MONICO DATA ANALYSIS

Digvijay Alluri
Nicho Naugle
Michael Cubriel

MACHINE LEARNING PREVENTATIVE MAINTENANCE FINAL REPORT

REVISION 1

Table of Contents

Execution Plan

Name	Owner	9/22	9/29	10/6	10/13	10/20	10/27	11/3	11/10	11/17	11/24	12/1	12/8
Concept of Operations	All												
Functional System Requirements	All												
Interface Control Document	All												
Milestones and Validation Plan	All												
Midterm Presentation	All												
Subsystem Breakdown	All												
Complete Subsystem Introduction Projects	All												
Designed Neural Network with Defined API parameters	Michael												
Complete mDNS and Websockets server	Nicho												
Data Reformatting with definied API parameters	Digvijay												
Progress Update 1	All												
Interface scripts for sending formatted data to the neural network for training	Michael												
Preliminary API commands and asynchronous client management	Nicho												
Training the model	Michael												
Start model feedback framework for API	Digvijay												
Progress Update 2	All												
Modify Model using first training run	Michael												
Synchronize API with data management for processing	Digvijay												
Finish API commands and link with data management codebase	Nicho												
Retrain model and optimize Reccurrent nodes	Michael												
Progress Update 3	All												
Data storage management script	Digvijay												
Create NN model output integration with API websockets server	Michael												
Improve prediction accuracy	Michael												
Complete API and Networking Connections with task management	Nicho												
Test Max client count and live streaming	Nicho												
Final Presentation	All												
Work on final presentation	All												
Final validation checks for all individual subsystems	All												
Finish Final Presentation Preparation	All												
Work on Subsystem Presentations	All												
Work on report	All												
Finish Subsystem Presentations	All												
Subsystem Presentations	All												
Finish Final Report	All												
Final Report	All												

MONICO DATA ANALYSIS

Digvijay Alluri
Nicho Naugle
Michael Cubriel

CONCEPT OF OPERATIONS

REVISION – 1

CONCEPT OF OPERATIONS
FOR
Monico Data Analysis

TEAM 41

APPROVED BY:

_____
Project Leader Date

_____
Prof. Kalafatis Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1.0	9/14/2024	Entire Team	PASS	Draft Release

Table of Contents

List of Tables

No table of figures entries found.

List of Figures

1. Executive Summary

The project sponsor, Monico, Inc., has asked that edge computing and machine learning technologies be used to improve heavy machinery maintenance and monitor conditions. The current challenges include rules-based analytics, necessitating ongoing care from internal machinery subject matter experts (SMEs). On the other hand, standard cloud-based machine learning solutions are hampered by inadequate internet access and maintenance resources. This project will create an Edge application for Monico's mCoreSDR to solve these problems. This application will use edge computing power to run ML models which will help with predictive maintenance. Moreover, a middleware program will be installed utilizing virtual machine technology on a safe cloud server via the internet or within a client's private network. For Monico, this strategy will optimize resource utilization and boost machine dependability.

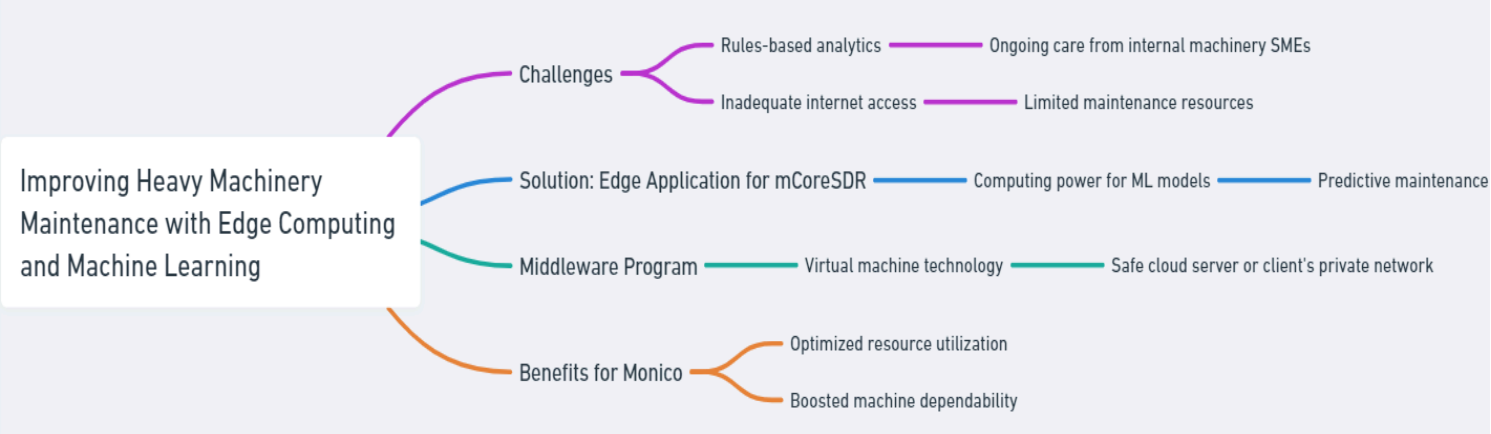


Figure 1: Summary of the entire project

2. Introduction

This document describes the creation of Edge and Middleware applications for Monico's mCoreSDR, a computer used for data collecting in heavy industrial machinery including wheel loaders, dozers, and mine haul trucks, is covered in this article. This project uses sophisticated AI/ML models on a local server to enhance equipment maintenance. With the help of this technology, real-time monitoring and predictive maintenance are made possible without the need for continual human supervision or cloud connectivity, as opposed to the current reliance on manual, rule-based analytics and restricted network access.

2.1. Background

Monico provides services to companies that have high-risk machinery, like mining and drilling equipment, generators, compressors, and generators. Predefined phrases have a

major role in the existing system's ability to identify possible problems. This system has a low degree of adaptability to changing operating conditions and relies heavily on manual activities. In addition, a lot of industrial locations have problems accessing the cloud-based monitoring system and encounter limitations when trying to connect to the internet.

Monico's computer, the mCoreSDR, already has data-collecting capabilities, but it is not equipped with sophisticated machine-learning algorithms for demanding situations. This project presents a novel application to improve machine learning by utilizing the local processing capacity of McoreSDR. The objective is to overcome the drawbacks of static rules-based systems as previously mentioned and instead use real-time data to train the AI model and inform decision-making.

As a result, the model must be in a server application that may run in a cloud server via virtual machine technology and manage numerous mCoreSDR apps. In addition to increasing operating efficiency and adhering to stringent regulatory requirements, this will take the role of manual intervention. The AI model will predict when a machine will fail, and this research is essential to satisfy the increasing demand for more autonomous, efficient monitoring systems where uptime and dependability are crucial.

2.2. Overview

There is software in place now, but it depends too much on limitations like networks, human verification, and rule-based analytics. With the use of our machine learning algorithm, which processes real-time data to determine whether a machine may fail in the future, our solution will assist Monico in making future scenario predictions.

