
Monitoring and Analytics for the whole Lifecycle, on Models, Hardware, and Software

# D5.2 Open-source reference implementation of LLM-driven analysis (continuously updated)

| | |
|---|---|
| **Deliverable Due Date** | 2026-06-30 |
| **Deliverable Type** | Software |
| **Deliverable Number** | D5.2 |
| **Authors** | Wasif Afzal (MDU, Sweden), Batur Alp Akdoğan (LTG, Turkey), Tóth László (Uni. Szeged), Mathias Axling (CNET) |
| **Dissemination Level** | Public |

# Contributors

| Name | Short Affiliation |
|---|---|
| Wasif Afzal | MDU, Sweden |
| Tóth László | Uni. Szeged, Hungary |
| Batur Alp Akdoğan | LTG, Turkey |
| Mathias Axling | CNET, Sweden |

# Reviewers

| Name | Short Affiliation | Date |
|---|---|---|
| Tóth László | Uni. Szeged, Hungary | 2026-06-23 |
| Batur Alp Akdoğan | LTG, Turkey | 2026-06-26 |

# Document Revision Log

| Version | Revision | Date | Description | Author |
|---|---|---|---|---|
| 01 | 01 | 2026-06-17 | First Draft | Wasif Afzal (MDU, Sweden) |
| 02 | 02 | 2026-06-23 | Revision | Tóth László (Uni. Szeged, Hungary) |
| 03 | 03 | 2026-06-25 | Revision | Mathias Axling (CNET, Sweden) |
| 04 | 04 | 2026-06-26 | Revision | Batur Alp Akdoğan (LTG, Turkey) |
| 05 | 05 | 2026-06-29 | Final Version | Wasif Afzal (MDU, Sweden) |
| 05 | 05 | 2026-06-29 | Submitted | Wasif Afzal (MDU, Sweden) |

# Table of Contents

# 1. Executive Summary of Deliverable

The purpose of D5.2 is to continuously document the analysis done as a result of processing of diverse log files coming from Mona Lisa's use case providers' systems. The analysis is done through design and implementation of various analysis techniques including the use of large language models (LLMs) as well as other artificial intelligence (AI) based techniques such as machine learning (ML).

# 2. Introduction

The main objective of work package 5 is to provide the semantic layer for MONA LISA's analytics and business logic. This WP receives various data (e.g., code traces and logs) from the infrastructure (WP3) and models from the model reengineering layer (WP4) and performs advanced analyses using methods from the subdomains of Artificial Intelligence (AI), e.g, Machine Learning (ML), Retrieval Augmented Large Language Models (RA LLM) and model-based approaches from WP4. The analysis then presents results for the visual front end (WP6). Figure 1 shows the interactions of various WPs with WP5.

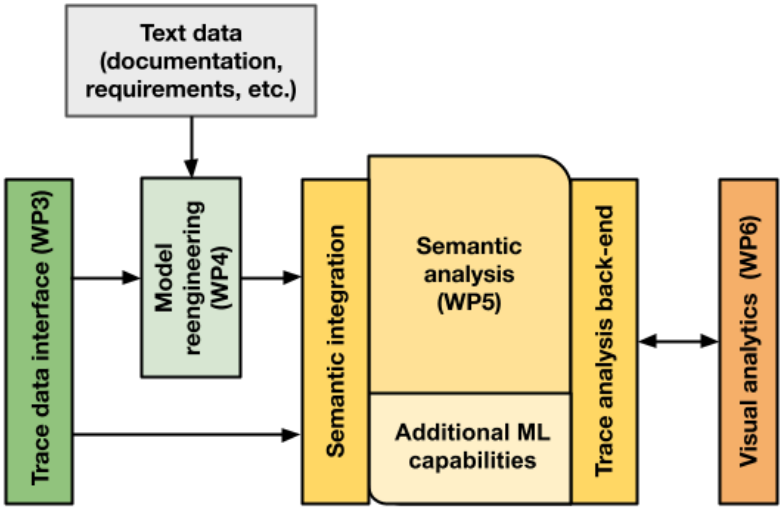


**Figure 1.** ***Interaction between the analysis work packages (WP4, WP5) and their role within the overall pipeline.***

WP5 has following two objectives:

- *Objective 1:* Semantic data integration using Knowledge Graphs and Language Models. Developers map data sources and analysis tools based on their knowledge of data semantics with assistance from ontology-based data access tools using domain-specific knowledge graphs (KG). Additional automation will be developed through Large Language Models (LLMs).
- *Objective 2:* Data analysis with ML and Retrieval-augmented LLMs. The aim is to integrate the semantically unified data model with both ML and LLM backends and tools. These tools and libraries will provide the analysis capabilities of the MONA LISA solution, giving users new insights into data traces.

