# D5.2 Technical Platform Design

| | |
|---|---|
| **Lead Partner:** | ATHENA |
| **Version:** | ToC |
| **Dissemination Level:** | PU |
| **Work Package:** | WP5 |
| **Due date:** | 30/09/2021 |
| **Submission Date:** | |

**Abstract:**
Deliverable 5.2 is a fundamental document of the MORE project that outlines the technical design and the architecture of the MORE platform. It describes the functional and the non-functional specifications, both for the cloud based MORE platform running in a data center and the MORE edge platform running on edge nodes in the RES installations.
The deliverable presents an overview of the MORE logical architecture, it describes each software module and presents the data flows between them. For each software module we presented detailed technical specifications, the sub-systems that comprise them, the interfaces (APIs) between modules and we establish the dependencies. The functional and non-functional specifications for each module are presented in a clear and standardized way. Finally, we provide an overview of the Common Integration Infrastructure, which is based on continuous integration (CI) and DevOps practices, and the targeted users of the platform are discussed.

ICT-H2020-51-2020 - Big Data technologies and extreme-scale analytics

## Document Revision History

| Date | Version | Author/Editor/Contributor | Summary of main changes / Status |
|---|---|---|---|
| 16/7/2021 | 0.1 | Giorgos Alexiou, George Papastefanatos | ToC |
| 30/7/2021 | 0.2 | Giorgos Alexiou, George Papastefanatos | Initial Draft for circulation |
| 01/09/2021 | 0.3 | Giorgos Giannopoulos, Alexandros Kalimeris, Stavros Maroulis, Vassilis Stamatopoulos, Vassilis Kaffes, Ioannis Psarros, Panagiotis Gidarakos, (ATHENA), Seshu Tirupathi (IBM), Christian Nygaard (MOD), Christian Thomsen, Søren Kejser Jensen (AAU) | Contributions from partners |
| 06/09/2021 | 0.4 | Giorgos Alexiou(ATHENA) | Preparation of final document |
| 24/09/2021 | 0.5 | Julien Masson, Nicola Girard (LABORELEC), Gregory Doumenis , Gregory Kotsis (InAccess) | Comments provided |
| 27/09/2021 | 0.6 | Giorgos Alexiou, George Papastefanatos, Manolis Terrovitis | Final version prepared |
| 30/09/2021 | 1.0 | Manolis Terrovitis | Deliverable Submitted |

## Disclaimer

## Copyright

# 1 Introduction

The goal of this deliverable is to present the architecture of the MORE platform, the technical characteristics and how this architecture achieves the technical objectives of the MORE project as well as to provide the functional specifications of its modules. MORE will deliver a platform that addresses the technical challenges in time series and stream management, focusing on the RES industry. More specifically, MORE's platform will introduce an architecture that combines edge with cloud data processing with the aim to provide scalability to massive datasets and thus achieving increased performance, interactive and sophisticated analytics.

WP5 is led by Athena RC and further AAU, InAccess, IBM-IE, Perd and Laborelec participate. The objectives of deliverable 5.2 are the following:

**Objective 1: Provide the design of the MORE architecture.**
This deliverable provides an overview of the architecture by dividing it in four main areas, namely:

i) System Logical Architecture
ii) Edge Layer
iii) Cloud Layer and
iv) Application Layer

Each area is further analyzed in the respective subsection. Also, it explains how the architectural design choices serve the technical challenges of the project.
.
**Objective 2: Provide the technical specifications of the MORE platform.**
The technical specifications of the platform are given in the form of functional and non-functional specifications. Specifically, for each module and sub-module of the platform a list of specifications about its input, output and technical details is provided (functional specifications) along with a list of criteria for evaluating how the modules should perform and what quality attributes should have (non-functional specifications).

**Objective 3: Outline the targeted users of the MORE platform.**
The targeted users of the MORE platform are divided in two main categories: i) the Data Scientists and ii) the Business Users. The latter include module constructors, park constructors, park operators, investors, and other types of stakeholders without specialized knowledge on data and knowledge engineering, while the former include domain experts, data analysts and generally any kind of stakeholder with specialized knowledge on data science and analytics.

**Objective 4: Provide an overview of the Continuous Integration and Deployment plan in MORE.**
To facilitate the optimal code engineering and to ensure that developers will not spend their time in package building and tedious debugging processes, as well as to ensure the best possible collaboration between the development teams of MORE, we will use DevOps and CI\CD practices.

This deliverable builds upon the activities and the preliminary results of the WP5's activities, which have been reported in D5.1. "Use Cases and Requirements". D5.1 presented the two sets of use cases; the first concerns wind parks and the second concerns photovoltaic parks. D5.1 also presented the specific data characteristics, available infrastructure, and functional requirements, which have been collected during the requirements elicitation process. The use cases and their requirements have been analyzed and the technical specifications targeting the required functionality are presented in this deliverable.

# 2 Architecture Overview

The key concept of the MORE project is to create a platform for time series management that will disrupt the way data from sensors is managed today. The vast amounts of data pose challenges in the responsiveness of the data management systems, but at the same time they provide the necessary resources to create accurate prediction and diagnostics tools. MORE will allow stakeholders in industry sectors with huge volumes of sensor data, especially the RES industry, to: **a) scale the management of streaming and historical time series ranging over an order of magnitude beyond the state-of-the-art and b) to perform forecasting, prediction and diagnostics using the whole data that is available to them with accuracy that outperforms existing approaches.** To achieve that MORE will combine scalable analytics, that will be able to scale beyond state-of-the-art systems, with sophisticated machine learning and pattern recognition methods.

## 2.1 Architecture principles

Current big data systems are unable to cope with the data created in RES installations. RES operators would like to receive and analyze readings from multiple sensors at much higher frequencies. Sensors can support higher frequencies, but companies lack the bandwidth, and the data management tools to ingest, transfer, and analyze greater data volumes. *MORE's key technical objectives are to consume billions of streams and petabytes of data without compromising the overall performance, the accuracy, and the analytics capabilities of the system.* These objectives are addressed by the following architectural decisions and techniques, that we follow in the design of the MORE platform:

- ***Processing compressed data at the edge*** *to cope with high frequency multidimensional streams*. MORE platform will employ data reduction techniques for time series, such as aggregation, summarization and compression, at the edge. The data will be modelled and summarized according to varying time granularities in the whole data processing pipeline and analytics will be directly applied to the summaries. This will provide support for real-time ingestion of high-frequency data streams from sensors even in limited hardware resources that are available at the edge nodes and eventually it will allow the efficient performance of analytics over compressed data on the edge, such as simple event detection.
- ***Algorithms parallelization for Scalable and complex data analytics on the cloud****.* MORE central processing will be primarily based on a distributed cloud architecture that can scale out (in terms of disk, memory, and CPU resources) for addressing the storage and processing requirements of RES big data. Cloud nodes can offer the necessary elasticity for scaling out the processing. On top of this pattern, however, MORE will achieve increased performance by working on highly parallelizable algorithms, namely, pattern extraction methods, and motif extraction techniques which are developed especially for time series data. The parallelization, either via node distribution or via GPU acceleration, will offer the ability to fine-tune the execution of complex analytics over limited memory and CPU cloud resources, not only for streaming summarized data but even for batch processing of raw historical data. By extracting motifs from the collected historical data, we can identify patterns that are linked to important events in the RES installation or reveal properties of the modules. Patterns, in this context, are indicative of correlations or rules between readings within single- or even multi-dimensional streams of possibly different nature (temperature, wind speed, active produced power, global irradiance, cloud cover, etc.).
- ***Increased accuracy over summarized data by employing AutoML.*** Creating a scalable platform will also enable us to create accurate models for prediction, forecasting and diagnostics. Moreover, the MORE platform will achieve its aim for accuracy in prediction and diagnostics by focusing on incremental machine learning algorithms that can scale better than

most existing approaches and are updated continuously. Also, it will combine incremental Machine Learning (ML) with AutoML techniques that customize the inference to the incoming data characteristics (e.g., granularity of the data) and achieve increased accuracy by selecting the most appropriate models.

- ***Self-service visual and interactive analytics for the end users.*** The MORE platform aims to enable *intuitive* and *fast interaction* of the user with the underlying data. For that reason, it will not only work on novel UI interfaces and user exploration capabilities for working with time-series data, but it will also develop visual-aware indexes that will be hosted on the cloud and can enable efficient interaction (e.g., exploration of streaming time series, annotation and visual detection of events, etc) with big volumes of multidimensional time series.

These key objectives of MORE will be achieved by building the system presented in Figure 1. Data is generated at RES installations and immediately processed by edge computing nodes running a lightweight ModelarDB edge node. Depending on the workload, simple analytics are performed locally, and the modelled data is then forwarded to the cloud infrastructure. In the cloud infrastructure, advanced analytics and learning take place. Incremental machine learning algorithms work directly on incoming modelled data and offer forecasts and diagnostics in real time by applying their inference model and, simultaneously, they continuously update their model. Pattern identification-based diagnostics and predictions are also applied directly on incoming data, producing libraries of identified patterns (motifs) and outliers (discords) on simple or multidimensional streams. Data, machine learning models and extracted patterns are then stored persistently to be used asynchronously in ad hoc analytics and pattern extraction/detection algorithms. In parallel, machine learning models and pattern indexes are periodically transmitted back to the respective edge nodes, to support real time forecasting and outlier/pattern/event detection tasks.
In the following, we present the MORE platform architecture and explore solutions to the challenges enacted by the user requirements. First, we present an overview of the MORE logical architecture, identify the main MORE software modules and the data flow paths between them. Next, we provide a more detailed view along with technical specifications of each module, the sub-systems that comprise them and we establish the dependencies.

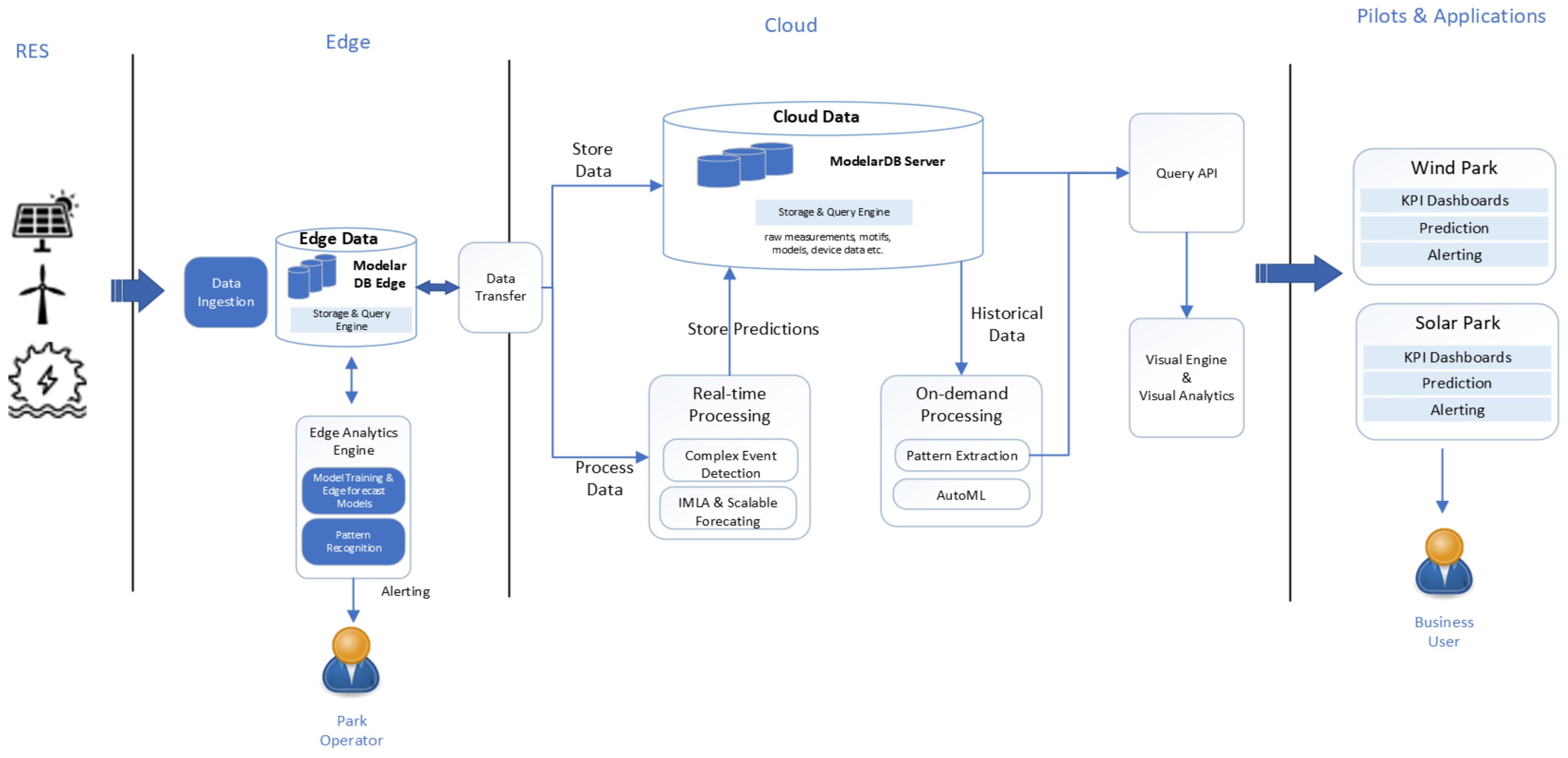

Figure 1. More Application Overview and Dataflow

## 2.2 Logical Architecture of the MORE platform

In this section, we introduce the main parts of the MORE system, present a high-level overview of its logical architecture, and identify the main data flow paths.

