



5G ExPerimentation Infrastructure hosting Cloud-native Network Applications for public proTecton and disaster RELief

Innovation Action – ICT-41-2020 - 5G PPP – 5G
Innovations for verticals with third party services

D2.6: 5G-EPICENTRE Analytics Engine

Delivery date: December 2023

Dissemination level: Public

Project Title:	5G-EPICENTRE - 5G ExPerimentation Infrastructure hosting Cloud-native Network Applications for public proTecton and disaster RELief
Duration:	1 January 2021 – 31 December 2023
Project URL	https://www.5gepicentre.eu/



Document Information

Deliverable	D2.6: 5G-EPICENTRE Analytics Engine
Work Package	WP2: Cloud-native 5G NFV
Task(s)	T2.5: ML-driven experimentation analytics
Type	Report
Dissemination Level	Public
Due Date	M36, December 31, 2023
Submission Date	M36, December 31, 2023
Document Lead	Luigi D'Addona (IST)
Contributors	Anna Maria Spagnolo (IST) Domenico Dato (IST)
Internal Review	ADS UMA

Disclaimer: This document reflects only the author's view and the European Commission is not responsible for any use that may be made of the information it contains. This material is the copyright of 5G-EPICENTRE consortium parties, and may not be reproduced or copied without permission. The commercial use of any information contained in this document may require a license from the proprietor of that information.

Document history

Version	Date	Changes	Contributor(s)
V0.1	29/09/2023	Initial deliverable structure	Luigi D'Addona (IST) Anna Maria Spagnolo (IST)
V1.0	14/12/2023	Internal Review Version	Luigi D'Addona (IST) Anna Maria Spagnolo (IST) Domenico Dato (IST)
V1.1	18/12/2023	First Internal Review comments and feedback	Jorge Márquez Ortega (UMA)
V1.2	19/12/2023	Second Internal Review comments and feedback	Laurent Drouglazet (ADS)
V1.5	20/12/2023	Version including feedback from the internal review	Luigi D'Addona (IST) Anna Maria Spagnolo (IST)
V1.6	21/12/2023	Quality check, including formatting and proof-reading	Konstantinos Apostolakis (FORTH)
V2.0	27/12/2023	Version including feedback from the quality check – Final version for submission	Anna Maria Spagnolo Luigi D'Addona

Project Partners

Logo	Partner	Country	Short name
	AIRBUS DS SLC	France	ADS
	NOVA TELECOMMUNICATIONS SINGLE MEMBER S.A.	Greece	NOVA
	Altice Labs SA	Portugal	ALB
	Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V.	Germany	HHI
	Foundation for Research and Technology Hellas	Greece	FORTH
	Universidad de Málaga	Spain	UMA
	Centre Tecnològic de Telecomunicacions de Catalunya	Spain	CTTC
	Istella SpA	Italy	IST
	One Source Consultoria Informatica LDA	Portugal	ONE
	Iquadrat Informatica SL	Spain	IQU
	Nemergent Solutions S.L.	Spain	NEM
	EBOS Technologies Limited	Cyprus	EBOS
	RedZinc Services Limited	Ireland	RZ
	OptoPrecision GmbH	Germany	OPTO
	Youbiquo SRL	Italy	YBQ
	ORamaVR SA	Switzerland	ORAMA
	Hewlett-Packard Italiana Srl	Italy	HPE

List of abbreviations

Abbreviation	Definition
5GTSM	5G Traffic Simulator Manager
AE	Analytics Engine
AI	Artificial Intelligence
API	Application Programming Interface
CNN	Convolutional Neural Network
DAGMM	Deep Autoencoding Gaussian Mixture Model
DL	Deep Learning
EM	Expectation-Maximization
GA	Grant Agreement
GMM	Gaussian Mixture Model
HSPF	Holistic Security and Privacy Framework
IP	Internet Protocol
JSON	JavaScript Object Notation
K8S	Kubernetes
KDE	Kernel Density Estimation
KPI	Key Performance Indicator
LSTM	Long Short-Term Memory
ML	Machine Learning
MQTT	Message Queuing Telemetry Transport
NS	Namespace
PPDR	Public Protection and Disaster Relief

PV	Persistent Volume
PVC	Persistent Volume Claim
QoE	Quality of Experience
QoS	Quality of Service
REST	Representational State Transfer
ROC-AUC	Area Under the Receiver Operating Characteristic Curve
SC	Storage Class
TSDB	Time Series Database
UC	Use Case
USAD	Unsupervised Anomaly Detection
VAE	Variational Autoencoders
VQ-VAE	Vector Quantized Variational Autoencoders

Executive summary

The 5G-EPICENTRE project was specifically focused on constructing a comprehensive 5G experimentation platform tailored to the Public Protection and Disaster Relief sector. The primary goal was to facilitate the analysis of data gathered by monitoring probes throughout the utilisation of the facilities, enabling the testing and validation of vertical applications by end-users. This report encapsulates the endeavours carried out under Task 2.5 "*ML-driven experimentation analytics*," with a primary objective of harnessing Artificial Intelligence for the thorough analysis of experimental data.

At the core of Task 2.5 lies the Analytics Engine, a pivotal component designed to monitor both the experimental conditions of vertical applications and the underlying infrastructure. The Analytics Engine is tasked with computing Key Performance Indicators (KPIs), to evaluate the attainment of targets established by experiments, and scrutinising measurements from the infrastructure to identify potential anomalies in network conditions.

The Analytics Engine employs a decentralised architecture that enhances the scalability and reliability of the system, reducing network overhead and latency. Various data processing modules within the engine, such as the Analytics Driver, KPI Monitor, Quality of Service / Quality of Experience (QoS/QoE) Monitor and Analytics Aggregator, work cohesively to collect, filter, aggregate and analyse data generated by experimental applications and infrastructure components.

The Analytics Engine provides data analytics services tailored to tackle health and performance analysis within the 5G-EPICENTRE platform during experiment executions. Through a meticulous examination of metrics generated by vertical applications, the Analytics Engine computes KPIs, and facilitates the assessment and validation activities.

This document further offers an in-depth overview of the solutions implemented in the QoS/QoE Monitor module within the Analytics Engine. This module employs Deep Learning techniques based on Vector Quantized Variational Autoencoders, to detect anomalies in metrics gathered from the testbed infrastructure. The analysis occurs in near-real time, and any identified anomalies are promptly relayed to the Front-end modules for visualisation: this guarantees that end-users can access detailed information regarding network performance during experiments, facilitating a more thorough evaluation of KPIs from an appropriate perspective.

Table of Contents

List of Figures.....	9
List of Tables.....	10
1 Introduction.....	11
1.1 Mapping of project’s outputs.....	11
2 The Analytics Engine.....	13
2.1 Architecture.....	13
2.1.1 Analytics Driver.....	15
2.1.2 KPI Monitor.....	16
2.1.3 QoS/QoE Monitor.....	16
2.1.4 Analytics Service API Server.....	16
2.1.5 Analytics Aggregator.....	17
2.2 Deployment of the Analytics Engine.....	18
3 Data Analytics.....	20
3.1 KPI calculation and statistical analysis.....	20
3.1.1 Outlier removal.....	20
3.2 Anomaly detection.....	21
3.2.1 Anomaly detection in time-series: state-of-the-art.....	22
3.2.2 Identification of a suitable classification approach.....	24
3.2.3 Data collection and data handling.....	28
3.2.4 Classification process.....	28
4 Testing and validation activities.....	31
4.1 Results for KPI calculation and statistical analysis.....	31
4.2 Results for the anomaly detection.....	35
5 Conclusions.....	38
References.....	39
Annex I : Portal API documentation.....	40
Annex II : Integration with the Testbed.....	43
Annex III : Integration with the Front-end.....	48

List of Figures

Figure 1: Architecture of the Analytics Engine and data flow	14
Figure 2: Integration among internal and external components	15
Figure 3: Schema of the Kubernetes deployment of the AE components at each testbed	18
Figure 4: Example of outliers to be removed from the timeseries	21
Figure 5: Schematic view of the VQ-VAE model.....	25
Figure 6: ROC curves for our model and the selected competitors	27
Figure 7: Performance of our model in a selected time window.....	28
Figure 8: Anomaly detection process	29
Figure 9: Sliding windows for anomaly detection on real-time data	30
Figure 10: Time series for <i>message_delay</i> measurements during the experiment	32
Figure 11: Evolution of the <i>message_delay</i> KPI over the iterations	33
Figure 12: Graphical representation of the test case statistics produced by the AE	34
Figure 13: Example of data with simulated anomalies	36
Figure 14: Performance of the VQ-VAE model for anomaly detection	37
Figure 15: Example of API invocation for the subscription of the analytics service	41

List of Tables

Table 1: Adherence to 5G-EPICENTRE’s GA Task Descriptions	11
Table 2: Analytics Service API endpoints exposed by the Analytics Engine	17
Table 3: Results of the evaluation of different anomaly detection approaches on our dataset	27
Table 4: JSON format of the subscription request to the Analytics Network Application API.....	31
Table 5: Message from the Analytics Aggregator to the Front-end with KPI on a single iteration.....	33
Table 6: Message including overall KPI of the experiment and related statistics.....	34
Table 7: JSON message containing the notification of a detected anomaly sent towards the Front-end.....	37
Table 8: Endpoint for subscribing to the analytics services for KPI calculation and statistical analysis	40
Table 9: Endpoint for deleting subscription to the analytics services for KPI calculation and statistical analysis	41
Table 10: Endpoint for returning the descriptor associated with “subscription_id”	42
Table 11: Analytics Driver – Publisher interface message schema.	43
Table 12: Example of message with measurements from the vertical application.	45
Table 13: Example of message with measurements from the network infrastructure.	46
Table 14: Analytics Aggregator – 5G-EPICENTRE Portal interface message schema.	48
Table 15: Details of the format of the messages from the Aggregator to the Portal	52
Table 16: Example of message with statistics on measurements from the infrastructure.	52

1 Introduction

This report presents the outcomes of Task 2.5, “ML-driven experimentation analytics,” under the 5G-EPICENTRE project. The task is focused on the Analytics Engine, a key component of the 5G-EPICENTRE platform, designed to monitor and analyse data from the infrastructure and the vertical applications. It uses a decentralized architecture to enhance scalability and reliability, and includes modules for data collection, Key Performance Indicator (KPI) monitoring, Quality of Service (QoS) / Quality of Experience (QoE) monitoring, and data aggregation. The Analytics Services Application Programming Interface (API) Server allows customization of the Analytics Engine’s functionalities. As a network application, it enables verticals to request analytics on their specific KPIs through a Representational State Transfer (REST) API, integrating the Analytics Engine’s capabilities into their operations.

The document is structured as follows:

- **Section 1** serves as the gateway to the Task, introducing the Task’s objectives and providing a comprehensive mapping of the outputs.
- **Section 2** provides a thorough understanding of the Analytics Engine’s decentralised architecture and its constituent data processing modules. It offers insights on the specific role and features of each module. Furthermore, this section explains the deployment and integration process of the Analytics Engine within the 5G-EPICENTRE platform, thereby illustrating the mechanisms through which the module collects and analyses data.
- **Section 3** focuses on the data analytics services provided by the Analytics Engine. It discusses the calculation of KPIs and statistical analysis, and provides a detailed description of the anomaly detection approach, using deep learning (DL) techniques.
- **Section 4** describes the testing and validation activities carried out for the services provided by the Analytics Engine. It presents the results of the KPI calculation and statistical analysis for measurements provided by a first-party experiments, and validates the approach for the anomaly detection on infrastructure measurements during traffic simulation experiments executed on a testbed.
- The report concludes with **Section 5**, summarising the key findings and contributions of the Task 2.5 to the 5G-EPICENTRE project.

1.1 Mapping of project’s outputs

The purpose of this section is to map 5G-EPICENTRE Grant Agreement (GA) commitments within the formal Task description, against the project’s respective outputs and work performed.

Table 1: Adherence to 5G-EPICENTRE’s GA Task Descriptions

5G-EPICENTRE Task	Respective Document Chapters	Justification
T2.5 ML-driven experimentation analytics <i>“[...] Activities in this Task will be related to the development of the Analytics Engine, which will monitor experimental application condition [...]”</i>	Section 2.1 – Architecture	This Section presents the architecture and the functionalities of its building blocks. The Section also explains how the Analytics Engine is able to monitor the experimental application conditions, collecting and analysing data coming from the infrastructure and the vertical applications.

<p>T2.5 ML-driven experimentation analytics</p> <p><i>“[...] identify any anomalies and patterns; and perform continuous log analysis in order to anticipate future events [...]”</i></p>	Section 3.2 – Anomaly detection	This Section provides details about the approach to anomaly detection on measurements from the infrastructure. More specifically, it summarises the state of the art of anomaly detection on time series, and the work done to find the most suitable approach in the project.
	Section 3.1.1 – Outlier removal	This Section provides information about the approach for removing outliers, caused by transient errors in the measurements provided by the vertical applications.
	Section 4.2 – Results for the Anomaly detection	This Section addresses the validation of the anomaly detection feature of the Analytics Engine conducted on infrastructure measurements of one of the testbeds in the 5G-EPICENTRE platform.
<p>T2.5 ML-driven experimentation analytics</p> <p><i>“[...] The Analytics Engine will be able to process huge amounts of data coming from the continuous analysis of the parameters of the experiments to determine whether the targets set by the experiments are met, in terms of KPIs and QoS/QoE [...]”</i></p>	Section 3.1 – KPI Calculation and statistical analysis	This Section explains how the Analytics Engine computes the KPIs of the experiments, based on the measurements collected during the execution of the experiments.
	Section 4.1 – Results for KPI calculation and statistical analysis	This Section reports on the details of a specific validation activity for KPI calculation and statistical analysis, performed by the Analytics Engine on measurements collected during the running of a first-party experiment on one of the testbeds of the 5G-EPICENTRE platform.
<p>T2.5 ML-driven experimentation analytics</p> <p><i>“[...] As a result, meaningful data will be used as input of the Insights Tool for visualization [...]”</i></p>	Section 2.1.5 – Analytics Aggregator	This Section describes the submodule in charge of providing the output of the Analytics Engine to the Insight Tool for visualisation.

2 The Analytics Engine

The 5G-EPICENTRE project aims to build an end-to-end 5G experimentation platform tailored to the needs of the Public Protection and Disaster Relief (PPDR) sector. The framework enables the analysis and visualisation of the experimental data collected by monitoring probes during the use of the facilities, to test and validate the vertical applications for the final users. The Analytics Engine is the software component that monitors the experimental features of the vertical applications and the infrastructure. It computes KPIs, to assess whether the targets set by the experiments are met, and analyses measurements from the infrastructure to detect potential anomalies in the network conditions.

2.1 Architecture

The Analytics Engine is composed of several internal modules, each with specific functionalities:

- **Analytics Driver:** acts as the initial point of contact for data entering the Analytics Engine. Its role encompasses the collection, filtering and initial pre-processing of data generated by experimental applications and infrastructure components;
- **KPI Monitor:** focuses on monitoring KPIs from the collected data;
- **QoS/QoE Monitor:** responsible for monitoring the QoS and QoE, based on the data analysis;
- **Analytics Aggregator:** consolidates the processed data from the KPI Monitor and QoS/QoE Monitor at each testbed, providing information to the Front-end components for data visualisation;
- **Analytics Service API Server:** implements a REST API, for customising the functionality of the Analytics Services Network Application.

