



Grant Agreement N°857191

Distributed Digital Twins for industrial SMEs: a big-data platform

DELIVERABLE 3.2 – FIRST VERSION OF THE SIMULATION AND AI SERVICES FOR DIGITAL TWINS



Document Identification

Project	IoTwinS
Project Full Title	Distributed Digital Twins for industrial SMEs: a big-data platform
Project Number	857191
Starting Date	September 1st, 2019
Duration	3 years
H2020 Programme	H2020-EU.2.1.1. - INDUSTRIAL LEADERSHIP - Leadership in enabling and industrial technologies - Information and Communication Technologies (ICT)
Topic	ICT-11-2018-2019 - HPC and Big Data enabled Large-scale Test-beds and Applications
Call for proposal	H2020-ICT-2018-3
Type of Action	IA-Innovation Action
Website	iotwins.eu
Work Package	WP3 - AI services for distributed digital twins
WP Leader	UNIBO
Responsible Partner	UNIBO
Contributing Partner(s)	ESI, THALES, SAG/SAGOE
Author(s)	Andrea Borghesi (UNIBO), Michela Milano (UNIBO)
Contributor(s)	Vincent Thouvenot (THALES), Tobias Schuele (SAG), Christian Kern (SAG), Felix Moessbauer (SAG), Tobias Preclik (SAG), Morgan Cameron (ESI), Fatima Daim (ESI)
Reviewer(s)	Elisabetta Ronchieri (INFN), Alice Heliou (THALES)
File Name	D 3.2 – FIRST VERSION OF THE SIMULATION AND AI SERVICES FOR DIGITAL TWINS
Contractual delivery date	M12 – 31 August 2020
Actual delivery date	M15 – 17 November 2020
Version	2.2
Status	Final
Type	DEM: Demonstrator, pilot, prototype
Dissemination level	PU: Public
Contact details of the coordinator	Francesco Millo, francesco.millo@bonfiglioli.com

Document log

Version	Date	Description of change
V0.1	07/09/2020	First draft
V1.0	28/09/2020	All contributions added – ready for internal reviewing
V1.5	05/10/2020	First review received and integrated
V2.0	13/10/2020	Internal review completed
V2.1	30/10/2020	Second draft - ready for partners' check
V2.2	12/11/2020	Final version

Table of Contents

1	Introduction.....	6
2	Service development methodology	6
3	List of developed services.....	7
3.1	General services APIs.....	8
3.1.1	APIs	8
3.1.2	Software Requirements.....	11
3.2	Supervised and semi-supervised anomaly detection services	11
3.2.1	Methodologies.....	12
3.2.1.1	Semi-supervised anomaly detection	12
3.2.1.2	Supervised anomaly detection	13
3.2.2	APIs	14
3.2.3	Software Requirements.....	18
3.3	Unsupervised abnormal time series identification and outliers detection.....	19
3.3.1	Functional Isolation Forest	19
3.3.2	ADTK	20
3.3.3	APIs	20
3.3.4	Software requirements.....	20
3.4	Physics simulation and fault modelling services	21
3.4.1	Reduced-order model DoE generator	22
3.4.2	Reduced-order model generator.....	22
3.5	Services usage examples	24
3.5.1	General services usage	24
3.5.2	Anomaly detection in multi-variate time series using Functional Isolation Forest.....	27
3.5.3	Outliers and change points detection with ADTK	31
3.5.4	Simulation services for smart manufacturing	35
4	A case-study: AI Services for Energy Automation	36
4.1	System architecture.....	37
4.1.1	Overview.....	37
4.1.2	Components	38
4.1.2.1	IEC 61850.....	38
4.1.2.2	Message Bus	39
4.1.2.3	Simulators.....	39
4.1.2.4	Rule Engine & Database Writer	39

4.1.2.5 Time-series Database (InfluxDB)	39
4.1.2.6 AI Applications	40
4.1.2.7 Analysis Dashboard	41
4.2 Results	41
5 Open Issues & Corrective Actions	42
5.1 Edge Processing	42
5.2 Integration with Existing Platforms	42
5.3 Testbed Requirements	42
5.4 Optimization Services	43
6 Conclusions.....	44

1 Introduction

This deliverable describes the first version of the Artificial Intelligence (AI) and simulation-based services developed for the IoTwinS platform. The initial set of services developed and deployed on the platform was selected in accordance with the requirements expressed by the project partners and collected with interviews and questionnaires (see deliverable D3.2). This is an initial version of the services to be developed, and they will be modified and extended following the evaluation and feedback stage, where the services will be made available to WP4 and WP5 partners to be used.

The chosen set of services do not cover all the use-cases presented by the multiple pilots (see deliverable D3.1 for reference), but they were rather developed to be open, well-documented and easily extensible, in order to allow test-bed partners to develop and implement their own specific, pilot-tailored services. Hence, the focus has been on developing general services that could be re-used in different context and serve multiple purposes; additionally, more problem-specific and high-level services have been developed to offer guidelines describing the methodology to compose base services to obtain more sophisticated ones. A key part of this deliverable is hence the description of the methodology followed to develop the services.

The code for the AI & simulation services can be found at the following Git repository: <https://gitlab.hpc.cineca.it/iotwins/ai-services/>. The repository is hosted by CINECA, one of the partners of IoTwinS consortium, and it can be accessed after having required the credentials. We plan to make the code public in a following phase of the project, after having received the feedback from the test-bed partners.

In the rest of the section the development methodology and the developed services will be described.

2 Service development methodology

The services were developed as a collection of self-contained Docker containers; this choice was made to ensure a high degree of portability and re-use capability. The Docker technology¹ consists of a platform for deploying service products using OS-level virtualization to deliver software in packages, and allowing a high-level abstraction of the underlying hardware (HW); each AI-based service is isolated from the remaining ones and self-contained, that is, it is endowed with all necessary corollary software tools and modules. This choice clearly requires the IoTwinS infrastructure (developed in WP2) to be capable of managing Docker containers; this is a reasonable requirement as the Docker technology is widely adopted and open source, with computational demands that depend on the actual developed services – in practice, one can create containers with different computational requirements for different HW targets. It can be considered the *de facto* standard for Platform as a Service (PaaS) solutions, in line with the overall goal of the IoTwinS project.

In order to demonstrate the deployment of the AI-based services developed and described by this document, we opted for employing the cloud infrastructure and the PaaS orchestrator provided by one of the IoTwinS partners, namely INFN. The code used for creating the demonstrator is publicly available on a GitHub repository (different from the one used to host all the developed AI-based services): <https://github.com/iotwins-demo/AI-Services>. The actual demonstrator can be reached at the following link <https://indigo-paas.cloud.ba.infn.it/home/>. Currently, the demonstrator is hosted at the facilities of INFN cloud and it is not accessible without registering to the INFN cloud service – only users with correct credentials can access the AI-based services. As discussed as well in Section 5, the current demonstrator is a

¹ <https://github.com/docker/docker-ce>

software tools whose main aim is to allow other partners to start using the AI services and provide feedback. It is not a full production-ready suite of services. Namely, what is lacking at the moment is the integration within the IoTwinS platform that is still under development – the collaboration between WP2 and WP3 partners will proceed as planned, especially after the completion of D2.2 (which has the same deadline as the current D3.2 deliverable).

As the current demonstrator could not rely on the IoTwinS platform, we opted for focusing on cloud-based services, though we identified services that can be executed on the edge devices as well, according to the computational demands of the services; as the initial round of questionnaires for the testbeds (see D2.1 and D3.1) highlighted that edge devices among the different pilots are heterogeneous, the interaction with testbed providers after the publication of D3.1 will allow us to better tailor edge-level services to the specific cases. Another consequence of the fact that the IoTwinS overall platform is not ready yet (as expected), there was not a unified methodology to collect and store data shared among all testbeds – thus, a common method to feed data to the AI services has not been established, yet. To cope with this issue, we decided to allow the AI services users to directly upload the data to be analysed/processed in batches, as single or multiple files; the format of admitted files depends on the service – for instance, a prevalent option to many services is the usage of csv (Comma Separated Values) files as input data. Clearly, this is a temporary solution and has its own limitations (e.g. the INFN-provided cloud infrastructure imposes an upper bound limit on the maximum size of the uploaded file). In collaboration with WP2 partners we plan to revise and refine the services and by addressing the potential requests from testbed partners, such as the addition of useful services not yet deployed (e.g. fine-tuning of the parameters of an AI-based model) and the capability of handling data streams.

The different services are provided in different Docker containers and require different software modules and libraries, in the following section we will provide a more detailed description of the developed AI-based services. In order to keep this deliverable clear and short we have decided to describe just a subset of the technical features at the finest level of detail. For instance, the description of the code (e.g. the algorithm, the data structure, support files required to instantiate Docker containers within the INFN's PaaS environment) for implementing the services is not reported here. However, sufficiently detailed information concerning all technical aspects can be found in the documentation associated to the code and available in the online repository (<https://github.com/iotwins-demo/AI-Services>).

3 List of developed services

In this section we list the services developed so far, with their associated APIs. The proposed services are divided into macro-areas, ranging from general services that form the basis of more complicated workflows, to higher-level tasks such as anomaly detection and simulation models. For each service we specify if it can be performed on the cloud, on the edge or on both.

Some services have higher computational demands and thus cannot be executed on the edge devices but require to be deployed on cloud resources; for example, training accurate Machine Learning (ML) models for anomaly detection on large amount of data or running complex simulation models requires non negligible hardware resources. Edge devices can be instead used for real-time computation on streams of data (e.g. after the initial training phase, an ML model can be used to make predictions on live data – also referred to as *inference* – with much more limited computational demands).

General services:

1. `get_categorical_continuous_features` - identifies categorical and continuous features in a data frame (edge & cloud)
2. `encode_categorical_feature` - encodes a single categorical feature using one-hot encoding (edge & cloud)
3. `oneHot_encode` – encodes all categorical features in the data set using one hot encoding (edge & cloud)
4. `normalize` – normalizes data (cloud)
5. `preprocess` - preprocesses data frame (cloud)
6. `split_data_noLabel` - splits data frame in train, test and validation sets (cloud)
7. `split_data_withLabel` - splits data frame in train, test and validation sets (cloud)
8. `oversample` – oversamples a labeled data set (cloud)
9. `sanitize` – sanitizes input strings (edge & cloud)

Anomaly detection services:

1. `semisup_autoencoder` - creates a semi-supervised autoencoder to detect anomalies (cloud)
2. `semisup_detection_inference` - exploits a semi-supervised autoencoder to detect anomalies (edge & cloud)
3. `sup_autoencoder_classr` - creates a supervised model for anomaly detection, composed by an autoencoder plus a classifier layer (cloud)
4. `sup_detection_inference` - exploits a supervised classifier to detect anomalies (edge & cloud)

Time series API for abnormal time series detection and outliers detection

1. `abnormal time series detection` - identifies abnormal time series in a corpus of time series. Based on Functional Isolation Forest (cloud)
2. `outliers and change point detection` - detects some classical anomalies type in time series. Based on ADTK Python module (cloud)
3. `anomaly detection for power grids` - detects anomalies in electric power grids, e.g. intrushes, short circuits or power swings, based on voltage and current measurements (time series) using recurrent neural networks (Long Short-Term Memory – LSTM) (edge & cloud)

In the following sections, we provide more details about the developed services and their APIs.

3.1 General services APIs

The general services offer a wide spectre of functionalities of different levels of complexity, ranging from encoding of categorical data to whole data pre-processing. The services in this group have a high degree of reusability and can be helpful for a large range of ML tasks.

3.1.1 APIs

`get_categorical_continuous_features` - identifies categorical and continuous features in a data frame

- `param df_fname` : string
 - name of the file containing the data to be analysed
 - it must be a csv files, where each field is separated by a comma (',')

- **return:** lists of strings
 - column names of categorical features and list of column names of categorical features

encode_categorical_feature - encodes a categorical feature with one-hot encoding

- **param** df_fname : string
 - name of the file containing the data to be encoded
 - it must be a csv files, where each field is separated by a comma (',')
- **param** cat_feat : str
 - name of the categorical feature to be encoded
- **return:** pandas dataframe
 - data with encoded feature

oneHot_encode – encodes a categorical feature using one-hot encoding (applied to all categorical features in a dataframe)

- **param** df_fname : string
 - name of the file containing the data to be encoded
 - it must be a csv files, where each field is separated by a comma (',')
- **param** (optional) cat_feats : list of strings
 - column names of categorical features
 - by default, the encoding is applied to all categorical features of the input file
- **return:** lists of strings
 - data with one-hot encoded categorical features

normalize – normalizes data

- **param** df_fname : string
 - name of the file containing the data to be normalised
 - it must be a csv files, where each field is separated by a comma (',')
- **param** (optional) cat_feats : list of strings
 - column names of categorical features
- **param** (optional) cont_feats : list of strings
 - column names of continuous features
- **param** (optional) scaler_in : scikit-learn scaler object
 - the scaler used to normalize the data
- **param** (optional) scalerType : string
 - scaling type - supported types 1) 'minmax' and 2) 'std'
 - default value = 'minmax'
- **return:** pandas dataframe
 - normalized data
- **return:** scikit-learn scaler object
 - scaler object used to normalize the data

preprocess - preprocesses data frame, removing Not-a-Number (NaN) values, applying one-hot encoding of categorical features and normalization of continuous features (using minmax scaling)

- **param** df_fname : string
 - name of the file containing the data to be processed

- it must be a csv files, where each field is separated by a comma (',')
- **param** (optional) `y` : string
 - name of the feature to be used as label/target
- **param** (optional) `scaler_in` : scikit-learn scaler object
 - the scaler used to normalize the data
- **return**: pandas dataframe
 - normalized data
- **return**: numpy arrays
 - list for targets if the label/target option has been specified, otherwise the target list is an empty list
 - labels are not pre-processed (neither scaled nor encoded)
- **return**: scikit-learn scaler object
 - scaler object used to normalize the data

split_data_noLabel - splits dataframe in train, test and validation sets; this function does not expect labels, it operates on a single data frame

- **param** `df_fname` : string
 - name of the file containing the data to be split
 - allowed inputs are python lists, numpy arrays, scipy-sparse matrices or pandas data frames
 - the input file is expected to be a binary file serialized following the python pickle format²
- **param** (optional) `ratio`: float
 - the proportion of the data set to be included in the training set split
 - default value = 0.7
- **return**: python list
 - list containing train, test, and validation splits

split_data_withLabel - splits data frame in train, test and validation sets; this function works with labeled data; it expects a list (numpy array, pandas series) of labels

- **param** `df_fname` : string
 - name of the file containing the data to be split
 - allowed inputs are python lists, numpy arrays, scipy-sparse matrices or pandas data frames
 - the input file is expected to be a binary file serialized following the python pickle format
- **param** `y` : list of strings
 - labels list to be split (according to the input features split)
- **param** (optional) `ratio`: float
 - the proportion of the data set to be included in the training set split
 - default value = 0.7
- **return**: python list
 - list containing train, test, and validation splits

oversample - oversamples a labeled data set, that is re-balance an unbalanced data set

- **param** `X_fname` : string
 - name of the file containing the python list or numpy array of features to be oversampled

² <https://docs.python.org/3/library/pickle.html>

- the input file is expected to be a binary file serialized following the python pickle format
- **param** `y_fname` : string
 - name of the file containing the Python labels to be oversampled (associated to the input features)
 - the input file is expected to be a binary file serialized following the python pickle format
- **param** (optional) `method` : string
 - oversampling method -- supported methods: 'ADASYN', 'SMOTE', 'KMEANSMOTE', 'RNDM', 'SMOTENC', 'SVMSMOTE', 'BORDERSMOTE'
 - default value = 'SMOTE'
- **param** (optional) `strat` : string
 - oversampling strategy -- supported strategies: 'minority', 'not minority', 'not majority', 'all'
 - default value = 'minority'
- **return**: numpy arrays
 - data (input features) after oversampling
- **return**: numpy arrays
 - labels after oversampling

sanitize – sanitizes input string, by removing all non-alphanumeric characters

- **param** `input_str` : string
 - the input string to be sanitized
- **return**: string
 - the sanitized string

3.1.2 Software Requirements

As mentioned before, the developed services have been deployed as Docker containers. The general services were developed using Python language (Python >= 3.4) and using a set of libraries and Python modules not present in the native Python installation; clearly, the service container is equipped with all the required modules.

