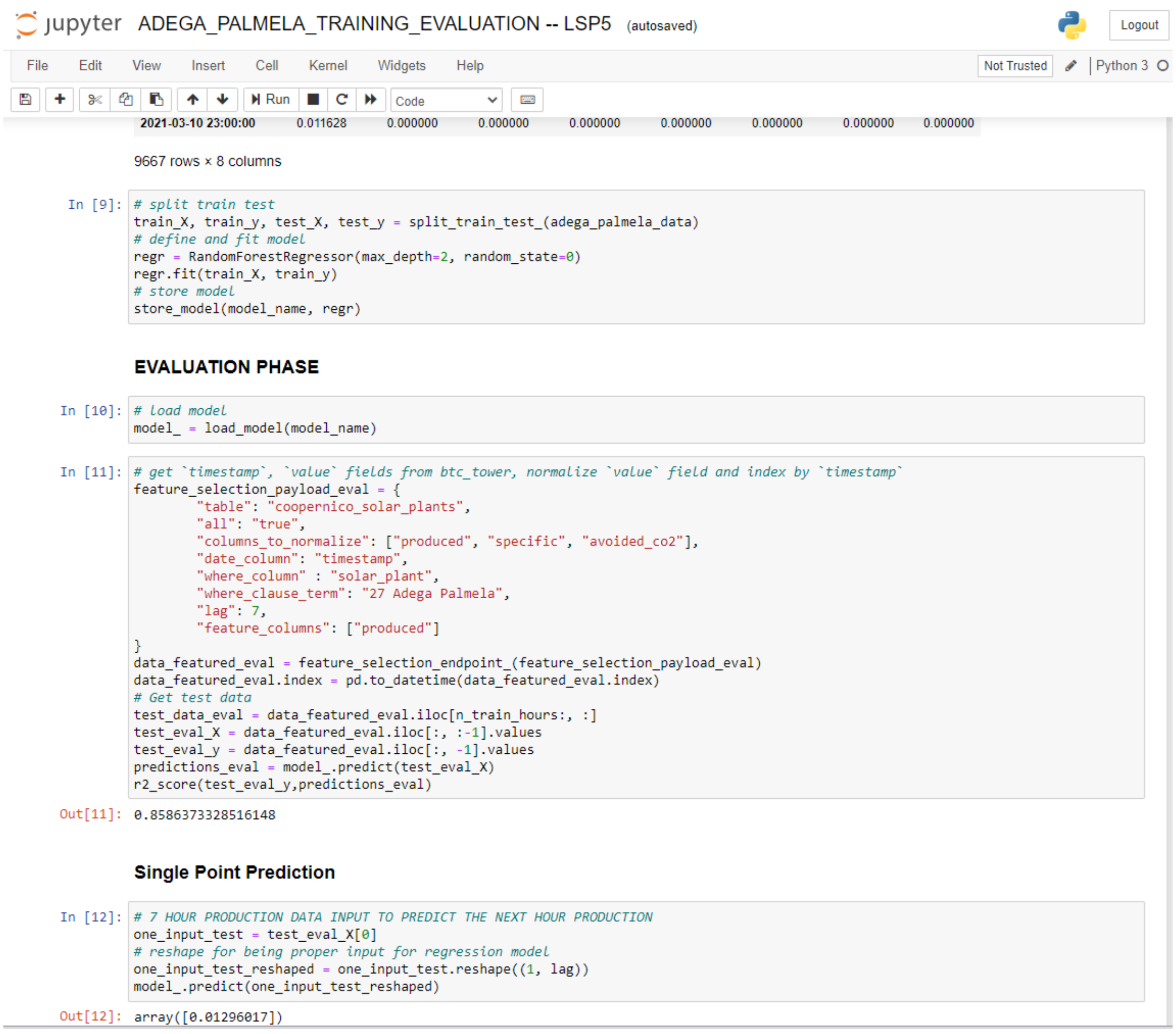
Figure 31: LSP1 Evaluation phase

[87] Python Requests, https://docs.python-requests.org/
[88] JSON encoder and decoder, https://docs.python.org/3/library/json.html
[89] Joblib: running Python functions as pipeline jobs, https://joblib.readthedocs.io/en/latest/
[90] Pandas, https://pandas.pydata.org/
[91] Numpy, https://numpy.org/