The Analytics Engine adopts a decentralised architecture, implying that data-intensive operations occur on each testbed (Infrastructure Layer), rather than on a centralised server. This approach reduces network overhead and latency, while enhancing the scalability and reliability of the Analytics Engine. The data processing modules (*i.e.*, Analytics Driver, KPI Monitor, QoS/QoE Monitor) are then deployed at the testbed level. These modules collect and process the data generated by the experimental applications and infrastructure components using various techniques, such as machine learning (ML) and statistical analysis, to extract useful information and patterns from the data.

The data processing modules communicate with an Aggregator module in the Back-End Layer, consolidating and harmonising data from different testbeds. The Aggregator module connects with data visualisation modules in the 5G-EPICENTRE Portal, which presents the data in a user-friendly and interactive way. The data visualisation module uses various tools, such as dashboards, charts, graphs and maps to display the data and the results of the analysis (more details can be found in the deliverable D3.2 “5G-EPICENTRE Front-end components”).

Aligning with other European ICT-41 projects, 5G-EPICENTRE followed the Network Applications approach [1]. The Analytics Services Network Application is an example of a Network Application that the 5G-EPICENTRE project provides to the verticals. It includes some of the components of the Analytics Engine, and allows the verticals to request analytics on their specific KPIs of interest. The Vertical can interact with the Analytics Services Application through a REST API, by sending requests and receiving responses in a standardised format.

Figure 1 below, describes the internal architecture of the Analytics Engine, illustrating how the input data from the testbed is processed, and the results are forwarded to the 5G-EPICENTRE Portal for visualisation.

The internal modules of the Analytics Engine are containerized applications deployed in a Kubernetes¹ environ-

¹ <https://kubernetes.io/>

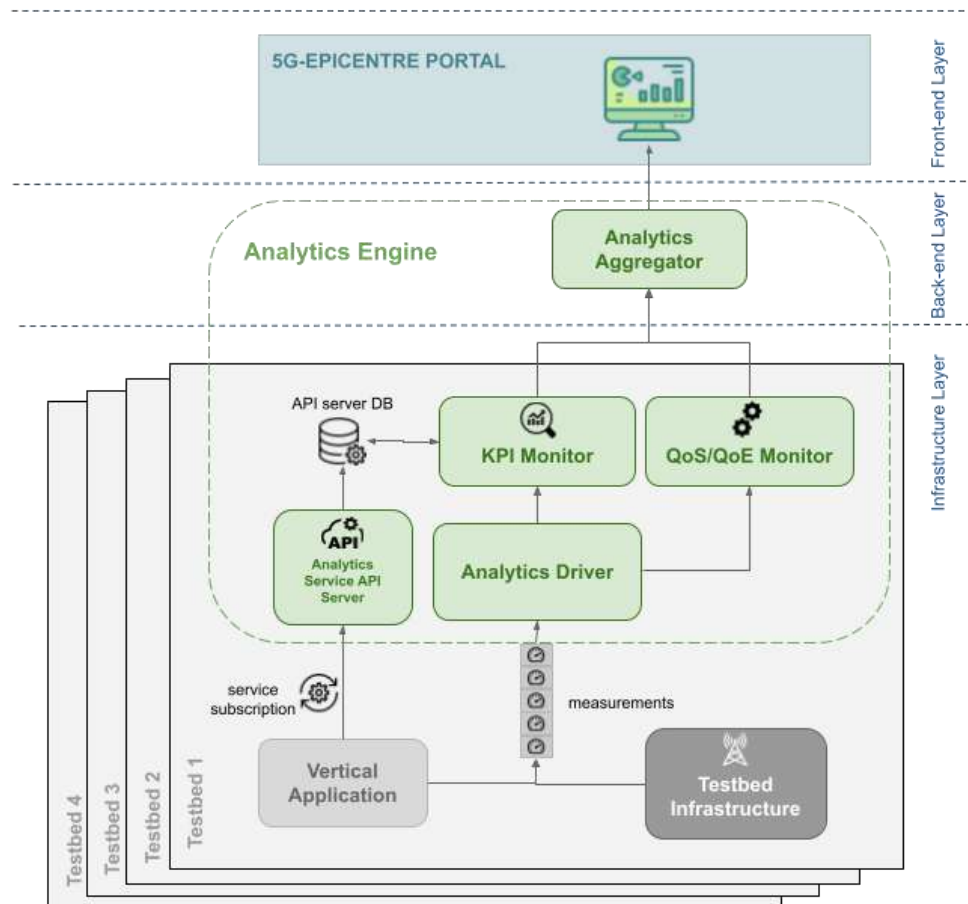


Figure 1: Architecture of the Analytics Engine and data flow

ment. These modules are implemented in Python, utilising specific libraries, such as PyTorch², NumPy³, Pandas⁴ and Scikit-learn⁵.

The Analytics Engine communicates with the rest of the platform through message queue systems, or REST APIs, exchanging payloads in a JavaScript Object Notation (JSON) format. Message queues are also employed for internal communication between the Analytics Engine's sub-components. More specifically:

- The Analytics Driver receives measurements via the message broker (*i.e.*, RabbitMQ⁶) exchange of the Publisher module (at each Testbed - more information about the Publisher module can be found in deliverable D2.5: “5G-EPICENTRE Experiment execution”), processes them for further analysis, and publishes the pre-processed measurements to the Analytics Engine's internal exchange.
- Both the KPI Monitor and the QoS/QoE Monitor receive data through the Analytics Engine's internal exchange, process them and publish their analytical results to the Aggregator exchange.
- The Analytics Aggregator processes data from each testbed's KPI Monitor and QoS/QoE Monitor modules on its “inbound” exchange, and provides its output through an “outbound” exchange. This output is subscribed to by the visualisation components in the Front-end Layer (see D3.2).

² <https://pytorch.org/>

³ <https://numpy.org/>

⁴ <https://pandas.pydata.org/>

⁵ <https://scikit-learn.org/stable/>

⁶ <https://www.rabbitmq.com/>

- Vertical applications can subscribe to the analytics services by communicating the required configuration for KPI calculation via a REST API provided by the Analytics Service API Server.

The modules at the Testbed Layer use an InfluxDB⁷ instance provided by the Testbed (outside the Kubernetes cluster), for storing data related to 5G networks and vertical applications.

The following Figure (Figure 2) depicts the integration of the Analytics Engine internal components with other components of the 5G-EPICENTRE architectural stack.

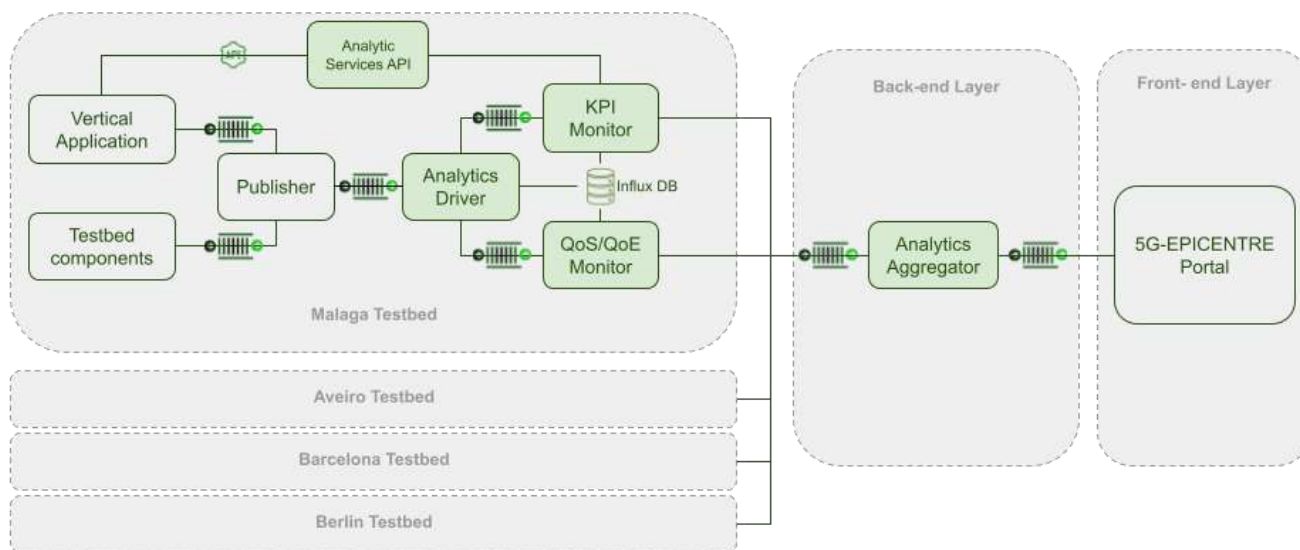


Figure 2: Integration among internal and external components

The following Sections provide a more detailed description of the Analytics Engine modules.

2.1.1 Analytics Driver

The Analytics Driver is the module responsible for collecting metrics generated at the testbed level by the infrastructure and the vertical applications. Its purpose is to pre-process the data, in order to prepare them for the other Analytics Engine tasks: KPI calculation and ML-based analysis. The Analytics Driver performs data validation and verification to ensure the quality and integrity of the data. It also records the received measurements into an InfluxDB, which is an open-source time series database (TSDB), suitable for recording metrics, events and analytics.

The metrics collected by the Analytics Driver are related to:

- **Network performance:** this includes metrics, such as *latency*, *bandwidth*, and *resource consumption*, reflecting the quality and efficiency of the network infrastructure and services used by the experiments.
- **Traffic conditions:** metrics such as *throughput*, *jitter* and *packet loss*, provide insights into the amount and quality of the data transmitted and received during the execution of the experiment.
- **Specific metrics of interest to the experimenters:** these may include *response time*, *accuracy* or *availability*. These metrics reflect the objectives and outcomes of the experiments, tailored to the specific needs and requirements of each vertical application. The vertical application components can request Analytics Engine services for KPI calculation and statistical analysis based on these metrics, following the Network Application model.

⁷ <https://www.influxdata.com/products/influxdb-overview/>

- **Security report:** this comprises a comprehensive report generated by the Holistic Security and Privacy Framework (HSPF). The HSPF is designed to provide protection for any Network Application deployed in a Kubernetes environment. More information about the HSPF can be found in deliverable D2.8: “*Cloud-native security final version*”.

The Analytics Driver receives metrics via asynchronous communication established with the Publisher, a component of each Testbed responsible for associating different kinds of metrics with the proper experiment metadata for identification purposes, and for notifying the Analytics Engine when an experiment starts and stops. The Publisher utilises a message broker (*e.g.*, RabbitMQ), to publish metrics and metadata to a common topic exchange.

The Analytics Driver subscribes to the Publisher's message queue, and receives both metrics and metadata. After data validation, the Analytics Driver further publishes metrics to both the KPI Monitor and the QoS/QoE Monitor, which are other internal modules of the Analytics Engine, responsible for calculating and evaluating KPIs and QoS/QoE indicators based on the measurements.

2.1.2 KPI Monitor

The KPI Monitor is an internal module of the Analytic Engine deployed on each Testbed. Its primary responsibility is to analyse metrics exposed by the Analytic Driver, overseeing the experiment by calculating KPIs. This calculation may occur based on a predefined number of iterations or time intervals. Additionally, the module computes statistics such as mean, median, standard deviation and percentiles, to furnish a comprehensive summary and overview of the metrics.

The configuration of KPI calculation is flexible, and aligns with the specific needs and requirements of each experiment through a designated API. Once computed, both KPIs and statistics are encoded in a standardised JSON format, and subsequently published to the Aggregator “inbound” RabbitMQ topic exchange.

2.1.3 QoS/QoE Monitor

The QoS/QoE module of the Analytics Engine is deployed at each Testbed, and is responsible for continuously analysing the pre-processed data received from the Analytics Driver. It provides a reliable and intelligent service for performing anomaly detection on infrastructure data. An anomaly is defined as a deviation from the normal, or expected behaviour, or performance of the system. Anomaly detection is important for identifying and diagnosing potential problems, faults, or threats that may affect the QoS or QoE of the experiments.

The QoS/QoE Monitor uses DL models to learn what constitutes ‘normal’ operation of the infrastructure of the 5G-EPICENTRE platform, and to detect and quantify anomalies in the data.

The module exposes the results corresponding to anomalous network conditions or trends to the Analytics Aggregator, using a RabbitMQ message queue system to publish its output data in a standardised JSON format. This mechanism enables the QoS/QoE Monitor to notify any detected anomalies to the Front-end components.

2.1.4 Analytics Service API Server

The Analytics Service API Server is an internal module of the Analytics Engine, deployed at the Infrastructure Layer, implementing a REST API used by the verticals to configure the functionality of the Analytics Services Network Application to align with their specific needs.

The **Analytics Services Network Application** includes the Analytics Driver, the KPI Monitor and the Analytics Aggregator: it provides KPI calculation and statistical analysis over the measurements sent by the vertical application during the experiment, allowing a certain degree of customization of the service.

Using the API, the vertical can subscribe to the analytics service and request analytics on specific KPIs of interest, providing details about the experimental data and the methodology to be applied for the KPI calculation, so to configure the KPI Monitor. In more detail, the API allows the vertical to specify the following information:

- The set of measurements provided by the vertical application;
- The number of iterations and the number of values per iteration for the KPI calculation;
- (Optionally) The thresholds for "upgradable" and "optimal" levels of the KPIs (see D1.6 *"Experiment evaluation strategy and experimentation plan"*);
- The option to turn on the automatic outlier removal feature, which removes possible measurement errors from the KPI calculation.

The API exposed by the Analytics Engine supports the capabilities/operations described in Table 2.

Table 2: Analytics Service API endpoints exposed by the Analytics Engine

Method	Endpoint	Description
POST	/kpi_service_subscribe	Endpoint for subscribing the KPI calculation services for a Vertical network application and a set of KPIs specified in the JSON payload.
GET	/kpi_service_delete/<id>	Endpoint for deleting the subscribed KPI calculation service.
GET	/kpi_service_get/<id>	Endpoint for listing the setting of the subscribed KPI calculation service

The detailed documentation of the API exposed by the Analytics Engine is reported in Annex I.

2.1.5 Analytics Aggregator

The Analytics Aggregator (or simply, "Aggregator") module is a centralised component of the 5G-EPICENTRE Analytics Engine, deployed at the Back-end Layer. Its role is to collect the data generated by the Analytics Engine's components deployed at each Testbed (Infrastructure Layer), and make them available in a unified output format to the 5G-EPICENTRE Portal for visualisation.

The Aggregator uses a RabbitMQ broker through which messages in JSON format are exchanged. The module provides an "inbound" queue, through which data is received from the testbeds, and an "outbound" queue, where the output data for the visualisation component in the Front-end is published.

The output data produced by the Aggregator is organised into three distinct types of data flows, each serving a unique purpose, and providing valuable insights into the system's operation and performance to the Front-end components. The data flows are detailed as follows:

- **KPIs:** This data flow contains KPIs based on measurements received from the first-, or third-party vertical applications.
- **Params:** This data flow provides statistics based on predefined time windows on measurements coming from the testbed infrastructure. These statistics offer a detailed view of the system's operation over time, allowing for trend analysis and performance evaluation.
- **Anomalies:** This data flow reports anomalies detected in measurements from the infrastructure. It identifies unexpected, or unusual patterns in the data, potentially indicating issues or areas for improvement.