The MORE system consists of the following logical subsystems as shown in Figure 1.

- **Renewable Energy Resources**: Where sensors are installed and continuously producing measurements related to the operation of a specific RES facility.
- **Edge Nodes**: Infrastructure that gathers, analyses, compresses, and stores the RES data, and sends them to the cloud.
- **Cloud**: The cloud module, where data is gathered, stored, and passed through ML and pattern extraction (PE) modules that are used to train and test prediction models.
- **Applications and Pilots**: The UI modules of the MORE platform, i.e., applications used for Visual Analytics, Interaction, prediction and alerting the user.

### 2.2.1 Edge Layer

In the following the Edge modules are enumerated.

- **Edge Data Ingestion Module**: This module ingests enormously large amounts of raw time series data, i.e., billions of values per day, and compresses the data as mDB models. Specifically, the data will be represented as coefficients of models, representing data points within an error bound. While the ingestion is done, the data can still be queried. The Initial version of the engine is expected to be a JVM with H2[1], while the final version will be Rust with DataFusion[2] query engine.
- **Edge Query Engine**: Model-based compressed data are stored into multi-layer storage, and later are used to support query processing. This module implements model-based, edge-optimized, resource-aware query processing and query optimization algorithms for basic continuous queries on models on the edge nodes. The algorithms will be able to efficiently evaluate (cross) stream queries and handle user-provided trade-offs between accuracy and response time. The edge query engine is also compatible with the Cloud Model Storage and Query Engine. The Initial version of the engine is expected to be a JVM with H2, while the final version will be Rust with DataFusion query engine.
- **Edge Analytics Engine**: This module offers lightweight Pattern Recognition and Machine Learning models that are used to make predictions on the edge node data that can help in avoiding critical failures. Models are trained on the edge utilizing only a subset of the available features containing only the most important ones, while the central model also explores the remaining features. The communication between the edge node and the central server is only performed for exchanging the model parameters during the training phase. The engine will leverage and extend the work on FL in H2020 Musketeer project[3] and utilize the AMQP protocol and the Apache Arrow Flights framework.
- **Data Transfer Module**: This module is responsible to transfer the model-based data representations (i.e., coefficients for the models) from edge nodes to the cloud. It will control what and when to transfer data, considering bandwidth limits and the urgency of the data/alerts. Also, shared prediction on the coefficients is applied to reduce bandwidth requirements, e.g., when the model transfer is dictated by the Workload Balancer using strategies provided by the Cloud Model Storage and Query Engine. Alerts are propagated

[1] https://www.h2database.com/
[2] https://arrow.apache.org/blog/2019/02/04/datafusion-donation/
[3] https://musketeer.eu/

directly to user applications in a dedicated high-speed and high-reliability pipeline. The module is expected to utilize the Apache Arrow Flights framework in order to achieve a fast data transport.

### 2.2.2 Cloud Layer

The Cloud layer contains modules that are used for the cloud analysis and storing of the data as well as for making predictions and visualizations. The modules are enumerated below.

- **Batch Processing:** Contains the Pattern Extraction and the AutoML modules. These modules are used to train and update the models that are stored in the Cloud Model Storage and Query Engine.
- **Pattern Extraction Module:** The Pattern Extraction module (PE) is responsible for analysing the RES time series data and extracting meaningful patterns in the form of motifs, discords, changepoints, etc. The module will incorporate the following core functionalities: (a) Calculating indexes (e.g., the Matrix Profile) on top of the time series that holds distance information between all different parts of a time series, considering specific but parameterizable pattern lengths; (b) Running variations of motif discovery algorithms in order to identify different types of patterns, including motifs, discords and changepoints; (c) Annotating time series areas via Annotation Vectors in order to incorporate domain knowledge; (d) Deploying traditional time series analytics, as well as ML algorithms (where meaningful) and jointly exploiting with motif discovery algorithms in the pattern extraction process; (e) Storing the output of the aforementioned processing (indexes, patterns) in the Patterns Repository. For the development process, several python libraries that calculate the MP and perform further processing/analytics on it will be used. In particular: STUMPY[4], MatrixProfile[5] and, SCAMP[6] and pyscamp[7]. The aforementioned libraries are implemented in python3 (or provide python3 bindings to C++ code), thus the PE module will adopt the same programming language. An installation of the CUDA toolkit[8] is necessary for deploying the GPU-accelerated versions of the aforementioned libraries.
- **Auto ML Module:** AutoML on incremental machine learning algorithms (IMLA) with federated learning will be explored for forecasting and prediction tasks on the cloud as well as on the edge. Fundamental IMLA models and FL will be initially explored. AutoML pipeline is built on top of these IMLA models coupled with FL. Development environment and integrations will leverage platforms and services like Kubernetes, RabbitMQ and Functions as a Service (FaaS). In addition, Cloud Object Store and DB2 are planned to be used for storing models and meta information of the models.
- **Real-time Processing:** Runs the Complex Event Detection and Incremental Machine-Learning models for scalable fast predictions against the fast-incoming data.
- **Complex Event Detection Module:** The Complex Event Detection module (CED) is responsible for exploiting outcomes of the Pattern Extraction module in order to (a) identify complex events within RES time series and (b) be able to identify specific types of events in near-real time. The CED module will incorporate the following core functionalities: (a) Updating existing indexes (e.g., the Matrix Profile), as new data points

[4] https://github.com/TDAmeritrade/stumpy
[5] https://github.com/matrix-profile-foundation/matrixprofile
[6] https://github.com/zpzim/SCAMP
[7] https://pypi.org/project/pyscamp/
[8] https://developer.nvidia.com/cuda-toolkit

are constantly added to the time series; (b) Deploying event detection algorithms that exploit the underlying MP index and additional precomputed information (e.g. top-k motifs and traditional time series analytics) in order to identify (nearly) real time patterns/events/incidents; (c) Deploying pattern recognition algorithms for identifying frequently cooccurring patterns (denoting complex patterns). For the development process, the same technologies and processes with PE module will be used.

- **IMLA & Scalable Forecasting Module:** IMLA & Scalable Forecasting provides a scalable platform for novel incremental algorithms and orchestration for federated learning models that would be deployed on the edge. The platform provides real time and on-demand forecasts and predictions. In addition, the service also orchestrates the training of federated learning models that can be deployed on the edge for scoring. IMLA & Scalable Forecasting Module is expected to use the same development process as AutoML, leveraging the same technologies and services, as its APIs are expected to have a great overlapping with the AutoML APIs.
- **Visual Engine and Visual Analytics Module:** The Visual Analytics Module (VA) will offer the functionality for MORE platform users to interact with the time-series data and be informed on various KPIs and complex events in real-time. This module will offer a dashboard-like UI for the visualization and interaction with various types of data, such as hierarchical multidimensional timeseries. The module will consist of a backend providing a REST API, built using the Sping Boot Framework in Java and a frontend, built using the React Javascript library. The D3.js library will be used for the time series visualization and LeafletJS for the visualization of the RES sensors on the map.
- **Cloud Model Storage and Query Engine:**
    - Cloud Model Storage: A distributed model storage solution running at the data center nodes. This module will extend the Apache Cassandra-based solution utilized by ModelarDB with improved indexing to efficiently support extraction of relevant models for value-based queries and high-level analytics also using the matrix profile suite of algorithms.
    - Cloud Query Engine: A distributed Cloud Query Engine as extension to ModelarDB, with Apache Spark utilized for distribution and parallelization. Two sets of extensions will be implemented: (a) Computation of matrix profiles directly on models to efficiently support the advanced analytics described in WP4 Task 4.1. By implementing support for computing the matrix-profile from models, the MORE platform can efficiently perform advanced analytics on time series; (b) Support for communicating with the Workload Balancers will be implemented as part of the cloud query engine. This will allow the query engine to decide what data must be transferred to the data center and at what precision the data must be provided. The cloud query engine will be able to dynamically decide if a query should be executed in the cloud or sent to the edge. In addition to transferring data on demand, the edge nodes will pre-emptively transfer relevant data to the data center based on interesting events detected in the data at the edge, what data have previously been utilized by queries, and user-defined strategies.

### 2.2.3 Application Layer

The MORE applications will support the two pilot use cases (Solar and Wind Parks) offering KPI monitoring via dashboards, prediction visualization and alerting functionality for users. It must be noted that the pilots are not part of the core functionality of the MORE platform; rather the main objective of the platform is to offer a flexible toolkit with APIs and functionality for building rich and scalable RES-analysis applications for different analysis needs and KPIS. As such, the specifications for the two pilots will be part of the deliverable 6.1 Pilot design, and evaluation strategy, which will provide the detailed functionality of the two pilots.

## 2.3 Dataflow overview

The dataflow is presented in Figure 1, with the use of arrows that denote the communication of the modules in the overall architecture. There are two distinct categories of dataflow and data processing. (a) Real-Time Dataflow and on-demand processing, which describes the flow of the data through each module of the system; (b) Batch processing, which describes the processing of stored historical data by the users of the system, for time-consuming activities, like training ML or Pattern Recognition models. A description of the dataflows on each module of the system is given below.

### 2.3.1 Real-Time Dataflow and on-demand processing

**RES:** Data sent from the Renewable Energy Sources is passed to the Data Ingestion Module of the Edge.

**Edge Nodes:** On the Data Ingestion Module of the Edge, data is represented as coefficients of models, representing data points within an error bound. This compression reduces the storage requirements of the system and aids in the system's scalability. The data ingestion module will be fast enough to keep up with enormous amounts of data. While the ingestion is done, the data can still be queried. Moreover, every Edge Node contains an Analytics module that houses the Pattern Recognition and Forecast Models. These, are run on incoming data and aid on avoiding critical failures by alerting the park operator.
Apart from real-time processing, data that are housed in the storages of the Edge Nodes and the cloud, can also be accessed on-demand for further processing and analysis via the Query Engine.
Finally, data pass through the Data Transfer module to be sent to the cloud. The bandwidth of the transfers as well as the type of data that are transferred (raw, compressed) is determined by the module's Workload Balancer, whose goal is to make the dataflow as optimal as possible.