For the sake of clarity and replicability, we list here the requirements of the services:

- Python 3.4
- Python modules:
 - numpy==1.19.1
 - pandas==1.1.0
 - scipy==1.5.2
 - scikit-learn==0.23.2
 - Imblearn==0.7.0

3.2 Supervised and semi-supervised anomaly detection services

The supervised and semi-supervised anomaly detection services have the goal of identifying anomalous values in a data set. The data set can correspond to a time-series, that is, every example in the data set has an associated time-stamp, with consecutive examples corresponding to temporally contiguous samples. As from the collection of requirements described in D3.1 it appeared that most of the testbeds were still striving

simply to differentiate between normal and anomalous data points. Hence, we have decided to begin by offering services for binary classification, that is for distinguishing between anomalous and normal behaviour without differentiating among multiple normal states (e.g. active machine versus idle) or potentially numerous fault conditions.

The main difference between the two macro-services is that in the supervised case the data must be fully labelled, that is for each entry there must be the associated class (normal or anomalous entry). In the semi-supervised case, the only requirement is to have partially labelled data, that is a sub-set of the entries must be labelled, and in particular they must be normal points, which are going to be used for training the Deep Learning (DL) models. Additionally, labelled data is also required to test and validate the trained semi-supervised models; this is not a requirement for the DL models *per se*, which can be trained using only a subset of normal labelled data, but it is a way to measure the accuracy of the models.

Albeit the services provided here deal with binary classification tasks, they could be easily extended to deal with multi-labelled classification problems, in case the testbeds show interested towards this end.

3.2.1 Methodologies

Both the supervised and the semi-supervised approaches were developed using DL models. Furthermore, in both cases the services have been decomposed in two stages: 1) the initial training phase when the DL models are trained by feeding them with a subset of the available data and 2) a second inference phase where the trained model can be used to perform anomaly detection on entries that have not been considered in the training phase. The training (and thus the corresponding service) has to take place on the cloud, as it tends to be quite demanding in terms of computational resources; the inference can instead happen on the edge, due to lower computational demands (clearly, the inference can be run on the cloud as well).

3.2.1.1 Semi-supervised anomaly detection

The core goal of the model is to learn the “normal” behaviour of the target system described by the data set, in order to detect anomalous conditions. The method is based on a type of DL model called **autoencoder**. More precisely, the proposed approach³⁴ is based on a critical assumption: an autoencoder can learn the representation of the normal state (looking at the measured features); afterwards, the model can be used to notice representation changes that underlie anomalous conditions. This happens because an autoencoder is a neural network trained to copy its input x to its output y . Internally it has hidden layers h encoding the representation of the data. An autoencoder is composed by two subparts: an encoding function $h = f(x)$ and a decoding function that reconstructs the input $y = g(h)$. Typically, autoencoders do not simply learn the identity function $g(f(x)) = x$, but they are designed to be unable to copy perfectly, thus the output of an autoencoder is generally different from its input; this difference is called reconstruction error. The reconstruction error is used to identify anomalies. During the training phase, an autoencoder learns the relationships among the features in the input set. If new, previously unseen, data is given to the trained autoencoder as input, it can reproduce it accurately (with a low reconstruction error), as long as the new data is similar to the one encountered in the training set – the correlation between the features of the new

³ Borghesi et al. [Anomaly detection using autoencoders in high performance computing systems](#), Proceedings of the AAAI Conference on Artificial Intelligence, 2019

⁴ Borghesi et al., [Online anomaly detection in hpc systems](#), 2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)

data must be similar to the one learned by the autoencoder. If this condition does not hold, the autoencoder cannot correctly reconstruct the input and the reconstruction error is greater.

Deciding the optimal hyperparameters that define a neural network is a notoriously challenging task, as for different learning tasks and domains different hyperparameters configurations could be better suited. In general, hyperparameter tuning requires effort and experience from Machine Learning (ML) practitioners. In this first version of the ML-based services we offer the users the chance of specifying the hyperparameters of the autoencoder network; alternatively, a default set of values is adopted. In future versions of the ML services, depending on the feedback gathered from the testbed partners, the services set could be extended via the addition of an automatic hyperparameter tuning mechanism.

Once we have observed that the reconstruction error for the anomalous points is much higher than the error obtained in normal samples⁵, we need to discuss how the reconstruction error can be used to detect an anomaly. We are not strictly interested in the absolute value of the reconstruction error but rather on the relative difference between normal and anomalous samples. Our goal is to identify an error threshold θ to discriminate between normal and anomalous behaviour. To do so we have looked at the distributions of the reconstruction errors; empirical observations show that the errors distribution of the normal data set is extremely different from the anomalous one. Since we can clearly distinguish the error distributions, we opted for a simple method to classify each data point: if the reconstruction error E for a data point is greater than a threshold θ , then the point is “abnormal”; otherwise the data point is considered normal. The next step is to identify the threshold used to classify each data point. We choose as a threshold the n -th percentile of the errors distribution of the normal data set, where n is a value that depends on the specific data set. In practice, the users of the ML service can either specify the desired n value to be used to compute the detection threshold, or can let the service automatically select the best value, computed using a subset of the test data. The algorithm to select n consists of a simple generate-and-test search strategy, based on a finite number of values opportunely tested, and then chose those guaranteeing the best results in term of classification accuracy on the validation set.

The autoencoder is trained using only normal data, with the goal of minimising the reconstruction error; as a loss function we adopted the mean absolute error (MAE). Afterwards, the threshold is selected as described earlier. Then, the model can be tested by feeding new data to the trained model and observing its prediction. In order to assess the accuracy of the model the test sample should contain both anomalous and normal points. The anomaly detection service expects to be provided with a data set containing also anomalous points exactly for this reason, that is evaluating the model’s accuracy. Finally, the output of the service that trains the DL models is a set of accuracy and performance metrics computed on the test set. Conversely, the inference service does not compute any accuracy metrics but rather returns the predictions made by the model on the newly arrived data set. The inference service makes its predictions using a previously trained DL model (the autoencoder plus the selected threshold – the users can specify a different threshold is they prefer), whose name must be provided by the service user, in order to retrieve the desired model.

3.2.1.2 Supervised anomaly detection

In the supervised case, the implicit assumption is that all data is labelled, thus each example is classified and annotated by domain experts. Since we are considering a binary classification task, each example can belong

⁵ Borghesi et al., [A semisupervised autoencoder-based approach for anomaly detection in high performance computing systems](#), Engineering Applications of Artificial Intelligence 85, 634-644, 2019

either to the normal class or to the anomalous ones. ML models for supervised tasks have higher accuracies than semi-supervised or unsupervised models, with the inherent increased cost due to the annotation process, which depends on domain knowledge and can be very time-consuming. There exist multiple ML techniques for supervised learning; one that in recent years has demonstrated success⁶ belongs to the DL area and relies on deep neural networks. For the supervised anomaly detection service developed for this demonstrator, we opted for a DL model composed by two main components: 1) an autoencoder and 2) a classifier network.

The autoencoder has a similar structure to the one described in the previous section (the semi-supervised anomaly detection case). In this supervised approach its purpose is to serve as a feature extractor; while striving to minimize the reconstruction error the autoencoder networks learns to focus on the most important features among the one fed as input, and to project them into its internal latent space. The classifier network is then attached to the latent features (as they should provide a simpler representation with respect to the original one) and a set of layers is added, until the final output one. The classifier layers are trained to distinguish between normal and anomalous points – binary classification.

Both the autoencoder and the classifier networks have their own set of hyperparameters. The users can either specify the desired parameters or just rely on the default values provided by the current implementation of the service. In the supervised setting, the autoencoder is trained with all types of entries, both the normal and the anomalous ones. The training is performed in two stages: 1) first the autoencoder is trained in order to minimise the reconstruction error (again, MAE as loss function); 2) the weights of the autoencoder are fixed and the classifier layers are trained, using binary cross-entropy as loss function. The test is performed on a subset of data not used during the training. As in the semi-supervised case, the service to train the DL model produces as output a collection of accuracy metrics to evaluate the performance of the model: the supervised inference services load a pre-trained DL model and use it to make predictions on new data, returning the list of predicted values, without computing any accuracy measure.

3.2.2 APIs

semisup_autoencoder - creates a semi-supervised autoencoder to detect anomalies; the idea is to train the autoencoder on the normal data only, then use the reconstruction error and a threshold to classify unseen examples (binary classification: allowed classes are only normal and anomaly). This model does not work on time-series but rather in a "combinatorial" fashion: after having been trained it makes the prediction (thus classifying the data point) based on the single test examples fed to it (disregarding precious data points).

- **param** df_fname : string
 - name of the csv file containing the dataframe (data to be used for training and testing)
 - it must be a csv files, where each field is separated by a separator symbol (see 'separator' parameter) and each row corresponds to a timestamp
 - one column must be termed "label" and it must contain the class of the example - 0 means normal data point, any other integer number corresponds to an anomalous data point
- **param** (optional) separator : string
 - separator used inside the csv file
 - default value ','
- **param** (optional) user_id : string

⁶ <https://arxiv.org/pdf/2007.02500.pdf>

- user identifier
- **param** (optional) task_id : string
 - task identifier
- **param** (optional) hparams_file: string
 - name of the pickle file containing the python dictionary with hyperparameters to be used to build the autoencoder
- **param** (optional) n_percentile : int
 - percentile to be used to compute the detection threshold; if no percentile is provided all values in the range [85, 99] will be explored and the one providing the best accuracy results will be selected
- **return**: string
 - name of the file containing saved Keras model, that is the trained autoencoder
- **return**: string
 - name of the pickle file containing the scikit-learn object with scaler object used to normalize the data
- **return**: string
 - name of the pickle file containing python dictionary with the summary of the detailed statistics computed on the autoencoder, not necessarily related to the classification task (which depends on the threshold as well), e.g. the reconstruction error for every data point, and the accuracy results
 - the statistics contain the 'n_percentile' value computed and used as threshold; the statistics do not contain the values of the hyperparameters used for training and creating the model

This is an example of a python dictionary containing the hyperparameters values for the **semisup_autoencoder** function (these are the default parameters):

```
default_hparams = {
    'epochs': 20, # number of training epochs
    'batch_size': 64, # batch size
    'shuffle': True, # shuffle data during training
    'overcomplete': True, # the autoencoder can be overcomplete (if the
    number of neurons in the hidden layer is larger than the number of input
    features) or undercomplete (if the number of neurons in the hidden layer
    is smaller than the number of input features)
    'nl_o': 3, # number of layers in the overcomplete case
    'nl_u': 4, # number of layers in the undercomplete case
    'nnl_o': 10, # the number of neurons per layer in the overcomplete
    autoencoder is computed as n_features multiplied by nnl_o
    'nnl_u': 2, # the number of neurons per layer in the overcomplete
    autoencoder is computed as n_features divided by nnl_u
    'actv': 'relu', # activation function
    'loss': 'mae', # loss function
    'l1_reg': 0.00001, # l1 regularization factor (only for overcomplete
    case)
    'lr': 0.0001, # learning rate
    'optimizer': 'adam', # optimizer
    'drop_enabled': False, # add Dropout layer
    'drop_factor': 0.1 # Dropout rate (only if drop==True)
}
```

semisup_detection_inference - exploits a semi-supervised autoencoder to detect anomalies; the autoencoder model needs to have been previously trained and the detection-threshold computed. This function can be only used at "inference" time, when new data arrive and need to be checked for anomalies.

- **param** df_fname : string
 - filename of the data to be classified in either anomaly or normal points
 - it must be a csv files, where each field is separated by a separator symbol (see 'separator' parameter) and each row corresponds to a timestamp
 - no labels are provided in the data
 - the data need to have been normalized
- **param** model_name: string
 - name of the file containing model (autoencoder) already trained and stored
- **param** (optional) separator : string
 - separator used inside the csv file
 - default value ','
- **param** (optional) user_id : string
 - user identifier
- **param** (optional) task_id : string
 - task identifier
- **param** (optional) n_percentile : float
 - threshold to be used to distinguish between normal and anomalous points. It should have been computed while training the autoencoder.
 - optional: if missing, the original threshold used during the model's creation will be used (found in the file containing the statistics computed at training tie)
- **param** scaler_in : scikit-learn scaler object
 - the scaler used to normalize the data
- **return:** Python list
 - the computed labels expressed as a list of integer values, one label for each data point included in the input dataframe
 - 0 indicates normal points, 1 indicates anomalous ones

sup_autoencoder_classr - creates a supervised model for anomaly detection composed by an unsupervised autoencoder (feature extraction), then connected to a classifier layer trained in a supervised fashion.

- **param** df_fname : string
 - name of the file containing the dataframe (data to be used for training and testing)
 - it must be a csv files, where each field is separated by a separator symbol (see 'separator' parameter) and each row corresponds to a timestamp
 - one column must be termed "label" and it must contain the class of the example - 0 means normal data point, any other integer number corresponds to an anomalous data point
- **param** (optional) separator : string
 - separator used inside the csv file
 - default value ','
- **param** (optional) user_id : string
 - user identifier
- **param** (optional) task_id : string

- task identifier
- `param` (optional) `hparams_file_ae`: string
 - name of the pickle file containing the python dictionary with hyperparameters to be used to build the autoencoder
- `param` (optional) `hparams_file_classr`: string
 - name of the pickle file containing the python dictionary with hyperparameters to be used to build the classifier
- `return`: string
 - name of the file containing saved Keras model, that is the trained autoencoder
 - the model contains both the autoencoder and the classifier (merged in a single model)
- `return`: string
 - name of the pickle file containing the scikit-learn object with scaler object used to normalize the data
- `return`: string
 - name of the pickle file containing python dictionary with the summary of the detailed statistics computed on the autoencoder, not necessarily related to the classification task (which depends on the threshold as well), e.g. the reconstruction error for every data point, and the accuracy results
 - the statistics do not contain the values of the hyperparameters used for training and creating the model

The following are examples of the python dictionaries containing the hyperparameters values for the autoencoder network and the classifier defined by the `sup_autoencoder_classr` function (the following are the default parameters as well).

```
default_hparams_ae = {
    'epochs': 20, # number of training epochs
    'batch_size': 32, # batch size
    'shuffle': True, # shuffle data during training
    'overcomplete': False, # the autoencoder can be overcomplete or
undercomplete
    'nl_o': 3, # number of layers in the overcomplete case
    'nl_u': 4, # number of layers in the undercomplete case
    'nnl_o': 10, # the number of neurons per layer in the overcomplete
autoencoder is computed as n_features multiplied by nnl_o
    'nnl_u': 2, # the number of neurons per layer in the overcomplete
autoencoder is computed as n_features divided by nnl_u
    'actv': 'relu', # activation function
    'loss': 'mae', # loss function
    'l1_reg': 0.00001, # l1 regularization factor (only for overcomplete
case)
    'lr': 0.0001, # learning rate
    'optimizer': 'adam', # optimizer
    'drop_enabled': True, # add Dropout layer
    'drop_factor': 0.1 # Dropout rate (only if drop==True)
}
default_hparams_classr = {
    'epochs': 10, # number of training epochs
    'batch_size': 32, # batch size
```

```

'shuffle': True, # shuffle data during training
'nl': 2, # number of layers
'nnl': 2, # the number of neurons per layer is computed as n_features
divided by nnl
'actv': 'relu', # activation function
'loss': 'binary_crossentropy', # loss function
'lr': 0.0001, # learning rate
'optimizer': 'adam', # optimizer
'drop_enabled': False, # add Dropout layer
'drop_factor': 0.1 # Dropout rate (only if drop==True)
}

```

sup_detection_inference - exploits a supervised classifier to detect anomalies; the classifier model needs to have been previously trained. This function can be only used at "inference" time, when new data arrive and need to be checked for anomalies.