2.2 Deployment of the Analytics Engine

Following the decentralised architecture specified in the Section above, the Analytics Engine components that perform data-intensive operations are deployed at each of the four testbeds of the 5G-EPICENTRE platform (the Infrastructure Layer), while one of its sub-components, the Aggregator, is centralised, and deployed at the Back-end Layer deployment testbed (UMA platform).

5G-EPICENTRE is designed as a federation of four separate testbeds/facilities, each characterised by its own 5G standalone implementation and technologies. However, they all share the capability to support the deployment of network functions and applications in a containerized environment. These deployments are managed using a Kubernetes⁸ cluster architecture, ensuring efficient operation, high availability and scalability. Therefore, the deployment of the Analytics Engine across different environments, such as different testbed infrastructures, utilises containerized modules, orchestrated via Kubernetes.

Each component of the Analytics Engine is containerized using Docker⁹, a popular containerization technology. These Docker images are stored in the project's Container Registry, from which they can be pulled for deployment. The following Figure (Figure 3) illustrates the deployment of the Analytics Engine components at each testbed.

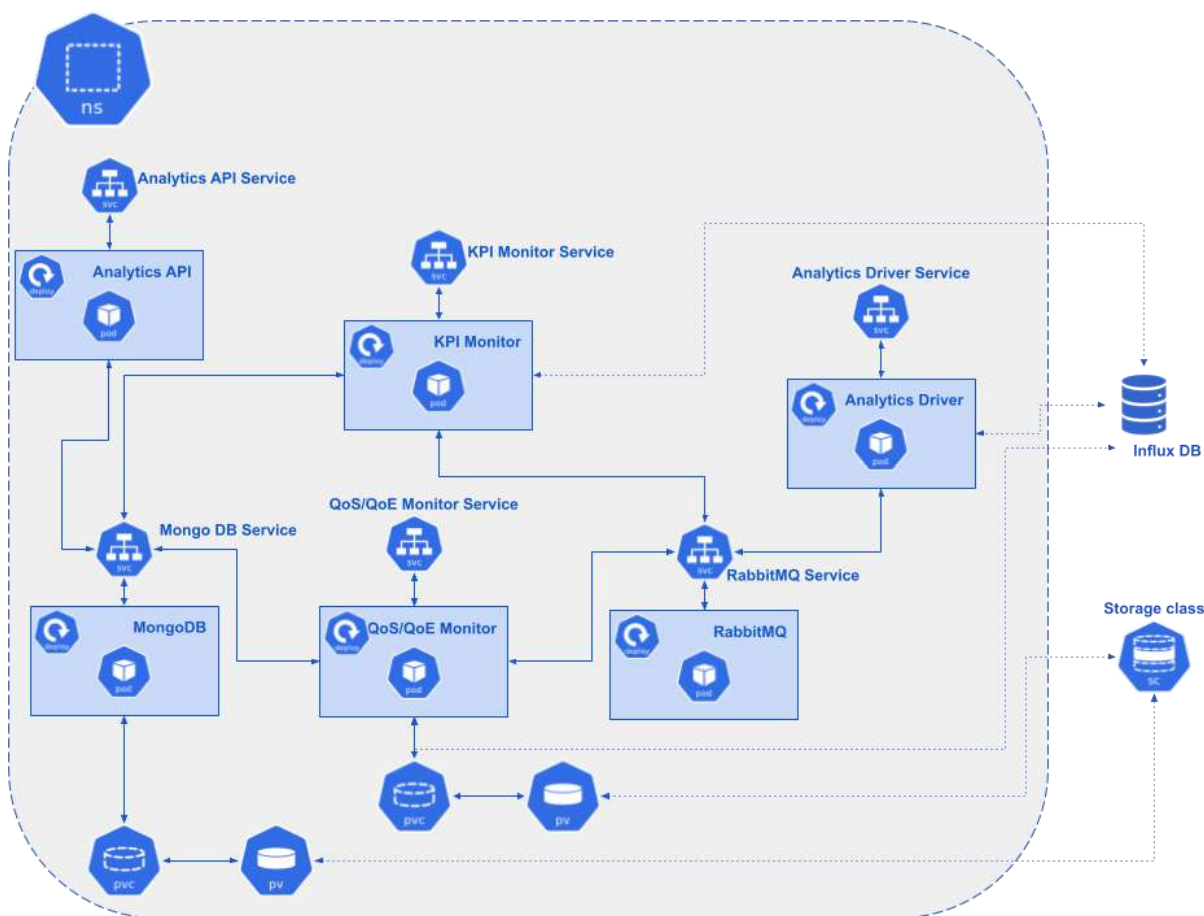


Figure 3: Schema of the Kubernetes deployment of the AE components at each testbed

⁸ <https://kubernetes.io/>

⁹ <https://www.docker.com/>

Each sub-component of the Analytics Engine is deployed as a pod, which is a group of one or more containers sharing resources and network. All these components are deployed within a specific namespace (NS) in the Kubernetes environment, ensuring isolation, high availability and ease of management. Kubernetes has also been used to define Services (SVC), which are abstractions that expose a set of pods to other pods, or external clients. Each service has a stable Internet Protocol (IP) address and port number, that can be used to access the sub-component of the Analytics Engine. Moreover, Kubernetes allows the definition of Deployments, which are controllers that manage the desired state and configuration of the pods and services. Each deployment specifies the number of replicas, the image source, environment variables and resource limits for the pods and services.

Some of the components of the Analytics Engine, namely the Analytics API server, the KPI Monitor, and the QoS/QoE Monitor, require a MongoDB¹⁰ instance, to store configuration settings for the analytics tasks. The MongoDB instance is deployed using a Persistent Volume Claim (PVC) provided by a storage class (SC), ensuring data persistence and durability.

The QoS/QoE Monitor, performing DL-based anomaly detection, also requires persistent storage for storing the DL models. This is achieved using a similar PVC, provided by a storage class.

In addition to MongoDB, the Analytics Driver, KPI Monitor and QoS/QoE Monitor also utilise an InfluxDB instance for time series data management. This InfluxDB instance is provided by the testbed owner and is located outside the specific namespace of the Analytics Engine.

¹⁰ <https://www.mongodb.com/>

3 Data Analytics

The Analytics Engine offers data analytics services designed to address the analysis of the health and performance of the 5G-EPICENTRE platform during experiment executions. Additionally, it aims to support the evaluation and validation of the experiments.

The following Sections will provide a detailed description of the data analytics functionalities.

3.1 KPI calculation and statistical analysis

The Analytics Engine computes the KPIs of the experiments based on the metrics collected from the testbeds during execution. The KPIs reflect the objectives and outcomes of the experiments according to the specific needs and requirements of each vertical application. They also enable the evaluation and validation of the experiments according to the 5G-EPICENTRE guidelines for KPIs validation (as defined in D1.6). In this case, the experimentation is performed by collecting multiple measurements following an iterative process. Each experiment defines the set of metrics, the number of iterations per metric, and the number of measurements per single iteration. This information, needed to configure the KPI calculation, can be sent to the Analytics Engine via the analytic services network application API.

The value of the KPI for the i -th iteration (mv) is defined as the average value of all the n measurements collected within the iteration, according to the following formula:

$$mv_i = \frac{1}{n} \sum_n v_{i,n} \quad (1)$$

Then, the overall KPI for the entire experiment (MV) is defined as the average of all iteration mv_i :

$$MV = \frac{1}{l} \sum_i mv_i \quad (2)$$

For each KPI, the Analytics Engine computes the average value at the end of each iteration, and sends it to the 5G-EPICENTRE Portal for data visualisation.

At the end of the experiment, when all the planned iterations have concluded, these indicators are averaged over iterations, to produce an overall test case KPI, accompanied by a confidence interval. The statistical analysis service also provides statistics about the entire experiment run: *max*, *min*, *median*, *standard deviation* and *percentiles*.

To provide more detailed KPI calculation, the KPI Monitor can aggregate measurements and provide specific KPIs for each individual device that is reporting measurements during the experiment. This feature enables a finer-grained, and more comprehensive view of the vertical application performance and behaviour at the device level, allowing for the comparison of KPIs across different devices, and the identification and diagnosis of potential issues that may affect specific devices or groups of devices.

Furthermore, the KPI Monitor can be configured to provide KPI calculation and statistical analysis on metrics from the infrastructure, based on predefined time intervals. This allows for more accurate information about the infrastructure environment during experiments.

3.1.1 Outlier removal

The Analytics Engine offers an optional outlier removal feature, that can be enabled to enhance the quality and precision of KPI calculations on experimental data and analytical outcomes. An outlier is a “point anomaly”, *i.e.*, a value that falls outside the normal range of a variable and may distort statistical analysis and KPI computation.

An anomalous value can be caused by an error in the measurement, which can be transient (in cases where there are no specific malfunctions of the measurement probes).

The Analytics Engine's outlier removal is based on the Z-score method, which exploits the standard deviation of the given distribution of recorded data points (per measurement), to determine whether a data point belongs to the expected value range. The Z-score is a measure of how many standard deviations a data point is away from the mean. Any value that is at least three standard deviations away from the mean is considered an outlier, and will be removed before applying analytical methods. This process is executed in real time, as soon as measurements are collected, and there is a sufficient amount of available data.

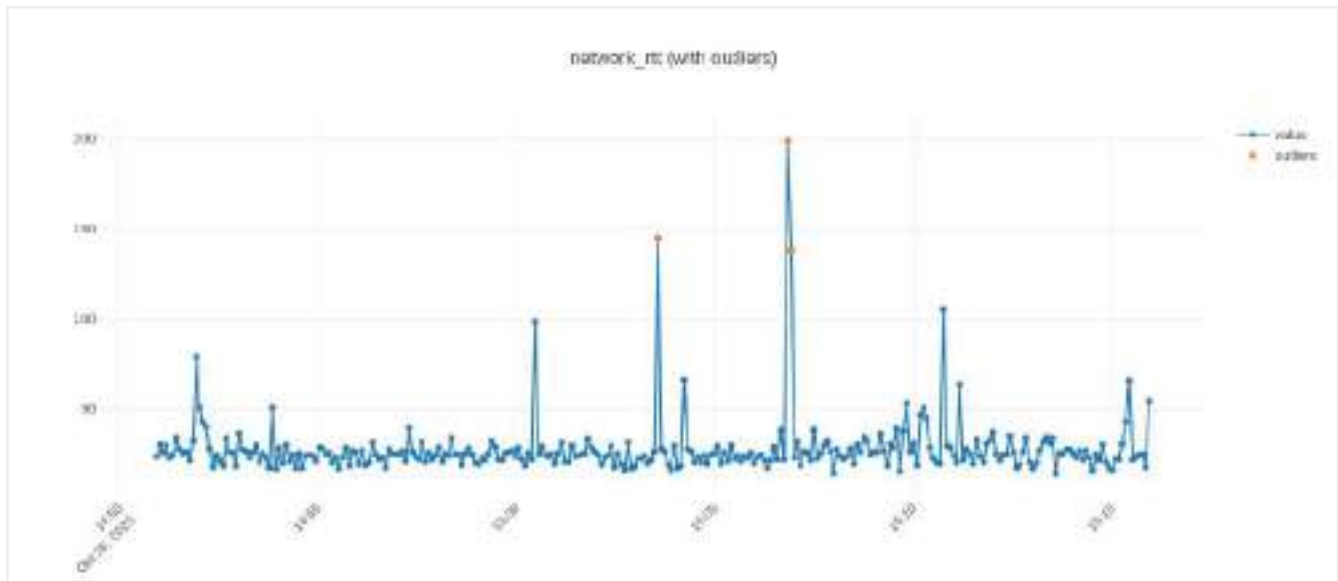


Figure 4: Example of outliers to be removed from the timeseries

The Z-score method is known for its robustness and consistency across different variables and distributions. However, it assumes that the data points follow a normal distribution, which may not always be true for some variables, or experiments. Additionally, it may remove some valid data points that are extreme, but not erroneous, leading to a potential reduction in the variability and richness of the data. The method may also fail to detect outliers that are close to the mean, but far from the majority of the data points, impacting the shape and skewness of the distribution. For this reason, the functionality of outlier removal has been made optional, allowing the experimenter to decide whether to activate it or not.

The functionality can be enabled via the Analytics Service API, by setting a specific parameter in the request for the analytics services. If the functionality is enabled, KPI calculations will be performed on measurements obtained after the outlier removal.

3.2 Anomaly detection

Detecting anomalies in data collected from network probes is a complex and critical task in performance monitoring. Anomalies, in this context, refer to patterns in the data, that deviate from what is expected or normal. These anomalies may result from various factors, such as network faults, or unusual user behaviour. Detecting these anomalies helps network administrators identify potential issues and respond to them promptly.

Detecting anomalies in data gathered from network probes poses multiple challenges:

- **Data Complexity:** network probe data is often high-dimensional, with numerous features and metrics being tracked. This complexity can make it challenging to identify which features are relevant for anomaly detection.
- **Dynamic behaviour:** network traffic patterns can change over time, and what is considered ‘normal’ can vary depending on the time of day, day of the week, or other factors. This variability complicates the establishment of a consistent baseline for anomaly detection.
- **Noise and variability:** network probe data can be noisy and highly variable. Minor fluctuations in network traffic are common, making it difficult to distinguish these from genuine anomalies.

In the 5G-EPICENTRE project, the data that are collected from the testbeds are time series, that are sequences of data points consisting of successive measurements made over a time interval. Anomaly detection in time series data poses other specific challenges, because it requires simultaneous consideration of temporal dependencies and relationships between variables. Temporal dependencies mean that the value at a given time point is dependent on previous time points, and this can make it difficult to identify anomalies that are based on these dependencies.

The approach in this task can be summarised in the following steps:

- **Choose an appropriate algorithm.** There are several algorithms available for anomaly detection in time series data, including autoencoders, Vector Quantized Variational Autoencoders (VQ-VAE), and DL-based approaches. The choice of algorithm depends on the specific dataset, and the complexity of the anomaly detection task.
- **Pre-process the data.** Pre-processing the data can help improve the accuracy of anomaly detection algorithms. This may include removing noise from the data, normalising the data, and handling missing data.
- **Use a sliding window approach.** A sliding window approach can be used to process the data in real time. This involves processing a fixed-size window of data at a time, and updating the anomaly detection algorithm with each new window of data.
- **Incorporate domain knowledge.** Domain knowledge can be useful in identifying anomalies in the data. Anomaly detection algorithms that incorporate domain knowledge may be more effective than those that do not.
- **Monitor the performance of the algorithm.** It is important to monitor the performance of the anomaly detection algorithm, to ensure that it is accurately identifying anomalies.

The following Sections describe in detail the solution developed for the QoS/QoE Monitor module of the Analytics Engine that uses DL techniques to perform anomaly detection on the metrics collected from the testbed infrastructure. The analysis is performed in near-real time, and the detected anomalies are notified to the Front-end components for data visualisation. This allows the final users to have more detailed information about the network performance during the experiment, so to be able to evaluate the KPIs of the experiment in the right perspective.

3.2.1 Anomaly detection in time-series: state-of-the-art

Anomaly detection is a complex field that has gained significant attention in recent years in many domains, because anomalies identification can often provide critical and actionable insights.

There are several types of anomalies that can occur in time series data, that can have significant implications for monitoring and observability mechanisms [2]. In particular, anomalies can have different types:

- **Point anomaly:** a data point that is anomalous compared to the rest of the data, as it deviates significantly from the expected pattern.
- **Contextual anomaly:** an anomaly that appears to be anomalous only in a specific context.