**Cloud:** From here, data follows in two different paths. First, it is stored on the Cloud Data Storage, from where it can be accessed through the Cloud Query Engine and used by ModelarDB Server. In addition, the data are also processed in real-time and are passed through the Complex Event Detection and IMLA modules. These models have been trained beforehand and can be found on the Cloud Storage where they have been stored. More details on the batch training of the models are given in paragraph **2.2.4.2.** The real-time prediction helps to quickly avoid potential failures. These predictions are also stored in the Cloud Storage. Due to the resource heavy nature of the operations that are performed on the Cloud and the need to ensure scalability, the elasticity of the cloud is crucial. For this reason, the cloud will provide the ability to gain or reduce computing resources such as CPU/processing, RAM, input/output bandwidth, and storage capacities on demand without causing system performance disruptions.

**Visual Analytics / Pilots & Applications:** Finally, after being stored on the Cloud, the MORE platform users will be able to interact with the timeseries data and be informed on various KPIs and complex events in real-time. The user will be presented with a dashboard like UI where various types of data, such as hierarchical multidimensional timeseries will be visualized in a way that optimizes the user experience.

### 2.3.2 Batch processing

**Edge Nodes:** The analytics module of every Edge Node, can also be used over historical data stored in the system to train the Forecast and Pattern Recognition Models that are used on the real-time processing. Edge nodes will be able to communicate with the central server as to exchange the model parameters that are used during the training phase of these models.

**Cloud**: Data stored in the cloud storage can be processed via the Pattern Extraction and AutoML modules. The Pattern Extraction and AutoML modules train models using historical data, find interesting motifs and patterns and persist them to the Storage for later use. The patterns and motifs found by these modules are used to train the real-time prediction models.

# 3 Technical Specifications of the MORE Platform

Following the Architecture overview, the functional specifications of the MORE Platform are grouped into the following categories: Edge Modules and Cloud Modules which are described in detail below. AS previously mentioned, pilot applications are software and processes that will be developed, with the aim to demonstrate the MORE platform's functionalities. However, their functionalities are not considered as a part of the offered functionalities of the MORE platform and therefore are not discussed in this deliverable, but rather they will be designed and presented in WP6.

## 3.1 Edge Data Ingestion Module

Data ingestion module on the edge receives raw time series data from edge devices and compresses them as mDB models. This module specializes ModelarDB's model-based data ingestion module for the edge infrastructure to be more efficient. Specifically, the data will be represented as coefficients of models, representing data points within an error bound. The data ingestion module will be fast enough to keep up with enormous amounts of data, i.e., billions of values per day. While the ingestion is done, the data can still be queried.

### 3.1.1 Functional Specifications

| Field | Short Description |
|---|---|
| ID | FS-Ingest-01 |
| Name | Compress raw time series data as models. |
| Purpose | To compress time series data to optimize storage and query performance (time) |
| Input | Raw time series data |
| Output | Model-based data representation |
| Operation | Time series are split into dynamically sized segments and a model is fitted to each segment within the error bound. |
| Used technologies | Initial version will be JVM with H2, final version will be Rust with DataFusion. |
| Target Users | Data Scientists |

| Field | Short Description |
|---|---|
| ID | FS-Ingest-02 |
| Name | Store Models |
| Purpose | Store model-based compressed data in persistent storage. |
| Input | Model-based data representation |
| Output | Binary files on disk. |
| Operation | Segments representing a sub-sequence of a time series using a model will be written to binary files on disk. Compaction will ensure small files are merged together. |
| Used technologies | Initial version will be JVM with H2 or Parquet, final version will be Rust and Parquet. |
| Target Users | Requires no user interaction. |

| Field | Short Description |
|---|---|
| ID | FS-Ingest-03 |
| Name | Combine Models |

| | |
|---|---|
| Purpose | Combine similar models into one to further increase compression ratio. |
| Input | Model-based data representation |
| Output | Model-based data representation |
| Operation | For a set of segments, the module will evaluate if each pair of models can be combined within the error bound. If so, one of the segments are discarded and the other segment is updated as necessary. |
| Used technologies | Rust with DataFusion. |
| Target Users | Requires no user interaction. |

| **Field** | **Short Description** |
|---|---|
| ID | FS-Ingest-05 |
| Name | Hierarchical model-based storage. |
| Purpose | Provide several layers of temporary storage, with different degrees of accuracy and indexing power. |
| Input | Raw time series data or model-based representation |
| Output | Model-based data representation |
| Operation | If raw data is available multiple models will be fitted to the same data points. If not, models with a higher error bound will be constructed from these models, however, models with a lower error bound cannot be constructed. |
| Used technologies | Rust with DataFusion. |
| Target Users | Data Scientists |

### 3.1.2 Non-functional Specifications

| ID | Name | Purpose | Module | Technical solution\mean of verification |
|---|---|---|---|---|
| NFS-Ingest-04 | Near real-time query execution | Allow execution of queries while ingesting data | Query Engine | Models are fitted to data points online. Segments containing the most recently fitted models are cached in-memory and combined with the segment on disk through a union operation performed by the query engine. |
| NFS-Ingest-05 | Scalable ingestion | Support ingestion of billions of data points per day | Ingestion Module | Models will be fitted to time series online using a sliding window. The efficiency of this approach has been verified by ModelarDB, but will be also be verified using an experimental evaluation. |
| NFS-Ingest-06 | Appropriate system requirements | Perform ingestion on edge nodes with 4 CPU Cores, 4 GiB Ram, and 250 GiB Hard Disk | Ingestion Module | The ingestion module will only keep a sub-sequence of each time series in memory during ingestion to reduce memory consumption. Also, the module will be implemented in Rust so memory allocation and deallocation can be manually tuned. The module will also be tested on similar hardware. |

## 3.2 Edge Query Engine

This module develops model-based, edge-optimized, resource-aware query processing and query optimization algorithms for basic continuous queries on models on the edge nodes. The algorithms will be able to efficiently evaluate (cross) stream queries and handle user-provided trade-offs between accuracy and response time. The module will: (1) guarantee scalability for very large numbers of sensors, and (2) optimize for various system-wide performance metrics (including, minimizing communication, latency, response time, and energy consumption). The query evaluation strategies will consider views and real-time KPIs that are computed on the edge nodes. The edge query engine is, while compatible with the data center engine, highly specialized and optimized for the edge nodes.

### 3.2.1 Functional Specifications

| Field | Short Description |
|---|---|
| ID | FS-Edge-Query-1 |
| Name | Simple Query Support |
| Purpose | Support simple queries (select, where etc.) and aggregated queries (sum, count, etc.) in order to facilitate basic analytics. |
| Input | Models and SQL queries |
| Output | SQL Result-sets |
| Operation | Execute SQL queries using a model-based representation of time series as the input. |
| Used technologies | Initial version will be JVM with H2, final version will be Rust with DataFusion. |
| Target Users | Data Scientists |

| Field | Short Description |
|---|---|
| ID | FS-Edge-Query-2 |
| Name | Query Performance Monitoring |
| Purpose | Provide information on query performance (response time) |
| Input | Models and SQL queries |
| Output | SQL Result-sets or a format specialized for monitoring |
| Operation | Initially, record metrics about query during execution and make them available as part of the query, and later store them separately and make them available through a monitoring interface. |
| Used technologies | Initial version will be JVM with H2, final version will be Rust with DataFusion. |
| Target Users | System Administrators |

| Field | Short Description |
|---|---|
| ID | FS-Edge-Query-3 |
| Name | User Specified Query Trade-offs |
| Purpose | Allow the user to specify a trade-off between accuracy and response time |
| Input | Models and SQL queries |
| Output | SQL Result-sets |

| | |
|---|---|
| Operation | Query engine will select the best models to use from the model hierarchy based on each queries' constraints. |
| Used technologies | Rust with DataFusion |
| Target Users | Data Scientists |

| Field | Short Description |
|---|---|
| ID | FS-Edge-Query-4 |
| Name | Simple Continuous Queries |
| Purpose | Support basic continuous queries on models across multiple streams on the edge nodes. |
| Input | Data points, models, and SQL queries. |
| Output | SQL Result-sets or Events? |
| Operation | Query engine runs continuously and returns a new result set when it changes. |
| Used technologies | Rust with DataFusion. |
| Target Users | Data Scientists |

### 3.2.2 Non-functional Specifications

| ID | Name | Purpose | Module | Technical solution\mean of verification |
|---|---|---|---|---|
| NFS-Edge-Query-5 | Response Time | The response time should be fast enough for the users and other modules | Edge Query Engine | Recent ingested and queried data will be cached in-memory as models, and queries that can executed directly on models instead of reconstructed data points will do so. |
| NFS-Edge-Query-6 | Appropriate system requirements | Perform query processing on edge nodes with 4 CPU Cores, 4 GiB Ram, and 250 GiB Hard Disk | Edge Query Engine | Time series in memory will be represented as models to significantly reduce their size and allow queries to be executed directly on the compressed format. Also, the module will be implemented in Rust so memory allocation and deallocation can be tuned. The module will be evaluated on a system with similar hardware. |

## 3.3 Edge Analytics Engine

Edge analytics engine provides real time forecasts and predictions that can help in avoiding critical failures. For training edge models, Federated Learning (FL) would be employed. Models are trained on the edge with the important features and the central model will explore the remaining features. The communication between the edge node and the central server mainly refer to the exchange of the model parameters during training phase. FL requires careful orchestration of the central server and the edge nodes and will leverage and extend the work on FL in H2020 Musketeer project[9] by introducing the following functionalities.

### 3.3.1 Functional Specifications

| Field | Short Description |
|---|---|

[9] https://musketeer.eu/

| | |
|---|---|
| ID | FS-Edge-Analytics-1 |
| Name | EAE Training |
| Purpose | Ability for a task member to actually participate in the training of that task's Federated ML model, either as aggregator or as participant |
| Input | Metadata or UID, Summarized data |
| Output | Model parameters |
| Operation | Ingests the data from ModelarDB to incrementally train a model. |
| Used technologies | AMQP/REST /Apache Arrow Flight |
| Target Users | Data scientists |

| **Field** | **Short Description** |
|---|---|
| ID | FS-Edge-Analytics-2 |
| Name | EAE Communication |
| Purpose | Ability for a task member to send and receive intermediate or final trained model parameters. |
| Input | Metadata or UID |
| Output | Model parameters |
| Operation | Transfers a client/edge model to the cloud and the updated model from the cloud with federated learning is received at the edge. |
| Used technologies | AMQP/REST |
| Target Users | Data scientists |

| **Field** | **Short Description** |
|---|---|
| ID | FS-Edge-Analytics-3 |
| Name | EAE Scoring |
| Purpose | Ability for a task member to score the given forecasting or prediction task. |
| Input | Metadata or UID |
| Output | Forecasting or prediction output |
| Operation | Ingests the data from ModelarDB to score a model generated by federated learning. |
| Used technologies | AMQP/REST /Apache Arrow Flight |
| Target Users | Data scientist / Business users |

## 3.3.2 Non-functional Specifications

| ID | Name | Purpose | Module | Technical solution\mean of verification |
|---|---|---|---|---|
| NFS-Edge-Analytics-1 | EAE Platform | Ability to run python code or docker containers for scoring and training the models on the edge. | EAE Training, EAE Scoring | Python code or corresponding docker code is deployed on the edge to test if training and scoring works without any errors. |

## 3.4 Data Transfer Module

This module is responsible to transfer the model-based data representations (i.e., coefficients for the models) from edge nodes running the improved ModelarDB edge-ingestion module to data center nodes running the extended ModelarDB cloud query and storage engines. Since bandwidth can be limited for RES installations in remote places, the data transfer module has to optimize the usage of bandwidth such that urgent data is transferred immediately while less important data are transferred in bulk later when the bandwidth is available. Also, shared prediction on the coefficients is applied to reduce bandwidth requirements. When to transfer the models is dictated by the Workload Balancer using strategies provided by the Cloud Query Engine (WP4 Task T4.4.). Alerts are propagated directly to user applications, e.g., for operators, in a dedicated high-speed and high-reliability pipeline.