# 3. Background

Many modern systems produce vast amounts of log files that could provide valuable insight for further processing, including the investigation of past issues, or in other words, Root Cause Analysis (RCA) and other types of analysis including anomaly detection. To make log files, often considered semi-structured data, interpretable, they are often parsed and preprocessed. However, many modern systems, including those in distributed embedded systems, consist of subsystems, components, and devices, and furthermore, each might produce data from various levels of the

system. Therefore, such a distributed embedded system might produce various types/artifacts of log files, each with specific considerations for processing and, as a result, pose additional processing challenges. Table 1 summarizes the analysis requirements in MONA LISA.

***Table 1. Types of MONA LISA analysis***

| Use case provider | Type of analysis |
|---|---|
| Alstom | Automated root cause analysis of defects |
| Eli Alps | Automated root cause analysis of defects |
| Bilecik Demircelik | Automated root cause analysis of defects & predictive analysis |
| CNET | Predictive analysis of remaining useful life |

# 4. Alstom

The on-going automated root cause analysis (RCA) of defects/incidents provided by the Alstom's dataset aims to:

- Analyze the data - log files, and a list of past issues, which include their root causes identified by an engineer.
- Preprocess the log files.
- Identify issues, and match them with an existing incident in the incident list.
- Assess further data transformation, and build a knowledge base.
- Build an AI agent and connect it with the knowledge base.
- Assess weak points of the AI agent, optimize them, and evaluate its performance.

## 4.1 RCA agent version 1.0

In order to build and evaluate Alstom's RCA agent, following distinct steps are taken:

- **Step 1: Data acquisition.** The dataset consists mostly of data provided by an RCA expert from Alstom; some additional documents were provided by researchers close to this company.
- **Step 2: Data preprocessing.** The dataset needs to be preprocessed across all its characteristics.
- **Step 3: Incident matching.** The incidents need to be identified, as the dataset does not contain any explicit cases. Once incidents are identified, they can be found in the stakeholder's list of all past incidents.
- **Step 4: Identifying ground truth for evaluation.** The matched description of each incident can then be used for various purposes, including evaluation.
- **Step 5: Identifying incident related log files.** Furthermore, the output of Step 4 can help assess the difference between the reported time and the actual time of the incident. This difference can be used to select an appropriate time window for relevant data.

- **Step 6: Chunking.** Once the relevant documents are assessed, they can be prepared for the knowledge base, breaking them into smaller chunks.
- **Step 7: Knowledge base.** These chunks are further stored in a database for retrieval.
- **Step 8: Retrieval.** The retriever is an important part of the pipeline, and its performance in the pipeline will be measured.
- **Step 9: LLM agent.** An appropriate LLM is selected and given access to the database, allowing it to retrieve relevant information. Furthermore, the AI agent will work in an iterative process, mimicking the RCA process.
- **Step 10: Context management of retrieved data.** Considering the iterative character of the RCA, the amount of retrieved context can be overwhelming even for modern LLMs; it is therefore important to manage the context in a way that keeps the relevant information while discarding the irrelevant.
- **Step 11: Evaluation.** The goal is to build an RCA agent; its output will be evaluated and compared with the ground truth obtained in Step 4.

Figure 2 shows these steps.