- **Collective anomaly:** a collection of data instances that appear to be anomalous with respect to the entire dataset. The individual data instances may not be anomalies themselves, but their occurrence as a collection is considered anomalous.

The approach for anomaly detection can vary greatly depending on the type of anomaly, the nature of the data (temporal or not temporal), and the number of variables involved (univariate or multivariate data, depending on the focus on a single variable, or multiple variables).

Mathematical and statistical methods have been the main focus of anomaly detection research for a long time. These methods, that use statistical tests to check if a data point is an anomaly based on the data distribution, can prove effective across various scenarios, often boasting simplicity and computational efficiency, particularly when dealing with univariate time series. However, they necessitate manual feature extraction and face challenges when confronted with high-dimensional data.

The landscape of anomaly detection in time-series data has undergone a significant transformation in recent years, thanks to the advent of ML algorithms. These algorithms operate by training a model to identify ‘normal’ patterns within the data, and subsequently assessing whether new data aligns with these established patterns. The effectiveness of these anomaly detection techniques is heavily influenced by the availability of labelled data. In many practical scenarios, such data is scarce. However, even a minimal amount of labelled data can be pivotal in achieving accurate anomaly detection. This is typically accomplished through *supervised learning*, which uses known (labelled) data to categorise information into ‘normal’ and ‘abnormal’ classes. Conversely, when labelled data is not readily available, *unsupervised learning* proves to be an invaluable tool: this approach works with unlabelled data, by identifying patterns and similarities without the need for predefined ‘normal’ and ‘abnormal’ classes [3] [4].

However, these traditional approaches are not without their limitations. They often grapple with the scarcity of labelled data, a common challenge in most industrial scenarios, where anomalies are rare and data annotation is both challenging and costly. Moreover, as industrial applications become increasingly automated and complex, the necessity to monitor multiple variables concurrently becomes paramount. Traditional methods often experience a significant decline in performance when dealing with high-dimensional data, a phenomenon known as the ‘curse of dimensionality’. Furthermore, some anomalies can only be detected by examining the relationships between different variables, a task that univariate time-series analysis is incapable of performing [5].

To address these limitations, researchers have turned to (deep) neural networks, to enhance classical techniques. DL-based models excel at learning representations of large-scaled sequences in an unsupervised manner, and identifying anomalies in the data. These methods can automatically extract features and handle high-dimensional data, making them particularly useful for anomaly detection tasks, even though they often require more data, and more computational resources.

In the context of the project, the anomaly detection task of the Analytics Engine is focused on real time (or near-real time) anomaly detection in time series data coming from monitoring probes on the network infrastructure. Anomaly detection in time series data adds complexity to the task, due to the need for simultaneous consideration of temporal dependencies and relationships between variables. Therefore, the focus is on detecting collective anomalies in multivariate data. Furthermore, the lack of precisely labelled data regarding the presence of anomalies, necessitates an unsupervised approach. For these reasons, DL models appear to be a suitable approach.

A review of the scientific literature on anomaly detection in an unsupervised context [6] [7] [8], pointed out that a possible solution to the problem is to develop a model capable of reconstructing the correct operating states of the system. In other words, a *regression model* should be created, that can reconstruct data observations in a state that does not deviate from the normal. The model’s reconstruction error can then be used as an indicator of anomalies during inference, as well as a cost function to minimise during training.

Specifically, the scientific literature suggests the use of a DL model known as an *Autoencoder*. This type of neural network is employed for unsupervised learning tasks, such as anomaly detection in time series data. The basic idea behind autoencoders is to learn a compressed representation of the input data, which can subsequently be used to reconstruct the original data. Anomalies in the data can be detected, by comparing the reconstruction error of the input data with the reconstruction error of the compressed representation.

The performance of autoencoders depends on several factors, including the quality of the data, the size of the dataset, and the complexity of the anomaly detection task.

Another type of autoencoder, the VQ-VAE, can also be used for unsupervised anomaly detection in time series data [10] [11]. The main difference between autoencoders and VQ-VAE lies in the way they learn the compressed representation of the input data. Autoencoders use a continuous latent space to learn the compressed representation, while VQ-VAE uses a discrete latent space. This means that VQ-VAE can learn a more structured and interpretable compressed representation of the input data than autoencoders. Recent research has shown that VQ-VAE can outperform autoencoders on several benchmark datasets [9].

In summary, both autoencoders and VQ-VAE can be used for unsupervised anomaly detection in time series data, but the performance of both methods depends on the specific dataset, and the complexity of the anomaly detection task.

3.2.2 Identification of a suitable classification approach

The approach to anomaly detection for the Analytics Engine began with the study of real data from a the 5G-EPICENTRE testbed in Malaga, leveraging the domain knowledge of the engineers working at the testbed to better understand the expected traffic patterns for a particular network, and the significant types of anomalies that are likely to occur.

The initial datasets gathered from the testbed were devoid of any substantial anomalies. They only contained isolated anomalies, which were not of significant concern to the testbed operators, as they were the result of temporary alterations in data configuration, or minor, inconsequential issues. To emulate substantial anomalies, the engineers employed a specialised tool¹¹, capable of introducing anomalies into the system. This allowed for the generation of a dataset, encompassing both normal and abnormal data, offering a realistic depiction of the system's operation.

The decision to use Autoencoders or VQ-VAEs, depends on the specific attributes of the collected data, and the type of anomalies that need to be identified. A thorough examination of the anomalies revealed that they could be classified as change point situations. Change points are a type of anomaly that represent a significant shift in the underlying properties of a time series. These shifts could be due to a variety of reasons, such as a sudden change in trend, level, or variance of the time series data. When it comes to choosing an approach for detecting anomalies that are change points, both autoencoders and VQ-VAEs can be used, but they have different strengths. In the context of change point anomalies, VQ-VAEs might be more suitable, as they can capture more complex patterns in the data.

We started our work by studying various methods proposed in the state-of-the-art, focusing on unsupervised approaches for anomaly detection on time series data based on autoencoders.

Audibert et al. [12] propose a three-stage anomaly detection method, USAD, in their research. The first stage involves data pre-processing, which is common to both training and detection. Here, the data is standardised and segmented into temporal windows of length K . The second stage is dedicated to training the model. The final stage involves anomaly detection, which is performed online using the model trained in the second stage. As a new time frame becomes available, the model calculates an anomaly score. If this score exceeds a set anomaly threshold, the new time frame is classified as *abnormal*. The model's design incorporates an autoencoder

¹¹ <https://jagt.github.io/clumsy/>

that enables unsupervised learning. The application of adversarial training [13], allows for rapid learning and effective anomaly isolation.

The Deep Autoencoding Gaussian Mixture Model (DAGMM) [14] employs a deep autoencoder for each input data point to generate a low-dimensional representation and reconstruction error. These are subsequently inputted into a Gaussian Mixture Model (GMM). The DAGMM optimises the parameters of both the deep autoencoder and the mixture model concurrently, in an end-to-end fashion. It utilises a distinct estimation network to aid in the learning of the mixture model's parameters. This approach replaces the two-stage training and the traditional Expectation-Maximization (EM) algorithm. Joint optimization allows the autoencoder to avoid less optimal local solutions, and further reduce reconstruction errors. This effectively strikes a balance between autoencoding reconstruction, density estimation of the latent representation and regularisation.

Munir et al. [15] propose a two-stage architecture for their DeepAnT approach. The initial stage is characterised by a time-series predictor, that employs a CNN to project the subsequent time stamp over a defined horizon, by taking a time series window and striving to predict the next time stamp. The subsequent stage is made up of an anomaly detection module, that uses the output from the first stage as input, and labels the corresponding timestamp as either *normal* or *abnormal*.

In their work, Lin et al. [16] propose a VAE-LSTM hybrid model as an unsupervised approach for anomaly detection in time series. Utilising the VAE model, they condense the local data of a brief window into a low-dimensional embedding. On the other hand, the LSTM model operates on these low-dimensional embeddings generated by the VAE model, to handle long-term sequential patterns. This proposed hierarchical structure enables them to detect anomalies occurring over both short and long durations.

We performed tests on these state-of-the-art methods, in order to estimate their performance. Then, building upon similar methodologies, we conducted experiments with a novel ML technique, that employs the VQ-VAE, specifically tailored for anomaly detection in 5G network data.

The following Figure (Figure 5) describes in detail the approach for anomaly detection using VQ-VAE.

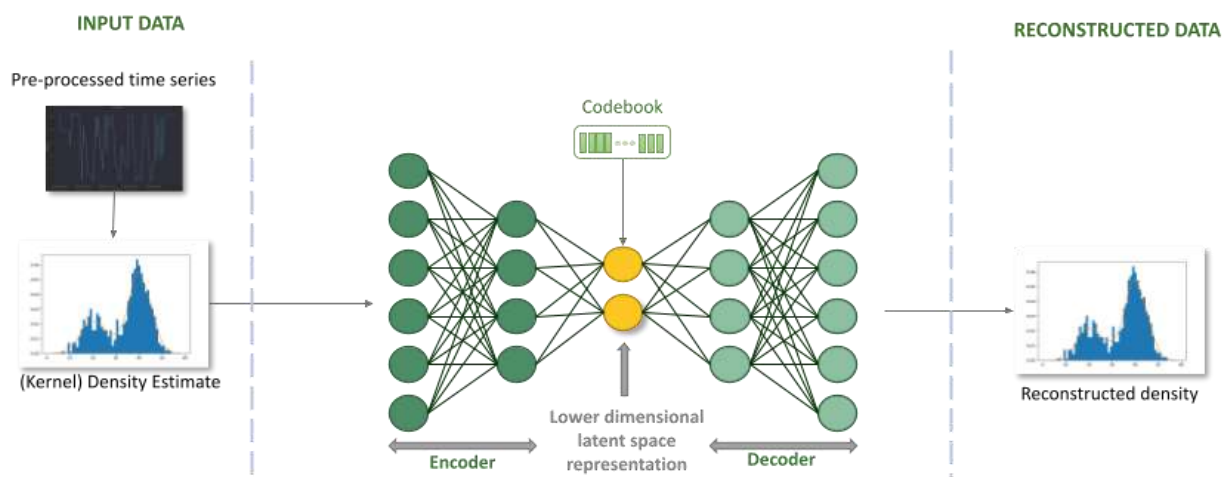


Figure 5: Schematic view of the VQ-VAE model

The VQ-VAE learns to encode multidimensional data into a lower-dimensional latent space. This is done by training an encoder network to map the input data to a discrete latent space, and a decoder network to reconstruct the original data from the latent representation.

The latent space is represented as a *codebook* of vectors, where each vector is an “embedding”. The encoder network outputs a continuous vector for each piece of input data, and this vector is then matched to the nearest embedding in the codebook through a process known as *vector quantization*.

The VQ-VAE can then be used to identify data that is significantly different from the data it was trained on, by looking at the distribution of codes in the latent space. More specifically, the Kernel Density Estimation (KDE) emerges as a non-parametric technique for approximating the probability density function of a random variable. This method proves particularly pertinent in the context of the VQ-VAE, where it is employed, to gauge the distribution of latent codes within the VQ-VAE's codebook [10]. The VQ-VAE, designed for encoding multivariate time series data, transforms this information into a lower-dimensional latent space, where each point corresponds to a unique code in the codebook. The application of KDE to this latent space facilitates the estimation of the probability density function associated with these codes.

Central to KDE is the smoothing parameter, also known as the *bandwidth*, a critical parameter dictating the trade-off between bias and variance in the density estimate. The bandwidth serves as a crucial factor in the precision of the estimation, influencing the smoothness of the resulting density function. In essence, it governs the sensitivity of the estimate to local fluctuations in the data.

The anomaly detection process within this framework hinges on a comparison between the density of a new data point (or code), and the estimated density function. Should the density at the new data point deviate significantly lower than the average density, it signals the identification of an anomaly. Essentially, this anomaly is situated in a low-density region of the latent space, indicating a substantial dissimilarity from the majority of the data upon which the VQ-VAE was trained. In this manner, the KDE methodology, with its bandwidth parameter and anomaly detection mechanism, proves instrumental in discerning outliers, or anomalous patterns within the latent space of the VQ-VAE.

In order to demonstrate the potential of our approach, we compared our method with the four unsupervised time series anomaly detection techniques described above: USAD [12], DeepAnT [15], LSTM-VAE [16] and DAGMM [14].

Our dataset was divided into three parts: training, validation and test. The training and validation sets were formed using data measurements with no detected anomalies for unsupervised training. The test set is composed of two subsets: a smaller one with no identified anomalies, and a larger one with detected anomalies. We applied our method to both subsets of the test set, but performance metrics were calculated solely on the annotated subset.

We used four evaluation metrics that have been widely used in previous works:

- *Precision*, which is the ratio of accurately identified anomalies to the total instances flagged as anomalies;
- *Recall*, the proportion of correctly identified anomalies to all actual anomalies;
- *F1 Score*, the harmonic mean of precision and recall;
- *Area Under the Receiver Operating Characteristic Curve (ROC-AUC)*, which shows the model's performance at all classification thresholds.

We evaluated the proposed methods by experimenting with various hyperparameter values, and employed different strategies for their selection. For the prediction window size, we tested multiple values, and ultimately chose the one that yielded the highest F1 score, which was 60. We opted for a stride of 15 between each window. The threshold value was determined by examining the performance on the validation set, with the threshold that achieved the best ROC-AUC score being chosen. The other parameters were selected based on the results from the validation set. We set the batch size to 32 and the learning rate to 10^{-4} .

We examined potential anomaly thresholds for each model, and reported the findings associated with the greatest F1 score, because not all anomaly detection techniques utilised for comparison offered a mechanism to choose anomaly thresholds.

The performance outcomes for each method on our dataset can be found in Table 3. These results indicate that our strategy employing VQ-VAE facilitates a 19.8% enhancement in terms of ROC-AUC when compared to the baseline.

Table 3: Results of the evaluation of different anomaly detection approaches on our dataset

Model	Precision	Recall	F1	ROC-AUC
DeepAnT	0.870	0.855	0.823	0.580
USAD	0.451	0.802	0.677	0.750
DAGMM	0.900	0.750	0.821	0.700
LSTM-VAE	0.880	0.804	0.843	0.760
VQ-VAE	0.930	0.923	0.926	0.862

The following Figure (Figure 6) shows the ROC curves for our model and the selected competitors, showing that the VQ-VAE outperforms other approaches in the state-of-the-art.