### 3.4.1 Functional Specifications

| Field | Short Description |
|---|---|
| **ID** | FS-Transfer-1 |
| Name | Model Transfer |
| Purpose | Transfer model-based data representations from the edge to the cloud. |
| Input | 1) Model-based data representations on the edge<br>2) Output from edge analytics engine |
| Output | Model-based data representation stored in the cloud<br>Alerts delivered to end-user sites |
| Operation | Segments will be transferred from the edge to the cloud using Apache Arrow Flight (gRPC) when there is enough bandwidth available. |
| Used technologies | Apache Arrow Flight (gRPC) |
| Target Users | System Administrator |

| Field | Short Description |
|---|---|
| ID | FS-Transfer-2 |
| Name | Alerts Handling |
| Purpose | Prioritize and transfer alerts |
| Input | Alerts from edge analytics engine |
| Output | Alerts delivered to end-user |
| Operation | Alerts will be transferred to the end-user before the segments are transferred to the cloud. |
| Used technologies | gRPC |
| Target Users | Data Scientists |

| Field | Short Description |
|---|---|
| ID | FS-Transfer-3 |
| Name | Combine Models |
| Purpose | Combine similar models into one to further increase compression ratio. |

| | |
|---|---|
| Input | Model-based data representations in the edge |
| Output | Model-based data representation stored in the cloud |
| Operation | See FS-Ingest-03 |
| Used technologies | Rust, DataFusion, and Apache Arrow Flight (gRPC) |
| Target Users | Data Scientists |

| **Field** | **Short Description** |
|---|---|
| ID | FS-Transfer-4 |
| Name | Shared prediction |
| Purpose | Prediction model coefficients instead of transferring them. |
| Input | Model-based data representations in the edge |
| Output | Predicted model coefficients stored in the cloud |
| Operation | Edge will not transmit a segment if a predictor stored on the edge and the cloud can predict the data it represents. |
| Used technologies | Apache Arrow Flight (gRPC) |
| Target Users | Data Scientists |

### 3.4.2 Non-functional Specifications

| ID | Name | Purpose | Module | Technical solution \ mean of verification |
|---|---|---|---|---|
| NFS-Transfer-01 | Bandwidth Limits | Support uplink with only 500kbps bandwidth | Data Transfer | Time series will be represented as models and thus be significantly compressed. Also, similar models will be combined before being transferred to the cloud. Shared prediction will be employed to reduce the number of models that must be transmitted. Last, work load balancer will prioritize alerts and transfer segments when bandwidth is available. The data transfer module will be evaluated on networks with different amounts of bandwidth. |

## 3.5 Auto ML Module

AutoML on incremental machine learning algorithms (IMLA) with federated learning will be explored for forecasting and prediction tasks on the cloud as well as on the edge. Fundamental IMLA models and FL will be initially explored. AutoML pipeline is built on top of these IMLA models coupled with FL. AutoML APIs are expected to be the same or a subset of the APIs of IMLA & Scalable Forecasting platform which are described in detail in the next section.

### 3.5.1 Functional Specifications

| **Field** | **Short Description** |
|---|---|

| | |
|---|---|
| ID | FS-AutoML-1 |
| Name | AutoML Model Selector API |
| Purpose | Select the best model for a given prediction task. |
| Input | Streaming dataset |
| Output | Output the best model with performance metrics (optional) |
| Operation | Given a streaming dataset, the function should be able to generate the best model for the prediction task using either a standard test set or a streaming test set. |
| Used technologies | AMQP/Apache Kafka/ Apache Arrow Flight |
| Target Users | Business Users / Data Scientists |

| Field | Short Description |
|---|---|
| ID | FS-AutoML-2 |
| Name | AutoML Feature Selector API |
| Purpose | Select the most appropriate features for a given streaming data source and prediction task. |
| Input | Streaming dataset |
| Output | Dynamic selection of the most useful features |
| Operation | Given a streaming dataset, the function should be able to select the best features for the prediction task. |
| Used technologies | AMQP/Apache Kafka/ Apache Arrow Flight |
| Target Users | Business Users / Data Scientists |

| Field | Short Description |
|---|---|
| ID | FS-AUTOML-3 |
| Name | AutoML Forecast Service |
| Purpose | Run the best performing forecasting and prediction model that using IMLA Forecast and Prediction Model API. |
| Input | Metadata or UID |
| Output | Time series data that consists of timestamps and forecasts/predictions generated by the best performing machine learning model. |
| Operation | Querying best performing machine learning model for the required task and score the models. |
| Used technologies | AMQP/Kafka /Apache Arrow Flight |
| Target Users | Data scientists |

### 3.5.2 Non-functional Specifications

The non-functional specifications of Auto ML are defined in the next section (4.6) since they are overlapping.

## 3.6 IMLA & Scalable Forecasting Module

IMLA Scalable Forecasting (WP3) provides a scalable platform for novel incremental algorithms and orchestration for federated learning models that would be deployed on the edge. The platform provides real time and on-demand forecasts and predictions. In addition, the service also orchestrates the training of federated learning models that can be deployed on the edge for scoring. Incremental

models will be built on Pyspark and Cassandra using the summarized data that is persisted in ModelarDB.
Software development will follow an agile approach with CI/CQ tools like Github and Jenkins for source code management and build and deployment. Development environment and integrations will leverage platforms and services like Kubernetes, RabbitMQ and Functions as a Service (FaaS). In addition, Cloud Object Store and DB2 are planned to be used for storing models and meta information of the models. Existing opensource software and python packages will be used extensively during code development. Deployment is expected to be as Docker containers or Python code.

**IMLA Model Service:**
IMLA model service mainly constitutes IMLA models for training and scoring on demand. Model instances are created and deployed using IMLA Forecast and Prediction Model API. Summarized data required for on demand forecasts and predictions are retrieved from Cloud Data Storage and Query API. Historical summarized data required for training/scoring the models are retrieved using IMLA Historical Data API. Forecasts are persisted to the Cloud Data Storage and Query API and can be accessed using IMLA Forecast API.

**FL Model Service:**
For edge learning models, FL (Federated Learning) will be employed. So, the cloud service will also integrate the FL platform required for FL training and orchestration. Edge data and data in Cloud Data Storage and Query API will be used for FL training. Trained models are stored in Cloud Object Storage and deployed on the edge for real-time forecasting. Integrating FL for training requires detailed functional requirements that describe the seamless interactions between the edge node and the cloud server. However, only the major APIs with reference to FL are mentioned below. For a detailed list of all the APIs corresponding to FL, the reader is referred to the deliverable in the EU project, Musketeer.

### 3.6.1 Functional Specifications

| Field | Short Description |
|---|---|
| ID | FS-IMLA-1 |
| Name | IMLA Historical Data API |
| Purpose | Querying and retrieval of summarized data from ModelarDB for training/scoring/visualization. |
| Input | Metadata or UID |
| Output | Historical time series data in Apache Arrow or similar formats |
| Operation | Historical data is used for benchmarking the incremental models, train the incremental models in an offline manner and generating test set for benchmarking |
| Used technologies | AMQP/REST/Kafka /Apache Arrow Flight |
| Target Users | Data scientists |

| Field | Short Description |
|---|---|
| ID | FS-IMLA-2 |
| Name | IMLA forecast and Prediction Model API |
| Purpose | Querying and retrieval of machine learning models for forecast and predictions. |
| Input | Metadata or UID |
| Output | Machine learning models in pickled format or other compressed formats. |

| | |
|---|---|
| Operation | Querying machine learning models that are required to score the models. |
| Used technologies | AMQP/REST/Kafka |
| Target Users | Data scientists |

| Field | Short Description |
|---|---|
| ID | FS-IMLA-3 |
| Name | IMLA Forecast Service |
| Purpose | Run the forecasting and prediction models that were registered using IMLA Forecast and Prediction Model API. Forecasts are stored using IMLA Forecast API. |
| Input | Metadata or UID |
| Output | Time series data that consists of timestamps and forecasts/predictions generated by the machine learning task. |
| Operation | Querying machine learning models for the required task and score the models. |
| Used technologies | AMQP/Kafka /Apache Arrow Flight |
| Target Users | Data scientists |

| Field | Short Description |
|---|---|
| ID | FS-IMLA-4 |
| Name | IMLA Forecast API |
| Purpose | Manage or query time series forecasts or predictions. |
| Input | Metadata or UID and time window (optional) |
| Output | Time series data for the given time window. |
| Operation | Time series data for visualization, analytics, and post processing. |
| Used technologies | AMQP/REST /Apache Arrow Flight |
| Target Users | Business Users |

| Field | Short Description |
|---|---|
| ID | FS-FL-1 |
| Name | FL Model API |
| Purpose | Manage or query FL training and orchestration. |
| Input | Depends on FL Task |
| Output | Depends on FL Task |
| Operation | FL orchestration to get the client/edge models and send the updated models back to the client nodes. |
| Used technologies | AMQP/Kafka |
| Target Users | Data scientists |

| Field | Short Description |
|---|---|
| ID | FS-FL-2 |
| Name | FL Task Status API |
| Purpose | Manage or query task status updates. |
| Input | • Metadata or UID<br>• Participant information<br>• Training status |

| | |
|---|---|
| Output | Success/Failure |
| Operation | Status update of a FL model to check if training is updated and the edge model is ready for scoring. |
| Used technologies | AMQP/Kafka |
| Target Users | Data scientists |

| Field | Short Description |
|---|---|
| ID | FS-FL-3 |
| Name | FL Edge Contributions API |
| Purpose | Ability to store model parameters and meta information of the participants |
| Input | • Metadata or UID<br>• Participant information<br>• Model parameters |
| Output | Success/Failure |
| Operation | Metadata of the client/edge is stored in the central server and used to match the clients that can be grouped for federated learning. |
| Used technologies | AMQP/Kafka |
| Target Users | Data scientists |

### 3.6.2 Non-functional Specifications

| ID | Name | Purpose | Module | Technical solution\mean of verification |
|---|---|---|---|---|
| NFS-IMLA-1 | Model Storage | Repository to store FL and IMLA models | IMLA forecast and Prediction Model API | Success/failure output on using APIs for storage and retrieval of FL and IMLA models. |
| NFS-IMLA-2 | Forecast Storage | Repository/database to store forecasts and predictions. | IMLA Forecast Service | Success/failure output on using APIs for storage and retrieval of forecasts. |
| NFS-IMLA-3 | IMLA Backup | Incremental backup and recovery of the repositories for model storage and forecasts. | IMLA Forecast Service, IMLA forecast and Prediction Model API | Backup cloud services will be managed to ensure all stored models and data can be recovered due to failures. |

## 3.7 Pattern Extraction Module

The Pattern Extraction module (PE) is responsible for analysing the RES time series data and extracting meaningful patterns in the form of motifs, discords, changepoints, etc. In the frame of the module, specialized pattern detection algorithms will be implemented that incorporate domain knowledge and exploit it to solve RES-specific (Wind Turbine and Solar Park) pattern recognition tasks.

The PE module will incorporate the following core functionalities: (a) Calculating an index (in particular, the Matrix Profile) on top of the time series that holds distance information between all different parts of a time series, having considered a window length m; (b) Running variations of motif discovery algorithms in order to identify different types of patterns, including motifs, discords and changepoints; (c) Annotating time series areas via Annotation Vectors in order to incorporate domain knowledge; (d) Storing the output of the aforementioned processing (indexes, patterns) in the Patterns Repository.

The input of the PE module comprises time series data. Indicative input data categories follow:

- The initial time series data
- Modelled (summarized) time series data produced by the respective ModelarBD module
- An annotation vector (essentially a time series of the same length with the initially input time series of interest) that serves as a user-input index on the time series that adjusts the importance of different time series areas w.r.t. processing and analysis

These can be provided in the following formats: (a) parquet files; (b) CSV files; (c) as a call to a dedicated API provided by the underlying ModelarDB module.