- **param** df_fname : string
 - filename of the data to be classified in either anomaly or normal points
 - it must be a csv files, where each field is separated by a separator symbol (see 'separator' parameter) and each row corresponds to a timestamp
 - no labels are provided in the data
 - the data need to have been normalized
- **param** model_name: string
 - name of the file containing model (autoencoder plus classifier) already trained and stored
- **param** (optional) separator : string
 - separator used inside the csv file
 - default value ','
- **param** (optional) user_id : string
 - user identifier
- **param** (optional) task_id : string
 - task identifier
- **param** (optional) scaler_in : scikit-learn scaler object
 - the scaler used to normalize the data
- **return**: Python list
 - the computed labels expressed as a list of integer values, one label for each data point included in the input dataframe
 - 0 indicates normal points, 1 indicates anomalous ones

3.2.3 Software Requirements

The supervised and semi-supervised anomaly detection services were developed using Python language (Python >= 3.4) as the general services. In this case as well non-native Python libraries were used, especially those required for creating neural networks and Deep Learning models. Again, the containers are endowed with all required modules, which are the following:

For the sake of clarity and replicability, we list here the requirements of the services:

- Python 3.4

- Python modules:
 - o numpy==1.19.1
 - o pandas==1.1.0
 - o scipy==1.5.2
 - o scikit-learn==0.23.2
 - o Tensorflow==1.14
 - o Keras==2.4.0

3.3 Unsupervised abnormal time series identification and outliers detection

This section describes two methods to analyse multi-variate time series, both working with time-series represented as batches:

1. Detect abnormal time series in a corpus of time series, using Functional Isolation forest (Staerman, Mozharovskiy, Cléménçon, & d'Alché-Buc, 2019)
2. Detect outliers and change points in a time series with ADTK Python module (ADTK: <https://github.com/arundo/adtk>, 2020)

3.3.1 Functional Isolation Forest

Isolation Forest (Liu, Ting, & Zhou, 2008)⁷ is a powerful unsupervised anomaly detection algorithm. It is based on tree ensemble and is based on decision trees. In these trees, partitions are created by first randomly selecting a feature and then selecting a random split value between the minimum and maximum value of the selected feature. In principle, abnormal points are less frequent than regular observations and are different from them in terms of values. That is why by using such random partitioning they should be identified closer to the root of the tree (shorter average path length, i.e., the number of edges an observation must pass in the tree going from the root to the terminal node), with fewer splits necessary. To gain in robustness, we train several isolation tree and compute the anomaly score : $s(x, n) = 2^{-E(h(x))/c(n)}$, where x is an observation, n the number of instances, $E(h(x))$ the expectation of the path length $h(x)$ of x which is measured by the number of edges x traverses an Isolation Tree from the root node until the traversal is terminated at an external node and $c(n)$ a normalization constant. This score is between 0 (normal point) and 1 (abnormal point). If the score of all observation are nearly 0.5, then there is no anomaly. The hyperparameters are the number of isolation tree and a subsampling parameter, because above a threshold, it is useless to keep all data. For each Isolation Tree a new subsample is randomly chosen. The evaluation stage consists in passing a new observation x in all the Isolation Tree and make an empirical estimation of $E(h(x))$.

An adaptation to time series of Isolation Forest (Staerman, Mozharovskiy, Cléménçon, & d'Alché-Buc⁸) is called the Functional Isolation Forest. The idea behind their article is to project the time series in a well-chosen dictionary according to a chosen scalar product and follow the same idea as Isolation Forest. The dictionary is chosen to be rich enough to explore different properties of data and well suited to be sampled in a representative manner. The projection of a time series on each member of the dictionary defines a feature that partially describes the time series. When considering all the functions of the dictionary, one gets a set of candidates split variables that provides a rich representation of the time series, depending on the nature

⁷ Liu, Ting, & Zhou. (2008). Isolation forest. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*.

⁸ Staerman, Mozharovskiy, Cléménçon, & d'Alché-Buc. (2019). Functional Isolation Forest.

of the dictionary. Dictionaries have been thoroughly studied in the signal processing community to achieve sparse coding of signals. They provide a way to incorporate a priori information about the nature of the data, and so expert knowledge. The choice of scalar products is driven by the anomalies we search. L2 scalar product allows for detection of location anomalies and the L2 scalar product of derivatives (or slopes) allows to detect anomalies regarding shape. It is possible to use a combination of these two scalar products with Sobolev scalar product.

3.3.2 ADTK

ADTK⁹ is a package for unsupervised/rule-based models of time series anomaly detection. It offers a set of common components that can be combined into various types of anomaly detection models for different scenarios. It allows to detect some classical anomalies type in time series:

- Outlier: data point whose value is significantly different from others
- Spike and Level Shift: An abrupt increase or decrease of value is called a spike if the change is temporary, or a level shift if the change is permanent
- Pattern change: change in volatility, in seasonality, in autoregressive relationship, etc.

3.3.3 APIs

A Dash application allows users to interact with the functions presented above. The Dash application and its parameters is described in subsection 3.5.2.

3.3.4 Software requirements

As in the case of the other services, the unsupervised anomaly detection for time-series services have been developed and deployed as Docker container. These services were developed using python language (python ≥ 3.8) and using additional libraries and python modules, detailed in the following list:

- python 3.8
- python modules:
 - o Plotly $\geq 4.6.0$
 - o dash $\geq 1.14.0$
 - o dash-table ≥ 4.9
 - o dash-upload-components $\geq 0.0.2$
 - o dash-core-components $\geq 1.10.1$
 - o dash-html-components $\geq 1.0.3$
 - o Flask $\geq 1.1.2$
 - o boto3 $\geq 1.14.36$
 - o botocore $\geq 1.17.36$
 - o numpy $\geq 1.19.1$
 - o pandas $\geq 1.1.0$
 - o scipy $\geq 1.5.2$
 - o gunicorn $\geq 19.9.0$

⁹ ADTK: <https://github.com/arundo/adtk>. (2020)

- o scikit-learn==0.23.2
- o urllib3==1.25.10
- o dill==0.3.2
- o adtk==0.6.2
- o cufflinks==0.17.3

3.4 Physics simulation and fault modelling services

To complement the ML-based functionalities being developed, services are being developed for IoTwinS that will enable the different test-bed users to exploit functionality based on physics-based simulation, fault-modelling and reduced-order modelling techniques.

Reduced-order modelling (ROM) methods have attracted increasing attention in recent years as a way to high-fidelity 3D simulation “in-time” to inform decision making about the operational performance or health of in-service assets.

System simulation offers an alternative way to simulate the physical behavior of products in operation. A key benefit of system simulation approaches is that components from heterogeneous physical domains are connected in the same model to represent the real system architecture. Since they do not use the geometric representation of the constituent components, the solution of system simulation models is generally fast, making them good candidates to fulfill the in-time computational requirements of the various IoTwinS test-beds. ESI’s SimulationX¹⁰ is a system simulation tool that offers a wide range of functionalities and model libraries for several different physical domains, including thermal, mechanics, hydraulics, fluid and electrical. Recently, a utility and accompanying model library (System Reliability Analysis) have been developed for SimulationX that enables users to explore the effects of fault on physical systems through system simulation¹¹. The new tool allows the user to augment their nominal system simulation models with additional model components that represent faults in, or between, different components in the system in order to represent degradation or failure in the physical system. The resulting fault-augmented model is parameterized in such a way that the user can vary the degree of fault present in different model components between e.g. 0 (nominal operation) and 1 (completely faulty). Within IoTwinS, system simulation methods using fault-augmented models are being used in the test-bed no. 1 for “Wind-farm predictive maintenance system”.

System simulation services hosted on the ESI Cloud platform will, amongst other things, allow the user to upload existing (fault-augmented) system simulation models as well as a simulation service for running fault-augmented simulation models, thus realizing a kind of reliability analysis simulation engine.

A number of the simulation and fault-modelling services that are being developed for IoTwinS will be hosted on ESI’s proprietary cloud platform. In this way, the ESI cloud platform manages the computational resources required for running the services as well as any requisite licence. To provide access to these functionalities for the different test-beds, “bridge” services are being developed, they will manage the interaction between the IoTwinS and ESI Cloud platforms. In implementing the connection to an externally hosted platform, the IoTwinS partners involved in WP3 aim to demonstrate the extensibility of the IoTwinS platform.

¹⁰ <https://www.simulationx.com/>

¹¹ A. Kolesnikov et al Systematic Simulation of Fault Behavior by Analysis of Vehicle Dynamics, Proceedings of the 13th International Modelica Conference 2019, doi: 10.3384/ecp19157451

In the following sections, sample functionality for generation and exploitation of reduced-order modelling techniques that has been developed for the ESI Cloud platform is shown. In the examples shown below, the services are demonstrated as they exist within the ESI Cloud platform. In each case, a bridge service is under development to enable the connection to the IoTwinS platform.

3.4.1 Reduced-order model DoE generator

In order to build a model learner based on non-intrusive reduced-order model techniques, it is necessary to generate a design of experiment (DoE) for the simulations required to train the model, which connects the parameters of the model (inputs) to its quantities of interest (outputs). The notion of “non-intrusivity” in this case describes the fact that these methods do not require any modification of the solver itself. Instead, the model is constructed based purely using the simulation results. A clear benefit of this approach is that the ROM methods are, in general, agnostic to the simulation tool used to generate the results.

The MOR (model-order reduction) application on the ESI Cloud platform offers a number of different ROM techniques to the user according to their requirements. Depending on the method chosen, a DoE will be generated using the parameters and quantities of interest identified by the user. Figure 3.1 below shows this process:

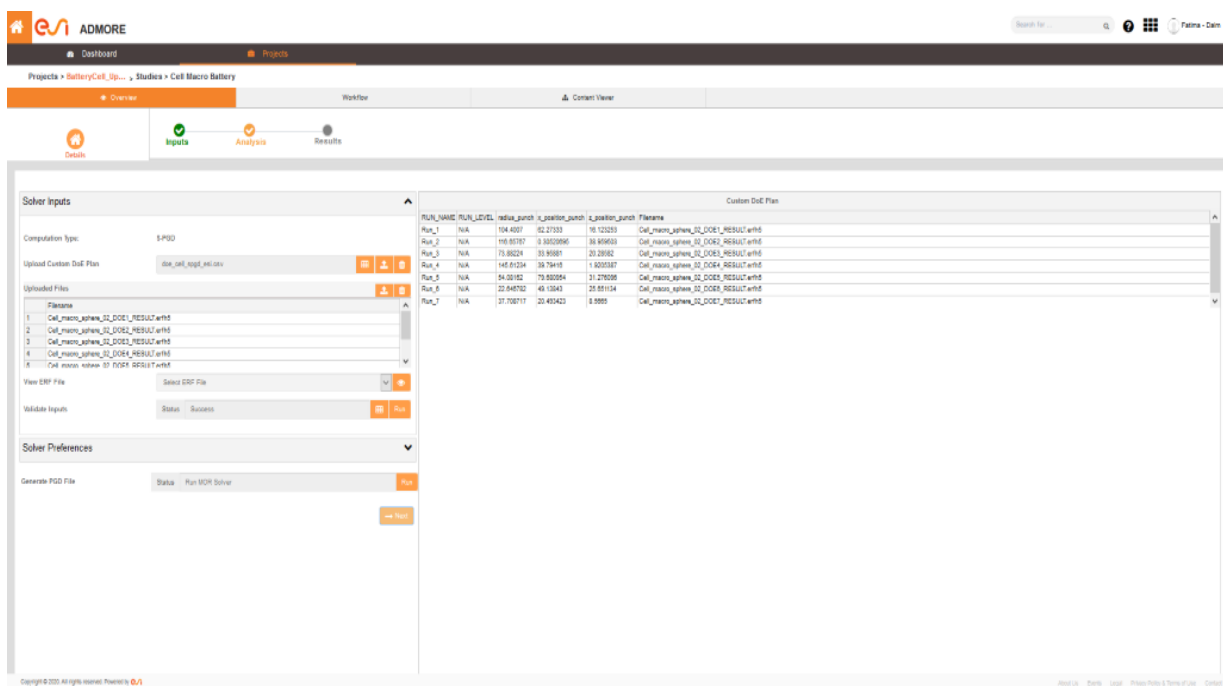


Figure 3.1: Generation of a DoE

3.4.2 Reduced-order model generator

Once the simulations listed in the generated DoE have been run, the collection of simulation results files should be uploaded to the MOR application. Results files of either ERFH5 (ESI results files based on H5 file format) or CSV formats are accepted. The application will then generate the ROM according to the chosen

method: sPGD¹² (sparse proper generalized decomposition) or SSL-PGD¹³ (sparse subspace learning proper generalized decomposition). Figure 3.2 shows this process.

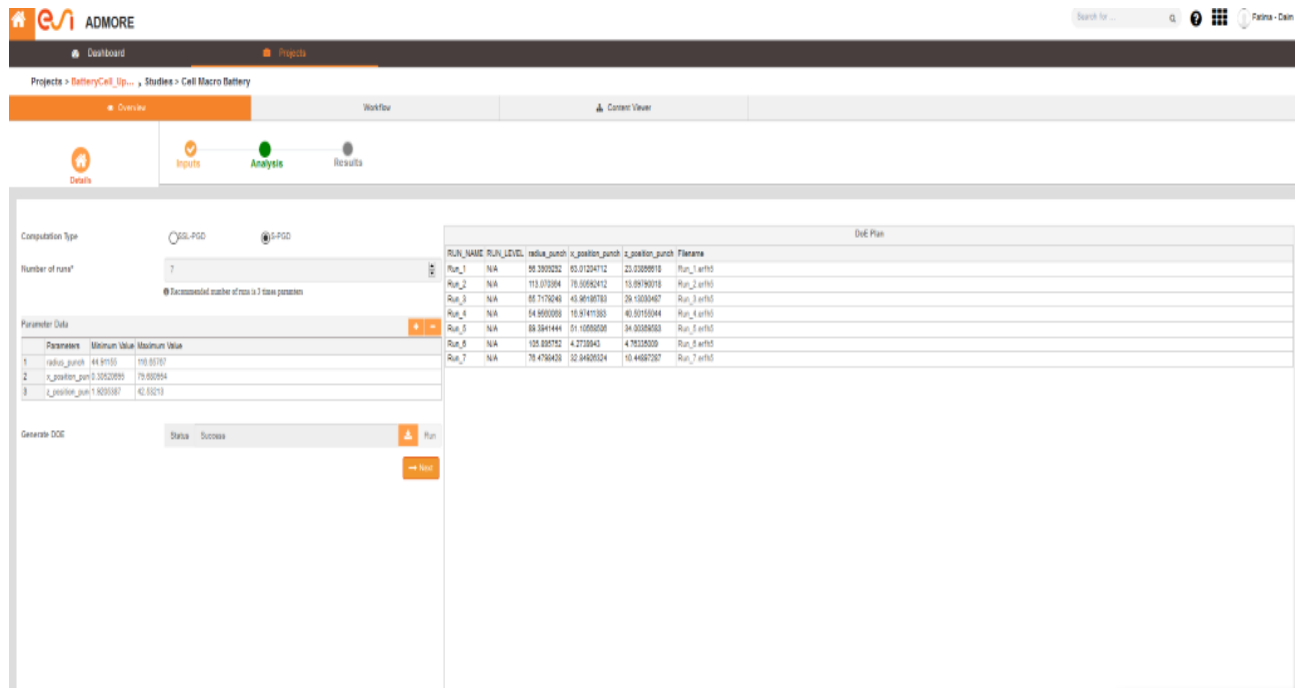


Figure 3.2: Generation of a reduced-order model

Once the ROM construction has been run, a parametric solution is stored in the MOR application for later exploitation by the user or their collaborators. This parametric solution is an abacus where quantities of interest can be computed for any set of parameter values that lie within the range of the parameters of the study. Depending on the method chosen, the quantities of interest may be calculated on the same level of geometric discretization (mesh) as that used for the original simulations.