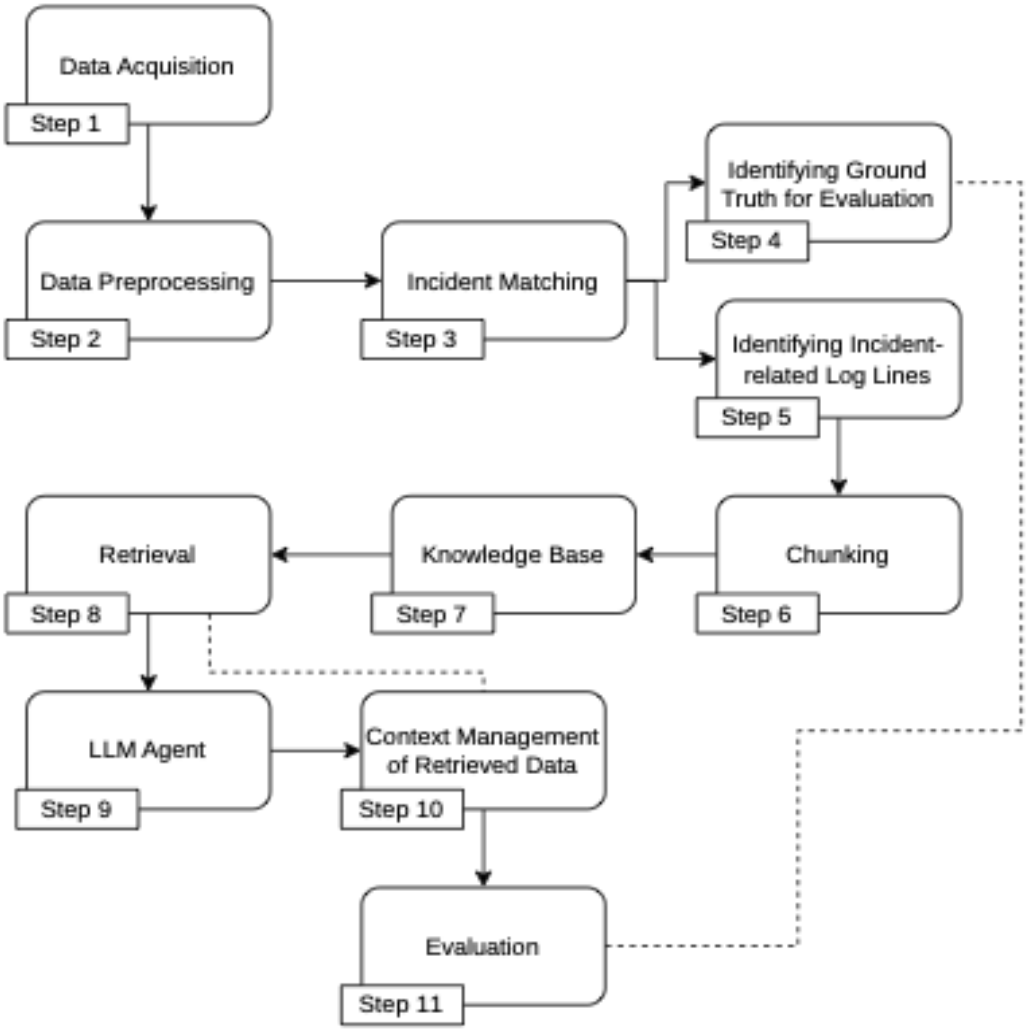


**Figure 2.** ***Process of building and evaluating an AI agent for RCA of Alstom's dataset.***

# 5. Eli Alps - Hybrid Neuro-Symbolic Log Anomaly Detector

It is a fully local log anomaly detector built on Google ADK. It narrows a large log file down to a handful of suspect rows with cheap, deterministic detectors, then has local LLMs (via Ollama) that explain **only** the small context windows around those rows. The result is a structured anomaly report — with an overall score, the affected log lines, root-cause analysis, and recommended actions — produced without sending your logs to any cloud service.

The whole point is **cost control**: LLM token usage stays proportional to the number of flagged anomalies (**O(k)**) instead of the total log size (**O(n)**), so it runs on consumer hardware.

## 5.1 How it works at a glance

CSV → PreFilterAgent (5 deterministic detectors) → context windows
→ 3 parallel LLM analyzers → SummarizerAgent → AnomalyReport JSON

A `SequentialAgent` runs three stages: a no-LLM **PreFilterAgent** (statistical/symbolic detectors + context extraction), a **ParallelAnalyzerAgent** (three concurrent LLM agents: per-line, multi-line patterns, knowledge-base rules), and a **SummarizerAgent** that synthesizes the final report. Agents communicate only through session state.