2.3. Referenced Documents and Standards

AI Model Development and Training:

1. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8830986/
2. https://ieeexplore.ieee.org/document/10183012
3. https://medium.com/@sanjay_dutta/understanding-the-number-of-hidden-layers-in-neural-networks-a-comprehensive-guide-0c3bc8a5dc5d
4. https://discord.com/channels/@me/1278081852896182293/1284671020934103052

Networking for IoT and Edge Computing:

1. https://www.researchgate.net/publication/378321885_Edge_Computing_for_IoT
2. https://www.sciencedirect.com/science/article/pii/S2590123024001282
3. https://www.researchgate.net/publication/310463974_IoT_Networking_Technologies_and_Research_Challenges

Pre-Conditional Data Filtering for Neural Networking Training:

1. https://arxiv.org/abs/2011.10231
2. https://www.sciencedirect.com/science/article/abs/pii/S0925231219309981
3. https://arxiv.org/abs/2403.07965

3. Operating Concept

3.1. Scope

The scope of the project is to develop an edge computing and machine learning solution for predictive maintenance of reciprocating compressors powered by Caterpillar G3612 A3 natural gas engines. The project will utilize Monico's mCoreSDR device to process real-time data from 176 key data points related to the engine, compressor, and supporting systems. This data will be used to transition from a rules-based maintenance system to a machine learning-driven model, providing insights into potential machine failures and reducing the need for human oversight. The data dashboard can also be accessed locally on the network through Websockets for any live data dashboard configuration

3.2. Operational Description and Constraints

To enable on-site data analysis without continual supervision, the Monico Data Analysis System will be deployed on Monico's mCoreSDR devices that are mounted on machinery. By spotting possible problems before they become failures, Monico's approach enables it to provide its clients with real-time insights, enabling them to maintain equipment. By doing this, Monico and its clients' operational expenses are decreased as well as the reliance on SMEs is decreased.
The possible constraints are:

1) **Hardware limitations:** Due to the mCore SDR's limited computational power, the machine learning method needs to be tailored to the device's capabilities. Strict control over processing speed and memory utilization is needed to guarantee that the program runs effectively.
2) **Data Quality:** The completeness and quality of the historical data that is accessible determines how effective the algorithm is. Its ability to make predictions could be hampered by inconsistent data.
3) **Connectivity Restrictions:** Considering network circumstances can change, the application must allow secure data transfer and not rely on high-speed bandwidth. Moreover, the local network must support mDNS discovery protocols and WebSockets support

3.3. System Description

Our system is broken into three fundamental pieces all wrapped into one package called predictAI. The three parts are data classification and training, tensorflow model, and API. The API will be exposed to the end user and act as an intermediary for any local dashboard/GUI to send commands and stream real-time sensor data.

Data Classification and Testing

The process starts with inputting a CSV file containing 186 variables and more than 60 data points every minute, around the clock. Data will be averaged out into a single node every minute and fed back into a CSV file for storage or future fetching through the API

TensorFlow
Each of the 186 input nodes will be trained on the historical compressor data and will predict when a machine will fail based on the past machine failures trained into the neural network. The result will be a single node with an estimation on when the overall system failure will occur (in hours).

API
The output from a single node will be sent over a Secured Websockets connection to a Front-end data dashboard, allowing users to access real-time user diagnostics and a gateway for contributing data to the model from other machines.

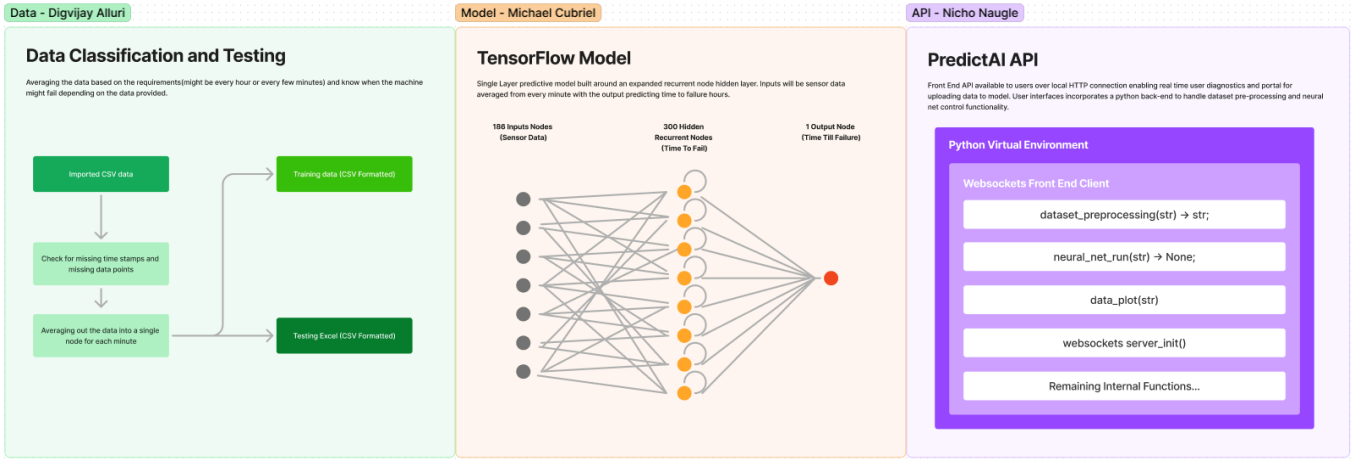


Figure 2: Block Diagram for data analysis

3.4. Modes of Operations

The system will operate in two modes to support both real-time monitoring/prediction and the ability to periodically train and update the model to increase accuracy. Both modes of operation can easily be triggered by the end client through a customized data dashboard that is connected over WebSockets.

1. Normal Operation Mode: In this mode, the mCoreSDR device continuously collects data from 176 data points across the engine. The system processes this data locally using a machine-learning model designed to predict potential failures before they occur. During normal operation, the mCoreSDR device autonomously handles data processing and analysis without needing internet connectivity or direct human supervision. In this mode, the

system will also provide real-time diagnostic feedback to the front-end API, allowing users to monitor the health of the compressor via a UI.

2. Model Training and Update Mode: In this mode, reiterative training of the machine learning model is initiated through the API. The system processes historical data from stored CSV to update the model's ability to make accurate predictions. In this mode, compressed historical data is used to test and update the model's capability. Once the updated model is ready, it is deployed back to the mCoreSDR device to resume normal operation, using the latest model for real-time predictions.

3.5. Users

The system will need to be implemented as an application on the local edge device during the development phase of the software. Given the nature of the software package, it can be very easily implemented into Linux, Windows, or MacOS environments through the use of a python virtual environment. Once instantiated by following the setup guide, the user will only have to set up the network interface, whether it be wifi or ethernet.

A data dashboard will also be monitored by maintenance professionals in the field who want to help avoid failures and ensure that the equipment remains operational as much as possible. Through this setup, very little intervention is required from non-technical users, past the initial development of the hardware compute module (mCore SDR) and its respective software which should be completed before the final product is delivered to be used in the field.

3.6. Support

Support for users of the system would be provided through several channels, ensuring both technical and non-technical users can develop and maintain the software effectively.

1. Setup Guide: A comprehensive manual will be provided to guide users through the initial setup of the software, focusing on instantiating the Python virtual environment and configuring the network interface (Wi-Fi or Ethernet). This guide would include step-by-step instructions and troubleshooting tips to ensure the installation process is as smooth as possible.

2. Hardware Installation Manual: Since the system is deployed on the mCore SDR hardware compute module, a detailed guide will cover the installation of the hardware components, basic configuration, and connection to the local edge device.

3. Data Dashboard Manual: A dedicated user guide will explain how to monitor and interpret the data dashboard for maintenance professionals. This manual will focus on avoiding equipment failures and ensuring operational efficiency, providing simple instructions on what metrics to watch and how to act on various alerts or warnings. Moreover, the dashboard will include built-in help “buttons” to describe functionality when prompted.

4. Scenario(s)

4.1. Preventive Maintenance for Sensor Based Machinery

The primary objective of this project is to develop an application that can be deployed to an edge computing device, incorporating a low-footprint neural network designed to process incoming data and continuously retrain itself when instructed to do so. This retraining enables the system to make predictive assessments regarding potential machinery failures based on specified sensor parameters. Our neural network is specifically optimized for the sensor data being utilized in the Monico Compressor application, to ensure high accuracy in its predictions. By leveraging this predictive capability, the system maximizes the operational lifespan of a field-deployed compressor engine, ensuring maintenance is performed only when necessary. This approach minimizes unnecessary downtime and reduces overall maintenance costs, optimizing resource allocation and operational efficiency.

5. Analysis

5.1. Summary of Proposed Improvements

The proposed system will include several notable improvements over the current rules-based analytics system and will reduce the need for human intervention.