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

8.4.2 LSP5

For this LSP, the ML model for LSP5 Adega Palmela dataset was created, and was uploaded into the dedicated SFTP folder. After that, Adega Mangualde ML model is created also. The important libraries, like NumPy[92], Pandas[93], Joblib[94], PyHive[95], are included in the Anaconda environment and used in the evaluation process. They enable working and managing the chosen ML models, as well as connection to the Data Handler's REST APIs and retrieving the data for the evaluation of the chosen model (Figure 32).



```
2021-03-10 23:00:00   0.011628   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000

9667 rows × 8 columns
```

```
In [9]:  # split train test
         train_X, train_y, test_X, test_y = split_train_test_(adega_palmela_data)
         # define and fit model
         regr = RandomForestRegressor(max_depth=2, random_state=0)
         regr.fit(train_X, train_y)
         # store model
         store_model(model_name, regr)
```

EVALUATION PHASE

```
In [10]: # load model
         model_ = load_model(model_name)
```

```
In [11]: # get `timestamp`, `value` fields from btc_tower, normalize `value` field and index by `timestamp`
         feature_selection_payload_eval = {
                 "table": "coopernico_solar_plants",
                 "all": "true",
                 "columns_to_normalize": ["produced", "specific", "avoided_co2"],
                 "date_column": "timestamp",
                 "where_column" : "solar_plant",
                 "where_clause_term": "27 Adega Palmela",
                 "lag": 7,
                 "feature_columns": ["produced"]
         }
         data_featured_eval = feature_selection_endpoint_(feature_selection_payload_eval)
         data_featured_eval.index = pd.to_datetime(data_featured_eval.index)
         # Get test data
         test_data_eval = data_featured_eval.iloc[n_train_hours:, :]
         test_eval_X = data_featured_eval.iloc[:, :-1].values
         test_eval_y = data_featured_eval.iloc[:, -1].values
         predictions_eval = model_.predict(test_eval_X)
         r2_score(test_eval_y,predictions_eval)

Out[11]: 0.8586373328516148
```

Single Point Prediction

```
In [12]: # 7 HOUR PRODUCTION DATA INPUT TO PREDICT THE NEXT HOUR PRODUCTION
         one_input_test = test_eval_X[0]
         # reshape for being proper input for regression model
         one_input_test_reshaped = one_input_test.reshape((1, lag))
         model_.predict(one_input_test_reshaped)

Out[12]: array([0.01296017])
```

Figure 32: Evaluation of the selected ML model for LSP5

8.5 Serving Framework

BentoML is integrated with the Evaluation Framework, particularly with the Anaconda environment. In this way, developers can create bundles of models to be served on the selected port with enabled Swagger API. This process can be done directly in the Jupyter notebook. In the background, Yatai and