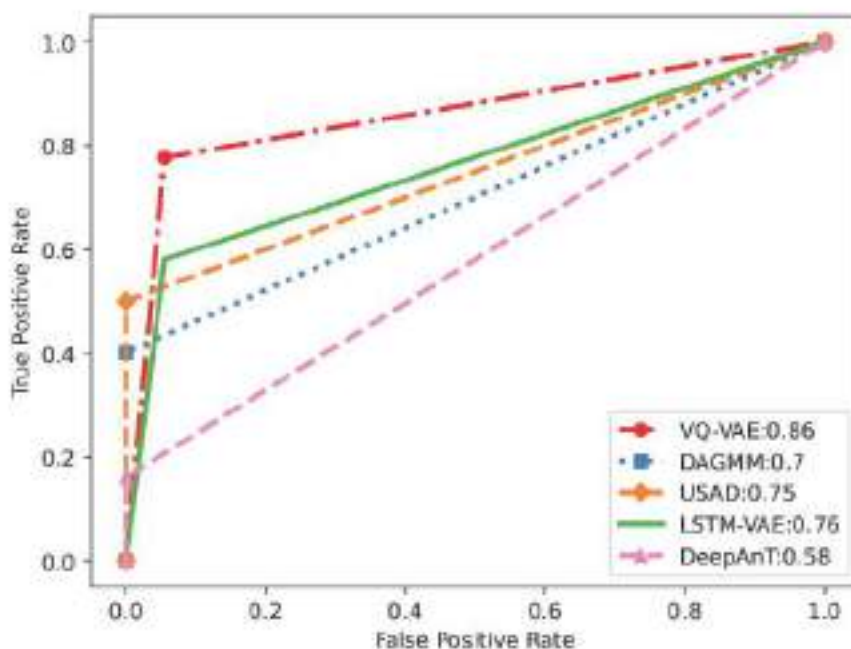


Figure 6: ROC curves for our model and the selected competitors

To generate the test dataset, we chose to use a set of measurements containing known anomalies. The selected window had a temporal size of approximately 7800 instants. As our training method involved a window size of 60, we needed to divide this sample into roughly 130 windows. To visually demonstrate the performance of our

method on the test set, we chose a sub-window of this set, and observed the disparities between our model's predictions and the actual labels. As depicted in Figure 7, our method accurately detects anomalies:

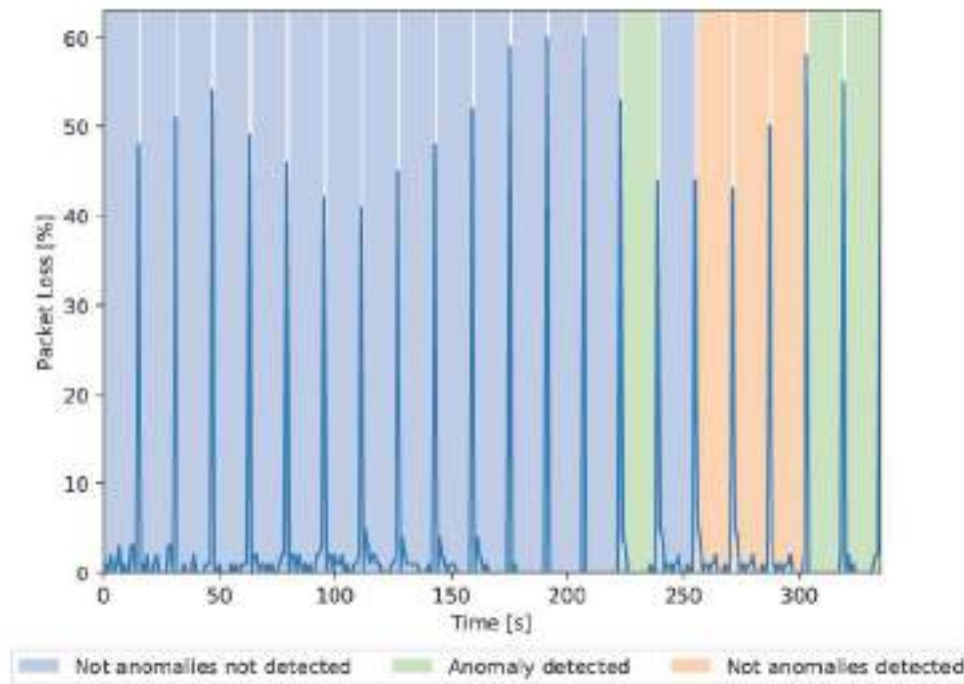


Figure 7: Performance of our model in a selected time window

More information about the validation of the selected approach and the achieved results can be found in Section 4.

3.2.3 Data collection and data handling

The input data for anomaly detection are multivariate time series, that are sequences of data that consist of multiple variables, or features that are measured over time.

The metrics are collected from various monitoring probes, and may exhibit differences of a few milliseconds, or have varying granularity. Therefore, prior to applying ML models to the data, the QoS/QoE monitor module must engage in pre-processing activities, to merge data from different monitoring probes, and synchronise them in time.

In the current configuration, the QoS/QoE Monitor module operates with a synchronisation granularity of one second. This implies rounding off each data point's timestamp to the nearest second. If a measurement contains multiple data points within the same second, the pre-processing module calculates their average value and assigns it to that specific second. As a result, each measurement will contain, at most, one data point per second, enabling straightforward synchronisation among different measurements.

The synchronisation makes sure that the data from different monitoring probes are consistent and comparable in time, which is essential for the next steps.

3.2.4 Classification process

The anomaly detection process within the Analytics Engine comprises two stages (Figure 8):

- *Training*, performed offline on previously collected data.
- *Detection*, executed on the live data stream.

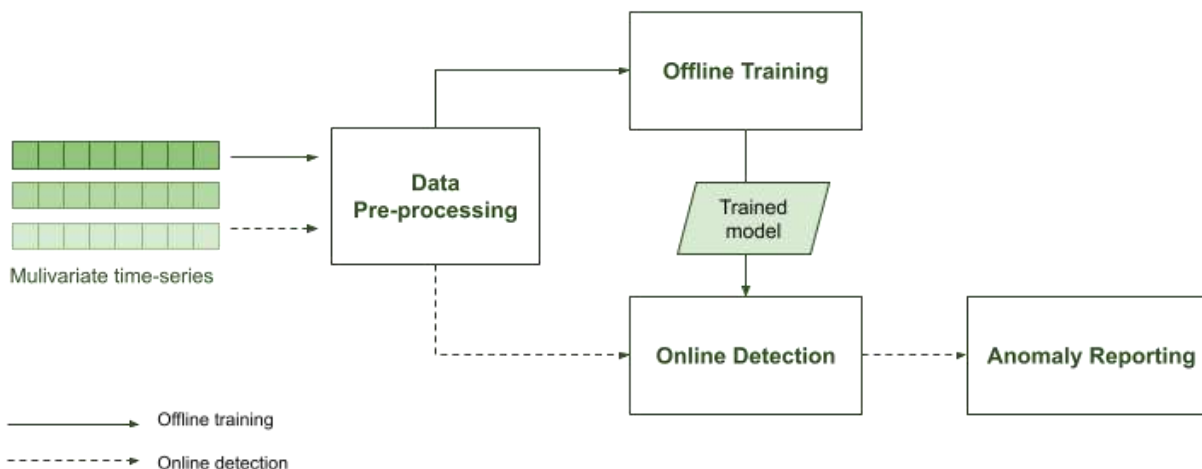


Figure 8: Anomaly detection process

Both stages utilise the same data processing module, which conducts data synchronisation and normalisation to harmonise the original data.

In the offline training stage, the model is trained on normal data collected from a continuous period (typically several hours) of normal system operation on the testbed infrastructure. More specifically, measurements assessing network performance for training purposes have been collected during traffic simulation conducted on the testbed using the 5G Traffic Simulation Manager (5GTSM) module (more details in deliverable “D2.5: 5G-EPICENTRE Experiment execution”). The VQ-VAE model can learn the normal patterns of multivariate time series from the training data: it encodes the input data into a latent space and reconstructs it. The latent vector is mapped to the nearest quantized vector, creating a low-dimensional representation. The offline training stage could be executed automatically on a regular basis, but it should ensure that the training data do not contain an excessive number of anomalies. For this phase, the support of experts is therefore necessary (in our case it was provided by the testbed network management team), to evaluate the collected data and ensure the absence of relevant anomalies (at least macro anomalies, identifiable by visually inspecting the data).

In the online detection stage, the Analytics Engine employs overlapping sliding windows to segment the incoming time series into window sequences (see Figure 9). The multivariate time series windows are treated as probability distributions, and the VQ-VAE model acts as an estimator for potential changes in distribution. Basically, the VQ-VAE structure is used to perform deep reconstruction of multivariate time series for the anomaly detection task, as the model can predict the reconstruction of the input time series window, by acting as an encoder–decoder at each moment.

For each new window sequence, the model outputs an anomaly score for each observation. If the time series window sequence has no anomalies, the reconstruction error at each moment is very low, resulting in a similar anomaly probability, and a relatively low anomaly score. Conversely, if the window sequence contains anomalies, the reconstruction error at normal moments is excessively high. If the anomaly score surpasses the predefined threshold, the observation is classified as anomalous; otherwise, it is classified as normal.

Upon detecting an anomaly, the Analytics Engine notifies the Front-end components via the Aggregator module, providing additional information for visualisation in the 5G-EPICENTRE Portal (see also D3.2).

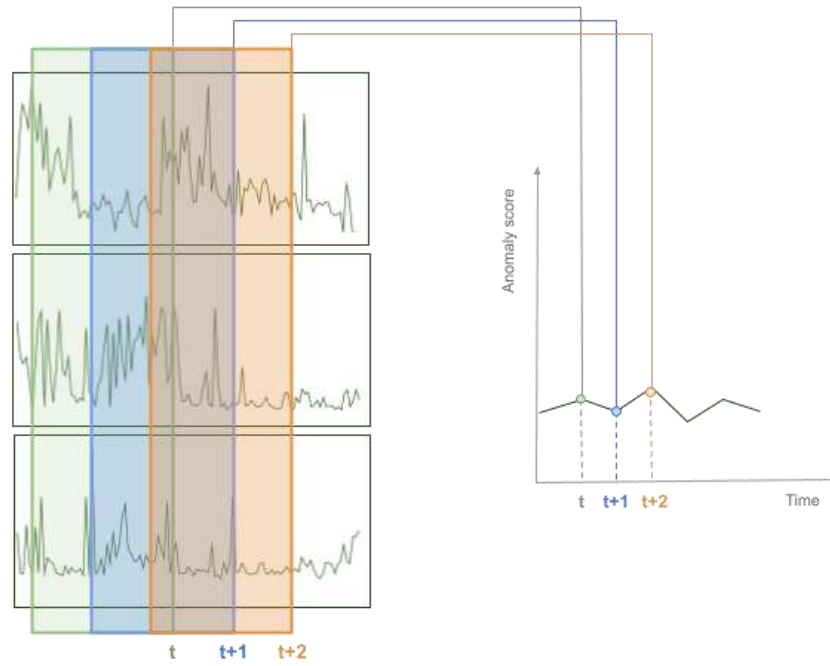


Figure 9: Sliding windows for anomaly detection on real-time data

4 Testing and validation activities

This Section describes the validation activities of the services provided by the Analytics Engine. It focuses on two particular scenarios: one involves the KPI calculation and statistical analysis of data gathered from a first party experiment; the other pertains to the identification of anomalies in infrastructure metrics during traffic simulation tests conducted on a testbed. The validation activities aim to evaluate the functionality and accuracy of the analytics services, as well as to demonstrate the approach and results obtained in a realistic scenario.

4.1 Results for KPI calculation and statistical analysis

This Section describes the testing and validation activities for the KPI calculation and statistical analysis performed by the Analytics Engine during an experimentation of the Use Case 4¹² in the ALB testbed.

During the experiment, the vertical application provided measurements for the calculation of the following KPIs:

- *network_rtt*
- *message_delay*
- *net_rssi*
- *net_rsrp*
- *net_rsrq*

According to the validation approach proposed in the project (see D1.6) for obtaining representative results of the tests on vertical applications, the experimentation procedure is based on multiple iterations, each involving numerous measurements used by the Analytics Engine for KPI calculation. The number of iterations (and the corresponding number of measurements per iteration) can be selected according to the need of the experiment and may vary for different KPIs. The KPI Monitor can be pre-configured to comply with the experiment's specifications in terms of iterations. Table 4 illustrates the request sent to the Analytics Service API, to preconfigure the KPI Monitor for the specific KPIs of the experiment.

Table 4: JSON format of the subscription request to the Analytics Network Application API.

```

1  {
2    "netapp_id": "mobitrust",
3    "testbed_id": 2,
4    "use_case_id": 4,
5    "measurement_list": [
6      {
7        "name": "net_rssi",
8        "kpi_type": "iteration_based",
9        "num_iterations": 10,
10       "num_values_per_iteration": 20,
11      },
12     {
13       "name": "net_rsrp",
14       "kpi_type": "iteration_based",
15       "num_iterations": 10,
16       "num_values_per_iteration": 20,

```

¹² The UC4 is an IoT application for improving first responders' situational awareness and safety provided by ONE

```

17     },
18     {
19         "name": "net_rsrq",
20         "kpi_type": "iteration_based",
21         "num_iterations": 10,
22         "num_values_per_iteration": 20,
23     },
24     {
25         "name": "net_rtt",
26         "kpi_type": "iteration_based",
27         "num_iterations": 10,
28         "num_values_per_iteration": 25,
29     },
30     {
31         "name": "message_delay",
32         "kpi_type": "iteration_based",
33         "num_iterations": 10,
34         "num_values_per_iteration": 20,
35     }
36 ]
37 }

```

The start of the experiment is notified to the Analytics Engine via a “special” message on its RabbitMQ exchange. This allows to prepare the Analytics Engine for the analysis of the experimental data.

The data collected by the Analytics Engine during the experiment consists of time series, similar to the one depicted in Figure 10, illustrating the measurement of *message_delay*.

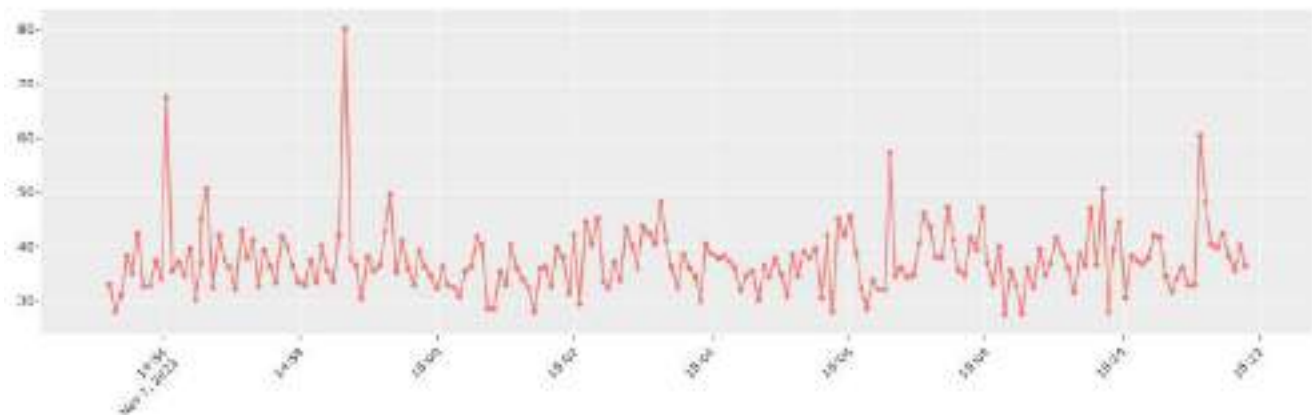


Figure 10: Time series for *message_delay* measurements during the experiment

At each iteration, the Analytics Engine aggregates the data to calculate the KPIs as the mean value (Figure 11), and sends the results to the Front-end for data visualisation (Table 5).

At the conclusion of the experiment, a “special” notification allows the Analytics Engine to finalise any remaining iterations, and compile the final results of the experiment.