¹² V. Limousin, X. Delgerie, E. Leroy, R. Ibáñez, C. Argerich, F. Daim, J.-L. Duval and F. Chinesta, Advanced model order reduction and artificial intelligence techniques empowering advanced structural mechanics simulations: application to crash test analyses, *Mechanics & Industry* 20, 804 (2019). <https://doi.org/10.1051/meca/2020009>

¹³ D. Borzacchiello, J.V. Aguado, & F. Chinesta, Non-intrusive Sparse Subspace Learning for Parametrized Problems. *Arch Computat Methods Eng* 26, 303–326 (2019). <https://doi.org/10.1007/s11831-017-9241-4>

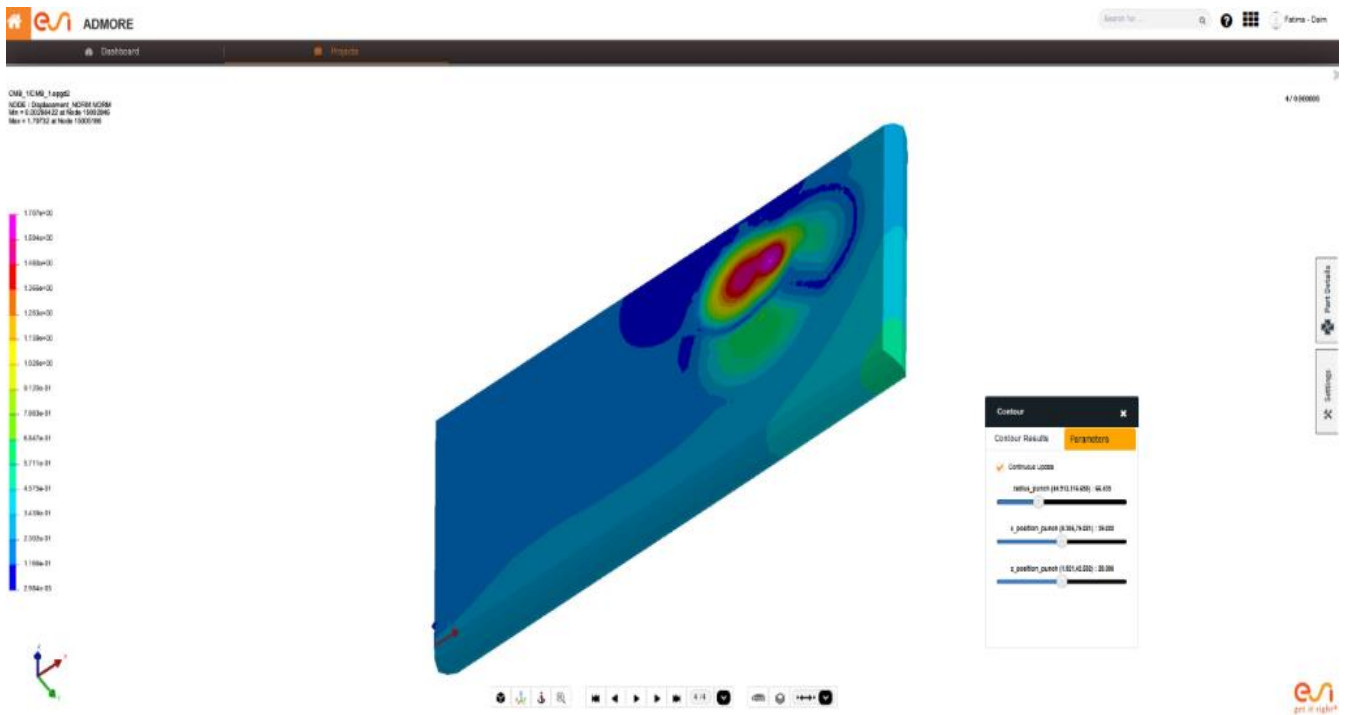


Figure 3.3: Exploitation of a parametric solution

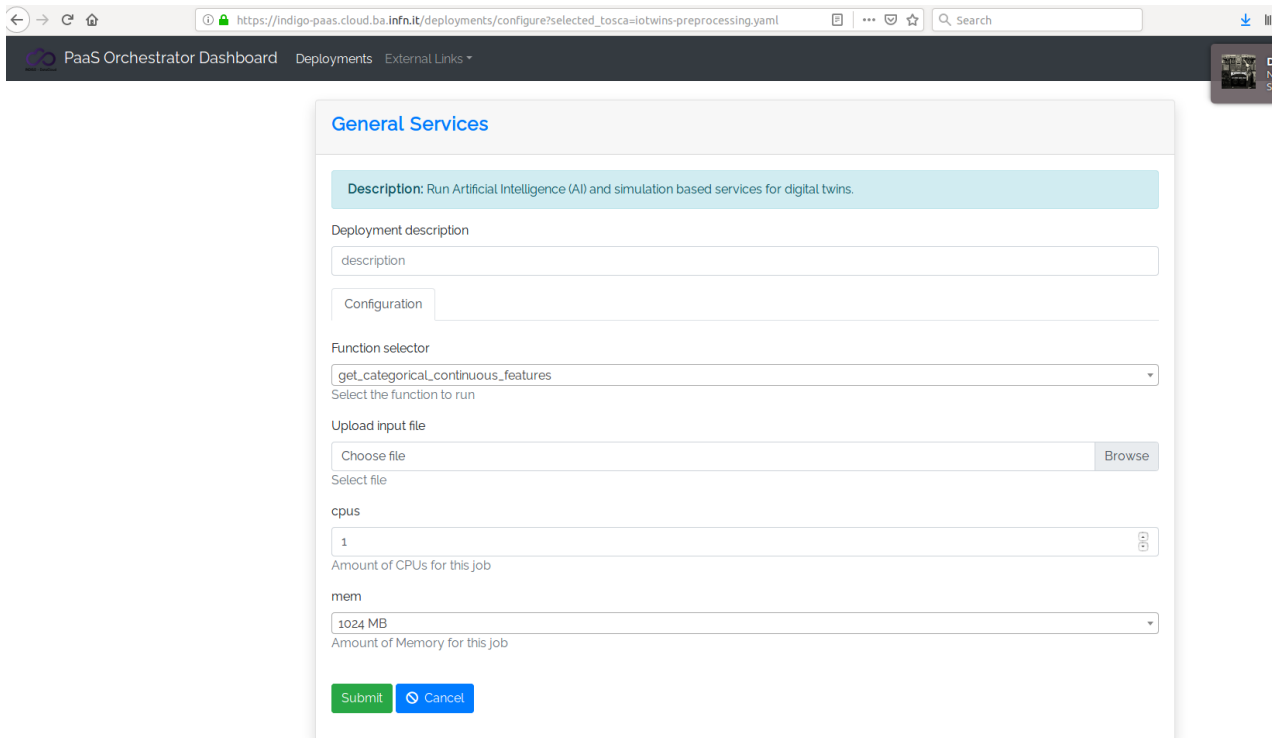
3.5 Services usage examples

In this section we will describe how to use some of the services previously introduced. We will not describe every single service, but we will focus on the most exemplary ones.

3.5.1 General services usage

The general services are relatively simple software components that perform basics tasks of data preparation, with the purpose of making the data suitable for usage in a Machine Learning setting.

The use of the general services begins by selecting them on the demonstrator dashboard, by clicking on the corresponding button ("General Services – IoTwinS"). After the selection, the following view (Fig. 3.4) is displayed:



General Services

Description: Run Artificial Intelligence (AI) and simulation based services for digital twins.

Deployment description

description

Configuration

Function selector

get_categorical_continuous_features

Select the function to run

Upload input file

Choose file Browse

Select file

cpus

1

Amount of CPUs for this job

mem

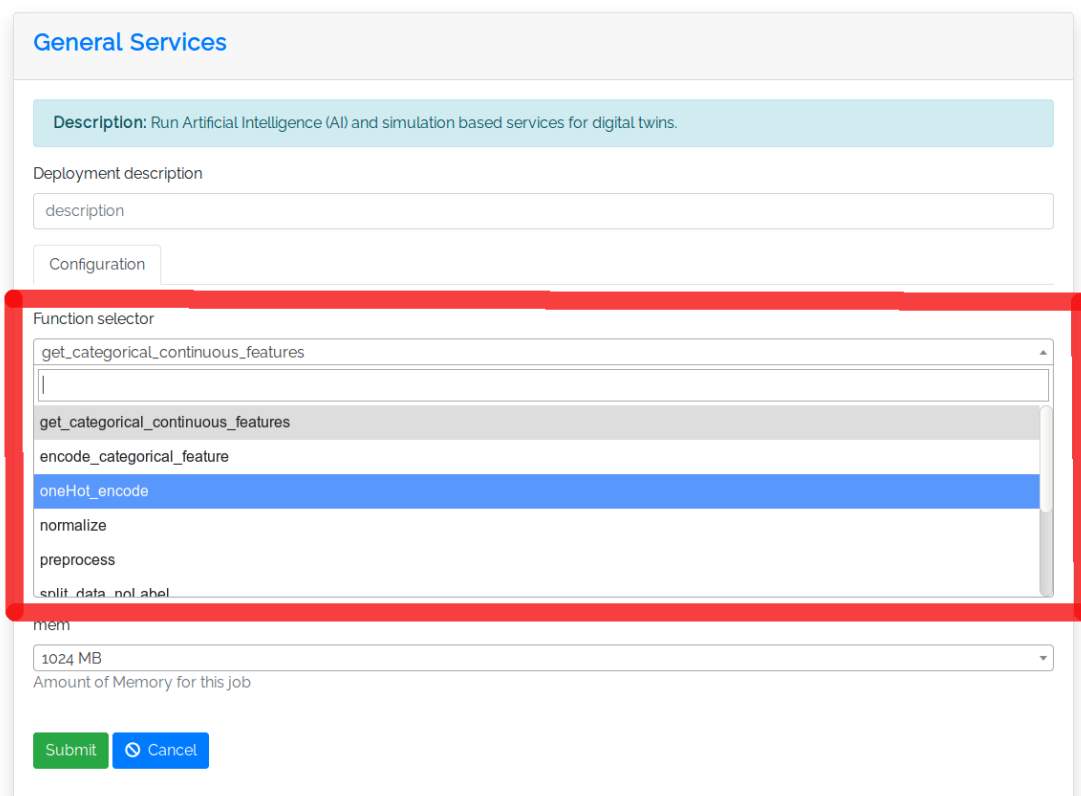
1024 MB

Amount of Memory for this job

Submit Cancel

Figure 3.4: General service configuration view

In this webpage the user must specify the name for the instance of the service to be created, the number of CPUs required and the amount of memory (RAM) desired, acting on the relative buttons. Additionally, in this form, it is possible to select the desired service among a list of candidates, using the drop-down button “Function selector” (as reported in Fig. 3.5).



General Services

Description: Run Artificial Intelligence (AI) and simulation based services for digital twins.

Deployment description

description

Configuration

Function selector

get_categorical_continuous_features

get_categorical_continuous_features

encode_categorical_feature

oneHot_encode

normalize

preprocess

split_data_not_label

mem

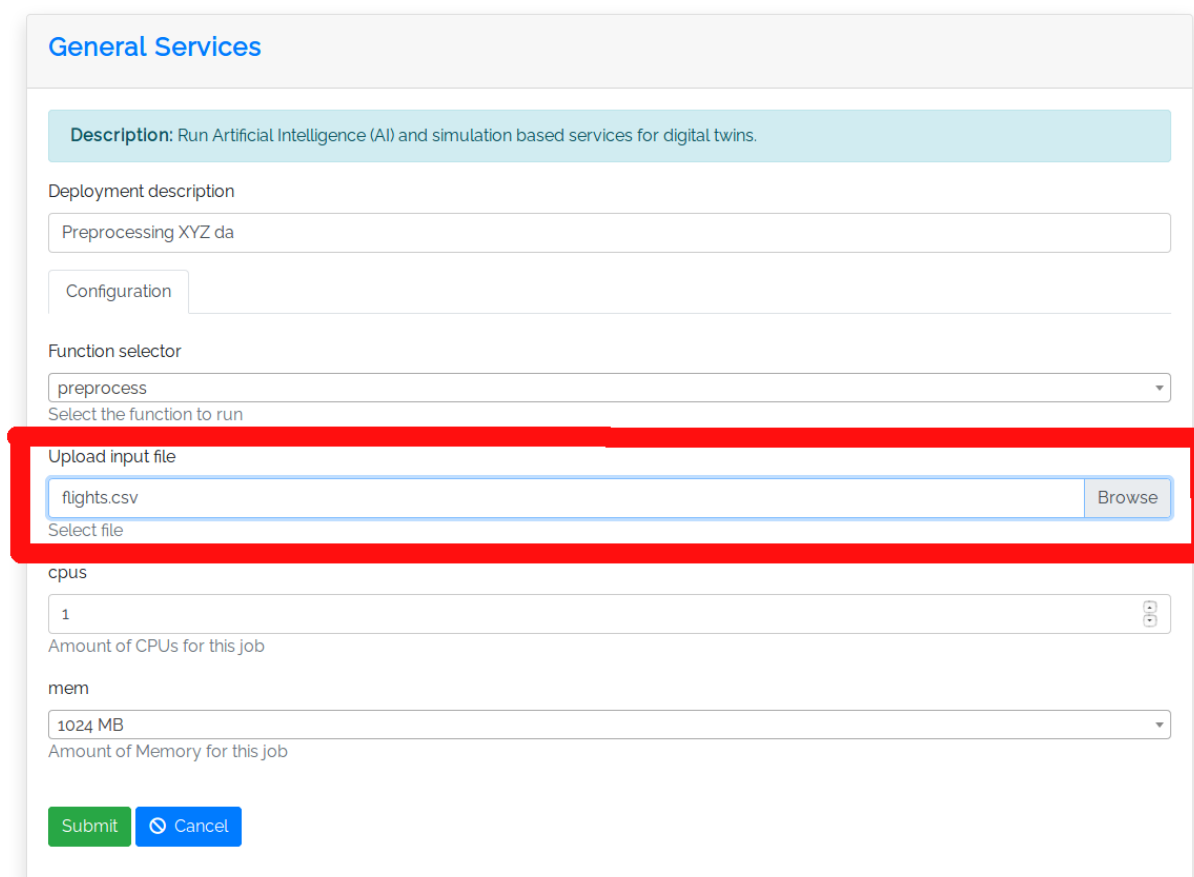
1024 MB

Amount of Memory for this job

Submit Cancel

Figure 3.5: General service configuration (service selection)

After having selected the desired general service, the data to be processed must be provided (as shown in Fig. 3.6). As aforementioned, these services expect a csv file as input.



General Services

Description: Run Artificial Intelligence (AI) and simulation based services for digital twins.