The PE produces the following types of output, all provided in the form of time series data, with potential metadata accompanying each time series output. Indicative output data categories follow:

- A MP index on the input time series that can be next utilized for performing analytics on the time series
- An updated set of patterns (motifs, discords, changepoints) that may comprise input for analytics algorithms
- Annotations on the input time series in the form of points identified on it or a new time series that annotates the input one

Similarly, the output of the module will be provided in the following formats: (a) parquet files stored on disk; (b) CSV files stored on disk; (c) as a call to a dedicated API provided by the underlying ModelarDB module.
No particular API is currently prescribed for the PE module; instead, utilizing the APIs of other modules/modules of the MORE platform (e.g., ModelarDB) will be an option as the integration of the module proceeds.

The module will be built on top of the Matrix Profile[1] (MP) framework and the respective suite of algorithms that exploit it in order to perform efficient analytics on top of time series data. To this end, several python libraries that calculate the MP and perform further processing/analytics on it, in particular: STUMPY[2], MatrixProfile[3], and SCAMP[4]. The aforementioned libraries are implemented in python3 (or provide python3 bindings to C++ code), thus the PE module will adopt the same programming language.
An installation of the CUDA toolkit[5] is necessary for deploying the GPU-accelerated versions of the libraries.

### 3.7.1 Functional Specifications

| Field | Short Description |
|---|---|
| ID | FS-PE-01 |
| Name | Sample a time series in various time scales |

| | |
|---|---|
| Purpose | The user will be able to sample the time series in a parameterized granularity level |
| Input | Initial multivariate time series in the form of a dataframe;<br>Desired sampling interval (in seconds) |
| Output | New time series with the new granularity |
| Operation | The initial time series is sampled according to the granularity level the user has input, measured in seconds |
| Used technologies | Python/Flask |
| Target Users | Data scientists |

| Field | Short Description |
|---|---|
| ID | FS-PE-02 |
| Name | Preprocess/filter a time series |
| Purpose | The user will be able to perform several preprocessing and filtering operations on top of a multivariate time series, using several variables of the time series |
| Input | Multivariate time series in the form of a dataframe;<br>Variable of interest and threshold value |
| Output | New, preprocessed/filtered time series |
| Operation | A new time series is generated by performing a series of actions on the initial one, including filtering by minimum or maximum value, on each of the variable of the time series |
| Used technologies | Python/Flask |
| Target Users | Data scientists/Business users |

| Field | Short Description |
|---|---|
| ID | FS-PE-03 |
| Name | Annotate a time series by using a custom user defined annotation function. |
| Purpose | The user will be able to define their own annotation function by providing their own filters/rules as a function of selected variables of the time series. The function will be assigning weights on different segments on the time series that denote the importance of each segment for the pattern discovery process |
| Input | Univariate/multivariate time series in the form of a dataframe;<br>User defined function for annotation |
| Output | Annotation Vector characterizing the initial time series |
| Operation | The function will support the definition of specific rules and weights to be used as input to the Annotation Vector functionality of the Matrix Profile suite of algorithms, allowing the incorporation of domain specific information as input for the motif (pattern) discovery process |
| Used technologies | Python/Flask |
| Target Users | Data scientists |

| Field | Short Description |
|---|---|
| ID | FS-PE-04 |

| | |
|---|---|
| Name | Annotate a time series by using a set of pre-implemented annotation functions for generic use on RES time series. |
| Purpose | The user will be able to pick one of the pre-implemented annotation vector functions provided and use it to annotate the time series at hand. The annotation will be denoting the importance of each segment in the pattern discovery process. Those pre-implemented annotation vector functions will be providing a collection of generally useful functionalities on RES time series (e.g., annotating the constant regions of a time series as “low importance” regions, annotating a region depending on the local standard deviation compared to the global one etc.) |
| Input | Univariate/Multivariate time series in the form of a dataframe;<br>Selected pre-implemented annotation function |
| Output | Annotation Vector characterizing the initial time series |
| Operation | The function will support the selection of a set of pre-implemented functions to produce an Annotation Vector for a time series, in order to filter out regions with trivial behavior from the pattern extraction and discovery process |
| Used technologies | Python/Flask |
| Target Users | Data scientists/Business users |

| Field | Short Description |
|---|---|
| ID | FS-PE-05 |
| Name | Automatically identify optimal Annotation Vector parameters |
| Purpose | The user will be able to automatically identify annotations that maximize the identification of optimal patterns for a specific task |
| Input | Multivariate time series in the form of a dataframe;<br>Selected objective function to be optimized for the specific task for annotation (in the form of individual, task specific functions that are applied on patterns identified on the time series) |
| Output | Annotation vector that optimizes the task’s objective, characterizing the initial time series as its metadata |
| Operation | An iterative optimization process will take place that will search the hyperparameter space of the AV and MP algorithms with respect to optimizing the produced solution on a specific task. This process will produce a set of optimal hyperparameters (e.g. motif length) to be used during the execution of the algorithms in similar tasks |
| Used technologies | Python/Flask |
| Target Users | Data scientists |

| Field | Short Description |
|---|---|
| ID | FS-PE-06 |
| Name | Normalize a time series |
| Purpose | The user will be able to normalize a univariate time series based on standardized normalization schemes |
| Input | Univariate time series in the form of a dataframe;<br>Selected normalization scheme |

| | |
|---|---|
| Output | New, normalized time series |
| Operation | The function will produce a new time series, normalized according to a selected normalization scheme |
| Used technologies | Python/Flask |
| Target Users | Data scientists/Business users |

| **Field** | **Short Description** |
|---|---|
| ID | FS-PE-07 |
| Name | Approximate a time series |
| Purpose | The user will be able to approximate a univariate time series based on either a theoretical formula or a regression function |
| Input | Multivariate time series in the form of a dataframe;<br>Selected model for approximation (in the form of individual, task specific functions that are applied on the initial time series) |
| Output | Approximate time series |
| Operation | The function will produce a second time series that will approximate the initial one, according to either a pre-defined theoretical formula, or a set of pre-defined regression models |
| Used technologies | Python/Flask |
| Target Users | Data scientists/Business users |

| **Field** | **Short Description** |
|---|---|
| ID | FS-PE-08 |
| Name | Extract statistics on a time series |
| Purpose | The user will be able to extract statistics from a multivariate time series |
| Input | Multivariate time series in the form of a dataframe; |
| Output | A set of statistics characterizing the time series as its metadata |
| Operation | Th function will provide an “overview” of the input time series, considering the variable it consists of and their statistical properties, such as min/max/mean, etc. |
| Used technologies | Python/Flask |
| Target Users | Data scientists/Business users |

| **Field** | **Short Description** |
|---|---|
| ID | FS-PE-09 |
| Name | Compare two univariate time series |
| Purpose | The user will be able to compare two univariate time series by measuring their difference in terms of standardized time series measures (e.g. MAPE, NMSE) |
| Input | Two univariate time series in the form of dataframes;<br>The desired comparison measure |
| Output | The value of the comparison measure |
| Operation | The function will take as input two time series and will compare them using a selected measure (from a set of pre-implemented ones), providing a value that quantifies the similarity of the two compared time series |
| Used technologies | Python/Flask |

| | |
|---|---|
| Target Users | Data scientists/Business users |

| Field | Short Description |
|---|---|
| ID | FS-PE-10 |
| Name | Compare two multivariate time series |
| Purpose | The user will be able to compare two multivariate time series by measuring and weighting their difference in terms of standardized time series measures (e.g. MAPE, NMSE) considering a selected set of their variables |
| Input | Two multivariate time series in the form of dataframes;<br>The desired comparison measure;<br>The selected variables to be considered for comparison and the respective weights |
| Output | The value of the comparison measure |
| Operation | The function will take as input two time series and will compare them using a selected measure (from a set of pre-implemented ones), providing a value that quantifies the similarity of the two compared time series |
| Used technologies | Python/Flask |
| Target Users | Data scientists/Business users |

| Field | Short Description |
|---|---|
| ID | FS-PE-11 |
| Name | Identify the k-Nearest Neighbors of a univariate time series with respect to a standardized evaluation measure |
| Purpose | The user will be able to find the k most similar time series to a reference univariate time series comparing them with a standardized time series measure (e.g. MAPE, NMSE, z-normalized Euclidean Distance and Dynamic Time Warping) |
| Input | The reference time series;<br>A set of candidate time series;<br>The desired comparison measure;<br>The desired number of results k |
| Output | A ranked list of the k most similar time series and the respective similarity scores |
| Operation | The function will return the top-k most similar time series of a reference time series, with respect to a selected comparison measure (FS-PE-09) |
| Used technologies | Python/Flask |
| Target Users | Data scientists/Business users |

| Field | Short Description |
|---|---|
| ID | FS-PE-12 |
| Name | Identify the k-Nearest Neighbors of a multivariate time series with respect to a standardized evaluation measure |
| Purpose | The user will be able to find the k most similar time series to a reference multivariate time series comparing them with a standardized time series measure (e.g. MAPE, NMSE) |

| | |
|---|---|
| Input | The reference time series;<br>A set of candidate time series;<br>The desired comparison measure;<br>The desired number of results k;<br>The selected variables to be considered for comparison and the respective weights |
| Output | A ranked list of the k most similar time series and the respective similarity scores |
| Operation | The function will return the top-k most similar time series of a reference time series, with respect to a selected comparison measure (FS-PE-10) |
| Used technologies | Python/Flask |
| Target Users | Data scientists/Business users |

| **Field** | **Short Description** |
|---|---|
| ID | FS-PE-13 |
| Name | Index a univariate/multivariate time series via Matrix Profile |
| Purpose | The user will be able to index a time series with the Matrix Profile algorithm in a parameterized way (e.g. w.r.t. the pattern's length) |
| Input | Univariate/multivariate time series in the form of a dataframe;<br>A list of selected pattern lengths; |
| Output | A list of Matrix Profile indexes corresponding to different motif lengths, characterizing the time series as its metadata |
| Operation | The function will create a MP index that will allow the extraction of different types of motifs and discords |
| Used technologies | Python/Flask |
| Target Users | Data scientists/Business users |

| **Field** | **Short Description** |
|---|---|
| ID | FS-PE-14 |
| Name | Extract interesting patterns (motifs) from a univariate/multivariate time series |
| Purpose | The user will be able to identify interesting patterns (motifs) in a parameterized way (e.g. w.r.t. the pattern's length) |
| Input | Univariate/multivariate time series in the form of a dataframe;<br>Selected pattern's length;<br>Number of different motif types to be identified;<br>Number of instances of each motif type to be retrieved |
| Output | A list of identified motif types in the form of time series segments characterizing the time series as its metadata;<br>A list of instances (positions on the initial time series) for each motif type; |
| Operation | The function will use an existing MP index to extract different types of interesting motifs. In particular, it will allow the selection of the number of motif types and the number of instances per motif type to be returned by the system |
| Used technologies | Python/Flask |
| Target Users | Data scientists/Business users |

| Field | Short Description |
|---|---|
| ID | FS-PE-15 |
| Name | Extract interesting patterns (motifs) from a summarized univariate time series |
| Purpose | The user will be able to identify interesting patterns (motifs) in a parameterized way (e.g. w.r.t. the pattern's length) within a summarized (modelled) time series |
| Input | Univariate summarized time series;<br>Selected pattern's length;<br>Number of different motif types to be identified;<br>Number of instances of each motif type to be retrieved |
| Output | A list of identified motif types in the form of time series segments characterizing the time series as its metadata;<br>A list of instances (positions on the initial time series) for each motif type; |
| Operation | The function will use an existing MP index on summarized time series to extract different types of interesting motifs. In particular, it will allow the selection of the number of motif types and the number of instances per motif type to be returned by the system |
| Used technologies | Python/Flask |
| Target Users | Data scientists/Business users |