1. **Automated Predictive Maintenance:** By using rules-based analytics as the foundation for the machine learning model, the proposed system will detect patterns in real-time data that indicate potential failures. This approach will reduce maintenance costs and help prevent unplanned downtime.

2. **Real Time Monitoring:** The edge computing capabilities of the mCoreSDR will allow real-time data processing locally. This eliminates the need for an internet connection and will provide a more secure and reliable environment.

3. **Scalability:** The system will utilize a vast historical dataset containing over 5.5 billion data points to test and train the machine learning model. This large volume of data ensures that the system can adapt to changing conditions. Additionally, the system supports the ability to retrain the model iteratively based on new data, further increasing accuracy and enabling deployment across various environments.

5.2. Disadvantages and Limitations

While our proposed system offers many advantages in efficiency and cost, there are certain disadvantages that should be taken into consideration.

1. **Hardware limitations:** The mCoreSDR device, while offering edge computing capabilities, has limited computational power compared to cloud-based solutions. The machine learning model will need to be optimized to run within the mCoreSDR's processing and memory constraints and should ideally be a lightweight program.

2. **Data Processing and Quality**: Our machine-learning model will rely heavily on the quality and completeness of the data provided. Inconsistent or missing data can lead to inaccurate predictions or system failures. Additionally, due to the size of the dataset, significant computational power will be required to process it effectively.

5.3. Alternatives

The AI model leverages local edge computing and machine learning capabilities to reduce reliance on cloud-based connectivity while providing accurate and insightful predictions. However, a fully cloud-based system could offer greater computational power and faster model training. Additionally, data would be easier to access and manage in the cloud, though this comes with potential security risks. Another alternative currently in place is the rules-based analytics system, which relies on predefined thresholds for detecting failures but lacks flexibility and requires human oversight to function effectively.

5.4. Impact

The system will have minimal direct environmental impacts because of the nature of the system being almost primarily software. However by optimizing machine performance by extending equipment lifespan, it is in turn contributing to more sustainable industry practices. Furthermore, it also has the potential to increase worker safety by predicting failures before they occur, leading to a safer work environment. The main ethical concern is keeping Monico's proprietary information safe and secure. Our system will be carefully designed to prevent unauthorized access or misuse of data.

MONICO DATA ANALYSIS

Digvijay Alluri
Nicho Naugle
Michael Cubriel

FUNCTIONAL SYSTEM REQUIREMENTS

REVISION – 2

24 September 2024

FUNCTIONAL SYSTEM REQUIREMENTS

FOR

Monico Data Analysis

PREPARED BY:

_____

Author Date

APPROVED BY:

_____

Project Leader Date

_____

John Lusher, P.E. Date

_____

T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1.0	9/14/2024	Monico Data Analysis	PASS	Draft Release
2.0	9/24/2024	Monico Data Analysis	PASS	Revision 1

Table of Contents

List of Tables

List of Figures

6. Introduction

6.1. Purpose and Scope

The goal of this project is to create Edge and Middleware applications that will improve Monico, Inc.'s capacity for heavy machinery condition monitoring and predictive maintenance. These models will not require continual human intervention or cloud infrastructure; instead, they will analyze data in real-time and provide actionable insights. Using a virtual machine environment, the Middleware component will safely handle data transfers, model updates, and system performance monitoring across the Internet or a client's private network. The range of work encompasses integrating machine learning algorithms designed for preventive maintenance, deploying systems on edge devices, and creating safe connections to middleware housed on private or cloud servers. In addition, the project will prioritize maximizing resource use, guaranteeing flawless real-time data processing, and improving machine dependability by cutting down on maintenance requirements and downtime. The technical specifications outlined in this specification are required to provide a scalable and effective solution for large industrial machinery.

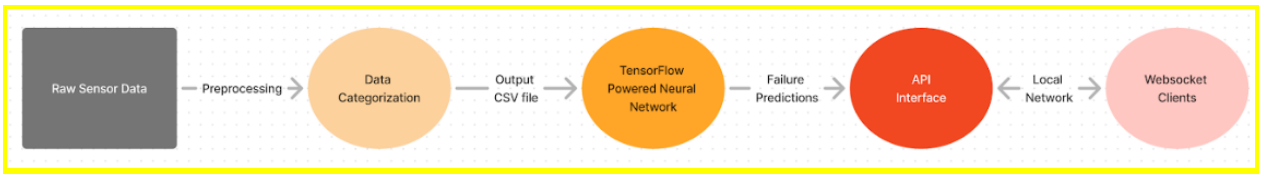


Figure 1. Your Project Conceptual Image

6.2. Responsibility and Change Authority

The project leader, Nicho Naugle, and Professor John Lusher alone have the power to make modifications, and the team leader will be in charge of ensuring that all requirements are fulfilled.

Subsystem	**Responsibility**
Data Classification	Digvijay Alluri
Neural Network	Michael Cubriel

API	Nicho Naugle

Table 1: Subsystem and Responsibility

7. Applicable and Reference Documents

7.1. Applicable Documents

The following documents, of the exact issue and revision shown, form a part of this specification to the extent specified herein:

Document Number	Revision/Release Date	Document Title
10.1007/s42979-022-01043-x	02/10/2022	AI-Based Modeling: Techniques, Application and Research Issues Towards Automation, Intelligent and Smart Systems
0c3bc8a5dc5d	06/05/2024	Understanding the Number of Hidden Layers in Neural Networks: A Comprehensive Guide
10.5120/ijca2016912181	11/2016	IoT: Networking Technologies and Research Challenges
101875	03/2024	An open source IoT edge-computing system for monitoring energy consumption in buildings

7.2. Reference Documents

The following documents are reference documents utilized in the development of this specification. These documents do not form a part of this specification and are not controlled by their reference herein.

Document Number	Revision/Release Date	Document Title
979-8-3503-9926-4	2023	Predictive Maintenance using Machine Learning: A Case Study in Manufacturing Management
2019.06.084	11/06/2019	Unsupervised pre-trained filter learning approach for efficient convolution neural network

7.3. Order of Precedence

In the event of a conflict between the text of this specification and an applicable document cited herein, the text of this specification takes precedence without any exceptions.

All specifications, standards, exhibits, drawings, or other documents that are invoked as “applicable” in this specification are incorporated as cited. All documents that are referred to within an applicable report are considered to be for guidance and information only, except ICDs that have their relevant documents considered to be incorporated as cited.

8. Requirements

This section defines the minimum requirements that the development item(s) must meet. The requirements and constraints that apply to performance, design, interoperability, reliability, etc., of the system, are covered.

8.1. System Definition

Providing a predictive maintenance and status monitoring system for heavy machinery is the aim of the project. This approach leverages machine learning and real-time data to overcome the limitations of existing rule-based analytics and unnecessary human intervention. This project aims to reduce downtime, increase operational efficiency, and lower the amount of human intervention needed for maintenance tasks by giving Monico an intelligent machine monitoring and breakdown prediction system.

The data categorization subsystem, the TensorFlow-powered neural network, and the previously mentioned API will be the three primary subsystems. The first file the system receives is a CSV file with thousands of rows and columns. After averaging the data into one node every minute or five minutes, etc., it is processed to look for errors by catching blocks. The data is then saved in a CSV file before moving on to the next subsection. By combining all of the nodes into a single node based on training on past data, the neural network will aid in the simplification of the data. This model uses a tensor flow model to anticipate the timing of machinery failures. After that, the data and output of the model will be sent to an API, enabling interaction with a front-end dashboard for user management and monitoring.

The subsystems interconnect through a seamless flow of data: raw sensor data is sent into the classification system, which prepares the data for model training and real-time monitoring, allowing the subsystems to be seamlessly connected. Failure predictions are generated by the TensorFlow model and sent to the user interface via the API for useful insights.