Deployment description
Preprocessing XYZ da

Configuration

Function selector
preprocess
Select the function to run

Upload input file
flights.csv
Browse

Select file

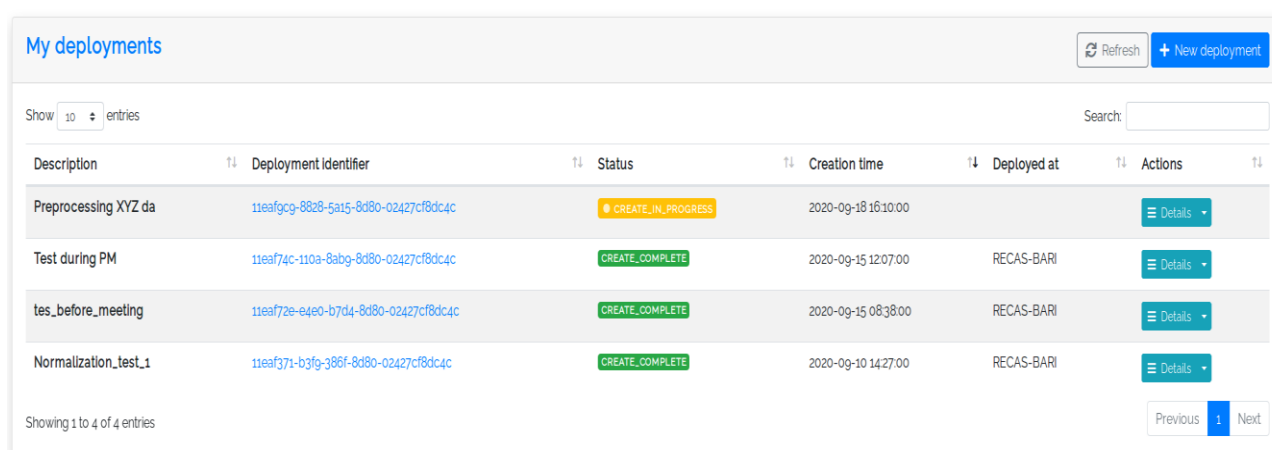
cpus
1
Amount of CPUs for this job

mem
1024 MB
Amount of Memory for this job

Submit Cancel

Figure 3.6: General service configuration (uploading data)

Finally, the user can either submit the request to execute the service, by clicking on the “Submit” green button or decide to cancel the request by clicking on the blue “Cancel” button. In case of submission, the PaaS orchestrator build an instance of the docker container with the specified service and run the newly created container when its hardware resources requirements can be met (that is, when enough spare resources are available). Until the resources are not available the request is in the “pending” status (recognizable from the yellow highlighting rectangle in Fig. 3.7); once the resources are available the container is deployed and the services commences its execution.



My deployments Refresh + New deployment

Show 10 entries Search:

Description	Deployment Identifier	Status	Creation time	Deployed at	Actions
Preprocessing XYZ da	11eafgcg-8828-5a15-8d80-02427cf8dc4c	CREATE_IN_PROGRESS	2020-09-18 16:10:00		Details
Test during PM	11eaf74c-110a-8abg-8d80-02427cf8dc4c	CREATE_COMPLETE	2020-09-15 12:07:00	RECAS-BARI	Details
tes_before_meeting	11eaf72e-e4e0-b7d4-8d80-02427cf8dc4c	CREATE_COMPLETE	2020-09-15 08:38:00	RECAS-BARI	Details
Normalization_test_1	11eaf371-b3fg-386f-8d80-02427cf8dc4c	CREATE_COMPLETE	2020-09-10 14:27:00	RECAS-BARI	Details

Showing 1 to 4 of 4 entries Previous Next

Figure 3.7: General services submitted and waiting to be executed

The results produced as output by the service (see the API specification for details) is then available as a downloadable compressed archive, hosted in the cloud storage space, where it can be retrieved by users by selecting the "Details" action from the right-most column, and then choosing the output tab – as shown in Fig. 3.8.

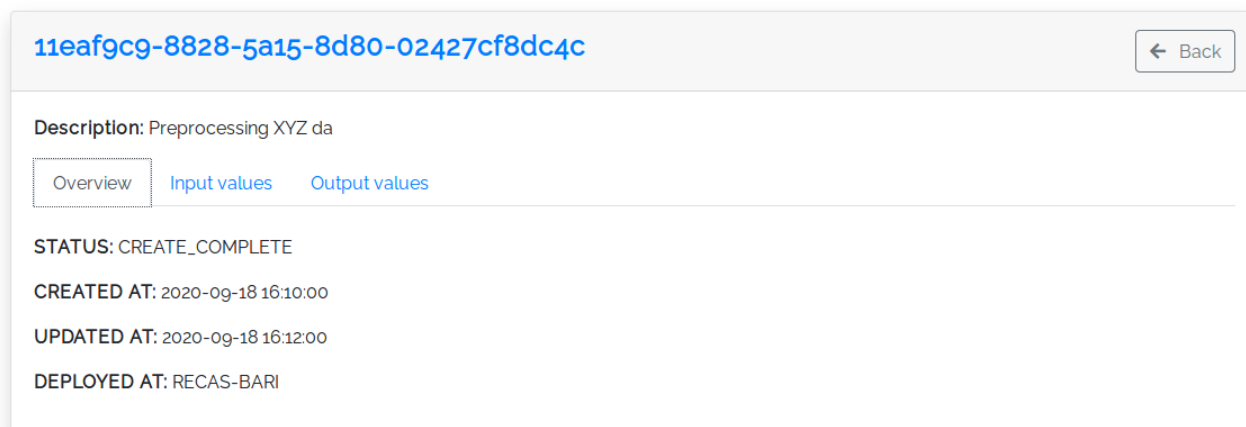


Figure 3.8: General service overview

3.5.2 Anomaly detection in multi-variate time series using Functional Isolation Forest

When the application is running, the user can open it in a browser, as displayed in Fig. 3.9. On the first tab, there is the sub-component that allows detecting abnormal time series in a corpus of time series, based on Functional Isolation Forest.

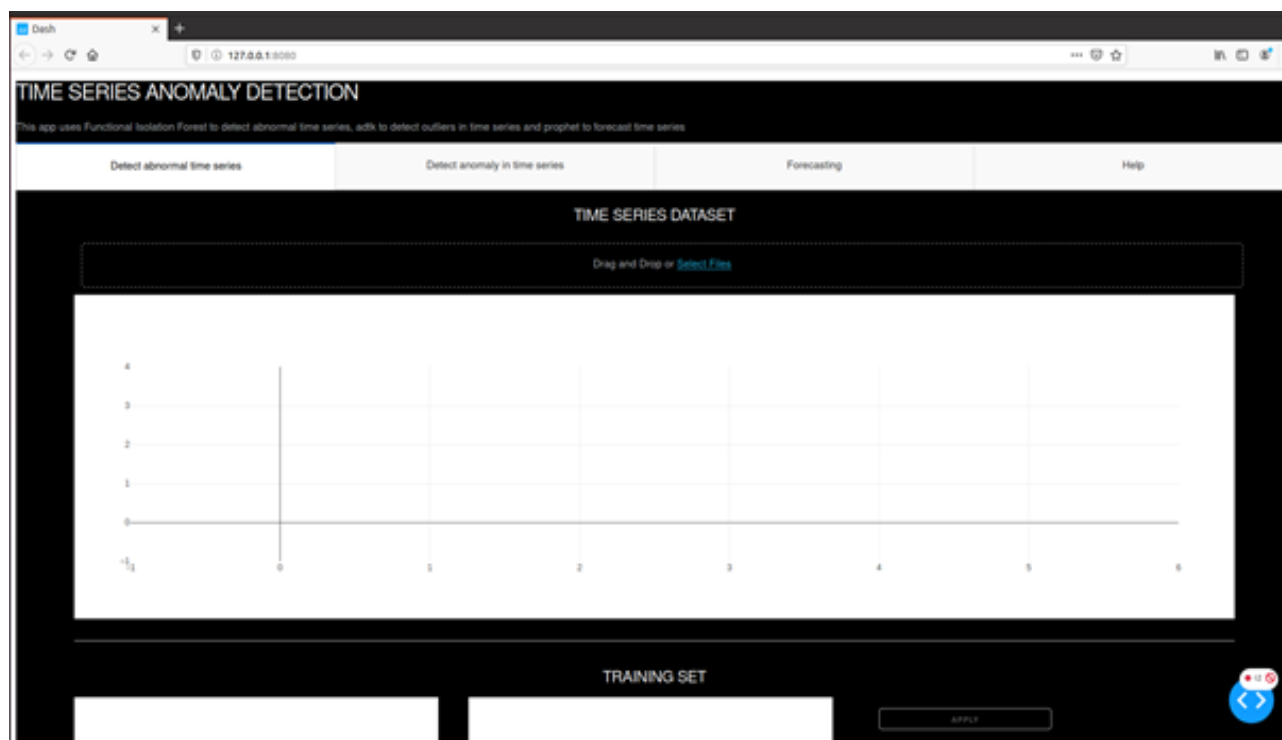


Figure 3.9: Anomaly detection with FIF service overview

With the button “Drag and Drop data or select files”, the user can load some data (Fig. 3.10).

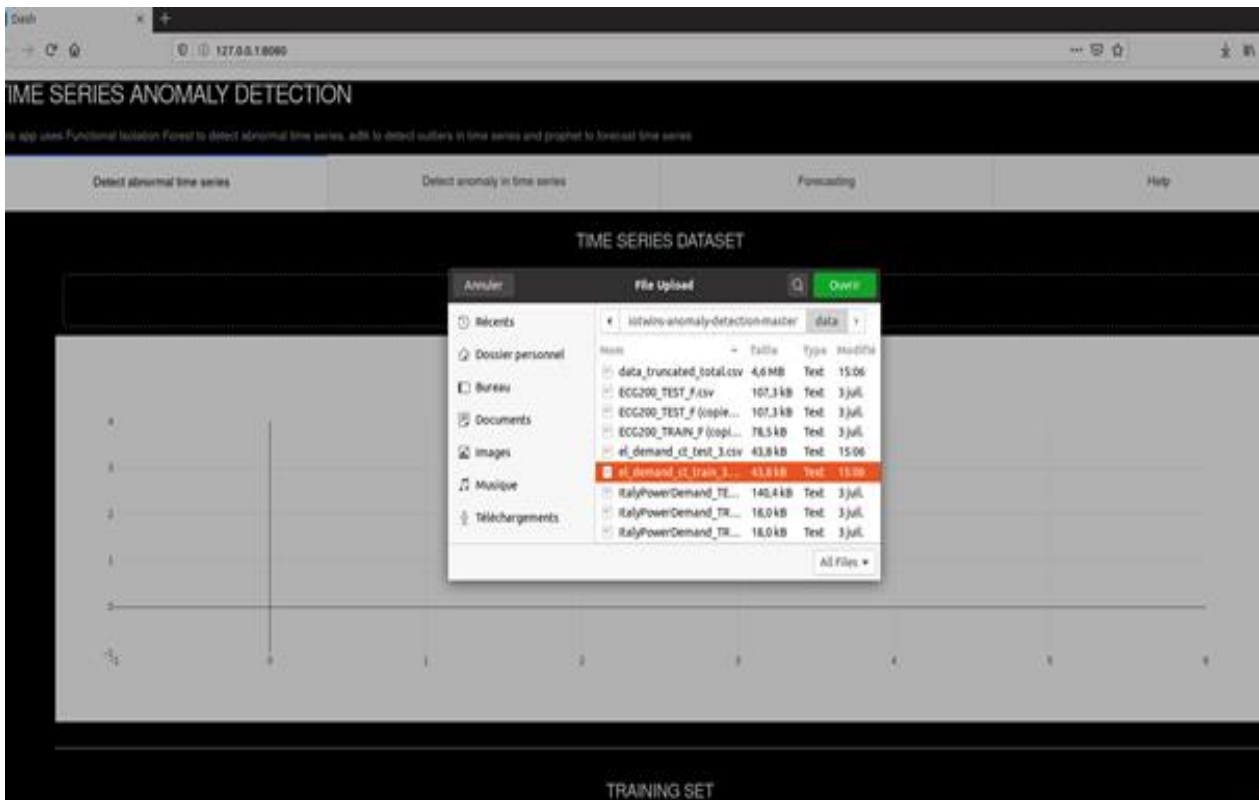


Figure 3.10: Anomaly detection with FIF service (data load)

The format of the data should have for rows a time series and for columns the time, with no column header – as shown in Fig. 3.11.

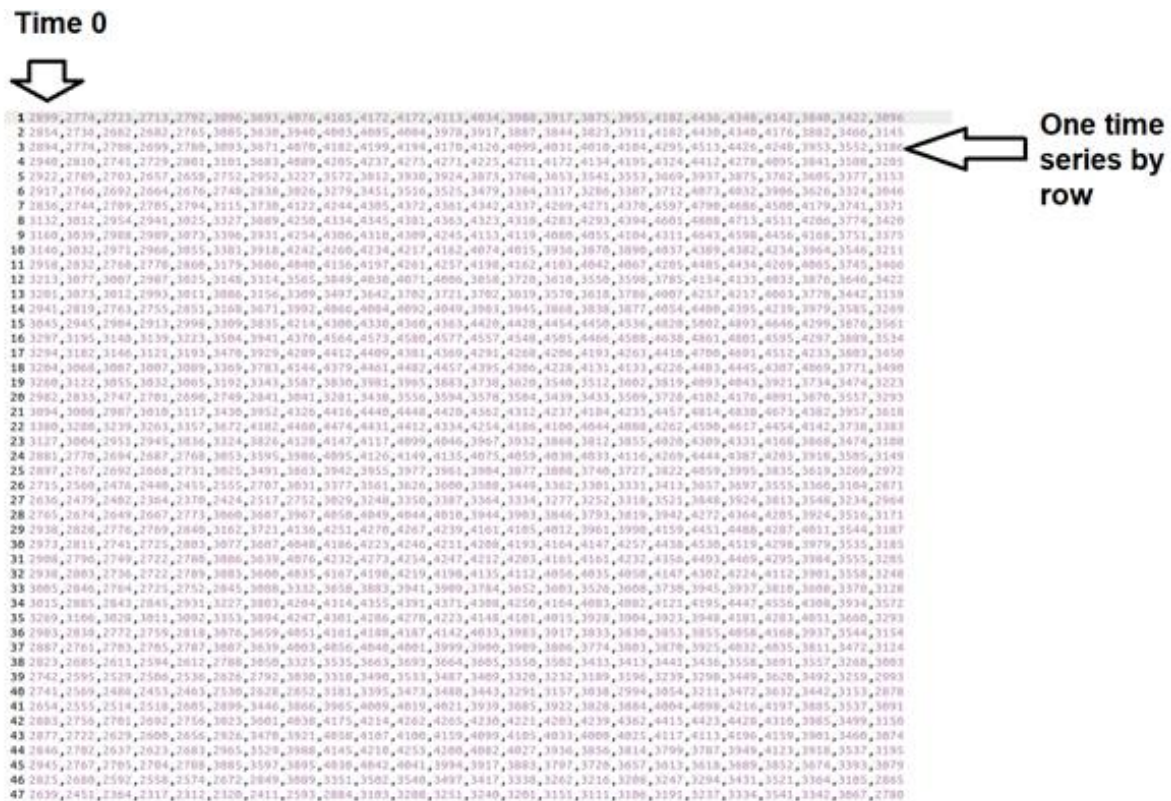


Figure 3.11: Anomaly detection with FIF service (data format)

When the data are loaded, the time series of the corpus are plotted on the graph and the first time series values are given in the table (Fig 3.12).