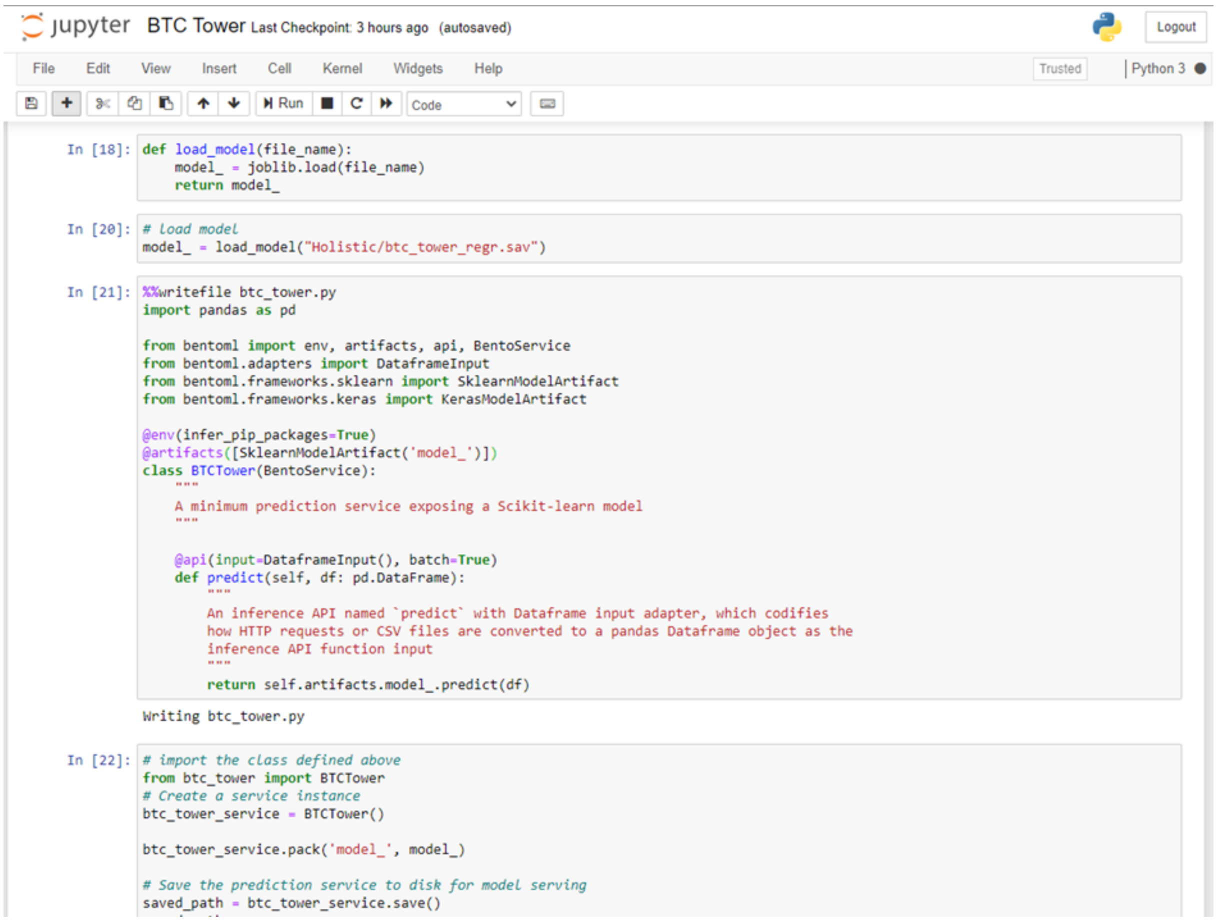
[92] NumPy, https://numpy.org/
[93] Pandas, https://pandas.pydata.org/
[94] Joblib, https://joblib.readthedocs.io/en/latest/
[95] PyHive, https://pypi.org/project/PyHive/

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

PostgreSQL are running as Docker containers, and they are enabling the serving of the selected set of the MATRYCS ML models.

8.5.1 LSP1

To serve models, the selected models should be imported into BentoML and saved for prediction service. Model serving with BentoML is easily created. There are several ways in which the model serving can be defined. In the following figure (Figure 33), one of the possible processes of including the evaluated ML model for LSP1 into the BentoML is presented.



```
In [18]: def load_model(file_name):
             model_ = joblib.load(file_name)
             return model_

In [20]: # Load model
         model_ = load_model("Holistic/btc_tower_regr.sav")

In [21]: %%writefile btc_tower.py
         import pandas as pd

         from bentoml import env, artifacts, api, BentoService
         from bentoml.adapters import DataframeInput
         from bentoml.frameworks.sklearn import SklearnModelArtifact
         from bentoml.frameworks.keras import KerasModelArtifact

         @env(infer_pip_packages=True)
         @artifacts([SklearnModelArtifact('model_')])
         class BTCTower(BentoService):
             """
             A minimum prediction service exposing a Scikit-learn model
             """

             @api(input=DataframeInput(), batch=True)
             def predict(self, df: pd.DataFrame):
                 """
                 An inference API named `predict` with Dataframe input adapter, which codifies
                 how HTTP requests or CSV files are converted to a pandas Dataframe object as the
                 inference API function input
                 """
                 return self.artifacts.model_.predict(df)

         Writing btc_tower.py

In [22]: # import the class defined above
         from btc_tower import BTCTower
         # Create a service instance
         btc_tower_service = BTCTower()

         btc_tower_service.pack('model_', model_)

         # Save the prediction service to disk for model serving
         saved_path = btc_tower_service.save()
```