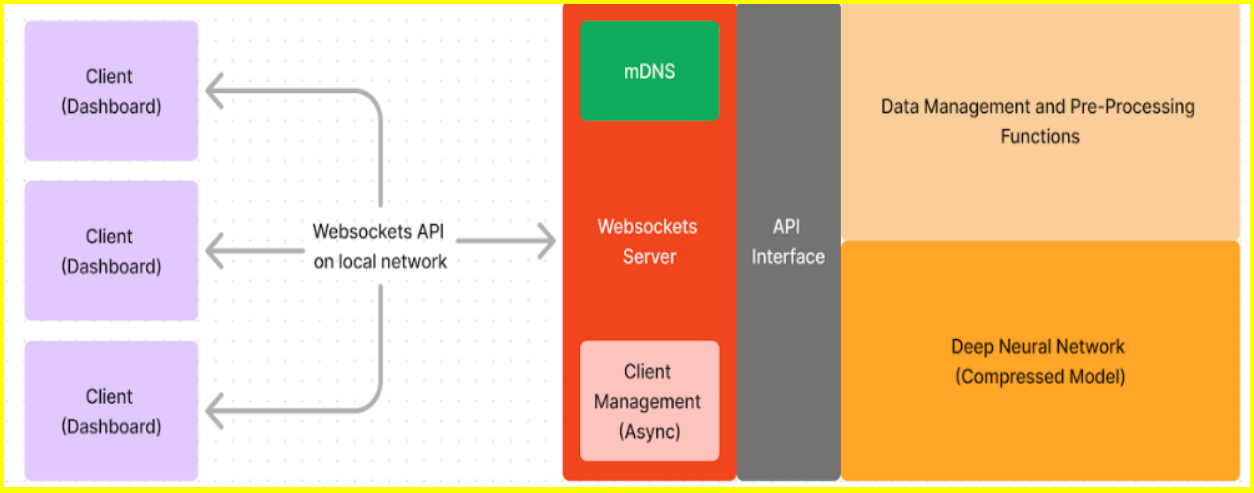


Figure 2. Block Diagram of System

8.2. Characteristics

8.2.1. Functional / Performance Requirements

Real-Time Data Ingestion:

- Every minute, sensor data from heavy machinery is ingested, including 186-250 variables for each machine.

Data Classification:

- Create single-node averages from sensor data once every minute, or as often as the user requests, and save the results in CSV format for model training.

TensorFlow Model:

- Predicts machine failure in hours with ±10% accuracy using 186-250 input nodes trained from historical data.

Edge Device Compatibility:

- Utilize the mCoreSDR device from Monico to operate with a latency of less than 1 second for forecasts, all without depending on the cloud.

Predictive Maintenance:

- Support scheduling maintenance based on machine learning insights with a 6-12 hour failure prediction window.

Scalability:

- Utilize dynamic machine management to manage data from up to 50 machines at once.

User Interface Integration:

- Integrate the dashboard seamlessly and refresh the data at least once every minute in real-time.

Data sampling frequency: 1-minute intervals

Failure prediction: Hours before machine failure

Data streams: 186-250 variables per machine

Predictive window: 6-12 hours before failure

Search Probability of Detection

The system is expected to achieve a 90% accuracy threshold in predicting machinery failure within a ±5% tolerance under typical operating conditions.

Rationale: This is the fundamental need for system performance. These circumstances include machine usage patterns, sensor data fluctuation, and real-time data intake from equipment like wheel loaders, dozers, and mine transport trucks.

Operational Monitoring Range

The system shall support real-time monitoring of machinery across a data sampling range from 1 data point per minute to 60 data points per minute.

Rationale: The system processes sensor data with a wide range of frequencies to accommodate different types of heavy machinery, each requiring distinct monitoring levels

8.2.2. Physical Characteristics

Mass

The mass of the Monico Data Analysis project shall be less than or equal to 5 kilograms.

Rationale: The system will weigh no more than 5 kg because it only uses a laptop and an external hard disc.

Volume Envelope

The Monico Data Analysis project's volume envelope must measure less than or equal to 15 inches in diameter, with dimensions of 15 and 13 inches for width and length, respectively.

Rationale: Since this system is an ML project, it will only require a laptop.

Mounting

There will be no equipment mounting requirements for the Monico Data Analysis project on machinery. All that needs to be done is upload the code to Monico's state-of-the-art mCoreSDR computer.

8.2.3. Electrical Characteristics

Inputs

a. The presence or absence of any combination of the input signals in accordance with ICD specifications applied in any sequence shall not damage the Monico Data Analysis System, reduce its life expectancy, or cause any malfunction, either when the unit is powered or when it is not.

b. No sequence of command shall damage the Monico Data Analysis System, reduce its life expectancy, or cause any malfunction except the end program prompt.

Rationale: By design, should limit the chance of damage or malfunction by user/technician error.

Power Consumption

The maximum peak power of the system shall not exceed 70 watts used for charging the laptop.

Rationale: The requirement will guarantee appropriate power consumption, preventing the laptop from dying.

Input Voltage Level

The input voltage level for the Monico Data Analysis shall be 0 VDC.

Input Noise and Ripple

The input noise and ripple do not apply to the Monico Data Analysis System.

External Commands

The Monico Data Analysis System shall document all external commands in the appropriate ICD.

Rationale: The ICD will capture all interface details from the low level electrical to the high-level packet format.

Outputs

Data Output

An API interface that allows users to track data inputs and outputs will be included in the Monico Data Analysis System.

Rationale: The Monico Data Analysis System passes information directly to the customer's system.

Diagnostic Output

The Monico Data Analysis System shall include a diagnostic interface for control and data logging.

Rationale: Provides the ability to control things for debugging manually and a way to view/download the node map with associated potential targets.

Raw Video Output

The Monico Data Analysis System central unit will not include a raw video interface to support external recording.

Connectors

The Monico Data Analysis system shall use no external connectors.

Wiring

The Search and Rescue System shall need no wiring.

8.2.4. Environmental Requirements

The Monico Data Analysis System shall be designed to withstand and operate in the environments and laboratory tests specified in the following section.

Pressure (Altitude)

The only place the Monico Data Analysis System can operate is on the ground with the necessary data, WiFi, and the mCoreSDR attached to the machinery.

Rationale:The system needs to remain stationary on the machinery in order for it to work and gather data, which is then used by the machine learning algorithm to decide when repair is required. Examples of this kind of machinery include bulldozers and mining equipment.

Thermal

As long as the mCoreSDR is operational, the Monico Data Analysis System will function, and it should be ensured that the mCoreSDR does not overheat.

Rationale: Monico owns a computer called mCoreSDR, which will power the data analysi system. This means that the data analysis system will function as long as the computer does.

External Contamination

The Monico Data Analysis system relies on API for external communication, which enables real time interaction between mCoreSDR, and user interfaces.

Rain

The Monico Data Analysis System will not work if exposed to rain.

Rationale: Any computer will not work if exposed to rain because of its electrical characteristics.

Humidity

A relative humidity of 30-40% is ideal for the Monico Data Analysis System to work.

Rationale: The data analysis system won't function if heated air gets inside the mCoreSDR, perhaps leading to short circuits and hardware damage.

8.2.5. Failure Propagation

The Monico Data Analysis System shall not allow propagation of faults beyond the Monico Data Analysis System interface.

Failure Detection, Isolation, and Recovery (FDIR)

Built In Test (BIT)

The Monico Data Analysis System may have an internal subsystem that will generate test inputs and shall evaluate the machine learning algorithm responses and determine if there is a failure.

BIT Critical Fault Detection

The BIT shall be able to detect a critical fault in the Monico Data Analysis System 90 percent of the time.

Rationale: This would allow the system to detect any failures.

BIT False Alarms

The BIT shall have a false alarm rate of less than 10 percent.

Rationale: This will limit the number of false outputs sent to the user interface.