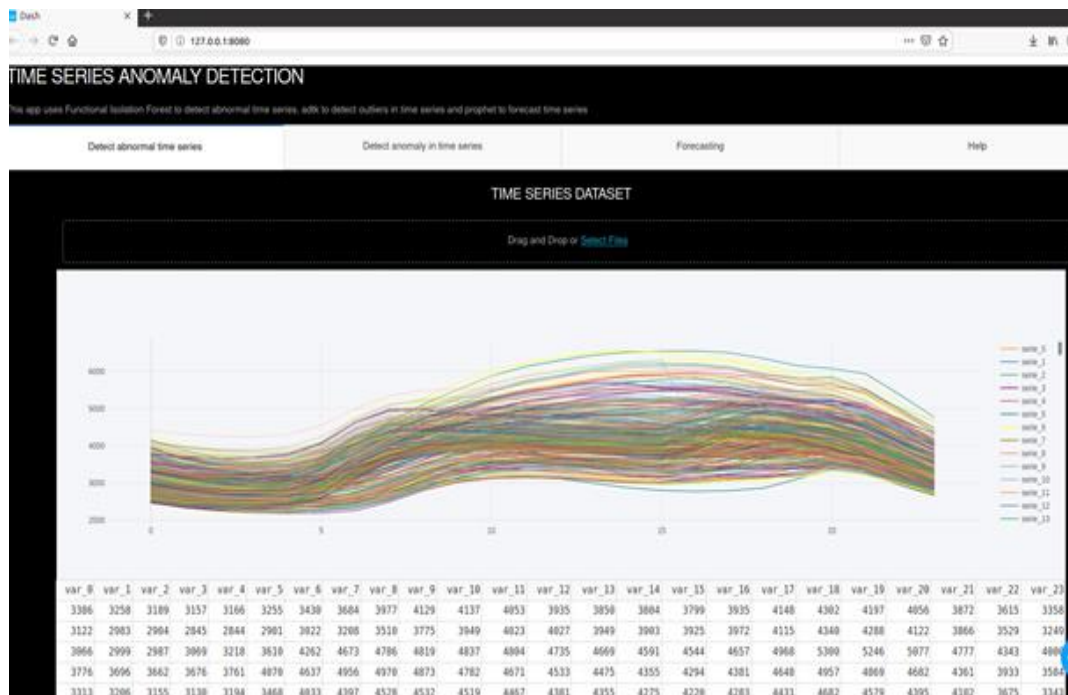


Figure 3.12: Anomaly detection with FIF service (displaying the data)

By scrolling down the tab, the user gets to the Functional isolation forest hyperparameters (Fig. 3.13). Between several possible dictionary one can choose the scalar products. This value corresponds to the trade-off in the Sobolev scalar products: when selecting 1, it corresponds to the L2 norm for the function, 0 to the L2 norm for the derivate, and between 0 and 1 the weight given to the two L2 norm. The user can choose the percentage of abnormal time series targeted. If she indicated 4% for instance, the 4% time series with the highest anomaly score will be considered as abnormal. When the user has chosen the hyperparameters, one can click on apply button.

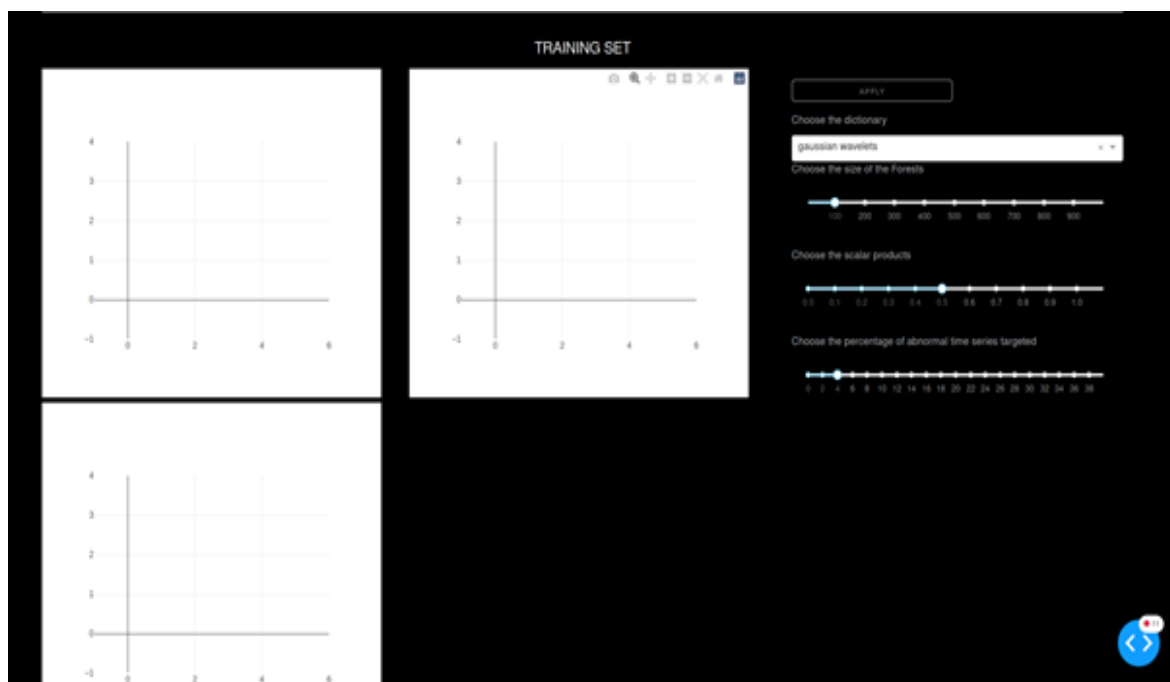


Figure 3.13: Anomaly detection with FIF service (hyperparameter selection)

When the Functional Isolation Forest is learnt, 3 plots and one table are given, displayed by Fig. 3.14. On the top left chart, all the time series considered as normal are plotted. On the top right chat, the abnormal time series are plotted. For these two charts, it is possible to select some series by double clicking on their legend. On the bottom left chart, the anomaly score of each series is represented. In the Table, the series with strongest anomaly score is given. It is possible to wave the charts, zoom some points, etc.

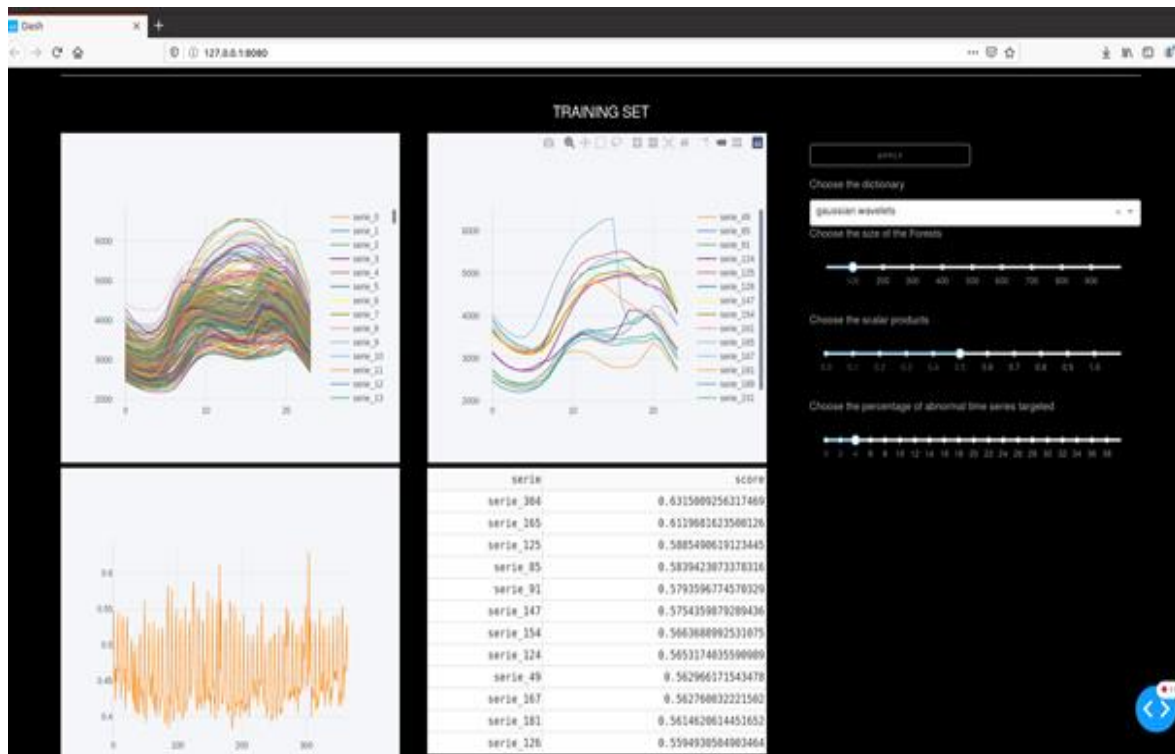


Figure 3.14: Anomaly detection with FIF service (after training)

It is possible to use the Functional Isolation Forest, which has been previously learnt, on a new data set of time series – this is a task commonly known in ML terminology as *transfer learning*. For that, the user has to scroll down the tab again (shown in Fig. 3.15). The only parameter one has to choose is the percentage of abnormal time series targeted. By clicking on “Drag and Drop or Select Files”, the user can choose the new time series corpus, which has to respect the same format that the previous data format. The given charts and Table are the same as for the training set.

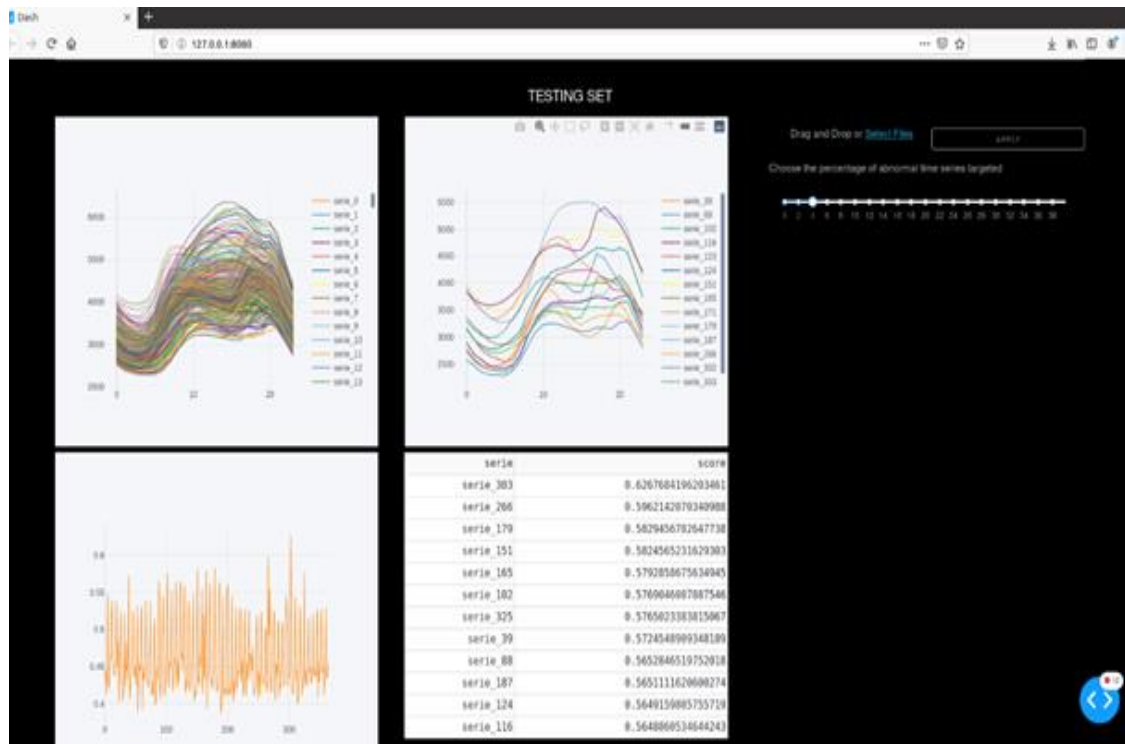


Figure 3.15: Anomaly detection with FIF service (transfer learning)

3.5.3 Outliers and change points detection with ADTK

The second tab offers the dashboard for using ADTK. The objective here is to detect outliers and change points in one time series (see Fig. 3.16).

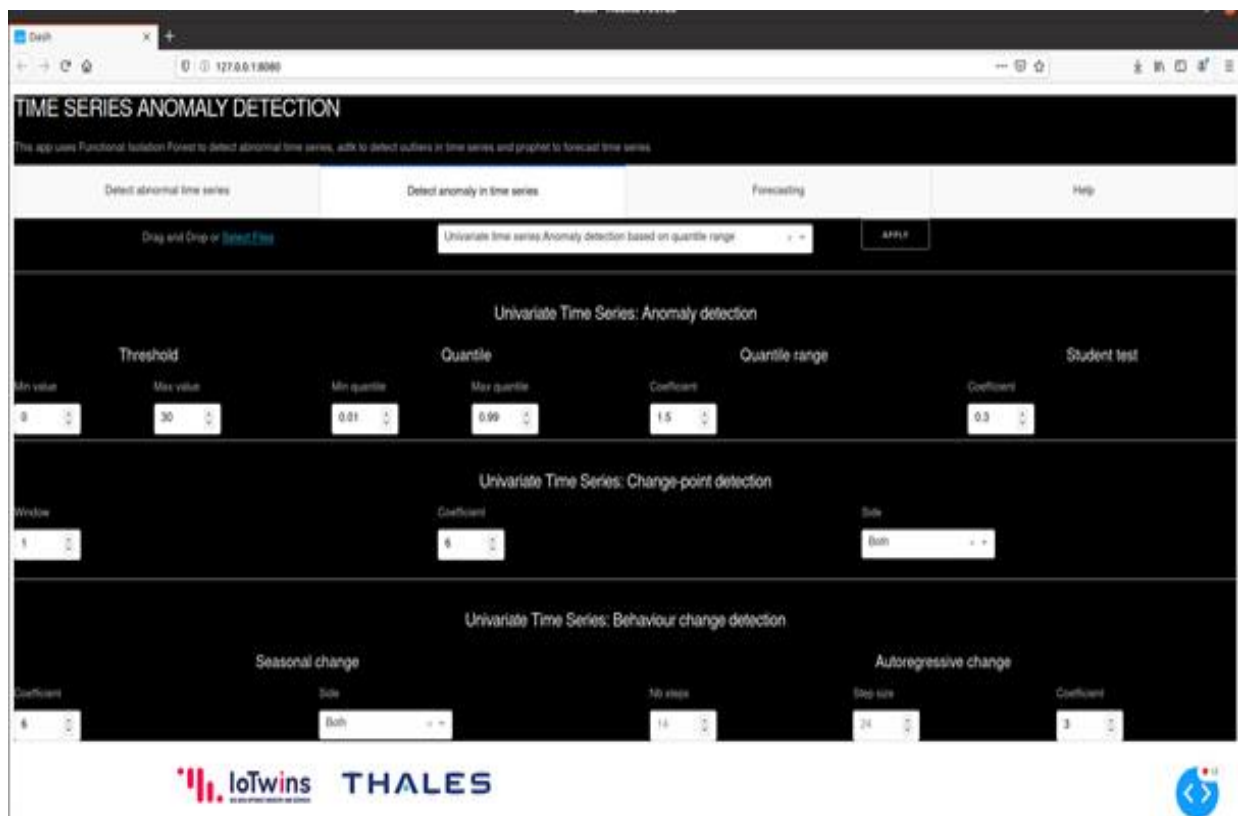


Figure 3.16: Anomaly detection with ADTK service overview

In Table 3.1 we describe the hyperparameters of the dashboard which allows to use the ADTK module (see Figure 3.16), and to perform different types of analysis on the time-series data. The column 'Objective' consists in an item from the drop-down list. The column 'Description' describes what the approach does. The column 'Hyperparameter' lists the hyperparameters, which are associated to the objective and the column 'Hyperparameter Description' describes each hyperparameter.