### 5.1.1 Diagram

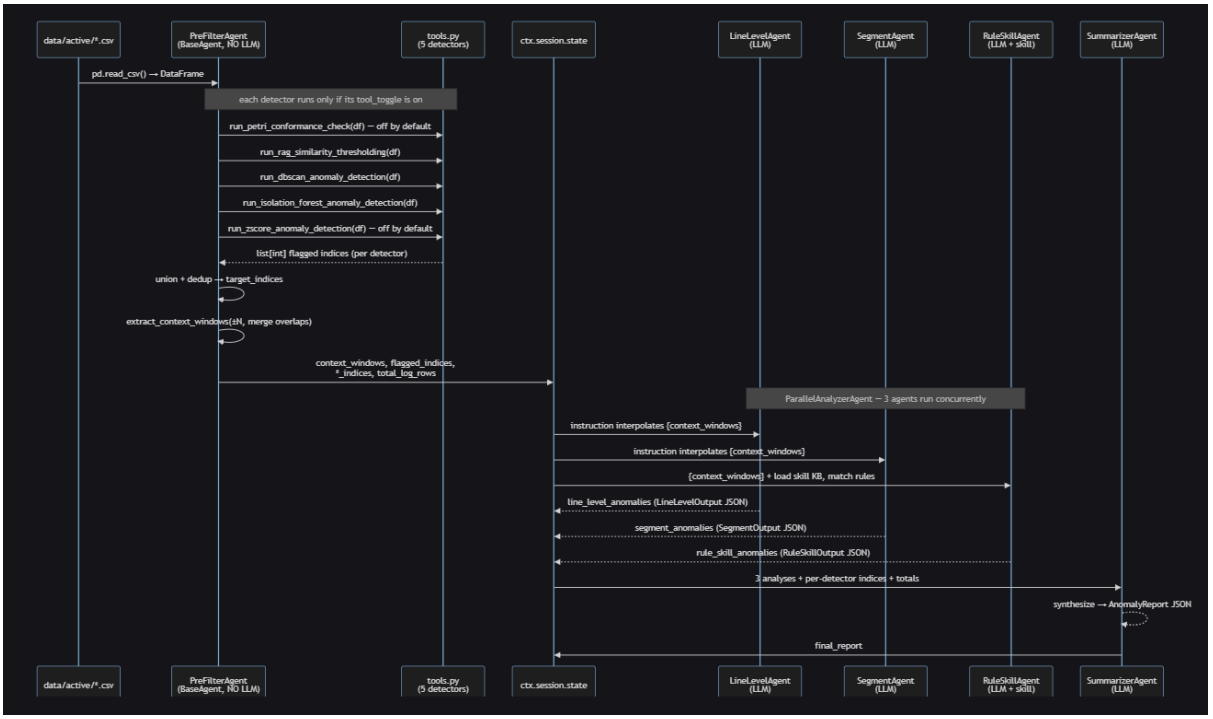


**Figure 3.** ***Log anomaly detector process for Eli Alps.***

# 6. Bilecik Demircelik

Bilecik Demircelik is a use case provider in MONA LISA for induction furnace monitoring and analysis. The use case focuses on processing operational data collected from furnace systems in order to detect abnormal operating behaviour and support root cause analysis of fault conditions.

The analysis mainly uses SCADA/PLC time-series data, operator-entered downtime records, fault indicators, and related operational logs. Instead of analysing all available data manually, the proposed approach aims to select the most relevant signals, time intervals, and log entries for each analysis task. This reduces the amount of data to be reviewed and provides engineers with a clearer view of furnace behaviour.

## 6.1 Anomaly Detection Classification

The Bilecik Demircelik use case focuses on detecting abnormal operating patterns in induction furnace data. The anomaly detection approach will process high-frequency SCADA and PLC signals collected from the furnace infrastructure and classify operating behaviour into normal and abnormal categories. The main objective is to reduce large volumes of time-series data into clearly identified abnormal operating states, affected signals, and anomaly classes that can be further examined by engineers.

The analysis will target general abnormal operating conditions that may indicate degradation, instability, or unsafe behaviour in furnace operations. These abnormal patterns are expected to appear through changes or unusual combinations in water flow, cooling water inlet and outlet temperatures, phase currents, phase voltages, power consumption values, ground leakage indicators, and operational fault flags. Since the furnace generates a large number of signal tags, the anomaly detection process will focus on identifying patterns in the most relevant operational and safety-critical parameters.

SCADA/PLC time-series data → preprocessing and synchronization → feature extraction → classification of operating states → anomaly class assignment → anomaly classification report

The process starts by acquiring historical and real-time sensor data from the MongoDB environment. The raw signals are then cleaned, synchronized, and transformed into a common time-series format. Missing values, inconsistent timestamps, duplicate records, and abnormal sensor readings caused by data acquisition problems will be handled during preprocessing.