BIT Log

The BIT shall save the results of each test to an Excel file that shall be stored in the Monico Data Analysis System for retrieval and clearing by maintenance personnel.

Rationale: This will assist in determining the initial cause of the machine's faults.

Isolation and Recovery

The Monico Data Analysis System should provide for fault isolation and recovery by enabling subsystems to be reset or disabled based upon the result of the BIT.

Rationale: In case the user wants the program to fully reset.

9. Support Requirements

Hardware Requirements (Laptop):

Processor: Intel Core i5 or equivalent

RAM: 8 GB

Storage: 256 GB SSD

Operating System: Windows 10 or Linux (Ubuntu 18.04 or newer)

Network: Stable internet connection with at least 10 Mbps for real-time data monitoring and remote updates.

Edge Device:

The software will be pre-installed on Monico's mCoreSDR edge device, which powers the system. For the device to gather and process data, it needs to be linked to the machine's sensor network. To reduce data loss or interference, make sure the mCoreSDR has a sufficient power supply and is positioned in a secure area close to the machinery.

Appendix A: Acronyms and Abbreviations

BIT	Built-In Test
UI	User Interface
ICD	Interface Control Document

Appendix B: Definition of Terms

MONICO DATA ANALYSIS

Digvijay Alluri
Nicho Naugle
Michael Cubriel

INTERFACE CONTROL DOCUMENT

REVISION – 1

INTERFACE CONTROL DOCUMENT
FOR
Monico Data Analysis

PREPARED BY:

_____
Author Date

APPROVED BY:

_____
Project Leader Date

_____
John Lusher II, P.E. Date

_____
T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1.0	9/26/2024	Whole Team		Draft Release

Table of Contents

List of Figures

List of Tables

10. Overview

This document describes the interfaces between the different subsystems of the Monico data analysis system. The subsystems will be integrated to form a solution that provides real time monitoring and predictive maintenance. The subsystems include the data preprocessing, which is responsible for cleaning and preparing the raw data for the prediction; the AI/ML model, which will make an estimate in hours of the remaining life left in the machine; and the API which will facilitate communication between the subsystems using websockets. The interfaces defined in this document cover data formats, communication protocols, and software interactions, ensuring that all subsystems can operate efficiently as part of the larger system.

11. Data Specifications and Interface

11.1. Raw Data

The raw data inputs are gathered from various sensors on a single-stage reciprocating compressor driven by the Caterpillar G3612 A3 Natural Gas engine. There are a total of 176 sensor inputs ranging from engine temperatures, pressure, vibration level, and diagnostic codes. The data was recorded and collected every second for approximately three years lending us to over 5 billion data points for this project. The data was received in CSV format and is presented in a time series format.

11.2. Preprocessing

Along with the ML/AI program for predictive maintenance the mCoreSDR device will also contain a program for preprocessing the raw data. Preprocessing includes handling null/missing values, unit conversions, averaging the data, and appending necessary columns/labels for the ML/AI model to function. Finally with the data in a numpy array, the ML model will be called.

12. Hardware Interface

12.1. G3612 Gas Compression Engine & Ariel Reciprocating 4-Throw Compressor

The G3612 Gas Compression Engine is an industrial 12-cylinder engine developed by Caterpillar specifically for gas compression purposes. In our application, it is driving an Ariel Reciprocating 4-Throw compressor which provides the mechanical force for compressing the natural gas. The data is collected from an array of sensors placed to monitor important parameters. This data is transmitted wirelessly to the mCoreSDR where it undergoes preprocessing.

12.2. mCoreSDR

The mCoreSDR, developed by Monico, is an industrial edge computing device capable of high-performance real-time data processing and collection from a wide array of industrial

equipment. For this project, the mCoreSDR will house the AI and preprocessing tasks allowing for these processes to be run locally. By running these processes locally, they are kept isolated from any cloud based connectivity and will be accessible through a front end provided through the API.

13. Software Interface

13.1. API Communication

The API will serve as the means of communication between the mCoreSDR device and the front end of the network. By using the API, can remotely administer commands and receive predictive insights without directly accessing any hardware. This will be achieved through the use of WebSocket protocols to allow the API to maintain a persistent connection between the mCoreSDR and user. Example documentation generated by FastAPI can be seen below

Figure 1: API Documentation

13.2. API Commands

The API provides a set of commands that allow the user to interact with the McoreSDR and its AI/ML software. These commands are designed to give the user control of various features of the program without giving them direct access.

13.2.1. Prediction_Run:

This command initiates the AI/ML model on the mCoreSDR device. When this command is issued by the user the model will begin processing the data and generating predictions. Upon completion, the mCoreSDR will issue back the number of hours left until the machine is predicted to fail along with other metrics such as confidence, accuracy, and f1 score.

13.2.2 Stop_Model:

This command triggers the AI/ML model to halt any ongoing processes it is currently running. This command can be used for general purposes when the program needs to stop running.

13.2.3 Train_Model

This command instructs the model to enter training mode, where it processes new, previously unseen data to enhance its predictive capabilities. By allowing the model to learn from additional data, this mode strengthens the model's accuracy and adaptability, ensuring that it remains effective even as new patterns and trends emerge in the data. Utilizing this training mode is crucial for continuously refining the model, enabling it to provide more reliable and precise predictions over time.

14. Communications / Device Interface Protocols

14.2. WebSocket

Our project will use the WebSocket protocol to facilitate bidirectional communication between the mCoreSDR device and the user. WebSocket is chosen for its ability to maintain an open, persistent connection between the client and the server, enabling continuous data exchange. The persistent connection ensures that users receive these updates immediately, without the trouble of repeatedly opening and closing connections as would be required with traditional HTTP requests.

Appendix A: Acronyms and Abbreviations

Document Name	Revision/ Release Date	Publisher
Python3 Documentation	3.9	Python Software Foundation
Websockets API Documentation	1.3	MDN Web Documents

Appendix B: References

API - Application programming interface is a way for two or more programs to interface.

CSV - Comma Separated Values is the format in which the data is received.

AI - Artificial Intelligence

ML - Machine Learning

SDR - Secure Data Router

MONICO DATA ANALYSIS

Digvijay Alluri
Nicho Naugle
Michael Cubriel

Data Preprocessing Subsystem Report

COMPLETED BY: DIGVIJAY ALLURI

REVISION – 1

Table of Contents

List of Figures

1.1. Subsystem Introduction

The Data Analysis Subsystem is designed to process and analyze data collected from various system components. It ensures efficient handling of both numerical and non-numerical data, providing valuable insights into system performance. The subsystem is equipped with the capability to detect anomalies and identify fault conditions based on specific error codes. It also generates trend lines and visual representations of the data to assist in understanding system behavior over time. The reliability and performance of the subsystem have been verified through comprehensive testing, ensuring it meets the requirements for supporting data-driven decision-making within the system.

2. Data Analysis Sub-System

2.1. Subsystem Overview

The Data Analysis Subsystem is a critical component responsible for processing, validating, and preparing sensor data to ensure its suitability for machine learning (ML) model training. This subsystem performs a comprehensive set of operations designed to streamline data analysis and ensure high-quality outputs.