Figure 33: ML model to be served for LSP1

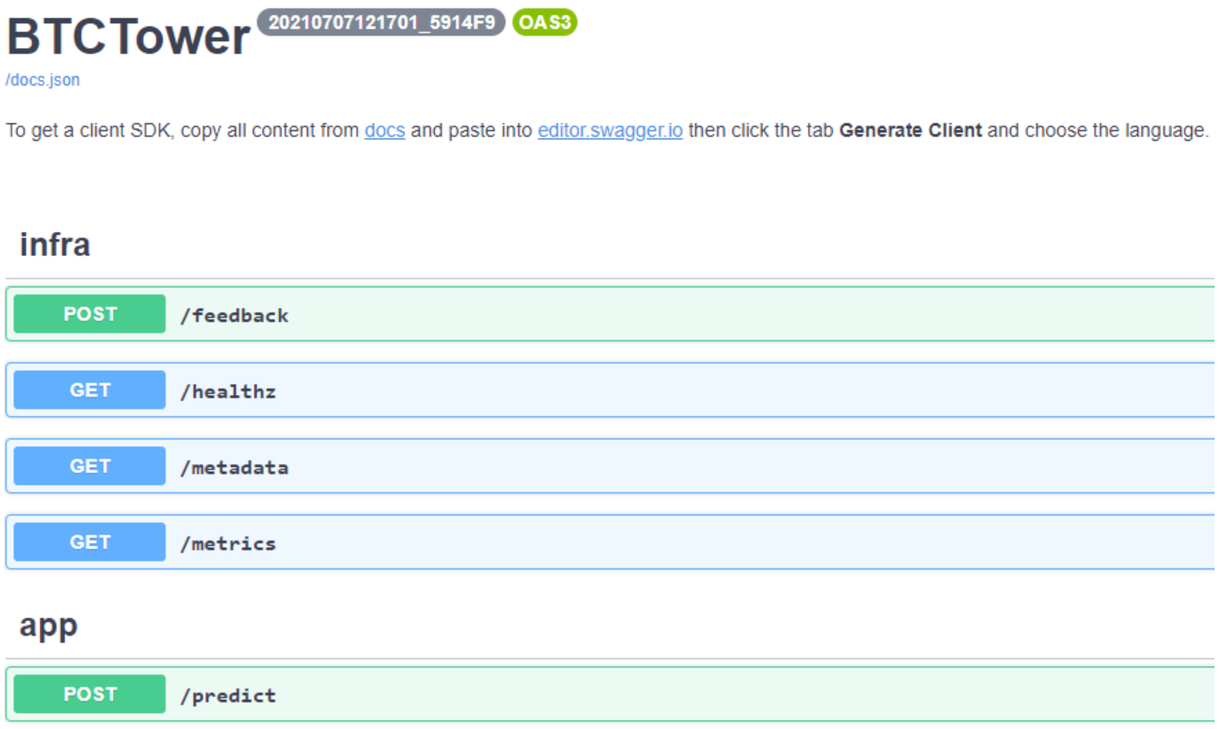
After creating the service bundle, the models then can be containerized and served in that way, or using the command:

Table 16: Basic BentoML command to serve the ML model in production mode

```
bentoml serve-gunicorn BTCTower:20210707121701_5914F9 --port 3002 --
enable-swagger --yatai-url 217.172.12.158:8891
```

The result of this process is presented in the following figure (Figure 34).

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158



BTCTower 20210707121701_5914F9 OAS3

/docs.json

To get a client SDK, copy all content from docs and paste into editor.swagger.io then click the tab **Generate Client** and choose the language.