After preprocessing, classification-based anomaly detection algorithms analyse the selected furnace signals to determine whether the observed behaviour corresponds to normal operation or to a predefined anomaly class. These may include sudden spikes, unexpected drops, unstable fluctuations, threshold violations, unusual thermal gradients, abnormal electrical imbalances, or

combinations of signals that deviate from normal operating behaviour. For example, an abnormal increase in cooling water temperature combined with changes in electrical parameters may indicate a potential operational anomaly.

When an abnormal pattern is identified, the system extracts a context window around the relevant time interval. This window includes the affected sensor values before, during, and after the anomaly. The purpose of this step is to provide engineers with the operational context surrounding the classified abnormal behaviour rather than analysing isolated data points.

Each detected abnormal pattern is then assigned to an anomaly class based on its signal characteristics and relation to known abnormal behaviour patterns. The output is structured as an anomaly classification report containing the affected furnace, timestamp, suspicious signal group, anomaly class, and the relevant context window. This report will support engineers in understanding the type of abnormal behaviour.

The expected output of the anomaly detection component is a structured report that includes:

- anomaly classification
- timestamp or time interval
- affected furnace or equipment unit
- affected signal names
- anomaly type

This anomaly detection classification layer will reduce the amount of data that needs to be manually reviewed and will provide a focused starting point for further analysis. The detected abnormal operating states and affected signals will also be transferred to the traceability and visualization environment, allowing engineers to inspect anomalies together with operational logs and other process data on a synchronized timeline.

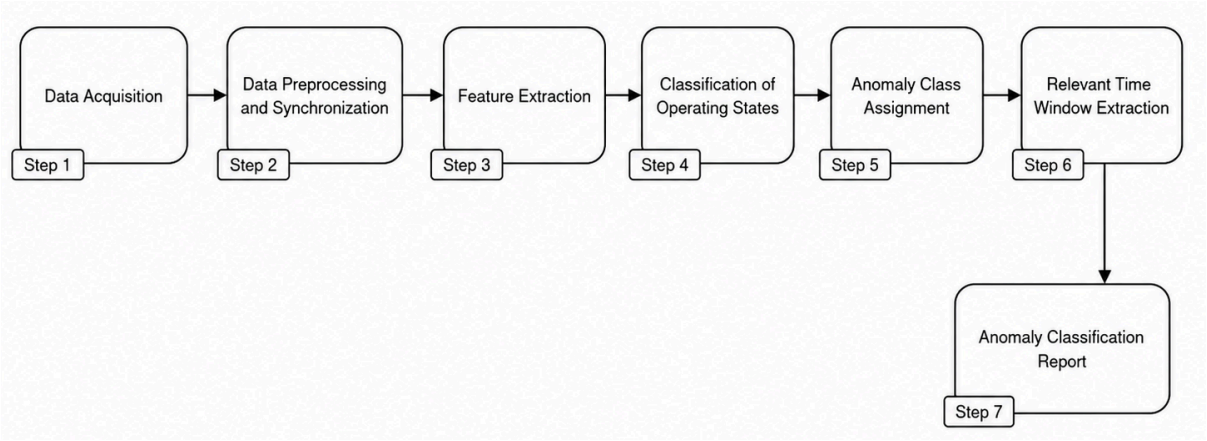


**Figure 4.** ***Anomaly detection process for BDC.***

## 6.2 Root Cause Analysis

The Root Cause Analysis component for the Bilecik Demircelik use case will focus on explaining the possible causes of furnace faults by analysing operational failure logs, fault indicators, sensor trends, and related process data.

The main objective of the RCA component is to support engineers in understanding the sequence of events that led to a fault condition and to identify the most probable underlying causes. The analysis will use relevant time windows, affected signal groups, operator-entered downtime records, and furnace fault information to explain the observed behaviour rather than directly linking it to predefined fault categories.

Operational data selection → operational log matching → relevant signal and event selection → sequence of events reconstruction → root cause hypothesis generation → RCA report

The process starts with the selection of relevant operational data only for performing fault analysis and understanding the root cause. This includes information such as affected furnace, timestamp or time interval, and related signal names. Based on this information, the RCA component searches for related operational logs and fault records within the same or nearby time interval.

After matching the operational data with logs, the system analyses the selected sensor signals and event records together only to perform fault analysis and understand the root cause. The purpose is to identify whether the behaviour was associated with thermal, hydraulic, electrical, or operational indicators. For example, a temperature increase, unstable water flow, electrical imbalance, or ground leakage signal may provide evidence for a specific failure mechanism.