2.2. Handling Non-Numeric Data

One of the primary functions of the subsystem is identifying and managing non-numeric data, such as error codes (e.g., E-270-3) or missing values. These anomalies are extracted and transferred into newly created columns at the end of the dataset. Each column is labeled with its respective source for easy traceability. This organization facilitates efficient data validation, ensuring that any discrepancies or anomalies are clearly identifiable for further analysis or rectification.

```python
def find_and_store_non_numerical(df):
    # Creates a dictionary to store non-numeric values for each column
    non_numeric_data = {col: ['' for _ in range(len(df))] for col in df.columns[1:]}  # Initialize storage for non-numeric va
    # Iterate over the DataFrame columns, starting from the second column (assuming 'Timestamp' is first)
    for column in df.columns[1:]:
        try:
            # Convert each column to numeric, forcing errors to NaN
            non_numeric = pd.to_numeric(df[column], errors='coerce')
            # Get the indices where non-numeric entries are found
            non_numeric_indices = non_numeric[non_numeric.isna()].index
            # For each index with non-numeric values, store the value to the corresponding position in non_numeric_data
            for idx in non_numeric_indices:
                non_numeric_data[column][idx] = df.loc[idx, column]
            # Replace the non-numeric values in the original column with NaN
```

Figure 1: Code for non-numeric data

2.3. Statistical Analysis

The subsystem calculates key statistical metrics for each column in the dataset, including the mean, median, standard deviation, and the highest and lowest values. These calculations provide an in-depth understanding of the dataset's distribution, variability, and overall behavior. This statistical insight is crucial for preprocessing the data, ensuring that it meets the requirements for accurate ML model training.

2.4. Data Averaging

Given the high frequency of data collection—recorded every second for 24 hours—the subsystem averages the data at regular intervals, such as every five minutes. This reduces the granularity of the dataset while preserving essential patterns and trends. Data averaging enhances computational efficiency and ensures that ML models can process the data effectively without compromising on critical information.

```
            raise ValueError("Duration for averaging must be greater than zero.")

        # Round down to the nearest multiple of duration_user
        df['Timestamp'] = (df['Timestamp'] // duration_user) * duration_user
        df_avg = df.groupby('Timestamp', as_index=False).mean()  # Calculate the mean for numeric columns
        # Convert the timestamp back to a readable format
        df_avg['Timestamp'] = df_avg['Timestamp'].apply(lambda x: datetime.fromtimestamp(x).strftime('%Y-%m-%dT%H:%M:%S'))
    except ValueError as e:
        logging.error(f"Value error: {e}")
        return None
    except Exception as e:
        logging.error(f"Failed to process timestamps: {e}")
        return None

    # Adding the non-numeric values from the original DataFrame to the averaged DataFrame
    for column in df.columns:
        if column.startswith("Non_Numeric_Values_"):
            # Using `first()` here assumes that all non-numeric values in each group are the same
```

Figure 2: Code for data averaging

2.5. Correlation Matrix

To uncover relationships between different variables, the subsystem computes a correlation matrix. This matrix quantifies how strongly sensor readings influence one another, revealing potential interdependencies. The insights gained from the correlation matrix are valuable for feature selection and optimization during the modeling phase.

```
    # Extract highly correlated pairs (> 0.7) to examine interdependencies
    high_corr_vars = correlation_matrix[correlation_matrix.abs() > 0.7].stack().reset_index()
    high_corr_vars = high_corr_vars[high_corr_vars['level_0'] != high_corr_vars['level_1']]
    high_corr_vars.columns = ['Variable 1', 'Variable 2', 'Correlation']
    high_corr_vars.drop_duplicates(inplace=True)

    # Display the high correlation pairs for further analysis
    print("Highly correlated variable pairs:")
    print(high_corr_vars)

.f   name   == "  main  ":
```

Figure 3: Code for correlation matrix

2.6. Trend Line Graphs

The Data Analysis Subsystem generates trend line graphs for selected columns, visually representing sensor data over time. These graphs help identify patterns, deviations, and anomalies, aiding in the early detection of potential issues. By tracking changes in the data, the subsystem allows for better monitoring of system performance, ensuring irregularities are addressed before becoming significant problems. The trend analysis offers clear, real-time insights, supporting informed decision-making and enhancing the overall effectiveness of the data analysis process. This functionality plays a key role in maintaining consistent system performance and improving reliability.

2.7. Fault Relay Monitoring

The Data Analysis Subsystem includes a vital feature to monitor machine failures by identifying when the fault relay column transitions to 1, signaling a system shutdown. When this occurs, the subsystem logs the event with its corresponding timestamp and any associated error codes from the dataset. This capability ensures a detailed record of failures, supporting effective root cause analysis.

This feature enhances system reliability by providing valuable insights into failure patterns, helping diagnose issues and enabling timely corrective measures. The logged fault data becomes an integral part of the processed dataset, aligning with other analysis outputs to prepare the data for machine learning (ML) models. By incorporating fault events into ML training, the subsystem enables predictive maintenance and proactive fault prevention. This seamless integration ensures the system's data is not only well-prepared but also enriched with actionable insights for future applications.

```python
def print_faulty_entries(df):
    # Checks for faulty machines indicated by '1' in the 'Ramsey C4701E.Fault Relay' column
    if 'Ramsey C4701E.Fault Relay' not in df.columns:
        logging.error("Column 'Ramsey C4701E.Fault Relay' not found in the DataFrame.")
        return  # Exit if column is not found
    # Check for rows with a '1' in the fault relay column
    faulty_entries = df[df['Ramsey C4701E.Fault Relay'] == 1]
    # If there are faulty entries found, which indicates that the machine has failed
    if not faulty_entries.empty:
        print("When fault relay is 1 (component has failed):")
        for index, row in faulty_entries.iterrows():
            try:
                # Get the timestamp
                timestamp = row['Timestamp']
                # Get non-numeric values for that row
                non_numeric_values = {col: row[col] for col in df.columns if pd.isna(pd.to_numeric(row[col], errors='coerce')
                # Print the timestamp and only the non-numeric values
                if non_numeric_values:  # Only print if there are non-numeric values
                    print(f"Timestamp: {timestamp}, Non-numeric values: {non_numeric_values}")
                else:
                    print(f"Timestamp: {timestamp}, no non-numeric values found.")
            except Exception as e:
                logging.warning(f"Error while processing faulty entry at index {index}: {e}")
    else:
```

Figure 4: Code for fault relay

2.8. Output Compilation

The Data Analysis Subsystem compiles all processed data into a structured CSV output file, ensuring readiness for downstream integration with machine learning (ML) pipelines. This comprehensive output includes separated non-numeric data, such as error codes, categorized into labeled columns for easy identification. Key statistical metrics, including mean, median, standard deviation, and extreme values, provide insights into data distribution. Trend line data highlights patterns and anomalies over time, while a correlation matrix explores relationships between variables to understand system dependencies. High-frequency data is averaged over intervals, such as five minutes, simplifying large datasets while retaining essential trends. Additionally, fault relay events, marked by a "1," indicating machine failure, are logged with timestamps and error codes for diagnostic purposes. By consolidating these elements, the subsystem ensures the dataset is clean, validated, and enriched with actionable insights, supporting effective ML model training and integration with other project subsystems.