Objective	Description	Hyperparameter	Hyperparameter Description
Anomaly detection based on threshold	The value smaller and greater than two thresholds are considered as abnormal	Threshold min value	Value for which the lower values are considered abnormal
		Threshold max value	Value for which the higher values are considered abnormal
Anomaly detection based on quantile	The value smaller and greater than two chosen quantiles are considered as abnormal	Quantile min value	Quantile for which the lower values are considered abnormal
		Quantile max value	Quantile for which the higher values are considered abnormal
Anomaly detection based on quartile range	A value is said abnormal when it is out of the range defined by $[Q1 - c \times IQR, Q3 + c \times IQR]$ where $IQR = Q3 - Q1$ is the difference between 25% and 75% quantiles	Quantile range coefficient	<p>Coefficient c of objective description. Default value is 3.</p> $\min\left(\frac{Q_1 - a}{IQR}, \frac{b - Q_3}{IQR}\right) \geq c$ $\max\left(\frac{a - Q_3}{IQR}, \frac{Q_1 - b}{IQR}\right) \leq c$ <p>With a and b respectively the min and max of the values considered.</p>
Anomaly detection based on student test	Detects anomaly based on generalized extreme Studentized deviation (ESD) test. Note a key assumption of generalized ESD test is that normal values follow an approximately normal distribution	Student test coefficient	Significance level. Classical value is 0.05.
Change point detection based on recent observation	Compares each time series value with its previous values	Univariate time series: change point detection window	Size of the preceding time window. Default is 1.
		Univariate time series: change point detection coefficient	Factor used to determine the bound of normal range based on historical interquartile range.
		Univariate time series: change point detection side	<p>If "both", to detect anomalous positive and negative changes. If "positive", to only detect anomalous positive changes. If "negative", to only detect anomalous negative changes.</p>

Objective	Description	Hyperparameter	Hyperparameter Description
Change point detection based on level shift	Detects shift of value level by tracking the difference between median values at two sliding time windows next to each other.	Univariate time series: change point detection window	Size of the preceding time window. Default is 1.
		Univariate time series: change point detection coefficient	Factor used to determine the bound of normal range based on historical interquartile range.
		Univariate time series: change point detection side	If "both", to detect anomalous positive and negative changes. If "positive", to only detect anomalous positive changes. If "negative", to only detect anomalous negative changes.
Change point detection based on variance shift	Detects a shift of volatility level by tracking the difference between standard deviations at two sliding time windows next to each other.	Univariate time series: change point detection window	Size of the preceding time window. Default is 1.
		Univariate time series: change point detection coefficient	Factor used to determine the bound of normal range based on historical interquartile range.
		Univariate time series: change point detection side	If "both", to detect anomalous positive and negative changes. If "positive", to only detect anomalous positive changes. If "negative", to only detect anomalous negative changes.
Change in seasonal behavior	Detects anomalous violations of seasonal pattern	Seasonal change coefficient	Factor used to determine the bound of normal range based on historical interquartile range.
		Seasonal change side	If "both", to detect anomalous positive and negative changes. If "positive", to only detect anomalous positive changes. If "negative", to only detect anomalous negative changes.
Change in autoregressive behavior	Detects anomalous changes of autoregressive behavior in time series.	Autoregressive change Nb steps	Number of steps (previous values) to include in the model
		Autoregressive change Step size	Length of a step. For example, if n_steps=2, step_size=3, X_[t-3] and X_[t-6] will be used to predict X_[t].
		Autoregressive change Coefficient	Factor used to determine the bound of normal range based on historical interquartile range.

Table 3.1: Anomaly detection with ADTK service (hyperparameter dashboard)

The data to be provided as input to the service must adhere to the following format: `<timestamp, value>`. The timestamp represents the datetime when the measurement took place, formatted as YYYY-MM-DD HH:mm:ss (YYYY indicates the year, MM the month, DD the day, HH the hours, mm the minutes, and ss the seconds). The value field contains the actual measurement. In practice, an example of the input data file is the csv file showed in Figure 3.17:

```
requirements.txt
1 "Date", "Temperature"
2 2003-03-01 00:00:00,730.29898874167
3 2003-03-01 01:00:00,817.753772304501
4 2003-03-01 02:00:00,855.079080335969
5 2003-03-01 03:00:00,867.720246326289
6 2003-03-01 04:00:00,776.017568838348
7 2003-03-01 05:00:00,491.384347195859
8 2003-03-01 06:00:00,-100.979593866936
9 2003-03-01 07:00:00,-348.893874253559
10 2003-03-01 08:00:00,-176.360898214914
11 2003-03-01 09:00:00,-218.665011837928
12 2003-03-01 10:00:00,-454.820037924974
13 2003-03-01 11:00:00,-750.157853963576
14 2003-03-01 12:00:00,-1052.05979212799
15 2003-03-01 13:00:00,-1339.79371135485
16 2003-03-01 14:00:00,-1452.37243599261
17 2003-03-01 15:00:00,-1515.65708932586
18 2003-03-01 16:00:00,-1476.19634401177
19 2003-03-01 17:00:00,-1097.23831290074
20 2003-03-01 18:00:00,-597.014482369379
21 2003-03-01 19:00:00,-710.07798014728
```

Figure 3.17: Anomaly detection with ADTK service (data format)

Figure 3.18 gives an example of the results. Here, we detect some abnormal seasonal behavior.



Figure 3.18: Anomaly detection with ADTK service (output)

3.5.4 Simulation services for smart manufacturing

ESI and ENSAM as part of their contribution to the test-bed no. 8 “CETIM pre-industrial prototype on smart manufacturing”, are jointly developing a set of utilities that combine simulation and MOR methods to facilitate the optimization of the production of instrumented machining tools.

The proposed architecture (Fig. 3.19) for the testbed includes edge computer(s) that receive sensor data from data acquisition systems connected to the machine, at the same time interacting with the ESI Cloud to obtain the predictions of machining simulations and parametric solutions.

Once fully functional, the service will allow for a dynamic simulation of the machining process, with interaction between the tool and the part to be machined via the computed cutting forces. Geometric representations of the part to be machined, the tool and the tool path are taken as input to the solver, which in return produces the geometry of the final surface and the cutting forces as output. Figure 3.20 shows the post-processing of results of a Nussy2m simulation.

As noted in Section 2, it was decided to use the PaaS orchestrator and cloud infrastructure provided by INFN for the purposes of this initial demonstrator. As part of this deliverable, it was planned to demonstrate part of the functionalities developed for this testbed by preparing and integrating a Docker image of the Nussy2m software in the INFN cloud service. However, a current limitation of the INFN PaaS orchestrator is that, in order to register new services with the platform, it is necessary to publish the Docker images to a registry (Dockerhub) that is publicly accessible, even to those outside of the IoTwinS project. Furthermore, for the Nussy2m software to run, it is necessary to have access to a FlexLM licence server (usually at a fixed IP address) in order to verify the existence of a valid licence. To date, no solution has been found that would ensure a service managed by the INFN orchestrator has access to a fixed IP address. As mentioned in Section 3.4, this issue could be resolved by hosting the simulation software on the ESI Cloud platform, which would manage a licence server, and a licence server, and using ‘bridge’ services to manage the interaction between the IoTwinS and ESI Cloud platforms. Nevertheless, these issues have been communicated to partners in WP2 so that provision can be made in the final IoTwinS platform for accessing commercial, licensed software.

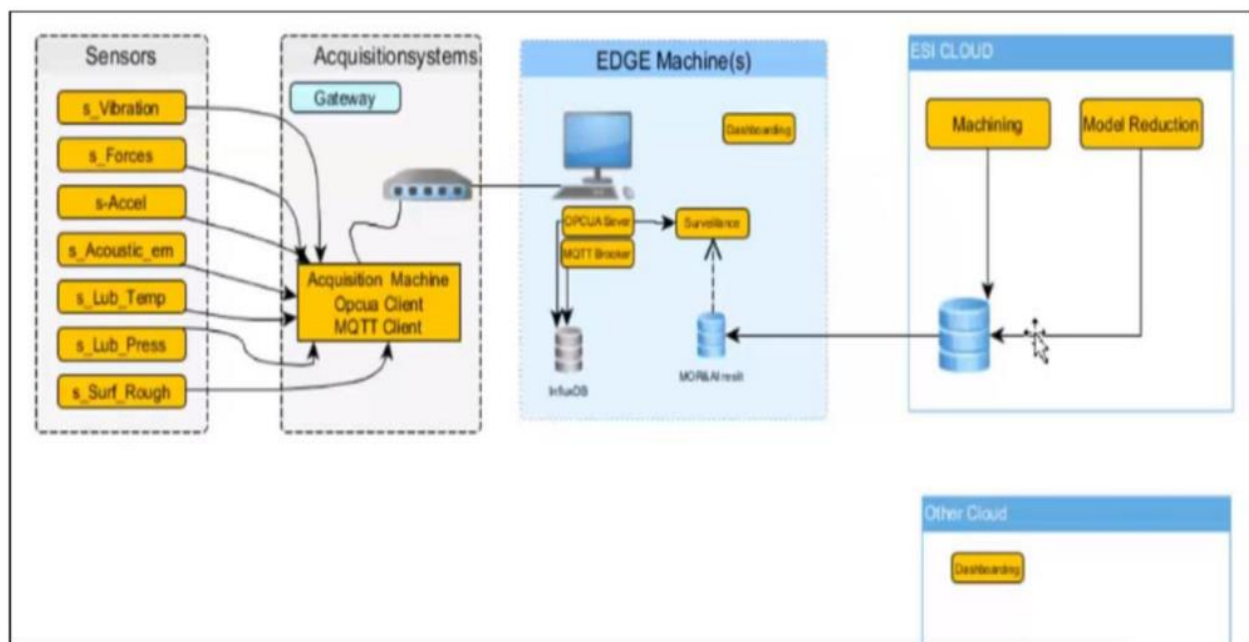


Figure 3.19: Proposed architecture for smart manufacturing test-bed (TB08)

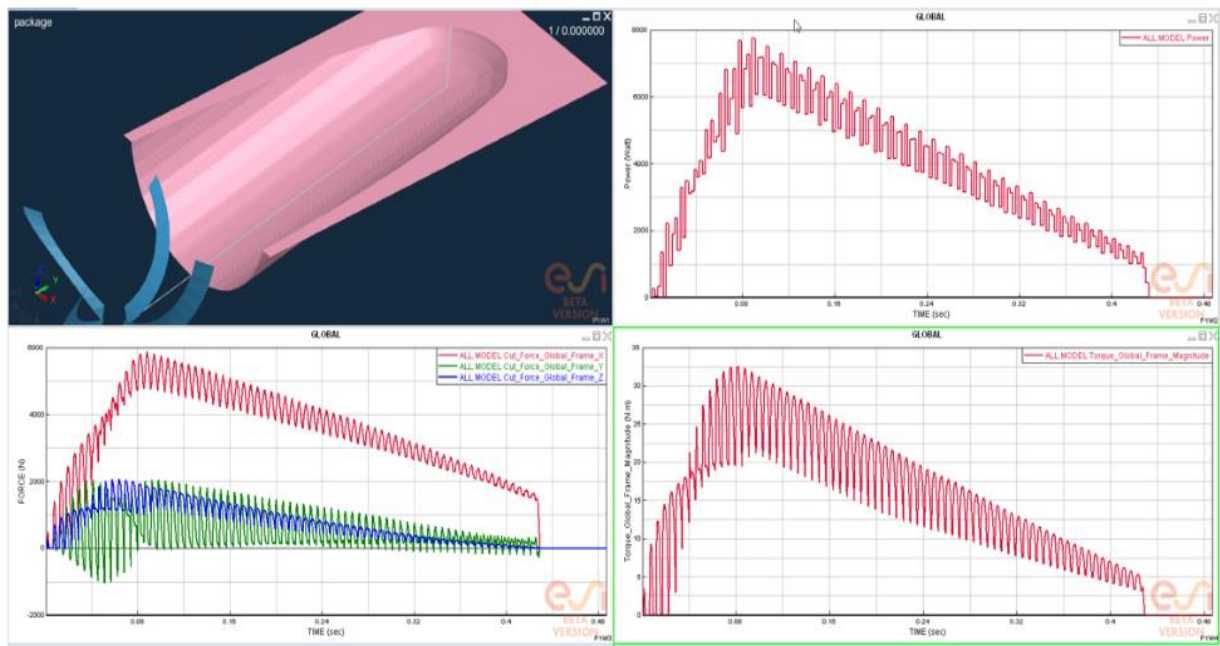


Figure 3.20: Post-processing results from Nassy2m solver

4 A case-study: AI Services for Energy Automation

As part of the collaboration with testbeds partners and with the goal of moving towards an overall infrastructure for handling the data flows (from IoT to cloud, passing through the edge) and applying AI-based services, we describe here a case study linked to testbed N.7 (WP5), “Smart grid – power quality management”.

Devices that protect, control, and monitor power grids have become essential for electric energy systems. Without them, managing and operating complex grids is not possible. Applications running on such devices are, for example:

- Protection: On dangerous events like short circuits, the affected parts of the grid must immediately be disconnected to avoid potential damage.
- Power quality: Parameters like voltage and frequency must be stable. Otherwise, power consuming devices might behave abnormally or even get damaged. For this reason, those parameters need to be monitored continuously and anomalies have to be reported.
- Failure location: On faults like a power outage due to lightning strike, one needs to determine the location of the fault, as distances in power grids may be huge.

Algorithms taking care of such tasks are often complicated, e.g. detecting a short circuit cannot be realized by a simple fuse cutting the power if some current threshold is reached. There are events that temporarily look like short circuits but are not, e.g. an inrush occurring when new consumers attached to the grid cause a high load. However, complicated algorithms result in high development costs. Additionally, domain experts are necessary to determine appropriate rules/configurations for various scenarios and setups. As power grids are becoming more and more diverse, more customizations are necessary and the requirements on the precision of those algorithms increase.

Machine Learning has had a revolutionary impact in various domains, e.g. for image recognition: instead of manually developing complex algorithms, the algorithms learn from provided examples. In general, this is also possible in the field of energy management. While we are not focusing on the development of concrete Machine Learning algorithms as part of IoTwinS, we investigate their deployment and execution on edge/field devices and show their applicability as a replacement for legacy algorithms. In the following, we describe the key requirements and sketch the architecture of a prototypical setup.

Usually, power grid data is analyzed with a high sampling rate, i.e., the system must be able to cope with a high throughput. Typical edge/field devices, however, are often rather restricted in computational power and available memory. Therefore, the algorithms need to be optimized for speed and memory efficiency. Also, they must be able to run on different hardware platforms, specifically on x86 and ARM processors. Even more importantly, safety-critical applications like protection are subject to real-time constraints. A protection algorithm is not allowed to delay the output but has to react deterministically within a specified time span.

4.1 System architecture

4.1.1 Overview

As part of IoTwinS, we have designed an end-to-end scenario that aims to address the above requirements in a realistic system setup. This includes the development of a demonstrator, which will subsequently be refined and extended. Figure 4.1 shows the high-level architecture. Measurement data is received by a simulator or a field device that digitizes the analog signals and broadcasts them using industry standard IEC 61850 (via Ethernet). Internally, the data is distributed to one or more apps via a message bus. Therefore, the ML algorithms can subscribe to the measurements in order to detect features such as inrush and short circuit. The results of the algorithms are again published via the message bus, e.g. for storing them in a database (InfluxDB¹⁴) and for their visualization on a dashboard. Note that most of the data is processed locally and only actual events (e.g. short circuits) along with the measurements around the related point in time are uploaded to the cloud for later inspection.