The RCA component then reconstructs the sequence of events around the operating condition only to perform fault analysis and understand the root cause. This includes identifying which signals changed first, which fault indicators appeared later, and whether any operator-entered downtime record confirms the observed behaviour. This step helps distinguish between the visible fault and the possible underlying cause.

Based on the matched evidence, the system generates a root cause hypothesis only to perform fault analysis and understand the root cause. The hypothesis may indicate whether the behaviour is more likely related to refractory degradation, earth leakage, Code 39-related electrical instability, cooling system issues, or another undefined operational problem.

The expected output of the RCA component is a structured report that includes:

- analysed operating condition,
- timestamp or time interval,
- affected furnace or equipment unit,
- related operational logs,
- affected signal groups,

- reconstructed event sequence,
- possible root cause,
- supporting evidence from sensor and log data.

This RCA layer will provide engineers with a structured explanation only to perform fault analysis and understand the root cause. Instead of reviewing sensor trends and operational logs separately, engineers will be able to examine the detected operating condition, related events, and possible root cause in a unified analysis output. The RCA results will also be transferred to the traceability and visualization environment, where they can be inspected together with signal trends and operational records on a synchronized timeline.

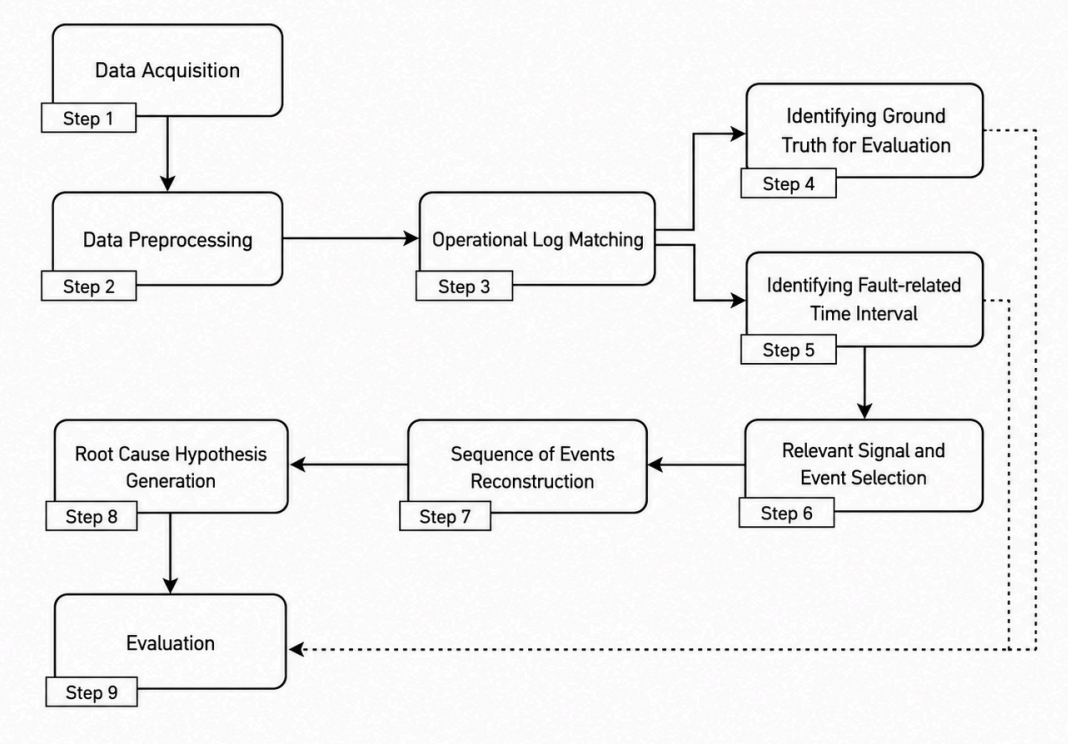


**Figure 5.** ***RCA process for BDC.***

# 7. CNET

## 7.1 Use Case

CNet is a use case provider in MONA LISA for structural health monitoring. This employs a digital twin (shadow) for ingesting, storing and analysing large amounts of high-frequency data, the core of which is the PYST SHM Engine. The structural health monitoring, condition assessment and predictive maintenance for critical infrastructures such as bridges will in this use case address enhancements in traceability and maintainability through all layers by using MONA LISA tools and methodologies.