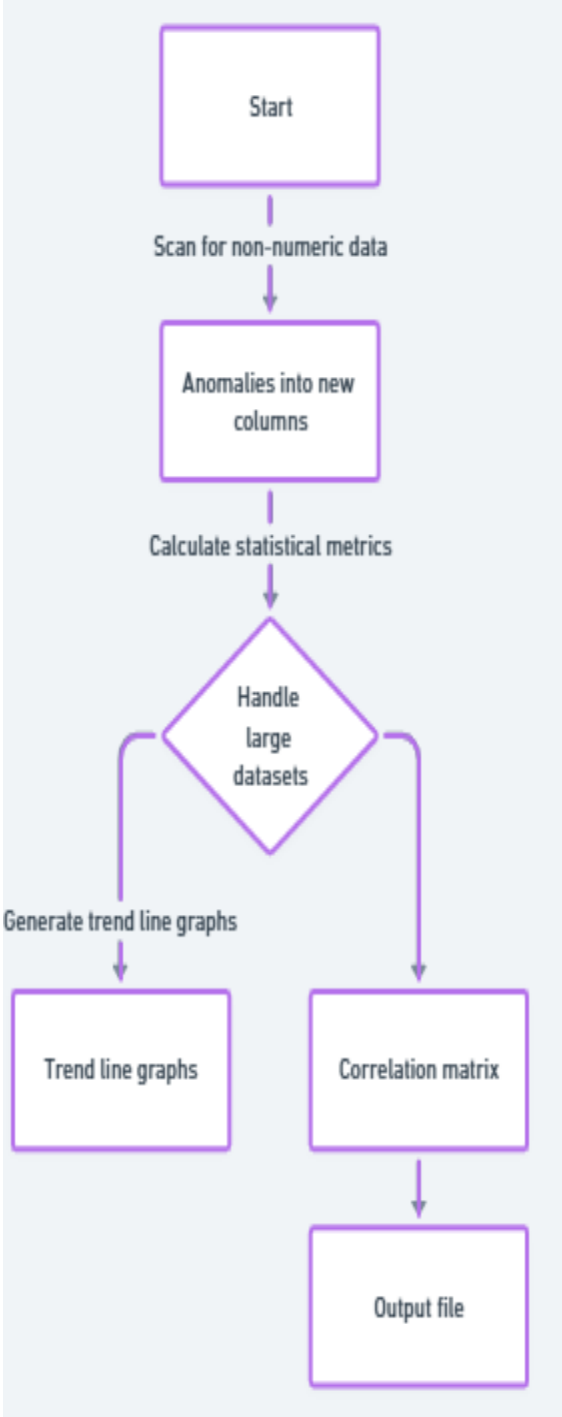
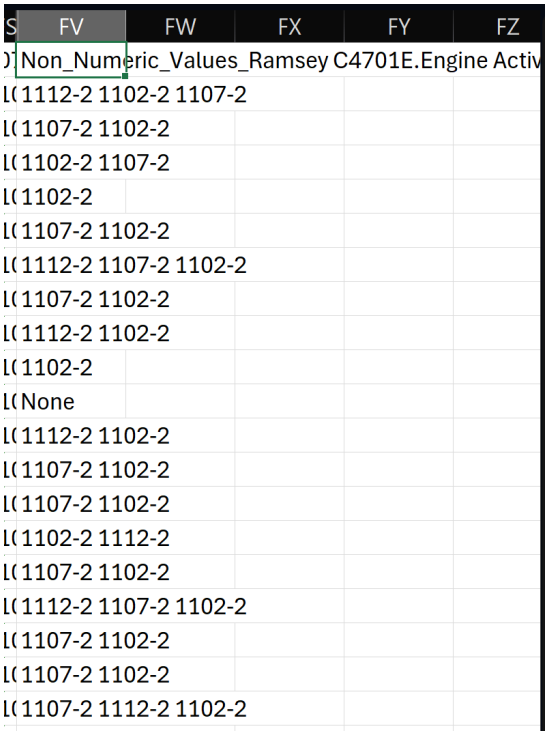


Figure 5: Functional diagram of the data analysis subsystem

3. Subsystem Validation

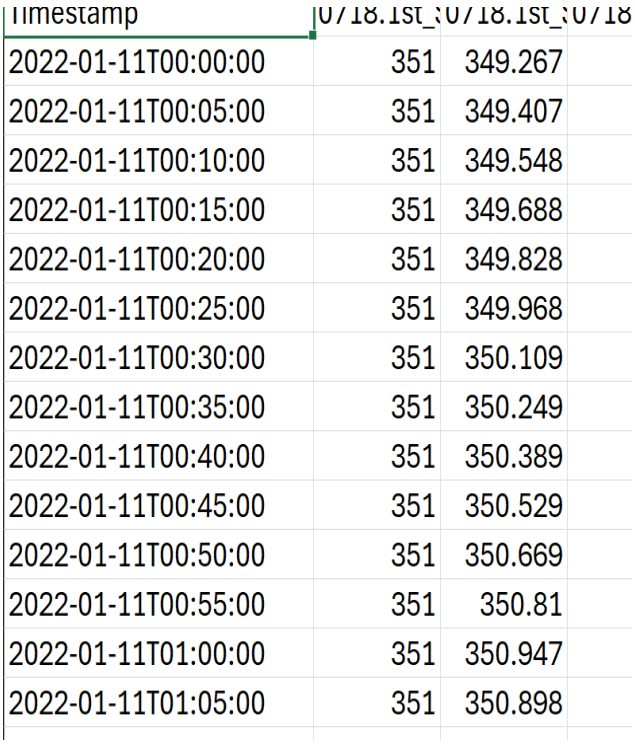
The Data Analysis Subsystem has been validated for its accuracy and reliability in processing sensor data. It identifies non-numeric data, such as error codes, and separates them into labeled columns for easier traceability (*Figure 6*). To handle high-frequency readings, the subsystem averages data over intervals, such as every 5 minutes, as shown in *Figure 7*, simplifying large datasets for further analysis and ML model training.

Additionally, the subsystem calculates key statistical metrics, including mean, median, standard deviation, and extreme values, ensuring a comprehensive understanding of the dataset's distribution (*Figure 9*). The correlation matrix (*Figure 8*) has been validated to reveal variable relationships, while trend analysis (*Figure 10*) effectively visualizes changes over time. These features confirm the subsystem's capability to deliver accurate, processed data ready for advanced modeling and analysis.



FV	FW	FX	FY	FZ
Non_Numeric_Values_Ramsey C4701E.Engine Activ				
1112-2 1102-2 1107-2				
1107-2 1102-2				
1102-2 1107-2				
1102-2				
1107-2 1102-2				
1112-2 1107-2 1102-2				
1107-2 1102-2				
1112-2 1102-2				
1102-2				
None				
1112-2 1102-2				
1107-2 1102-2				
1107-2 1102-2				
1102-2 1112-2				
1107-2 1102-2				
1112-2 1107-2 1102-2				
1107-2 1102-2				
1107-2 1102-2				
1107-2 1112-2 1102-2				

Figure 6: Error codes



Timestamp	0718.1st_3	0718.1st_3	0718
2022-01-11T00:00:00	351	349.267	
2022-01-11T00:05:00	351	349.407	
2022-01-11T00:10:00	351	349.548	
2022-01-11T00:15:00	351	349.688	
2022-01-11T00:20:00	351	349.828	
2022-01-11T00:25:00	351	349.968	
2022-01-11T00:30:00	351	350.109	
2022-01-11T00:35:00	351	350.249	
2022-01-11T00:40:00	351	350.389	
2022-01-11T00:45:00	351	350.529	
2022-01-11T00:50:00	351	350.669	
2022-01-11T00:55:00	351	350.81	
2022-01-11T01:00:00	351	350.947	
2022-01-11T01:05:00	351	350.898	

Figure 7: Data Averaging

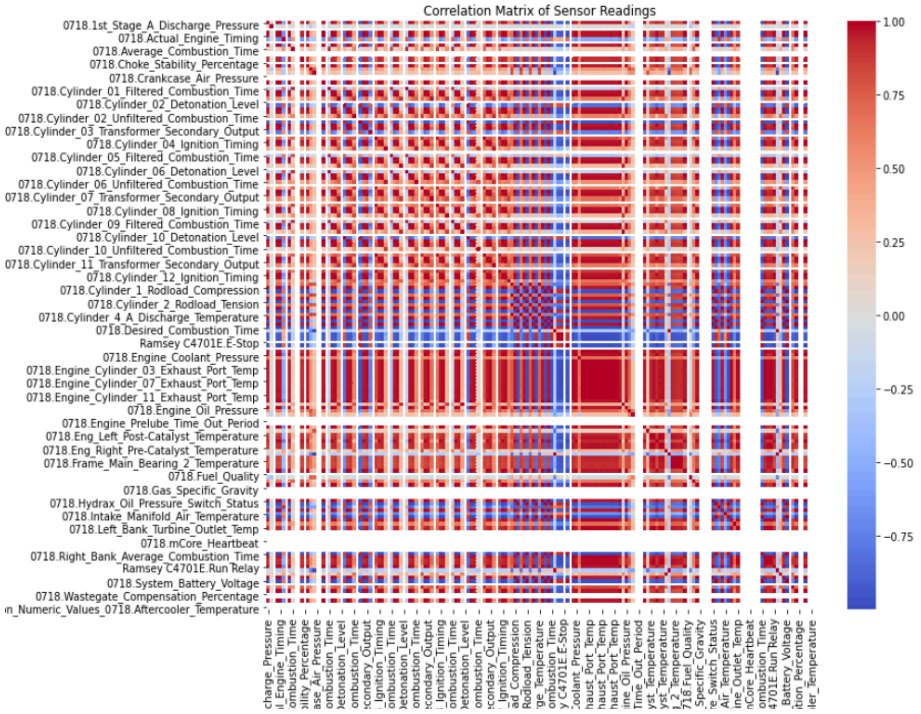


Figure 8: Correlation Matrix

	0718.Speed	0718.System_Battery_Voltage	0718.Total_Crank_Cycle_Time
count	288.000000	288.000000	288.0
mean	17.197975	24.989178	30.0
std	111.766380	0.070258	0.0
min	0.000000	24.500000	30.0
25%	0.000000	25.000000	30.0
50%	0.000000	25.000000	30.0
75%	0.000000	25.000000	30.0
max	800.260497	25.000000	30.0