infra

POST /feedback

GET /healthz

GET /metadata

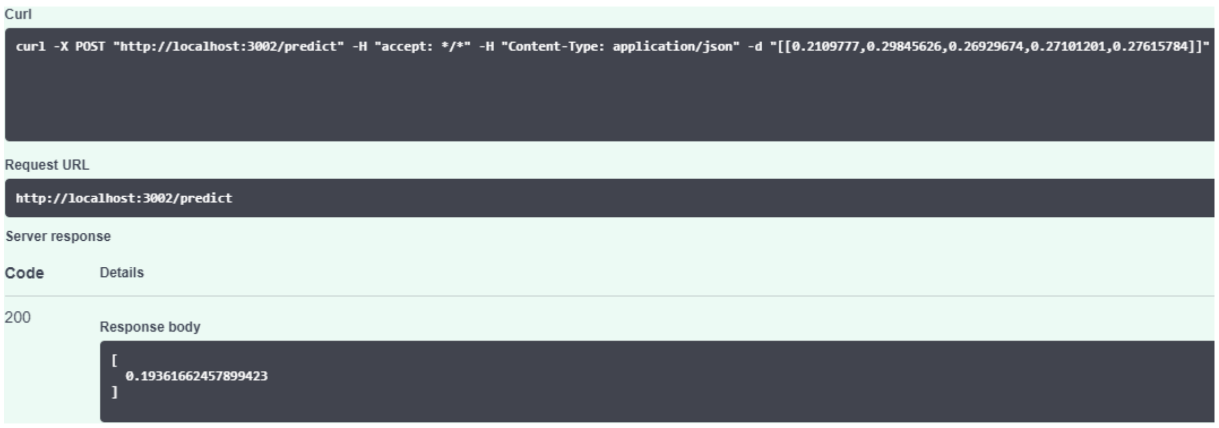
GET /metrics

app

POST /predict

Figure 34: Swagger API to access evaluated models from LSP1 served with BentoML

The Swagger REST API that is provided (Figure 35) is intuitive to use. Served ML model is available for the prediction service.



Curl

```
curl -X POST "http://localhost:3002/predict" -H "accept: */*" -H "Content-Type: application/json" -d "[[0.2109777,0.29845626,0.26929674,0.27101201,0.27615784]]"
```

Request URL

```
http://localhost:3002/predict
```

Server response

Code Details

200

Response body

```
[
  0.19361662457899423
]
```

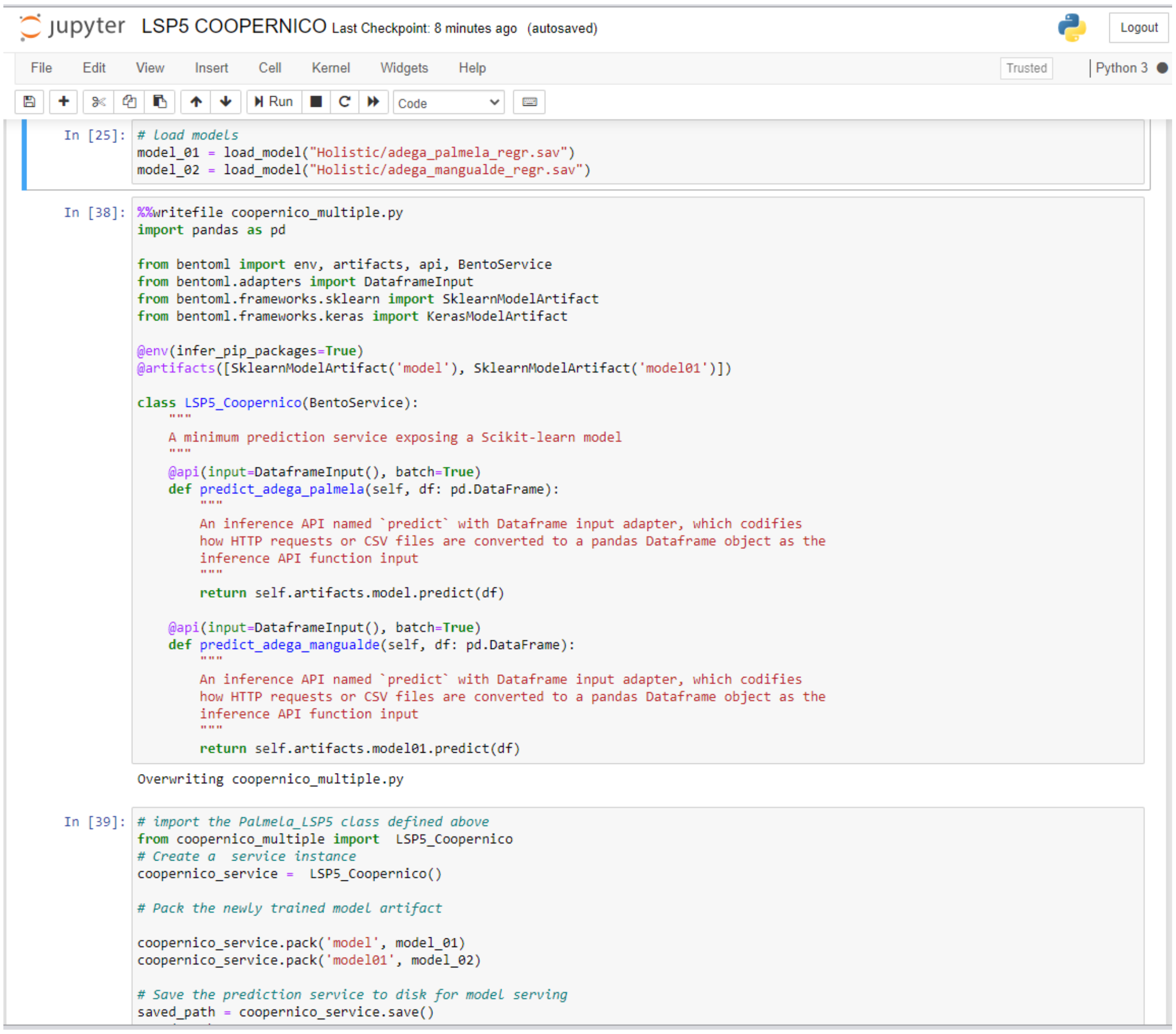
Figure 35: Example of prediction service used from BentoML's Swagger REST API for LSP1