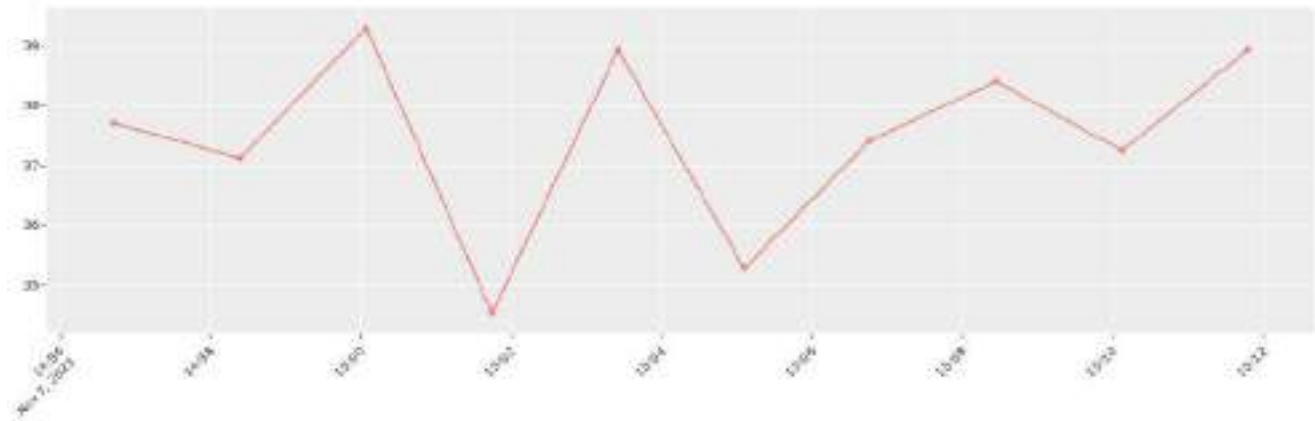
Figure 11: Evolution of the *message_delay* KPI over the iterations

Table 5: Message from the Analytics Aggregator to the Front-end with KPI on a single iteration.

```

1  {
2    "type": "kpis",
3    "category": "experiment",
4    "testbed_id": 1,
5    "scenario_id": 0,
6    "use_case_id": 4,
7    "experiment_id": 0,
8    "netapp_id": "mobitrust",
9    "data": [
10   {
11     "kpi_id": "net_rssi",
12     "timestamp": 1698332658924,
13     "origin": "ue",
14     "device_id": "mt-40234c",
15     "unit": "dbm",
16     "kpi_value": -39.5,
17     "iteration": 8
18   }
19 ]
20 }
```

In fact, upon the completion of the entire experiment, the Analytics Engine also provides the overall KPIs' values for the test case (averaged over iterations). This information is accompanied by a confidence interval and a statistical overview of the collected data, including percentiles, minimum, maximum, median and standard deviation calculated over all iterations (Figure 12). These statistical measures offer a thorough understanding of the data's distribution and variability, providing valuable insights into the performance and reliability of the test case.

The results of the KPI computation are packaged in a message using JSON format, and sent to the Analytics Aggregator. From there, eventually, they will reach the Front-end components for data visualisation. The following Table 6 illustrates an example of the messages provided by the Analytics Engine to the Front-end components, containing the final KPI for the entire experiment, and the associated statistics (more information about

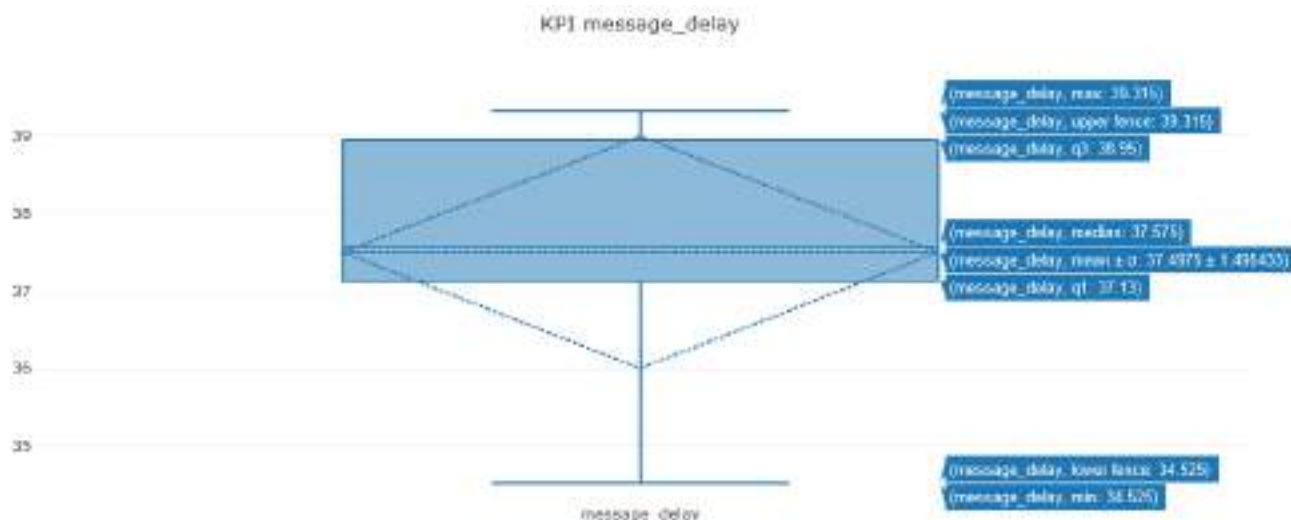


Figure 12: Graphical representation of the test case statistics produced by the AE

Table 6: Message including overall KPI of the experiment and related statistics.

```

1  {
2    "type": "kpis",
3    "category": "experiment",
4    "testbed_id": 1,
5    "scenario_id": 0,
6    "use_case_id": 4,
7    "experiment_id": 0,
8    "netapp_id": "mobitrust",
9    "data": [
10   {
11     "kpi_id": "net_rssi",
12     "timestamp": 1698332860381,
13     "origin": "ue",
14     "device_id": "mt-40234c",
15     "unit": "dbm",
16     "kpi_value": -39.720000000000006,
17     "kpi_ci": 0.19605362077088628,
18     "min": -40.05,
19     "max": -39.3,
20     "median": -39.75,
21     "sd_dev": 0.26000000000000045,
22     "p5": -40.027499999999996,
23     "p25": -39.95
24     "p75": -39.5625
25     "p95": -39.3
26   }

```

```
27     ]  
28 }
```

the data visualisation in the 5G-EPICENTRE Portal can be found in deliverable D3.2: “5G-EPICENTRE Front-end components”).

4.2 Results for the anomaly detection

The validation of the anomaly detection feature of the Analytics Engine, as reported in this Section, was conducted on the UMA testbed, one of the four testbeds of the 5G-EPICENTRE platform. The validation was performed on measurements from the infrastructure during traffic simulation experiments executed on the testbed. The traffic simulation experiments were set up using the 5GTSM, hosted on each testbed. This manager makes use of one or more remote iPerf agents, to generate dummy traffic between selected devices. The simulated scenario corresponds to one of the predetermined traffic profiles available on the 5G-EPICENTRE platform for the execution of the experiments. The aim of the activity was to test and validate the approach for anomaly detection, evaluating the results obtained on a specific dataset based on measurements collected in a realistic scenario.

The validation consisted of two phases: training phase and anomaly detection phase. The training phase aimed to create a deep learning model based on the VQ-VAE technology, which, as described in section 3.2.2, is a type of variational autoencoder that uses vector quantization to learn discrete latent representations of the data.

For the validation activity described in this Section, a four-hour session of traffic generated with 5GTSM was set up at the UMA testbed. The produced measurements about the network performances (*throughput*, *packet loss*, *jitter*, *ssSnr*, and *ssRSRP*) were collected by the Analytics Engine, and used as a dataset for the training of the model.

The training data should consist of normal network behaviour under high traffic conditions, which requires a lot of ground-truth labels. However, manually labelling the data is too hard and time-consuming, so it is not realistic in this case. The expert team of the testbed owner partner, provided the needed domain knowledge for the creation of the training dataset: they have knowledge about the expected traffic patterns and the types of anomalies that are likely to occur.

Therefore, the test assumed that there were no major anomalies in the data that the testbed engineers could notice, and treated all the data as *normal* for the training phase. The data were time series (one for each measurement), with different sampling rates and not well synchronised. The Analytics Engine performed a pre-processing step to transform them into a common format, before the training phase that generated the VQ-VAE model.

The anomaly detection phase aimed to test the ability of the DL model to detect anomalies in new data, which included simulated network problems. The network problems were simulated using Clumsy, a specific tool that introduced packet loss and random latency variation (with jitter variation) in the network. The simulated network problems caused anomalies in the measurements, which were represented by values still in a valid range, but having a different distribution with respect to the normal state. As a result of the simulation, a *silver standard* dataset has been created for validation purposes: this benchmark dataset is not manually annotated but we made the assumption that all data during the simulated network problems were anomalous, while the remaining data reflected normal network conditions. The silver dataset is a sequence of about 90 minutes of data containing simulated periods of anomalous data, with a balanced number of normal and anomalous data intervals.

The new data, including anomalies, were monitored by the Analytics Engine providing time windows of 60 seconds of data to the model in order to spot anomalies. In detail, for each window of time series data, the model encodes the data and then decodes it back to the original space. The reconstruction error, which is the difference

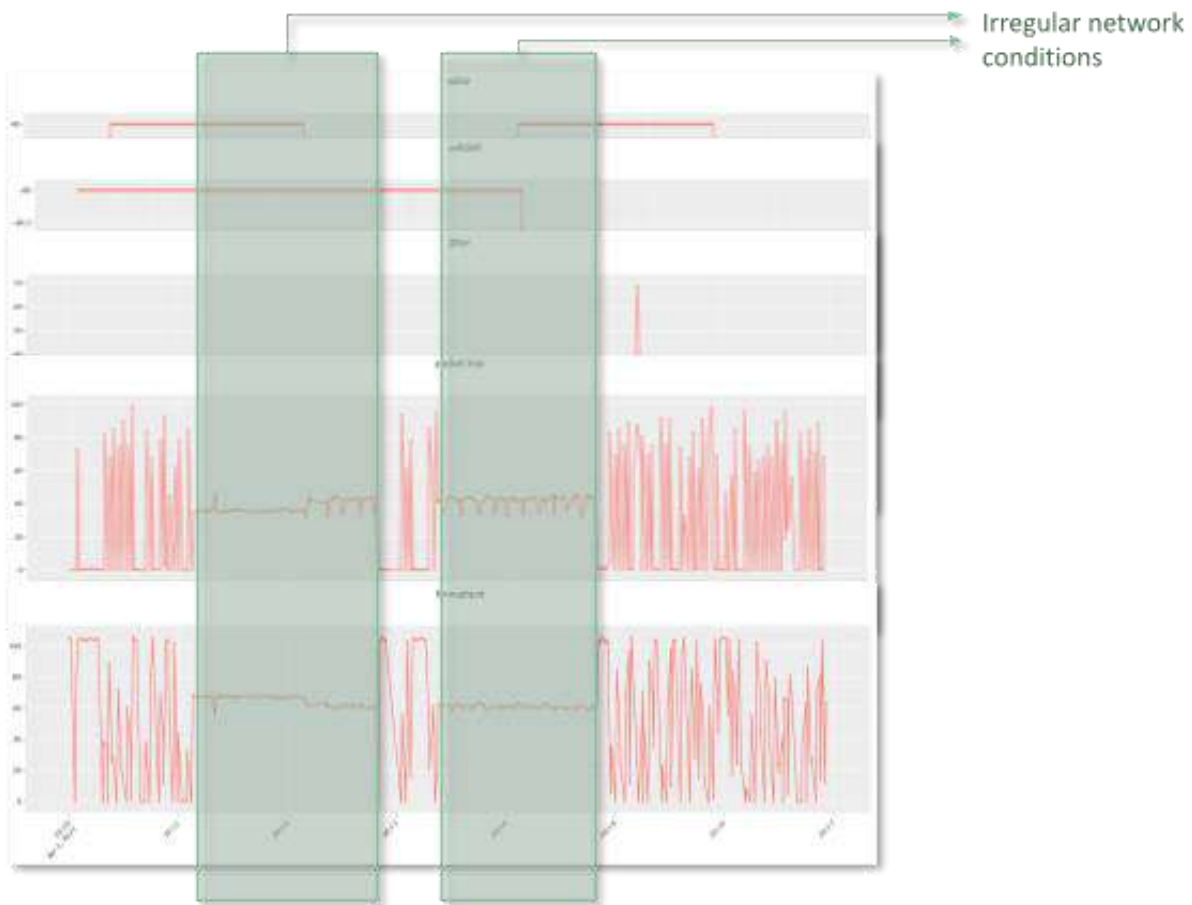


Figure 13: Example of data with simulated anomalies

between the original data and the reconstructed data, is then calculated. If the reconstruction error is significantly high, it means that the model found the data difficult to reconstruct, indicating that the data is anomalous. In other words, the model finds it difficult to reconstruct data, that it has not “seen” during training. A threshold for the reconstruction error is defined, to determine whether a data point is considered an anomaly. This threshold has been chosen carefully, to minimise false positives, which are instances where normal data points are incorrectly identified as anomalies. The choice of the threshold is a balance between sensitivity and specificity. A low threshold may result in many false positives (*i.e.*, normal data points identified as anomalies), while a high threshold may result in many false negatives (*i.e.*, anomalous data points identified as normal). It is important to note that the optimal threshold may vary depending on the specific characteristics of the data, and the cost associated with false positives versus false negatives. Therefore, the threshold is often determined empirically, by using a separate validation dataset, or through methods such as cross-validation.

Assessing unsupervised, or weakly supervised models can be particularly challenging, as there is no clear “ground truth” available for comparing their predictions.

In this validation activity, the silver dataset, which can be considered as a standard set of labelled data, has been developed to help evaluate the model. This dataset serves as a benchmark, providing a “ground truth” that is known with reasonable certainty. This means that the labels in this dataset are reliable, and accurately reflect the true nature of the data. Furthermore, this labelled dataset is representative of the task at hand. This implies that the characteristics of the data and the associated labels in this dataset closely mirror the type of data and tasks, that the models will encounter in real-world scenarios. Therefore, a model's performance on this dataset can be a good indicator of how it would perform on similar tasks in the future.

The validation results showed that the DL model based on VQ-VAE technology was able to learn a discrete latent representation of the normal state data, capturing the main features and patterns of the data. The DL model was able to detect anomalies in the new data, which deviated from the normal state representation. The following Figure (Figure 14) summarises the results of the anomaly detection model on the silver dataset:

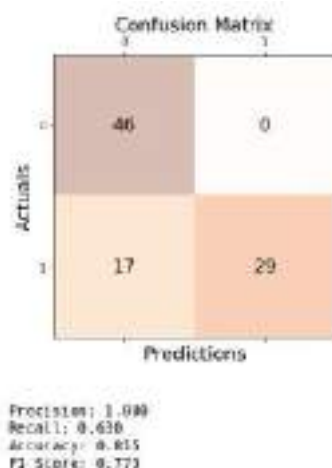


Figure 14: Performance of the VQ-VAE model for anomaly detection

The performance of the VQ-VAE model in the UMA testbed scenario differed slightly from the outcomes observed in the laboratory, and detailed in Section 3.2.2. This variance is attributed to our choice of a threshold, that minimises false positives, thereby improving the model's precision. While this decision aids in reducing false alarms, which are notifications of non-existent anomalies to the testbed owner, it may occasionally lead to the system failing to identify genuine anomalies that it could potentially recognize.

Notifications on the detection of anomalies have been created in the form of JSON messages sent towards the Analytics Aggregator, which propagated them to the visualisation components in the Portal (Table 7).