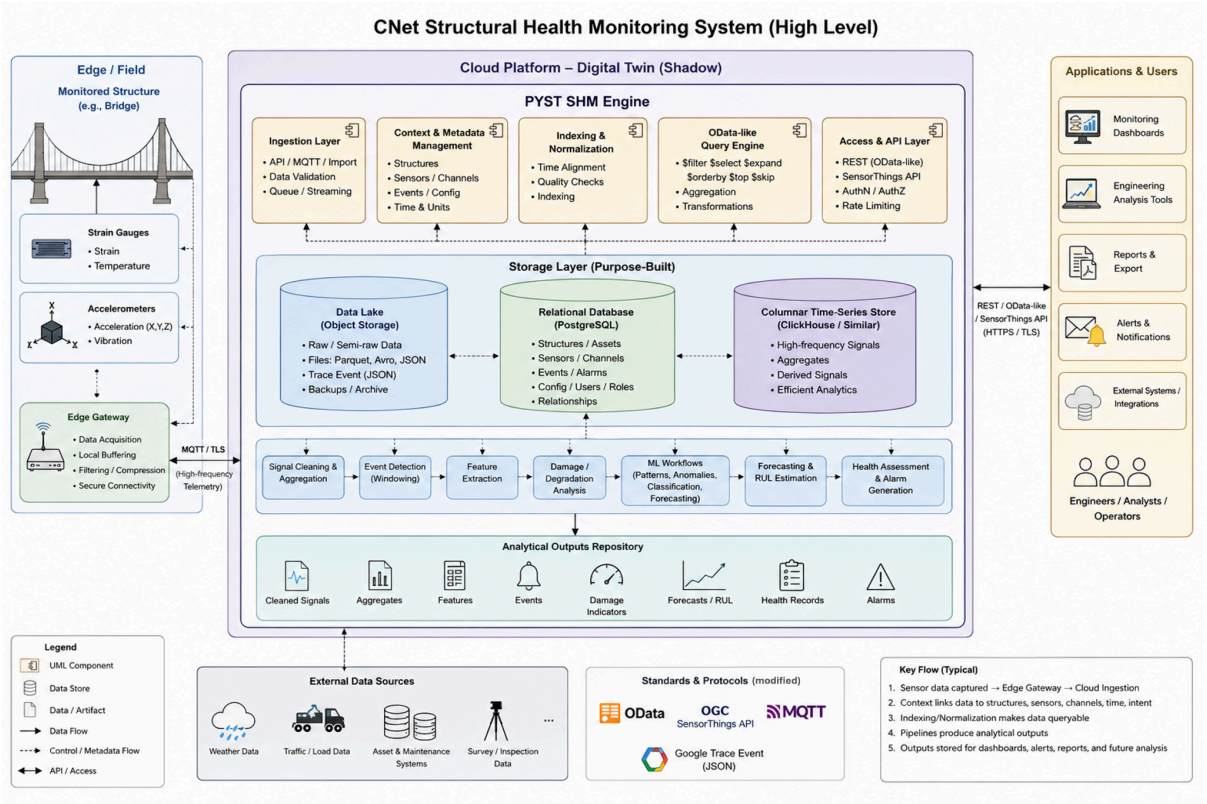


**Figure 6.** ***High level view of CNET's structural health monitoring system.***

Within the use case, MONA LISA will cover the areas of (1) structural health monitoring and (2) design-time and real-time monitoring in a distributed architecture. In Area 1, structural health monitoring, MONA LISA will improve estimation of remaining useful life (RUL) of equipment and optimise needed maintenance. Area 2 concerns design-time and real-time monitoring of functionality in a distributed architecture, which often is remapped between heterogeneous computing resources including special-purpose hardware. In this context, MONA LISA helps the designer by visualising the coupling between failures in the hardware, through software to a safety impact on the system. The goals of the use case are:

- Reduce development and deployment cost and time through unified tools, improved traceability and visual analytics
- Improve visualisation techniques for complex, interdependent sensor data
- Improve analytics, including AI-based models for structural health and predictive maintenance
- Support lifecycle management to reduce technical debt and maintain old bridges longer

## 7.1 PYST SHM Engine and Monitoring Overview

PYST is a Structural Health Monitoring (SHM) engine for turning raw sensor data into engineering analytics. It combines data lake storage, relational databases, columnar time-series storage, an OData-like custom query engine, processing pipelines, and machine-learning workflows.

The goal is to refine measurements into useful outputs: cleaned signals, aggregates, extracted features, detected events, damage indicators, forecasts, health records, and alarms.