Prediction requests can also be sent using the curl command, with Python and the Requests library, Postman etc.

8.5.2 LSP5

For the LSP5, two ML models are evaluated and because of that, the bundle of those two models is created in order for them to be served together. The APIs which contain the serving logic can be defined in the following way (Figure 36):

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158


jupyter LSP5 COOPERNICO Last Checkpoint: 8 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code

```
In [25]: # load models
         model_01 = load_model("Holistic/adega_palmela_regr.sav")
         model_02 = load_model("Holistic/adega_mangualde_regr.sav")
```

```
In [38]: %%writefile coopernico_multiple.py
         import pandas as pd

         from bentoml import env, artifacts, api, BentoService
         from bentoml.adapters import DataframeInput
         from bentoml.frameworks.sklearn import SklearnModelArtifact
         from bentoml.frameworks.keras import KerasModelArtifact

         @env(infer_pip_packages=True)
         @artifacts([SklearnModelArtifact('model'), SklearnModelArtifact('model01')])

         class LSP5_Coopernico(BentoService):
             """
             A minimum prediction service exposing a Scikit-learn model
             """
             @api(input=DataframeInput(), batch=True)
             def predict_adega_palmela(self, df: pd.DataFrame):
                 """
                 An inference API named `predict` with Dataframe input adapter, which codifies
                 how HTTP requests or CSV files are converted to a pandas Dataframe object as the
                 inference API function input
                 """
                 return self.artifacts.model.predict(df)

             @api(input=DataframeInput(), batch=True)
             def predict_adega_mangualde(self, df: pd.DataFrame):
                 """
                 An inference API named `predict` with Dataframe input adapter, which codifies
                 how HTTP requests or CSV files are converted to a pandas Dataframe object as the
                 inference API function input
                 """
                 return self.artifacts.model01.predict(df)

         Overwriting coopernico_multiple.py
```

```
In [39]: # import the Palmela_LSP5 class defined above
         from coopernico_multiple import  LSP5_Coopernico
         # Create a  service instance
         coopernico_service =  LSP5_Coopernico()

         # Pack the newly trained model artifact

         coopernico_service.pack('model', model_01)
         coopernico_service.pack('model01', model_02)

         # Save the prediction service to disk for model serving
         saved_path = coopernico_service.save()
```