| Field | Short Description |
|---|---|
| ID | FS-PE-16 |
| Name | Identify anomalies (discords) in a univariate/multivariate time series |
| Purpose | The user will be able to identify anomalies (discords) in a parameterized way (e.g. w.r.t. the pattern's length) |
| Input | Univariate/multivariate time series in the form of a dataframe;<br>Selected pattern's length;<br>Number of different discord types to be identified; |
| Output | The identified discord in the form of time series segment characterizing the time series as its metadata |
| Operation | The function will use an existing MP index to extract different types of interesting anomalies. In particular, it will allow the selection of the number of anomaly types to be returned by the system |
| Used technologies | Python/Flask |
| Target Users | Data scientists/Business users |

| Field | Short Description |
|---|---|
| ID | FS-PE-17 |
| Name | Identify anomalies (discords) in a summarized univariate time series |

| | |
|---|---|
| Purpose | The user will be able to identify anomalies (discords) in a parameterized way (e.g. w.r.t. the pattern's length) from a summarized (modelled) time series |
| Input | Univariate summarized time series;<br>Selected pattern's length;<br>Number of different discord types to be identified;<br>Number of instances of each discord type to be retrieved |
| Output | The identified discord in the form of time series segment characterizing the time series as its metadata |
| Operation | The function will use an existing MP index on summarized time series to extract different types of interesting anomalies. In particular, it will allow the selection of the number of anomaly types to be returned by the system |
| Used technologies | Python/Flask |
| Target Users | Data scientists/Business users |

| Field | Short Description |
|---|---|
| ID | FS-PE-18 |
| Name | Identify changepoints in a univariate/multivariate time series |
| Purpose | The user will be able to identify points on a univariate/multivariate time series where the behavior/distribution changes with respect to one or more variables |
| Input | Univariate/multivariate time series in the form of a dataframe;<br>Selected pattern's length; |
| Output | Index on the initial time series denoting the changepoint, characterizing the time series as its metadata |
| Operation | The function will identify areas of the time series with different distributions and will return the points on the time series where the distribution changes |
| Used technologies | Python/Flask |
| Target users | Data scientists/Business users |

| Field | Short Description |
|---|---|
| ID | FS-PE-19 |
| Name | Identify trends in a univariate/multivariate time series |
| Purpose | The user will be able to identify downward/upward trends on a univariate/multivariate time series with respect to one or more variables |
| Input | Univariate/multivariate time series in the form of a dataframe; |
| Output | Index on the initial time series denoting the starting point of a trend or maximal time intervals where trends are observed |
| Operation | The function will return points and/or intervals on the time series that denote an identified trend |
| Used technologies | Python/Flask |
| Target users | Data scientists/Business users |

| Field | Short Description |
|---|---|

| | |
|---|---|
| ID | FS-PE-20 |
| Name | Decompose a time series into Seasonal, Trend and Remainder modules. |
| Purpose | The user will be able to decompose a time series into three modules (seasonal, trend, remainder) with the usage of of traditional (additive/multiplicative decomposition) or more sophisticated (STL) techniques. |
| Input | A univariate time series.<br>The selected decomposition method. |
| Output | Three time series representing each respective module. |
| Operation | The function will decompose the initial time series into potentially identified (see FS-PE-19) seasonal, trend and remainder modules |
| Used technologies | Python/Flask |
| Target Users | Data scientists/Business users |

| Field | Short Description |
|---|---|
| ID | FS-PE-21 |
| Name | Segment a univariate/multivariate time series into regions of different behavior |
| Purpose | The user will be able to divide (segment) a time series into a number of k regions that present different behavior w.r.t. the patterns (motifs) they contain |
| Input | Univariate/multivariate time series in the form of a dataframe;<br>Number of segments k;<br>Selected pattern's length |
| Output | Indexes on the initial time series denoting the different segments, characterizing the time series as its metadata |
| Operation | The function will use the MP segmentation algorithm to identify areas of the time series with different distributions and will return the points on the time series where the distribution changes |
| Used technologies | Python/Flask |
| Target Users | Data scientists/Business users |

## 3.7.2 Non-functional Specifications

| ID | Name | Purpose | Module | Technical solution\mean of verification |
|---|---|---|---|---|
| NFS-PE-01 | Indexing time | Given a time series, the construction of the MP index should be completed in reasonable time (a few hours depending on the length of the time series and the available hardware) | Pattern Extraction | Apply or improve upon state-of-the-art algorithms for computing the Matrix Profile. |

| | | | | |
|---|---|---|---|---|
| NFS-PE-02 | Response time | Given a time series and/or its MP index and/or an annotation vector, all required events (e.g. motifs, discords, changepoints) should be extracted in reasonable time (a few seconds depending on the length of the time series and the available hardware) | Pattern Extraction | Apply or improve upon state-of-the-art algorithms for extracting information from a Matrix Profile index. |
| NFS-PE-03 | Documentation | The implemented tools will be accompanied with detailed implementation and execution documentation | Pattern Extraction | The documentation will be assessed by data scientist and business users. |

## 3.8 Complex Event Detection Module

The Complex Event Detection module (CED) is responsible for exploiting outcomes of the PE module in order to (a) identify complex events within RES time series and (b) be able to identify specific types of events in near-real time. In particular, the CED module will utilize the indexes built on top of the initial time series (Matrix Profile, Annotation Vectors, etc.), as well as precomputed patterns, like motifs, discords and changepoints. These will be exploited to identify inter-time series patterns (i.e. multivariate time series patterns) and to detect interesting patterns in new, incoming time series streams.

The CED module will incorporate the following core functionalities: (a) Updating existing indexes (e.g., the Matrix Profile), as new data points are constantly added to the time series; (b) Deploying event detection algorithms that exploit the underlying MP index and additional precomputed information (e.g. top-k motifs) in order to identify (nearly) real time patterns/events/incidents; (c) Deploying pattern recognition algorithms for identifying frequently cooccurring patterns (denoting complex patterns).

The input of the CED module comprises time series data. Indicative input data categories follow:

- The initial (or the modelled) time series data
- Indexed version of the initial (or the modelled) time series data (Matrix Profile, Annotation Vector)
- Precomputed patterns (motif, discords, changepoints) stored in the Pattern Repository
- Modelled (summarized) time series data produced by the respective ModelarBD module
- New data point of the time series in the form of stream data

These can be provided in the following formats: (a) parquet files; (b) CSV files; (c) as a call to a dedicated API provided by the underlying ModelarDB module.

The CED module produces the following types of output, all provided in the form of time series data, with potential metadata accompanying each time series output. Indicative output data categories follow:

- An updated MP index on the input time series that can be next utilized for performing analytics on the time series
- An updated set of patterns (motifs, discords, changepoints, sets of motifs), that may comprise complex events in the time series
- Annotations on the input time series in the form of points identified on it or a new time series that annotates the input one

Similarly, the output of the module will be provided in the following formats: (a) parquet files stored on disk; (b) CSV files stored on disk; (c) as a call to a dedicated API provided by the underlying ModelarDB module.
No particular API is currently prescribed for the CED module; instead, utilizing the APIs of other modules/modules of the MORE platform (e.g. ModelarDB) will be an option as the integration of the module proceeds

The module will be built on top of the Matrix Profile[6] (MP) framework and the respective suite of algorithms that exploit it in order to perform efficient analytics on top of time series data. To this end, several python libraries that calculate the MP and perform further processing/analytics on it, in particular: STUMPY[7], MatrixProfile[8] and, SCAMP[9] and pyscamp. The aforementioned libraries are implemented in python3 (or provide python3 bindings to C++ code), thus the CED module will adopt the same programming language.
An installation of the CUDA toolkit[10] is necessary for deploying the GPU-accelerated versions of the aforementioned libraries.

## 3.8.1 Functional Specifications

| Field | Short Description |
|---|---|
| ID | FS-CED-01 |
| Name | Update existing time series indexes |
| Purpose | The user will be able to periodically, automatically update existing univariate/multivariate time series indexes (e.g. Matrix Profile) with incoming (streaming) data points parameters |
| Input | Streaming univariate/multivariate time series;<br>Index on the time series |
| Output | Updated time series index |
| Operation | The function will periodically update/recompute an existing MP index |
| Used technologies | Python/Flask |
| Target Users | Data scientists/Business users |

| Field | Short Description |
|---|---|
| ID | FS-CED-02 |
| Name | Identify significant patterns/events on a univariate/multivariate time series nearly-real time |
| Purpose | The user will be able to exploit the underlying Matrix Profile and Annotation Vector indexes and the Pattern Repositoryto identify significant/interesting patterns (motifs) in (nearly) real time |

| | |
|---|---|
| Input | Streaming multivariate time series;<br>Indexes on the time series;<br>Precomputed patterns (motifs) form the Pattern Repository |
| Output | Identified pattern types and their positions on the streaming time series |
| Operation | The function will examine an incoming stream of data points and will identify patterns based on a set of pre-extracted motifs |
| Used technologies | Python/Flask |
| Target Users | Data scientists/Business users |

| Field | Short Description |
|---|---|
| ID | FS-CED-03 |
| Name | Identify significant anomalies on a univariate/multivariate time series nearly-real time |
| Purpose | The user will be able to exploit the underlying Matrix Profile and Annotation Vector indexes to identify anomalies and produce alerts in (nearly) real time |
| Input | Streaming multivariate time series;<br>Indexes on the time series;<br>Precomputed patterns (motifs/discords) form the Pattern Repository |
| Output | Identified anomaly (in the form of a time series segment) and their position on the streaming time series |
| Operation | The function will examine an incoming stream of data points and will identify anomalies based on a set of pre-extracted discords |
| Used technologies | Python/Flask |
| Target Users | Data scientists/Business users |

| Field | Short Description |
|---|---|
| ID | FS-CED-04 |
| Name | Identify changes in the time series behavior of a univariate/multivariate time series nearly-real time |
| Purpose | The user will be able to exploit the underlying Matrix Profile and Annotation Vector indexes to identify changepoints, i.e. identify changes in the time series behavior-distribution and produce alerts in (nearly) real time |
| Input | Streaming multivariate time series;<br>Indexes on the time series;<br>Precomputed patterns (motifs/discords) form the Pattern Repository |
| Output | Identified changepoint position on the streaming time series |
| Operation | The function will examine an incoming stream of data points and will identify changepoints based on existing patterns and statistics extracted from FS-PE-18 and FS-PE-21 |
| Used technologies | Python/Flask |
| Target Users | Data scientists/Business users |

| Field | Short Description |
|---|---|

| | |
|---|---|
| ID | FS-CED-05 |
| Name | Identify trends in a univariate/multivariate time series in nearly-real time |
| Purpose | The user will be able to identify downward/upward trends on a univariate/multivariate time series with respect to one or more variables in nearly-real time |
| Input | Streaming univariate/multivariate time series |
| Output | Index on the time series denoting the starting point of a trend or maximal time intervals where trends are observed |
| Operation | The function will examine an incoming stream of data points and will identify trends based on existing patterns and statistics extracted from FS-PE-19 |
| Used technologies | Python/Flask |
| Target Users | Data scientists/Business users |

| Field | Short Description |
|---|---|
| ID | FS-CED-06 |
| Name | Identify frequently cooccurring patterns on a univariate/multivariate time series |
| Purpose | The user will be able to exploit the underlying Matrix Profile and Annotation Vector and the Paterrn Repository indexes to identify frequently cooccurring patterns (denoting complex patterns) |
| Input | Streaming time series;<br>Indexes on the time series;<br>Precomputed patterns (motifs) form the Pattern Repository |
| Output | Identified pattern type groups and their positions on the streaming time series |
| Operation | The function will deploy frequent itemsets algorithms and sliding-window calculations to identify frequently co-occurring patterns (motifs) that might denote complex events |
| Used technologies | Python/Flask |
| Target Users | Data scientists/Business users |