### 7.1.1 What PYST Does

At a high level, PYST:

- ingests sensor data and related metadata from connected systems and data sources;
- keeps structures, sensors, channels, events, and configuration context queryable;
- stores large signal payloads in formats suited for high-volume time-series work;
- provides an OData-like query layer for filtering, selecting, expanding, aggregating, and transforming SHM data;
- runs pipelines for aggregation, feature extraction, event detection, damage analysis, forecasting, and alarm generation;
- supports machine-learning workflows for pattern discovery, classification, anomaly detection, forecasting, and model-assisted interpretation.

### 7.1.2 Data and Analytics Flow

PYST uses different storage layers for different jobs. Data lakes hold raw or semi-raw sensor material. Relational databases hold context and relationships. Columnar storage supports efficient work on dense time-series and derived signals.

A typical refinement flow is:

1. Sensor data lands in the system through data lake, API, import, or queue-based paths.
2. Context links the data to structures, sensors, channels, time ranges, and analysis intent.
3. PYST indexes or normalizes the data so it can be queried efficiently.
4. Pipelines turn raw and intermediate data into analytical outputs.
5. Those outputs are stored for review, monitoring, dashboards, and later analysis.

### 7.1.3 Query and Access Layer

PYST includes an OData-like (https://www.odata.org/) custom query engine designed for SHM data. It gives applications a consistent way to work across relational context, sensor streams, columnar data, larger stored payloads, and derived analytical records.

### 7.1.4 Analytics Supported

PYST supports SHM analytics such as signal cleaning, aggregation, event-window processing, feature extraction, ML-assisted pattern recognition, anomaly detection, damage and degradation indicators, forecasts, remaining-useful-life style outputs, and health-based alarms.

### 7.1.5 Data Formats

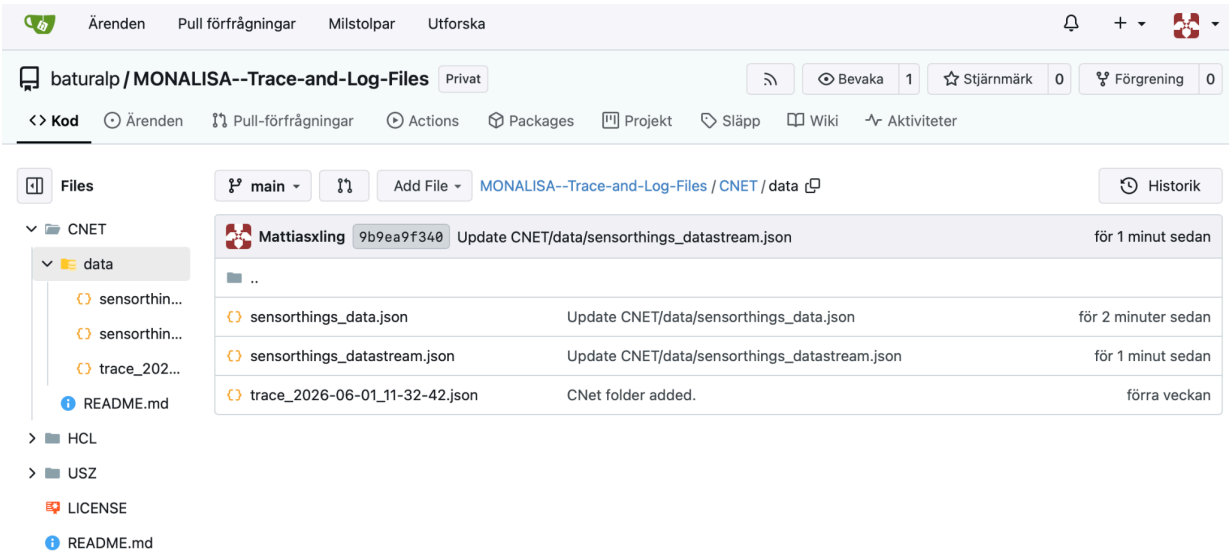


**Figure 7.** ***CNET's folder structure in Mona Lisa repository.***

The data format and API are based on OData and the OGC SensorThings API (https://www.ogc.org/standards/sensorthings/), with modifications.

Examples for datastreams and data can be found in the https://git.monalisaio.com/baturalp/MONALISA--Trace-and-Log-Files/src/branch/main/CNET/data folder together with example Google Trace Event JSON for the sensor logs.

Live Google Trace Event data from a test installation has been made available at an MQTT endpoint at (trapp.iotbridge.se). Contact CNet for connection details.