Table 7: JSON message containing the notification of a detected anomaly sent towards the Front-end.

```

1  {
2    "type": "anomalies",
3    "category": "experiment",
4    "testbed_id": 1,
5    "experiment_id": 1,
6    "netapp_id": "opentap_uma",
7    "data": [
8      {
9        "measurement_id": "packet_loss",
10       "timestamp": 1700496152784,
11       "origin": "UE",
12       "unit": "%",
13       "value": 20
14     }
15   ],
16   "anomaly": 1,
17 }

```

5 Conclusions

This deliverable describes the work carried out under Task 2.5 “*ML-driven experimentation analytics*” of the 5G-EPICENTRE project. The project’s focus was on building an end-to-end 5G experimentation platform tailored to the PPDR sector, and this Task has been devoted to the analysis of data collected during the use of the facilities, to test and validate the vertical applications of the final users.

The Analytics Engine, a key component of this Task, has been designed and developed for monitoring the experimental conditions of the vertical applications and the infrastructure. It computes KPIs to assess whether the targets set by the experiments are met, and analyses measurements from the infrastructure to detect potential anomalies in the network conditions.

The Analytics Engine’s decentralised architecture has been designed to reduce network overhead and latency, and improve the scalability of the system. The data processing modules of the Analytics Engine, namely the Analytics Driver, the KPI Monitor, the QoS/QoE Monitor, and the Analytics Aggregator, have been used to collect, filter, aggregate, and analyse the data generated by the experimental applications and the infrastructure components.

The data analytics services provided by the Analytics Engine have been instrumental in analysing the health and performance of the 5G-EPICENTRE platform during the experiment executions, as well as supporting the evaluation and validation of the vertical applications. The Analytics Engine’s ability to perform analysis of the metrics produced by the vertical applications to monitor the experiment by calculating the KPIs and statistics, has been tested and validated in real case scenarios.

The solution developed for the QoS/QoE Monitor module of the Analytics Engine, which uses innovative DL techniques to perform anomaly detection on the metrics collected from the testbed infrastructure, has shown promising results. The analysis is performed in near-real time, and the detected anomalies are notified to the Front-end components residing at the 5G-EPICENTRE Portal for data visualisation. This feature has enabled the final users to have more detailed information about the network performance during the experiment, enabling them to evaluate the KPIs of the experiment in the right perspective.

In conclusion, this deliverable provided a comprehensive report on the application of data analytics on experimental data collected on the 5G-EPICENTRE platform. The results achieved have not only demonstrated the practical utility of AI in data analysis for 5G network platforms, but also laid a solid foundation for future research and development in this field.

References

- [1] Sayadi, B., Chang, C.-Y., Tranoris, C., Iordache, M., Katsaros, K., Vilalta, R., et al. (2022). Network Applications: Opening up 5G and beyond networks [White paper]. Zenodo. <https://doi.org/10.5281/ze-nodo.7123919>
- [2] Braei, M., & Wagner, S. (2020). Anomaly Detection in Univariate Time-series: A Survey on the State-of-the-Art. ArXiv, abs/2004.00433.
- [3] Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys*, 41, 3, 1–58. doi: 10.1145/1541880.1541882.
- [4] Schmidl, S., Wenig, P., & Papenbrock, T. (2022). Anomaly detection in time series: A comprehensive evaluation. *Proc. VLDB Endow.*, vol. 15, no. 9, p. 1779–1797. <https://www.vldb.org/pvldb/vol15/p1779-wenig.pdf>.
- [5] Choi, K., Yi J., Park, C., & Yoon, S. (2021). Deep Learning for Anomaly Detection in Time-Series Data: Review, Analysis, and Guidelines. *IEEE Access*. PP. 1-1. 10.1109/ACCESS.2021.3107975.
- [6] Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., & Shroff, G. (2016). Lstm-based encoder-decoder for multi-sensor anomaly detection. <https://arxiv.org/pdf/1607.00148.pdf>.
- [7] Wei, Y., Jang-Jaccard, J., Xu, W., Sabrina, F., Camtepe, S., & Boulic, M. (2023). LSTM-Autoencoder-Based Anomaly Detection for Indoor Air Quality Time-Series Data. *IEEE Sensors Journal*, vol. 23, no. 4, pp. 3787–3800, 15 Feb.15, 2023, doi: 10.1109/JSEN.2022.3230361.
- [8] Provotar, O. I., Linder, Y. M., & Veres, M. M. (2019). Unsupervised Anomaly Detection in Time Series Using LSTM-Based Autoencoders. *IEEE International Conference on Advanced Trends in Information Theory (ATIT)*, Kyiv, Ukraine, 2019, pp. 513–517, doi: 10.1109/ATIT49449.2019.9030505.
- [9] Gangloff, H., Pham, M. -T., Courtrai L., & Lefèvre, S. (2022). Leveraging Vector-Quantized Variational Autoencoder Inner Metrics for Anomaly Detection. *26th International Conference on Pattern Recognition (ICPR)*, Montreal, QC, Canada, 2022, pp. 435–441, doi: 10.1109/ICPR56361.2022.9956102.
- [10] Marimont, S. N. & Tarroni, G. (2020). Anomaly Detection through Latent Space Restoration Using Vector-Quantized Variational Autoencoders. <https://doi.org/10.48550/arXiv.2012.06765>.
- [11] Van den Oord, A., Vinyals, O., & Kavukcuoglu, K. (2017). Neural discrete representation learning. *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 6309–6318.
- [12] Audibert, J., Michiardi, P., Guyard, F., Marti, S., & Zuluaga, M. A. (2020). Usad: Unsupervised anomaly detection on multivariate time series. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ser. KDD '20*. New York, NY, USA: Association for Computing Machinery, 2020, p. 3395–3404. [Online]. <https://doi.org/10.1145/3394486.3403392>.
- [13] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. C., & Bengio, Y. (2014). Generative Adversarial Nets. *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014*. Montreal, Quebec, Canada, 2672–2680.
- [14] Zong, B., Song, Q., Min, M. R., Cheng, W., Lumezanu, C., Cho, D., & Chen, H. (2018). Deep autoencoding gaussian mixture model for unsupervised anomaly detection. *International Conference on Learning Representations*, 2018. [Online]. <https://openreview.net/forum?id=BJLHbb0->
- [15] Munir, M., Siddiqui, S. A., Dengel, A., & Ahmed, S. (2019). DeepAnT: A Deep Learning Approach for Unsupervised Anomaly Detection in Time Series. *IEEE Access*, vol. 7, pp. 1991–2005, 2019, doi: 10.1109/ACCESS.2018.2886457.
- [16] Lin, S., Clark, R., Birke, R., Schönborn, S., Trigoni, N., & Roberts, S. (2020). Anomaly Detection for Time Series Using VAE-LSTM Hybrid Model. *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Barcelona, Spain, 2020, pp. 4322–4326, doi: 10.1109/ICASSP40776.2020.9053558.

Annex I: Portal API documentation

This Section reports the specification of the API exposed Analytics Services Network Application. The vertical can consume the API to subscribe to services for KPI calculation and statistical analysis provided by the Analytics Engine.

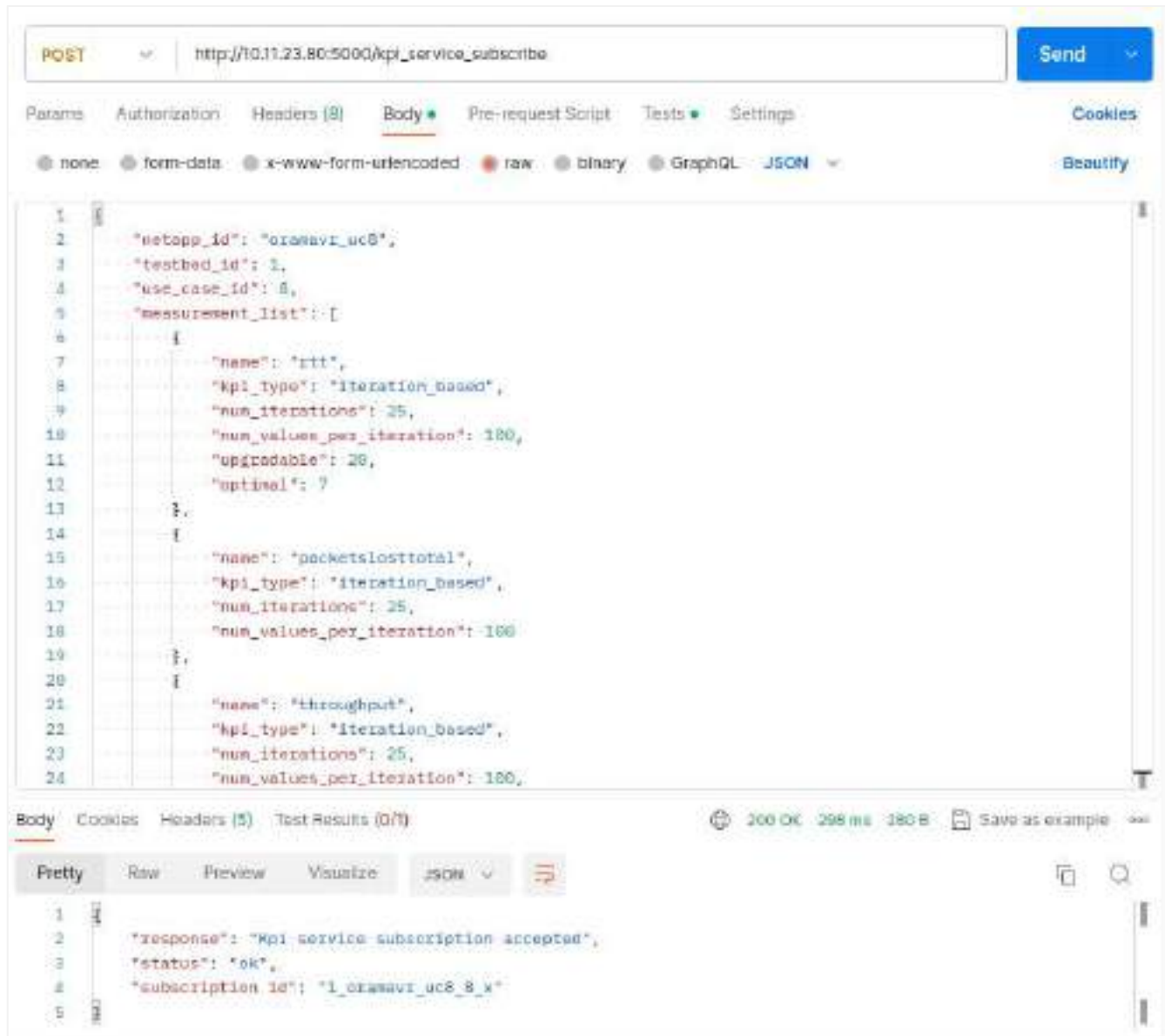
Subscription endpoint

This endpoint is used to send a request for the subscription to the Analytics Services Network Application. The request contains information about KPI calculation to be used to pre-configure the Analytics Engine to calculate KPIs and perform statistical analysis over the measurements provided by the subscribed vertical. If the request is not successful, the response contains the corresponding error code and message.

Table 8: Endpoint for subscribing to the analytics services for KPI calculation and statistical analysis

Subscription	
Method	POST
Endpoint	/kpi_service_subscribe
Request Headers	None
Request Body	<p>descriptor [Object] REQUIRED</p> <p>A JSON structure exchanged between the subscriber and the Analytics Network Application, to request KPI calculation and statistical services for the set of measurements specified in the JSON. It contains specific information about the <i>testbed_id</i>, the <i>experiment_id</i>, the <i>netapp_id</i> and the list of measurements to be collected during the experiment.</p> <p>For each measurement to be used for KPI calculation, according to the standard methodology (see deliverable D1.6) the JSON specifies:</p> <ul style="list-style-type: none"> • the expected number of iterations and the number of measurements per iteration • the “upgradable” and “optimal” thresholds (optional) • the outlier removal flag that indicates that the experiment requires automatic outlier removal during KPI calculation (optional)
Response	<p>200 The subscription was accepted. The response will contain the status “ok” and the <i>subscription_id</i>.</p> <p>400 The format of the descriptor is not valid.</p> <p>500 The server cannot process the request for an unknown reason.</p>

The following figure shows an example of subscription of the analytics services.



The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** http://10.11.23.80:5000/kpi_service_subscribe
- Body (Request):**

```

1
2  "netopp_id": "oraxavi_uc8",
3  "testbed_id": 1,
4  "use_case_id": 8,
5  "measurement_list": [
6    {
7      "name": "rtt",
8      "kpi_type": "iteration_based",
9      "num_iterations": 25,
10     "num_values_per_iteration": 100,
11     "upgradable": 20,
12     "optimal": 7
13   },
14   {
15     "name": "packetslosttotal",
16     "kpi_type": "iteration_based",
17     "num_iterations": 25,
18     "num_values_per_iteration": 100
19   },
20   {
21     "name": "throughput",
22     "kpi_type": "iteration_based",
23     "num_iterations": 25,
24     "num_values_per_iteration": 100,

```
- Response (Status 200 OK):**

```

1
2  "response": "Kpi service subscription accepted",
3  "status": "ok",
4  "subscription id": "1_oraxavi_uc8_8_x"
5

```

Figure 15: Example of API invocation for the subscription of the analytics service

Delete subscription endpoint

This endpoint is used to delete the subscription to the Analytics Services Network Application. If the request is not successful, the response contains the corresponding error code and message.

Table 9: Endpoint for deleting subscription to the analytics services for KPI calculation and statistical analysis

Delete Subscription	
Method	GET
Endpoint	/kpi_service_delete/:subscription_id
Request Headers	None

Request Body	None
Response	200 The deletion was successful.
	404 The “subscription_id” was not found.
	500 The server cannot process the request for an unknown reason.

Get Subscription endpoint

This endpoint is used to retrieve the information associated with “subscription_id”. For a valid “subscription_id”, the response will contain the descriptor of the subscription, a JSON structure containing the information about the measurements associated with the provided subscription_id. If the request is not successful, the response contains the corresponding error code and message.

Table 10: Endpoint for returning the descriptor associated with “subscription_id”

Get Subscription	
Method	/kpi_service_get/:subscription_id
Endpoint	None
Request Headers	None
Request Body	200 The query was successful. The response will contain the information associated with “subscription_id”, structured in a JSON format.
Response	404 The “subscription_id” was not found.
	500 The server cannot process the request for an unknown reason.
	/kpi_service_get/:subscription_id

Annex II: Integration with the Testbed

This section provides further details on the interface between the Analytics Engine and the modules at the testbed. The Analytics Driver is responsible for gathering real-time data from the vertical undergoing experimentation on the 5G-EPICENTRE platform, as well as from the network infrastructure of the testbed.

The communication interface is built upon an MQTT broker, which facilitates integration with the Publisher component at each testbed. Within the context of WP4, a message schema (in JSON format) has been established to define the format of the messages exchanged on the interface (see the following schema in Table 11).