¹⁴ <https://www.influxdata.com/>

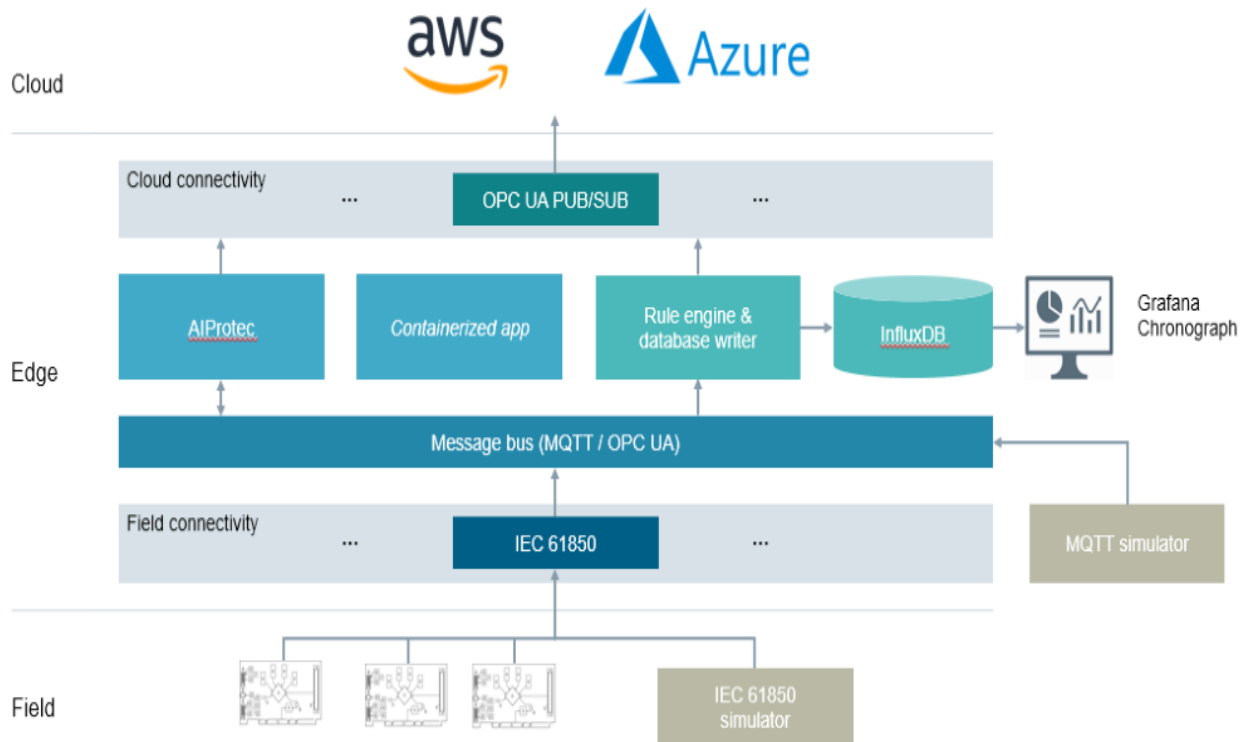


Figure 4.1: High-level architecture of edge-based analysis of power grid data

4.1.2 Components

In the following, we discuss the system components and protocols in more detail.

4.1.2.1 IEC 61850

IEC 61850 is an international standard defining communication protocols for electronic devices at electrical substations. This protocol is used to transmit sampled measured values (SMV) via standard Ethernet between the connected devices. Here, for each phase of a three-phase alternating current, both voltage and current are measured with a sampling rate of 4kHz. This also defines the requirement for all subsequent components in the data processing pipeline to handle data at this frequency.

For our demonstrator, we employ a Manufacturing Message Specification (MMS) stack, which provides two libraries for publishing and subscribing to IEC 61850 packages over Ethernet. These libraries are used for (1) dealing with measurements from the measurement hardware, where we solely need the subscriber component to translate the packages and to publish them on our internal message bus using MQTT¹⁵, and (2) for simulating data, where we replay sequences using the publishing component to create IEC 61850 packages. The simulator is deployed on a generator module (in our setup a Raspberry Pi), which broadcasts the generated data to the demonstrator network.

¹⁵ <http://docs.oasis-open.org/mqtt/mqtt/v3>

4.1.2.2 Message Bus

As mentioned previously, the central message bus is used to distribute SMVs, prediction data, and events along the participating components. In our setup, this is accomplished using MQTT (Message Queue Telemetry Transport), where a central broker handles the routing of the data between the producers (publishers) and consumers (subscribers) based on topics. A topic can be seen as a virtual bus or channel over which data is distributed. When a publisher publishes a message on a particular topic, all participating subscribers on this topic can read the message. This makes it an ideal solution for the intended use case, as not all participants are interested in all data. Hence, the total amount of data to be transferred can be reduced.

As MQTT messages are not typed (i.e. they are just variable-length byte arrays), all components must agree on a common data format (per topic). To fulfill the performance requirements, the encoding and decoding has to be efficient regarding space and time consumption. This led to the decision to use Google Protocol Buffers (protobuf) as data format. There are bindings for all common programming languages, which facilitates the integration of already existing components.

4.1.2.3 Simulators

For testing and fine tuning, we employ simulators that generate SMV data and can be attached at different places in the processing pipeline. One simulator generates IEC 61850 SMV data, which would otherwise be measured by a dedicated hardware. The other simulator directly publishes to the MQTT data bus (and hence bypasses the IEC 61850 to MQTT adapter). All simulators can read pre-recorded data from a file and generate synthetic data.

4.1.2.4 Rule Engine & Database Writer

The combined rule engine and database writer subscribes to SMV data as well as prediction (classification) results and writes interesting sections of the data into a database for recording and post-processing. In future versions, interesting sections will also be sent to the cloud for further analysis. The rule engine is responsible for dividing the data between those sent to the cloud and those stored locally.

4.1.2.5 Time-series Database (InfluxDB)

As SMV and prediction data refer to particular points in time, it is natural to store them in a time series database (TSDB). This allows, for example, to query data for specific time spans. In our setup, a non-functional requirement is to store data with a frequency of 4kHz. For the visualization, the database is connected to an analysis dashboard such that users can interactively browse the data. To inspect larger ranges of high-frequency data, down sampling is required. To this end, the chosen TSDB (InfluxDB¹⁶) supports efficient on-the-fly data aggregation when querying data. This enables users of the system to inspect both high resolution data (up to individual measurements) and to have a good overview of the data set.

¹⁶ <https://www.influxdata.com/>

4.1.2.6 AI Applications

The AI applications running at the edge are developed using standard toolchains such as TensorFlow, Keras and python. For our demonstrator, we use a neural network that detects faults in the power grid (see Section 3, time series service n.3). Here, a measurement, which serves as input for the model, consists of three current and three voltage values. For each time step, the model is applied to a measurement to obtain probabilities for certain fault patterns. Relevant patterns for the demonstrator are inrush, short circuit, and power swing.

Successive measurements are not processed independently of each other. The model keeps an internal state used as additional input in each inference step. Architecturally, the model is a Long Short-Term Memory (LSTM) with post- and preprocessing layers. LSTM models are well-suited to process sequential time series data, as they can learn context. Figure 4.2 shows a sample execution of the neural network. The upper diagram shows three current inputs, the diagram in the middle three voltage inputs, and the lower diagram shows the output of the network.

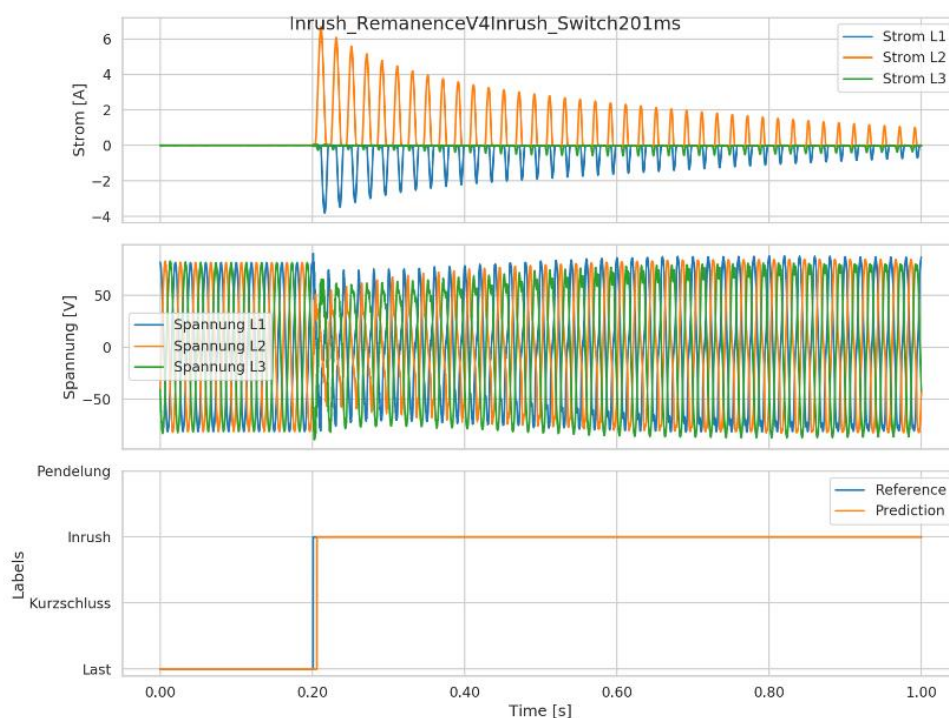


Figure 4.2: Input and output of neural network for power grid protection

Frameworks used for creating and training models are usually not well-suited for inference on field devices, as they consume significant amounts of memory and rely on dynamic memory allocation which renders them non-deterministic by concept. Also, they often have many runtime dependencies that may not always be available on embedded systems with stripped-down and customized operating systems.

Specialized inference frameworks fill this gap. For our demonstrator, we use an inference framework that is part of TensorFlow: TensorFlow Lite. This is a static library that can be linked to the target application and loads/executes the model. The model format is different to TensorFlow, but models can be converted easily if all layers used by the original model are supported.

We carried out basic measurements on a computationally limited field device (dual-core ARM-v7) and a more powerful edge device (dual-core Intel Celeron), which resulted in execution times of 5.7 ms and 1 ms for a single inference, respectively.

4.1.2.7 Analysis Dashboard

The analysis dashboard (see Figure 4.3) is connected to the TSDB and provides an interactive web interface to browse and inspect the data. By zooming and scrolling, the user can easily inspect the situation at a given time period and analyze the decisions of the AI algorithm. As the UI is decoupled from the data itself, new interfaces and visualizations can be setup in the future, working on already existing data or on different data sources.



Figure 4.3: Analysis dashboard for power grid protection

4.2 Results

First experiments with the demonstrator show that our system is capable of end-to-end processing in AI-based power grid protection. While we are currently unable to do this in real-time (4kHz), we identified the system parts that need refinement to achieve this goal. Foremost, we need to increase the processing speed of the neural network by a factor of four (the network's throughput on the target hardware currently allows to process input data with approx. 1kHz). This can be achieved, for example, by quantization or redesign of the network's architecture. It also turned out that the MQTT implementation (i.e. libmosquitto¹⁷) is not capable to deal with the desired throughput. As an alternative, the neural network may subscribe directly (or via a high-speed interface like shared memory) to the IEC 61850 library. For the database, one can still use MQTT although IEC 61850 packages should be aggregated and sent in a batch fashion.

¹⁷ <https://mosquitto.org/man/libmosquitto-3.html>

5 Open Issues & Corrective Actions

As mentioned in the title of this deliverable, the services introduced described by this document are an initial exploration for analysis to be used by testbeds (WP4 and WP5). After the empirical evaluation performed by the testbed partners, their feedback will be carefully considered and used to improve the following version of the services, which will be described in D3.3 (M30).

This section details some of the challenges and issues that are still open and will have to be addressed, following the comments of testbeds partners and in tight interaction with the WP2 partners, as the deployment of the AI-based services is based on the implementation of the IoTwinS platform (see D2.2).

5.1 Edge Processing

At this stage, it was important to have the initial set of services ready for being tested and used by testbed partners, albeit in a relatively limited fashion – namely, the services have not been tested yet on the edge and they operate only with batches of data (for example, csv files and or binary objects) and not on streams of data. This is due to the fact that D3.2 preparation is concurrent with the activity for the implementation of the IoTwinS platform performed in WP2 – hence, no actual edge device or data streaming processing infrastructure was ready during the development of the services, albeit some preliminary analysis has been conducted while working on the case-study described in Section 4. However, the services have already been classified depending on the possibility to be run on the edge rather than exclusively on the cloud.

In the following months we plan to tighten the interaction with WP2 partners, especially after having analysed the results in terms of implementation of the IoTwinS platform that is going to be described in D2.2. In this way, we will integrate the services developed so far in the whole infrastructure; the services will also be extended to operate on streams of data, where needed (for instance, in the case of services performing inference on live data using already trained DL models).

5.2 Integration with Existing Platforms

Another issue that arose during the development of the AI services concerns the integration of the IoTwinS platform (and the services deployed on top of it) and other already existing and in-production platforms and cloud services. For instance, many simulation services provided by ESI require to be integrated with the ESI cloud platform, which hosts the simulation engines needed for the services. This clearly creates some issues, especially in terms of data protection and data confidentiality, as industrial partners might not be willing to share data with other platforms other than the IoTwinS one. In the current demonstrator this problem does not cause great concern, as the users can test the demonstrator by providing realistic but non-confidential data. However, this is a challenge that needs to be dealt with in future versions of the services. Finding solutions will require a tight collaboration with the other partners in the consortium.

5.3 Testbed Requirements

The testbed requirements were collected in D2.1 and, specifically for the AI services, in D3.1. The requirements showed that heterogenous collection of services was required by testbed partners, with some common tasks that were identified, such as anomaly detection. For this reason, in this deliverable we focused

on the services that could be exploited by most of testbeds. However, it is only expected that the pilots partners might have additional requirements and specific needs, which emerged in the first year of the project and not yet captured by the initial phase of requirements gathering. Hence, more services will certainly need to be added after the feedback received by the partners on the first version of the AI services described in this deliverable. Additionally, the services developed and deployed so far are “general” purpose services (at least many of them), that is, they are not testbed-specific services. In real world applications it is very rare for AI-based services to be applied to different context without any tweaking or fine-grained tuning. Hence, it must also be expected that the current set of AI services will need to be refined after the interaction with the partners, in order to make them more tailored to the specific needs of each testbed. This is a development effort that will be performed by WP3, WP4, and WP5 partners jointly.

5.4 Optimization Services

In addition to the previously mentioned issues, another problem concerns the development of optimization-based services (see Task 3.5). The problem arose from the fact that in the requirements collected from the testbeds and gathered in D3.1, no optimization service that could have been applied to multiple testbeds was identified. Furthermore, it appears that at the stage when the requirements were collected, most of the testbeds struggle with the formalization of their desired goals and operative constraints as optimization problems. This is understandable as the formal definition of optimization problems drawn from industrial facilities and large-scale data centres is a non-trivial process. However, this lack of sufficiently detailed requirements greatly hampered the development of optimization-based services – which are currently absent from the demonstrator described in this document.

To correct this situation, the next actions will be the following:

- 1 initiate a second round of collection of testbeds requirements, focusing explicitly on the optimization problems, taking advantage of the additional knowledge about the target processes gathered in the months passed since the first requirement collection phase and of the insights that will be made possible by the newly available AI-based services;
- 2 identify and develop testbed-specific optimization services, at the risk of losing generality and re-usability.

Albeit the optimization services still need to be developed, the knowledge and technical expertise gained while developing ML- and simulation-based services during the creation of the first demonstrator will greatly help the development and deployment of optimization services as well. For instance, optimization services will be developed as Docker container self-contained and loaded with all required software packages and libraries. In this way, after the completion of the development phase, the deployment process of the optimization services will be very similar to the one adopted for the remaining services, greatly decreasing the time and effort required.

6 Conclusions

This document describes the first version of the AI and simulation services developed for the IoTwinS platform. The slight delay (1 month) in the delivery of the services was due to the Covid-19 emergency and the related nation-wide lockdown enforced by the Italian government, as the latter hindered the installation and configuration of the cloud infrastructure where the services were to be deployed.

The AI services were developed and deployed as Docker containers; the capability of running Docker containers will thus be an important prerequisite for the final IoTwinS platform. As such platform is still under development, the services described by this document have been deployed on a cloud infrastructure provided by one of IoTwinS partners, INFN – the current demonstrator has the purpose of showing the potential benefits of the services and allowing testbed partners to use the services and perform preliminary analysis on the data they have been collecting in the first year of the IoTwinS project. As noted in several passages, the current version of the services will need to be revisited according to the testbed partners' feedback and in tight interaction with WP2 partners, in order to integrate them in the final version of the IoTwinS platform. Following these considerations, next actions and remaining challenges were listed and the planned methods to address them have been identified.