| Field | Short Description |
|---|---|
| ID | FS-CED-07 |
| Name | Automatically identify optimal Matrix Profile parameters |
| Purpose | The user will be able to automatically identify Matrix Profile parameters (e.g. motif length, time series granularity, segmentation parameters) that maximize the identification of optimal patterns for a specific task |
| Input | Multivariate time series in the form of a dataframe;<br>Selected objective function to be optimized for the specific task (in the form of individual, task specific functions that are applied on patterns identified on the time series);<br>Set of parameters and their respective parameter space to be searched |
| Output | The set of optimal values for the input parameters |

| | |
|---|---|
| Operation | An iterative optimization process will take place that will search the hyperparameter space of the MP algorithms with respect to optimizing the produced solution on a specific task. This process will produce a set of optimal hyperparameters (e.g. motif length, time series granularity, segmentation parameters) to be used during the execution of the algorithms in similar tasks |
| Used technologies | Python/Flask |
| Target Users | Data scientists |

| **Field** | **Short Description** |
|---|---|
| ID | FS-CED-08 |
| Name | Automatically identify optimal Matrix Profile parameters |
| Purpose | The user will be able to automatically identify Matrix Profile parameters (e.g. motif length, time series granularity, segmentation parameters) that maximize the identification of optimal patterns for a specific task |
| Input | Multivariate time series in the form of a dataframe;<br>Selected objective function to be optimized for the specific task (in the form of individual, task specific functions that are applied on patterns identified on the time series);<br>Set of parameters and their respective parameter space to be searched |
| Output | The set of optimal values for the input parameters |
| Operation | An iterative optimization process will take place that will search the hyperparameter space of the MP algorithms with respect to optimizing the produced solution on a specific task. This process will produce a set of optimal hyperparameters (e.g. motif length, time series granularity, segmentation parameters) to be used during the execution of the algorithms in similar tasks |
| Used technologies | Python/Flask |
| Target Users | Data scientists |

| **Field** | **Short Description** |
|---|---|
| ID | FS-CED-09 |
| Name | Identify patterns on a univariate/multivariate time series based on sparsely supervised data |
| Purpose | The user will be able to exploit learned Matrix Profile parameterizations on pre-existing labelled (golden truth) data, in order to identify task-specific patterns on newly incoming time series data |
| Input | Streaming multivariate time series;<br>Indexes on the time series;<br>Precomputed patterns (motifs) form the Pattern Repository |
| Output | Identified pattern types and their positions on the streaming time series |
| Operation | The function will select a pre-calculated model from FS-CED-07 so that to build a MP with specific |

| | |
|---|---|
| | hyperparameterizations and utilize it to extract/detect patterns on new data |
| Used technologies | Python/Flask |
| Target Users | Data scientists/Business users |

## 3.8.2 Non-functional Specifications

| ID | Name | Purpose | Module | Technical solution\mean of verification |
|---|---|---|---|---|
| NFS-CED-01 | Update time | Each update on the current Matrix Profile index with new time series must be much faster than constructing the index from scratch. | Event Detection | The Matrix Profile index can be constructed by an incremental algorithm that efficiently supports insertions of new time series. |
| NFS-CED-02 | Testabilit y | The user should be able to verify the results easily. | Event Detection | Output format will be compatible with the relevant visualization tools, so that the user can visually verify that the results are meaningful. |
| NFS-CED-03 | Documen tation | The implemented tools will be accompanied with detailed implementation and execution documentation | Event Detection | The documentation will be assessed by data scientist and business users. |

## 3.9 Visual Engine and Visual Analytics Module

This Visual Analytics Module (VA) will offer the functionality for MORE platform users to interact with the timeseries data and be informed on various KPIs and complex events in real time. This module will offer a dashboard like UI for the visualization and interaction with various types of data, such as hierarchical multidimensional timeseries.

**Presentation Layer.** VA will employ web-based JS technologies for the rendering of the data and the interaction with users. A dashboard UI will contain a map for spatial exploration over the RES sensors, one or more panels for time series visualization of data and motifs, a panel for filtering, and auxiliary panels for alerting, user parameterization, etc.
**Visual Model and Engine.** VA will maintain in memory a visual model with all the necessary parameters for the visual representation and exploration of the data; i.e., parameters corresponding to the type of interaction each data type supports (e.g., timeseries will support window-based operations), user preferences related to the rendering or the viewport of a visualization, etc.
Also, the VA will build and keep in memory a visual-specific index which will help users avoid the over-plotting of data on the presentation layer and in the same time achieve a small response time in producing a visualization. For example, in a hierarchical exploration of multidimensional timeseries

data, this module will consider (through the index) the hierarchy of the dimensions and the level of details the users request to compute or fetch pre-aggregated data at the front end. This is seamless to the user performing the exploration.
Finally, caching and pre-fetching of frequently requested query outputs will empower the visual analytics module, such that the response time is kept small based on the locality-based interaction of the user (e.g., a user visually explores areas of interest that are close to each other, based on a distance function)
**Data Layer:** We distinguish the following types of data, to which VA will have access to.

- RES data from sensors; raw or aggregated (compressed) collected from the edge nodes and stored in a MORE Cloud Data Storage, i.e., ModelarDB.
- RES generated data, such as motifs, events, which are produced by T4.1, T4.2 and stored in the MORE Cloud Data Storage.
- Visual specific configuration data, such as user preferences, visual results, etc, which will be stored in an app-specific database (postgres).

Visual operations will be translated and executed in the underlying model-based representation of ModelarDB. The latter must support a declarative way (e.g., SQL, API) for the access to data, via the Cloud query engine and offer capabilities for interactive workloads (low-latency, caching, etc). However, we will also consider the deployment of a distributed interactive query engine (e.g., Apache Druid) which can be integrated with ModelarDB, for the support of the VA functionalities. For this case, data from Cloud Data Storage will be re-indexed according to the visual model and the needs that user interaction poses.

The Visual Analytics Module will consist of a backend providing a REST API built using the Sping Boot Framework in Java and a frontend built using the React Javascript library. For the visualization of the data the D3.js library will be used for the time series visualization and LeafletJS for the visualization of the RES sensors on the map.

## 3.9.1 Functional Specifications

| Field | Short Description |
|---|---|
| ID | FS-VA-01 |
| Name | Initialize the state of a visualization |
| Purpose | Given a specific type of visualization the platform will initialize the state of a visualization by deciding on the initial layout of the rendered data before the user starts the visual exploration. The initialization will take into account past user preferences or user characteristics (e.g., data related to the origin country of the user are initially presented to the user) or even existing summaries for the data that are relevant to the user in order to facilitate the beginning of the interaction. |
| Input | A custom visualization's ID. |
| Output | A set of data objects enriched (e.g., JSON format) with visual information which describes the initial state of the visualization |
| Operation | The user initializes a visualization, the platform fetches the underlying data and decides the initial state of the visualization. |

| | |
|---|---|
| Used technologies | React |
| Target Users | Data Scientists / Business Users |

| Field | Short Description |
|---|---|
| ID | FS-VA-02 |
| Name | Query Navigation |
| Purpose | The user will be able to query the database and retrieve data that satisfies their query. |
| Input | A set of filters and operations that specify what the user wants to retrieve. |
| Output | Tabular or list representation of the data objects representing the RES data and their characteristics |
| Operation | The user is presented with a set of categorical filters that can be applied on the RES data. After the selection they can view, explore, and analyse the data that was retrieved by their query. |
| Used technologies | React |
| Target Users | Data Scientists / Business Users |

| Field | Short Description |
|---|---|
| ID | FS-VA-03 |
| Name | Map Navigation |
| Purpose | A world map containing all RES will be presented to the user. Users will be able to geolocate specific RES, zoom-in and out, filter RES based on their characteristics and view their status. Users will be alerted on the status of an RES based on its color on the map (red, yellow, green). This way they can easily check on the current health of the RES or even prevent future failures. |
| Input | A bounding-box on the map and filters for specific queries. |
| Output | Data objects each representing an RES located within the bounding-box that also satisfy the filters provided by the user. |
| Operation | The user navigates to a specific location on the map and applies specific filters. Afterwards they are presented with points on the map each representing a different RES. The user can click on each point to inspect the characteristics of each RES. |
| Used technologies | React, Leaflet |
| Target Users | Business Users |

| Field | Short Description |
|---|---|
| ID | FS-VA-04 |
| Name | Time series Visualization |
| Purpose | Explore time series data for univariate/multivariate timeseries. Detect and analyze trends, patterns, anomalies and changes in the timeseries. Specify filters that aid in the exploration of the timeseries. |

| | |
|---|---|
| Input | Time series, dimension, filters, time interval, granularity. |
| Output | Visualization of timeseries data for the specified input. |
| Operation | The user selects a specific RES timeseries to analyse. From there they can:<br>• Apply Filters to the Timeseries Dimension<br>• Select a window (time interval) in the data and explore its characteristics<br>• Change the time-series granularity.<br>• Zoom in\out (change the time granularity) on a specific window Scroll left\right on the timeline<br>• Visualize timeseries patterns (e.g., motifs) which are detected and stored in the system, filters can also be applied on motif characteristics. |
| Used technologies | React, Highcharts |
| Target Users | Data Scientists / Business Users |

| **Field** | **Short Description** |
|---|---|
| ID | FS-VA-05 |
| Name | Time series patterns (motifs) and anomalies (discords) visualization. |
| Purpose | Aid the user in their exploration, by visually presenting the result of the matrix profile analysis. This includes the top motif visualization and best segmentation visualization. For the segmentation part, the user can also compare the found segmented dates with annotations of their own choosing, to identify changes in the behavior of the timeseries. |
| Input | • A Matrix Profile (univariate/multivariate from a list of precomputed choices.<br>• A single variable (m) and the choice of dimensions that is used for the creation of a new user-defined Matrix Profile. |
| Output | Visual representation of the motifs and discords found in the time series for each dimension. |
| Operation | The user chooses from a pre-defined set of matrix profile configurations or chooses to create a new matrix profile based on configurations they provide. The creation of a new MP is costly and cannot always be done online. Because of this, the user will be alerted when the new MP has been created. Patterns and anomalies found by the MP will be presented to the user so they can further analyse the timeseries data. |
| Used technologies | React, Highcharts |
| Target Users | Data Scientists |

| **Field** | **Short Description** |
|---|---|
| ID | FS-VA-06 |

| | |
|---|---|
| Name | Visualize time series changes in behavior using the underlying Matrix Profile and Annotation Vector. |
| Purpose | Aid the user in their exploration, by visually presenting the result of the matrix profile analysis. Present the user with the results of the matrix profile segmentation. The user can also compare the found segmented dates with annotations of their own choosing, to identify changes in the behavior of the timeseries. |
| Input | • A Matrix Profile (univariate/multivariate from a list of precomputed choices.<br>• A single variable (L) or a list of variables to give us input to the segmentation. |
| Output | Visual representation of the segmentations found in the time series and how they compare to the Annotated dates. |
| Operation | The user chooses from a pre-defined set of matrix profile configurations or chooses to create a new matrix profile based on configurations they provide. The creation of a new MP is costly and cannot always be done online. Because of this, the user will be alerted when the new MP has been created. The user provides the segmentation variable and is then presented with the result of the matrix profile segmentation. |
| Used technologies | React, Highcharts |
| Target Users | Data Scientists |

| **Field** | **Short Description** |
|---|---|
| ID | FS-VA-07 |
| Name | Visualize a time series with its Seasonal, Trend and Remainder modules. |
| Purpose | The user will be presented with a visual representation of the three modules (seasonal, trend, remainder) of a time series of their choosing. |
| Input | A univariate time series.<br>The selected decomposition method. |
| Output | Three time series representing each respective module. |
| Operation | The user selects the time series they want to compare and the decomposition method. They are then presented with a chart which displays the modules. |
| Used technologies | React, Highcharts |
| Target Users | Data Scientists / Business Users |

| **Field** | **Short Description** |
|---|---|
| ID | FS-VA-08 |
| Name | Complex Event Visualization |
| Purpose | • Present visual recommendations of potential complex events, based on the pattern extraction and ML modules. |

| | |
|---|---|
| | • Alert the user for potential complex events, which will be visualized in timeseries charts and which can be explored at different levels of time granularities. |
| Input | Event / Pattern to be discovered |
| Output | Visualization of patterns or complex events. |
| Operation | The user will be presented with a list of recommendations and predictions regarding the behaviour of newly incoming time series data. This way they can utilize learned Matrix Profile parameterizations on pre-existing labelled data to be alerted on complex events that will occur now or in the future. |
| Used technologies | React, Highcharts |
| Target Users | Data Scientists / Business Users |