Table 11: Analytics Driver – Publisher interface message schema.

```

1  {
2    "$id": "https://example.com/tb-adrv.schema.json",
3    "$schema": "https://json-schema.org/draft/2020-12/schema",
4    "$description": "The JSON schema for messages exchanged on Tb-Adrv interface",
5    "title": "tb-adrv",
6    "type": "object",
7    "properties": {
8      "category": {
9        "enum": ["5g_network", "nfv_mano", "vnf_chain", "experiment"],
10       "description": "Mandatory, filled by the Testbed or the Network Application"
11     },
12     "testbed_id": {
13       "enum": [1, 2, 3, 4],
14       "description": "Mandatory, filled by the Testbed (UMA: 1, ALB: 2, CTTC: 3, HHI: 4)"
15     },
16     "scenario_id": {
17       "type": "integer",
18       "description": "Optional, filled by the Testbed"
19     },
20     "use_case_id": {
21       "type": "integer",
22       "description": "Optional, only for first party experiments, filled by the Testbed"
23     },
24     "experiment_id": {
25       "type": "integer",
26       "description": "Optional, filled by the Testbed"
27     },
28     "netapp_id": {
29       "type": "string",
30       "description": "Optional, filled by the use case app"
31     },
32     "data": {
33       "type": "array",

```

```

34         "items": { "ref": "#/$defs/tb-adv_data_item" },
35         "minItems": 1
36     },
37 },
38 "required": ["category", "testbed_id", "data"],
39 "$defs": {
40     "tb-adv_data_item": {
41         "$id": "https://example.com/tb-adv-data-item.schema.json",
42         "$schema": "https://json-schema.org/draft/2020-12/schema",
43         "$description": "The JSON schema for messages exchanged on Tb-Adv interface",
44         "title": "tb-adv-data-item",
45         "type": "object",
46         "properties": {
47             "type": {
48                 "type": "string",
49                 "description": "mandatory, identifies the name of the metric"
50             },
51             "timestamp": {
52                 "type": "integer",
53                 "description": "mandatory, as UTC POSIX timestamp, in milliseconds"
54             },
55             "origin": {
56                 "enum": ["UE", "RAN", "5GC", "EPC", "main data server", "edge"],
57                 "description": "mandatory, origin of the metric"
58             },
59             "unit": {
60                 "enum": ["gbps", "mbps", "kbps", "bps", "s", "ms", "us",
61                     "ordinal", "rate", "%", "interval"],
62                 "description": "optional, unit of measurement of the metric"
63             },
64             "device_id": {
65                 "type": "string",
66                 "description": "optional, origin device identifier"
67             },
68             "value": {
69                 "anyOf": [{"type": "string"}, {"type": "integer"}
70                     {"type": "number"}, {"type": "boolean"}],
71                 "description": "value of the metric"
72             }
73         },
74         "required": ["type", "timestamp", "origin"],
75         "additionalProperties": {
76             "anyOf": [{"type": "string"}, {"type": "integer"}

```

```

77         {"type": "number"}, {"type": "boolean"}],
78         "description": "custom fields (key-value pairs as from native metrics)"
79     }
80 }
81 }
82 }

```

Certain fields of the message are populated by the vertical application, while others are filled by the Publisher with specific details about the ongoing experiment, which remain transparent to the experimenter.

A detailed explanation of the fields is provided below:

- **category**: specifies whether the measurements in the message come from the vertical application ("experiment") or from network infrastructure ("5g_network") or other testbed components ("nfv_mano", "vnf_chain"). This is a mandatory field.
- **testbed_id**: numeric identifier of the testbed (1: Malaga, 2: Aveiro, 3: Barcelona, 4: Berlin). This is a mandatory field.
- **scenario_id**: numeric identifier of the configuration scenario under which the measurements were produced.
- **use_case_id**: use case id number (only for measurements from first party experiments).
- **experiment_id**: identifier of the specific run of an experiment.
- **netapp_id**: identifier of the Network Application that produces the measurements.
- **data**: array of data items (mandatory). Each item contains a single measurement's value plus additional metadata.

Elements in data array can have the following fields:

- **type**: name of the measurement (mandatory).
- **value**: value of the measurement.
- **timestamp**: timestamp (UTC POSIX), in milliseconds, of the measurement (mandatory).
- **origin**: origin of the measurement, *i.e.*, user device or other infrastructure components (mandatory).
- **unit**: unit of measurement.
- **device_id**: for experiments involving multiple devices, allows to identify the specific device that produces the measurement.

The flexibility of the schema allows to include in the data items additional fields to adapt the message to specific needs (see examples below).

Table 12 illustrates an example of a message including measurements provided by the vertical application during the experimentation.

Table 12: Example of message with measurements from the vertical application.

```

1  {
2      "category": "experiment",
3      "testbed_id": 1,
4      "scenario_id": 3,
5      "experiment_id": 56,
6      "use_case_id": 1,
7      "netapp_id": "m6",
8      "data": [

```

```
9      {
10         "type": "m6_video_latency",
11         "timestamp": 1656498094331,
12         "origin": "UE",
13         "unit": "ms",
14         "value": 25,
15         "longitude": -4.49981645,
16         "latitude": 36.71677381,
17         "altitude": 87.33721923828125,
18         "asu_level": 72,
19         "level": 4,
20         "csi_rsrq": 2147483647,
21         "csi_rsrp": 2147483647,
22         "dbm": -68,
23         "csi_sinr": 2147483647,
24         "radio_type": "nr",
25         "bitrate": 870400,
26         "fps": 25,
27         "iframe_interval": 2,
28         "video_protocol": "mcs",
29         "viewers": 0,
30         "width": 640,
31         "height": 480
32     }
33 ]
34 }
```

Table 13 presents an example of measurements derived from the network infrastructure.

Table 13: Example of message with measurements from the network infrastructure.

```
1  {
2     "category": "5g_network",
3     "experiment_id": 1,
4     "testbed_id": 1,
5     "scenario_id": 0,
6     "netapp_id": "opentap_uma",
7     "data": [
8         {
9             "type": "throughput",
10            "timestamp": 1701419813898,
11            "origin": "UE",
12            "unit": "mbps",
13            "value": 89.4,
```

```
14     }
15   ]
16 }
17
18 {
19   "category": "5g_network",
20   "experiment_id": 1,
21   "testbed_id": 1,
22   "scenario_id": 0,
23   "netapp_id": "opentap_uma",
24   "data": [
25     {
26       "type": "jitter",
27       "timestamp": 1701419813898,
28       "origin": "UE",
29       "unit": "ms",
30       "value": 0.081,
31     }
32   ]
33 }
```

Annex III: Integration with the Front-end

This section provides further details on the interface between the Analytics Engine and the modules at the Front-end level. The Analytics Aggregator is responsible for gathering the results of the data analytics activities performed at the testbed level and making them available to the 5G-EPICENTRE Portal for visualisation.

As for other components, the communication interface is built upon an MQTT broker, which facilitates integration with the Visualization tools component in the Front-end layer. Within the context of WP4, a message schema (in JSON format) has been established to define the format of the messages exchanged on the interface (see the following schema in Table 14).

Table 14: Analytics Aggregator – 5G-EPICENTRE Portal interface message schema.

```

1  {
2    "$id": "https://example.com/tb-adrv.schema.json",
3    "$schema": "https://json-schema.org/draft/2020-12/schema",
4    "$description": "The JSON schema for messages exchanged on Aggr-Vis interface",
5    "title": "aggr-vis",
6    "type": "object",
7    "properties": {
8      "type": {
9        "enum": ["kpis", "params", "anomalies"],
10       "description": "specifies the type of the data flow"
11     },
12     "category": {
13       "enum": ["5g_network", "nfv_mano", "vnf_chain", "experiment"],
14       "description": "Mandatory, filled by the Testbed or the Network Application"
15     },
16     "testbed_id": {
17       "enum": [1, 2, 3, 4],
18       "description": "Mandatory, filled by the Testbed (UMA: 1, ALB: 2, CTTC: 3, HHI: 4)"
19     },
20     "scenario_id": {
21       "type": "integer",
22       "description": "Optional, filled by the Testbed"
23     },
24     "use_case_id": {
25       "type": "integer",
26       "description": "Optional, only for first party experiments, filled by the Testbed"
27     },
28     "experiment_id": {
29       "type": "integer",
30       "description": "Optional, filled by the Testbed"
31     },
32     "netapp_id": {
33       "type": "string",

```



```

34     "description": "Optional, filled by the use case app"
35   },
36   "testbed_id": {
37     "enum": [0, 1],
38     "description": "Optional, only for anomalies data flow"
39   },
40   "data": {
41     "type": "array",
42     "items": {
43       "oneOf": [
44         { "$ref": "#/$defs/aggr-vis_kpis_item" },
45         { "$ref": "#/$defs/aggr-vis_params_item" },
46         { "$ref": "#/$defs/aggr-vis_anomalies_item" },
47       ]
48     },
49     "minItems": 1
50   }
51 },
52 "required": ["type", "category", "testbed_id", "data"],
53 "$defs": {
54   "aggr-vis_kpis_item": {
55     "$id": "https://example.com/aggr-vis_kpis_item.schema.json",
56     "$schema": "https://json-schema.org/draft/2020-12/schema",
57     "$description": "The JSON schema for kpis items",
58     "title": "aggr-vis_kpis_item",
59     "type": "object",
60     "properties": {
61       "kpi_id": {
62         "type": "string",
63         "description": "mandatory, identifies the name of the metric"
64       },
65       "kpi_value": {
66         "anyOf": [{"type": "string"}, {"type": "integer"}
67           {"type": "number"}, {"type": "boolean"}],
68         "description": "value of the metric"
69       }
70     "timestamp": {
71       "type": "integer",
72       "description": "mandatory, as UTC POSIX timestamp, in milliseconds"
73     },
74     "unit": {
75       "enum": ["gbps", "mbps", "kbps", "bps", "s", "ms", "us",
76         "ordinal", "rate", "%", "interval"],

```

```
77         "description": "optional, unit of measurement of the metric"
78     }
79 },
80 "required": ["kpi_id", "kpi_value", "timestamp", "unit"],
81 "additionalProperties": {
82     "anyOf": [{"type": "string"}, {"type": "integer"}
83         {"type": "number"}, {"type": "boolean"}],
84     "description": "custom fields (key-value pairs as from native metrics)"
85 }
86 },
87
88 "aggr-vis_params_item": {
89     "$id": "https://example.com/aggr-vis_params_item.schema.json",
90     "$schema": "https://json-schema.org/draft/2020-12/schema",
91     "$description": "The JSON schema for params items",
92     "title": "aggr-vis_params_item",
93     "type": "object",
94     "properties": {
95         "param_id": {
96             "type": "string",
97             "description": "mandatory, identifies the name of the metric"
98         },
99         "timestamp": {
100             "type": "integer",
101             "description": "mandatory, as UTC POSIX timestamp, in milliseconds"
102         },
103         "unit": {
104             "enum": ["gbps", "mbps", "kbps", "bps", "s", "ms", "us",
105                 "ordinal", "rate", "%", "interval"],
106             "description": "optional, unit of measurement of the metric"
107         }
108     },
109     "required": ["param_id", "timestamp", "unit"],
110     "additionalProperties": {
111         "anyOf": [{"type": "string"}, {"type": "integer"}
112             {"type": "number"}, {"type": "boolean"}],
113         "description": "custom fields (key-value pairs as from native metrics)"
114     }
115 },
116
117 "aggr-vis_anomalies_item": {
118     "$id": "https://example.com/aggr-vis_anomalies_item.schema.json",
119     "$schema": "https://json-schema.org/draft/2020-12/schema",
```

```

120     "$description": "The JSON schema for anomalies items",
121     "title": "aggr-vis_anomalies_item",
122     "type": "object",
123     "properties": {
124         "measurements_id": {
125             "type": "string",
126             "description": "mandatory, identifies the name of the metric"
127         },
128         "value": {
129             "anyOf": [{"type": "string"}, {"type": "integer"}
130                 {"type": "number"}, {"type": "boolean"}],
131             "description": "value of the metric"
132         }
133         "timestamp": {
134             "type": "integer",
135             "description": "mandatory, as UTC POSIX timestamp, in milliseconds"
136         },
137         "unit": {
138             "enum": ["gbps", "mbps", "kbps", "bps", "s", "ms", "us",
139                 "ordinal", "rate", "%", "interval"],
140             "description": "optional, unit of measurement of the metric"
141         }
142     },
143     "required": ["param_id", "timestamp", "unit"],
144     "additionalProperties": {
145         "anyOf": [{"type": "string"}, {"type": "integer"}
146             {"type": "number"}, {"type": "boolean"}],
147         "description": "custom fields (key-value pairs as from native metrics)"
148     }
149 }
150 }
151 }

```

A detailed explanation of the fields is provided below:

- **type**: specifies the type of data contained in the message according to the three distinct data flows produced by the Aggregator. This is a mandatory field and can take the following values:
 - "kpis", for KPIs based on measurements received from the first party or third-party vertical applications;
 - "params", for statistics based on predefined time windows on measurements coming from the testbed infrastructure;
 - "anomalies", for anomalies detected in measurements from the infrastructure.
- **category**: specifies whether the measurements in the message come from the vertical application ("experiment"), network infrastructure ("5g_network") or other testbed components ("nfv_man0", "vnf_chain"). This is a mandatory field.

- **testbed_id**: numeric identifier of the testbed (1: Malaga, 2: Aveiro, 3: Barcelona, 4: Berlin). This is a mandatory field.
- **scenario_id**: numeric identifier of the configuration scenario under which the measurements were produced.
- **use_case_id**: use case id number (only for measurements from first party experiments).
- **experiment_id**: identifier of the specific run of an experiment.
- **netapp_id**: identifier of the Network Application that produces the measurements.
- **anomaly**: only for the “anomalies” data flow. The field can take the value 1 or 0 depending on whether an anomaly has been detected or not on a given time window.
- **data**: array of data items (mandatory). Each item contains a single measurement’s value plus additional metadata.

For each type of data flow, the schema provides a predefined set of fields for the items in the data array, as detailed in the following table. The schema can allow additional fields to adapt the message to specific needs.

Table 15: Details of the format of the messages from the Aggregator to the Portal

Item type	Item Fields
<i>params</i>	<ul style="list-style-type: none"> ● param_id: name of the measurement; ● timestamp: timestamp (UTC POSIX), in milliseconds, of the measurement; ● unit: unit of measurement.
<i>kpis</i>	<ul style="list-style-type: none"> ● kpi_id: identifies the name of the measurement; ● kpi_value: value of the measurement; ● timestamp: timestamp (UTC POSIX), in milliseconds, of the measurement; ● unit: unit of measurement;
<i>anomalies</i>	<ul style="list-style-type: none"> ● measurement_id: name of the measurement; ● timestamp: timestamp (UTC POSIX), in milliseconds, of the measurement; ● unit: unit of measurement; ● value: value of the measurement.

Table 16 illustrates an example of a message including “params”, *i.e.*, statistics on measurements from the infrastructure.

Table 16: Example of message with statistics on measurements from the infrastructure.

```

1  {
2    "type": "params",
3    "category": "5g_network",
4    "testbed_id": 1,
5    "scenario_id": 0,
6    "experiment_id": 0,
7    "origin": "kubernetes",
8    "data": [
9      {
10     "param_id": "container_cpu_usage_seconds_total",

```

```
11         "timestamp": 1702653370207,  
12         "unit": "s",  
13         "mean": 199259.37533946542,  
14         "std_dev": 699939.2227080354,  
15         "min": 0.007919052,  
16         "max": 6104993.473112935,  
17         "median": 5247.382495741,  
18         "p25": 0.041792569,  
19         "p75": 57246.414882951,  
20         "p05": 0.0172413868,  
21         "p95": 1031598.5211124117,  
22     }  
23 ]  
24 }
```