Figure 36: ML models bundle to be served for LSP1

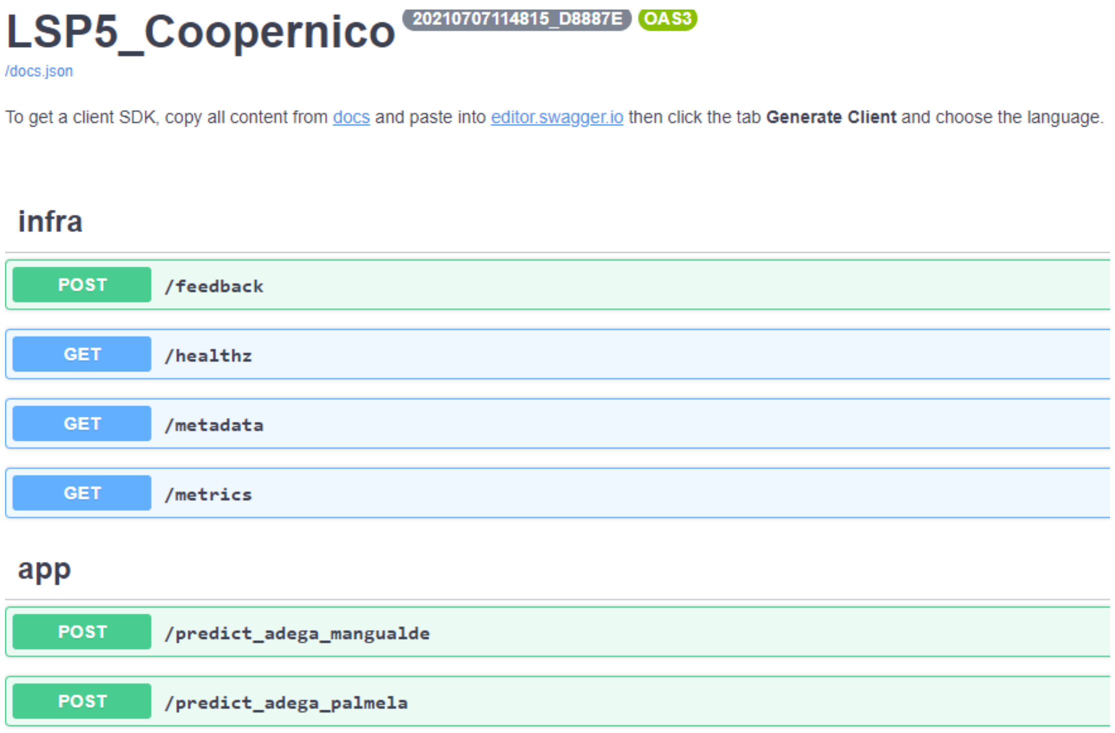
The command (Table 17) that can be issued to serve the bundled ML models is:

Table 17: Basic BentoML command to serve the bundle of ML Models in production mode

```
bentoml serve-gunicorn LSP5_Coopernico:20210707114815_D8887E --port 3003 -
-enable-swagger --yatai-url 217.172.12.158:8891
```

In contrast with the LSP1, for LSP5 there are two prediction services, for two evaluated models. The connections to APIs are enabled through an interactive user interface (Figure 37).

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158



LSP5_Coopernico 20210707114815_D8887E OAS3

/docs.json

To get a client SDK, copy all content from docs and paste into editor.swagger.io then click the tab **Generate Client** and choose the language.