### 3.9.2 Non-functional Specifications

| ID | Name | Purpose | Module | Technical solution\mean of verification |
|---|---|---|---|---|
| NFS-VA-01 | Interactivity | User response time must be less than 1s for most exploration operations on the map or the timeseries charts. | Pilots | An in-memory visual-specific index will be maintained which will help achieve a small response time in producing a visualization. Also, caching and pre-fetching of data likely to be requested based on the locality-based interaction of the user will improve the interactivity. |
| NFS-VA-02 | User Experience and friendliness | Users should view important information and summarized data at a glance and navigate easily to more details. | Pilots | Information will be intuitively organized in charts and in the dashboard, and appropriate data abstraction \ reduction techniques will be employed to provide overviews of the data and avoid visual clutter. |

## 3.10 Cloud Model Storage and Query Engine

Cloud Query Engine: A distributed Cloud Query Engine as extensions to ModelarDB with Apache Spark utilized for distribution and parallelization. Two sets of extensions will be implemented. Firstly, computation of matrix-profiles directly on models to efficiently support the advanced analytics described in WP4 Task 4.1. ModelarDB currently supports execution of simple aggregate queries (COUNT, MIN, MAX, SUM, AVG) on models which is more efficient than computation of simple aggregates on reconstructed data points. By implementing support for computing the matrix-profile from models, the MORE platform can efficiently perform advanced analytics on time series (e.g., discord and motif mining). Secondly, support for communicating with the Workload Balancers will be implemented as part of the cloud query engine. This allow it to decide what data must be transferred to the data center and at what precision the data must be provided. The cloud query engine can dynamically decide if a query should be executed in the data center or sent to the edge. In addition to

transferring data on demand, the edge nodes will pre-emptively transfer relevant data to the data center based on interesting events detected in the data at the edge, what data have previously been utilized by queries, and user-defined strategies.
Cloud Model Storage: A distributed model storage solution running at the data center nodes. This module will extend the Apache Cassandra-based solution utilized by ModelarDB with improved indexing to efficiently support extraction of relevant models for value-based queries and high-level analytics using matrix-profile.

### 3.10.1 Functional Specifications

| Field | Short Description |
|---|---|
| ID | FS-Cloud 1 |
| Name | Distributed model-based queries |
| Purpose | Support advanced SQL queries and analytics such as extraction of relevant models for value-based queries and high-level analytics using matrix-profile |
| Input | Models-based representation of time series and SQL |
| Output | SQL Result-sets |
| Operation | Initially execute value-based queries using a table scan and matrix-profile on reconstructed data points using Apache Spark. Later, value-based queries will be pruned using an index and matrix-profile will be computed directly from models. |
| Used technologies | Apache Spark, Apache Cassandra |
| Target Users | Data Scientists |

| Field | Short Description |
|---|---|
| ID | FS-Cloud 2 |
| Name | Workload Balancer |
| Purpose | Dynamically determine when and how to transfer data, transfer alerts, and execute queries. |
| Input | SQL and monitoring data. |
| Output | SQL Result-sets and Alerts |
| Operation | The module dynamically decides based on the available resources if a query should be executed in the cloud using existing data, if data must be requested from the edge, or if the query should be sent to the edge based on the available resources. The query execution will also take the priority of operations into account. |
| Used technologies | Apache Spark, Apache Cassandra |
| Target Users | System Administrator |

| Field | Short Description |
|---|---|
| ID | FS-Cloud 3 |
| Name | Distributed model storage |
| Purpose | Store the models in a distributed fashion to facilitate analytics of large datasets |
| Input | Model-based representations of time series |

| | |
|---|---|
| Output | Binary files on disk |
| Operation | Segments representing a sub-sequence of a time series using a model will be written to binary files on disk. Compaction will ensure small files are merged together. |
| Used technologies | Apache Spark, Apache Cassandra |
| Target Users | Requires no user interaction. |

### 3.10.2 Non-functional Specifications

| ID | Name | Purpose | Module | Technical solution\mean of verification |
|---|---|---|---|---|
| NFS-Cloud 4 | Scalability | | Cloud Query Engine and Cloud Model Storage | The cloud modules will use an existing proven distributed query engine, an existing proven distributed storage system, and the partitioning method proven by ModelarDB to ensure ModelarDB Cloud can scale as necessary. Scalability will be evaluated through an experimental evaluation. |
| NFS-Cloud 5 | Response Time | The response time should be fast enough to be useful for the users and other modules | Cloud Query Engine | Recent ingested and queried data will be cached in-memory as models, and queries that can executed directly on models instead of reconstructed data points will do so. |
| | | | | |

# 4 Continuous Integration and Deployment (CI\CD) in MORE

To facilitate the optimal code engineering and to ensure that developers will not spend their time in package building and tedious debugging processes and as well as to ensure the best possible collaboration between the development teams of MORE, we will use DevOps and CI\CD practices.

DevOps is a set of practices that combines software development and IT operations. It introduces processes, tools, and methodologies to balance needs throughout the software development life cycle, from coding and deployment, to maintenance and updates.

Automation is a core principle for achieving DevOps and CI/CD is a critical module. CI/CD comprises of continuous integration and continuous delivery. Put together, they will form a “CI/CD pipeline” which is a series of automated workflows that will allow our teams to reduce many manual tasks.

- Continuous integration (CI) automatically **builds**, **tests**, and **integrates** code changes within a shared repository; then
- Continuous deployment (CD) automatically **deploys** code changes to the platform.

The development teams in MORE will make CI/CD part of their development workflow with a combination of automated process, steps, and tools. Specifically, we will use the following CI\CD configuration:

1. **Version Control**: We will use shared repositories, where teams can collaborate on code using version control systems (VCS) like Git. A VCS keeps track of code changes and makes them easy to revert if something breaks. It also enables configuration as code, which will allow teams to manage testing, infrastructure, and more as versioned artifacts. In practice, we will create a Github organization (MORE project) and all our code repositories will reside within this Github organization.
2. **Builds**: We will use build tools automatically package up files and modules into release artifacts and run tests for quality, performance, and other requirements. Specifically, we will use Jenkins which is one of the most popular CI tools. It also has first class support for GitHub built into the default installation. Jenkins scans the entire GitHub organization and creates Pipeline jobs for each repository containing a Jenkinsfile—a text file that defines the process of building, testing, and deploying a project using Jenkins. Immediately after code is checked in or a new pull request is created, Jenkins will execute the Pipeline job and return the status to GitHub indicating whether it failed or succeeded. This process will allow the development teams to run a build and subsequent automated tests at every check. Catching bugs early and automatically will reduce the number of problems introduced into production, so our teams will be able to build better, more efficient software.

# 5 Users of the MORE Platform

In this section we present the outline of the targeted users of the MORE platform. The users are divided in two main categories: i) the Data Scientists and ii) the Business Users. The latter include module constructors, park constructors, park operators, investors, and other types of stakeholders without specialized knowledge on data and knowledge engineering while the former include domain experts, data analysts and generally any kind of stakeholder with specialized knowledge on data science and analytics.
MORE will allow stakeholders in industry sectors with huge volumes of sensor data, especially the RES industry, to: a) scale the management of streaming and historical time series at least an order of magnitude beyond the state-of-art and b) to perform forecasting, prediction and diagnostics using the whole data that is available to them with accuracy that outperforms existing approaches.

## 5.1 Business Users

What do the industry growth, evolvement and securitization mean for plants owners, investors, park constructors and operators? As mainstream investors group renewable assets into large portfolios with geographically dispersed plants totaling thousands of megawatts across the globe, scale is becoming the new currency. And key to scaling is big data, high-performance processing and powerful analytics. MORE will provide to these users a sophisticated renewable monitoring system enabling them to turn the massive streams of data from these multi-gigawatt-scale portfolios into an additional source of value. It will also provide the means to users without specialized knowledge on data and knowledge engineering to gain essential knowledge about a) patterns that show the optimal turbine and solar panel configurations to further improve the production optimization b) learning models and patterns that accurately indicate the condition of turbines and solar panels to improve diagnostic services, e.g., ice on wind blades, malfunctions in mechanical parts, underperforming modules, dirty/broken solar panels; c) prediction models that support predictive maintenance, i.e., methods to understand when a module will need replacement or service; and d) knowledge artefacts that can be linked to design and construction improvements for RES installations and modules.
For that purpose, MORE will develop a self-service visual analytics module for non-expert users to efficiently perform complex analytics in a visual manner. The ambition is to transform and optimize visual operations to efficient data access and analysis methods, that can be executed by MORE's query engine. Thus, allowing the stakeholders to get more precise KPI's from their plant, helping them to take better decisions for the production plan, the maintenance strategy and planning, and lifetime extension.

## 5.2 Data Scientists

MORE will also provide the means and the tools to expert-users like domain experts and data analysts to gain further insights on the functioning of their systems and fine-tune their configurations by setting optimal parameters to their assets. Expert users will be able to use and further fine-tune the ML and pattern recognition processes to gain and exploit insights for driving their R&D activities, further advancing their technologies, and potentially affecting their B.I. decisions. An indicative example is that MORE will provide the means to expert users to create Annotation Vectors (i.e., "weight" time series) on top of the initial time series through a function/algorithm that incorporates their domain knowledge, thus assigning different importance to different parts. This leads to the discovery of domain-specific motifs within the time series. The Annotation Vectors might comprise a simple seasonality dependent function, or more complex functions that encode complex/advanced domain knowledge of the expert.

# 6 Conclusion

The goal of the MORE project is to create a platform for time series management that will disrupt the way data from sensors is managed today. The vast amounts of data pose challenges in the responsiveness of the data management systems, but at the same time they provide the necessary information to create accurate prediction and diagnostics tools. Current big data systems are unable to cope with the data created in RES installations. Moreover, RES operators would like to receive sensor readings at much higher frequencies. Sensors can support higher frequencies, but companies lack the bandwidth and the data analytics tools to transfer and manage even greater data volumes. MORE will manage to consume billions of streams and 100s of petabytes of data by flexibly partitioning its system between cloud and edge and by adapting to the ratio of data traffic to computing cost. To this end, it will significantly extend the initial ModelarDB architecture and push the data ingestion and simple analytics to the edge nodes, creating an environment that combines cloud and edge computing. Moreover, MORE's architecture enables the provision of sophisticated analytics by implementing libraries that will add incremental machine learning capabilities and domain specific pattern extraction and detection methods. Specifically, MORE will achieve its aim for accuracy in prediction and diagnostics by focusing on incremental machine learning algorithms that can scale better than most existing approaches and are updated continuously. Moreover, it will also work on highly parallelizable pattern extraction methods, by focusing on motif extraction techniques which are developed especially for time series data.

In the present document we outlined the technical design and the architecture as well as the functional along with the non-functional specifications, including both the full MORE platform running in a cloud data center and the MORE edge platform running on edge nodes in the RES installations. Starting with an overview of the MORE logical architecture, we presented the main MORE software modules and the data flow paths between them. Next, we provided a more detailed view of each module along with its technical specifications, the sub-systems that comprise it, the interfaces (APIs) between modules and its dependencies. Finally, we discussed the targeted users of the MORE platform and provided an overview of our Common Integration Infrastructure strategy based on continuous integration (CI) and DevOps. Currently, we have successfully completed the technical design of the MORE platform and the overall architecture that combines edge with cloud data processing. Furthermore, we have also started the design and the implementation of the individual modules of the platform. Finally, the integration of all modules and the testing phase will start imminently.