infra

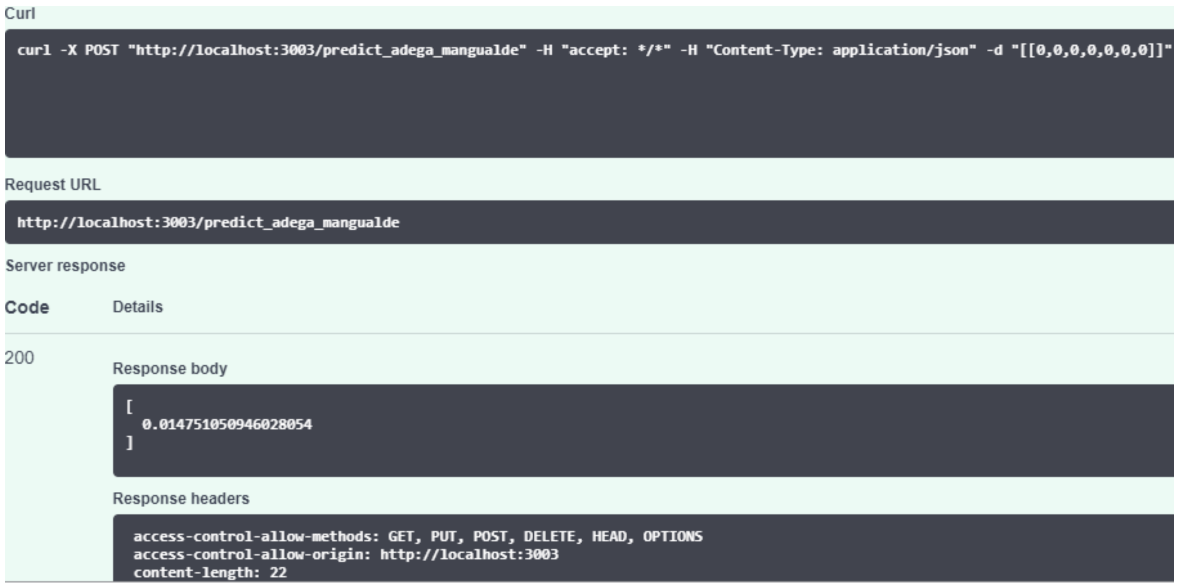
```
POST /feedback
GET  /healthz
GET  /metadata
GET  /metrics
```

app

```
POST /predict_adega_mangualde
POST /predict_adega_palmela
```

Figure 37: Swagger API to access evaluated models from LSP5 served with BentoML

The prediction functionalities are easily reachable via POST requests in Swagger REST API (Figure 38).



Curl

```
curl -X POST "http://localhost:3003/predict_adega_mangualde" -H "accept: */*" -H "Content-Type: application/json" -d "[[0,0,0,0,0,0,0]]"
```

Request URL

```
http://localhost:3003/predict_adega_mangualde
```

Server response

Code Details

200

Response body

```
[
  0.014751050946028054
]
```

Response headers

```
 access-control-allow-methods: GET, PUT, POST, DELETE, HEAD, OPTIONS
 access-control-allow-origin: http://localhost:3003
 content-length: 22
```

Figure 38: Example of prediction service used from BentoML's Swagger REST API for LSP5

8.6 Connection to MATRYCS-ANALYTICS Layer

The MATRYCS-PROCESSING is the aggregation of data management and AI services where the trained and stored models are exposed to the MATRYCS-ANALYTICS layer, through REST APIs, by using the infrastructure of the Serving Framework. The analytics services use the trained models in the backend, as decision support system and visualize the final results for the end users.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158

9 Future activities

As the 1st technology release has been described and analysed for the MATRYCS-PROCESSING layer, the main technologies that were used and the limitations that have arisen are well understood and lead to the future activities for an efficient and successful 2nd technology release.

Below, the future activities for MATRYCS-PROCESSING layer are listed:

Data Feed

- Add more functionalities for data preparation (e.g, exponential smoothing)
- Continuous integration activities with all MATRYCS-PROCESSING components
- Migrate from on premises installation to cloud installation
- Data expansion by adding more datasets
- Integration with End-to-End Security Framework

Model Development Module and ML Suite

- Install more ML libraries
- Continuous integration activities for advanced connection with all MATRYCS-PROCESSING layer's components
- Migrate from AWS to EGI ACE cloud (once positively evaluated by EGI ACE project)
- Store more ML/DL models
- Integration with End-to-End Security Framework

Serving and Evaluation Framework

- Transfer all models to official MATRYCS-PROCESSING model shared storage
- Test more ML libraries with BentoML
- Install and test new libraries based on the different models that will be developed during the project's ML development phase
- Integration with End-to-End Security Framework
- Migrate from on premises installation to cloud installation

The 2nd technology release of the MATRYCS-PROCESSING layer will be described in detail in D4.2 - *MATRYCS-PROCESSING (2nd technology release) (M22)*.

The MATRYCS project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no.101000158