Figure 9: Data Analysis

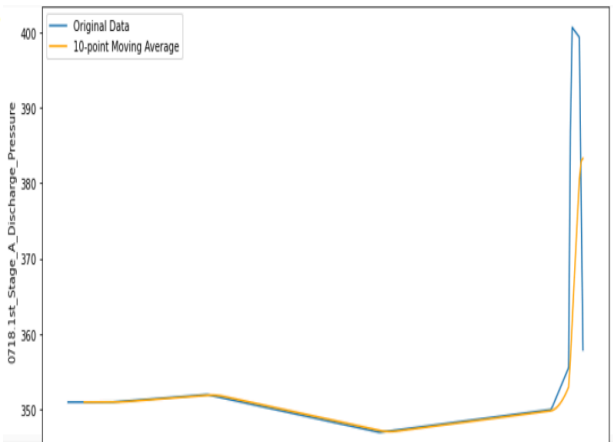


Figure 10: Trend Analysis

4. Subsystem Conclusion

The Data Analysis Subsystem has proven to be an essential component in the overall system, ensuring the integrity and readiness of sensor data for machine learning model training. Its ability to identify non-numeric data, such as error codes, and separate it into designated columns enhances data clarity and reduces potential issues during analysis. The subsystem's functionality extends to calculating critical statistical metrics, such as mean, median, standard deviation, and the highest and lowest values, providing comprehensive insights into the dataset's distribution and variability. Additionally, by averaging high-frequency data and generating trend line graphs, the system allows for more manageable datasets, making it easier to detect trends and anomalies in the data.

This approach of combining error handling, statistical analysis, and data reduction significantly improves the quality and reliability of the data. The subsystem's integration of these features ensures that the data is both accurate and prepared for further analysis or modeling. Its capability to process large datasets with various error codes, while maintaining accuracy and consistency, establishes its importance as a foundational tool in the broader system. As the project progresses, the subsystem will continue to play a pivotal role in streamlining data analysis and supporting subsequent tasks, enabling more effective decision-making and efficient use of sensor data.

5. Future Plans for the Subsystem

The Data Analysis Subsystem will be enhanced to facilitate seamless integration with the Machine Learning (ML) and API subsystems, alongside improvements to scalability and automation.

For the ML Subsystem, the focus will be on enabling real-time data preprocessing, where sensor data streams are processed dynamically to feed ML models continuously. Advanced feature engineering techniques will be integrated, leveraging statistical metrics and correlation insights to optimize model input. A feedback loop from the ML subsystem will help refine preprocessing workflows based on model performance.

For the API Subsystem, the system will support API-driven data access and processing, allowing external requests for trend analysis, statistical summaries, and anomaly reports. Preparations for cloud deployment will ensure the subsystem handles high-volume datasets efficiently while supporting multiple users simultaneously.

MONICO DATA ANALYSIS

Digvijay Alluri
Nicho Naugle
Michael Cubriel

API & Networking Subsystem Report

COMPLETED BY: NICHO NAUGLE

REVISION – 1

Table of Contents

1. Subsystem Introduction

The API and networking subsystem serves as the backbone of our application, providing critical features for communication and data management. It includes: a WebSocket server to stream real-time sensor data to multiple simultaneous connections, a RESTful API, which facilitates seamless integration with local dashboards and inter-device communication on the same network, and a mDNS resolver to identify and connect devices on dynamic IP networks by resolving them to static hostnames for reliability. The subsystem is implemented in Python, ensuring maintainability, and cross-platform compatibility across Linux, macOS, and Windows environments. Its lightweight design makes it ideal for deployment on edge devices. Additionally, all machine learning (ML) configurations and data streaming settings can be easily managed through the API.

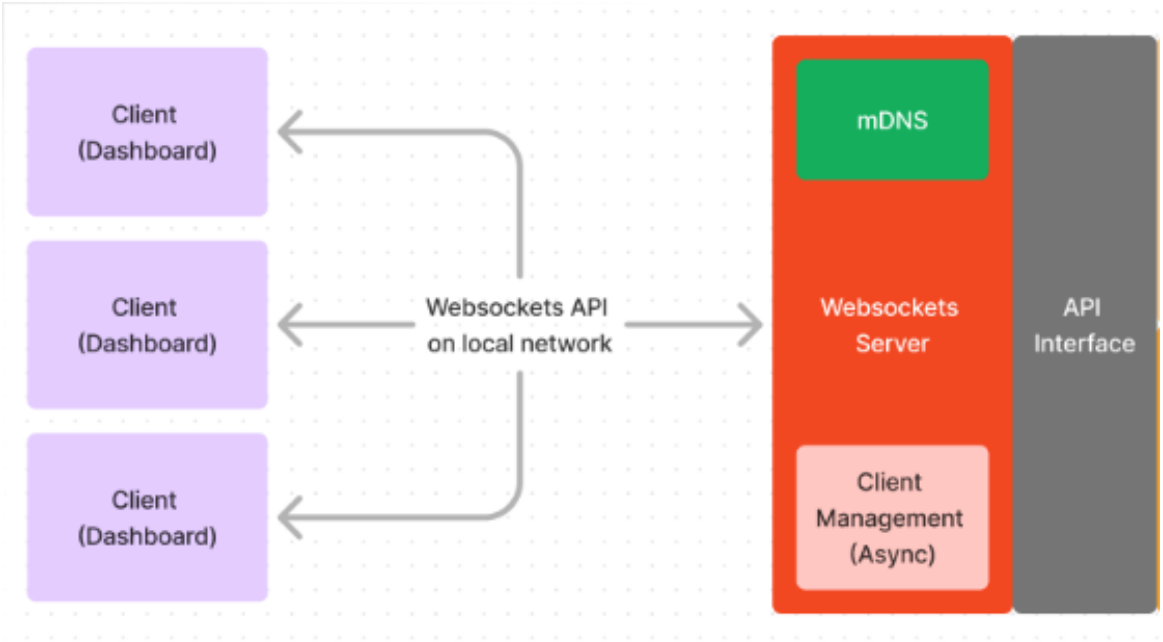


Figure 1: API & Networking complete functional diagram

2. Connectivity

With a Wi-Fi or Ethernet connection available, the API and networking features are automatically activated at startup. All parameters specified in the configuration JSON file are then applied to the server seamlessly.

2.1. mDNS Resolution

On startup, the system asynchronously performs mDNS resolution using a hostname defined in the configuration file. The format predictai-***** (where ***** is a unique, randomly generated identifier) prevents name collisions and simplifies device identification. While mDNS resolves collisions automatically, unique hostnames are especially helpful in managing multiple devices. Figure 2 provides an example of the resolved mDNS service attributes, including the corresponding IP address and port, both configurable via server_config.json.

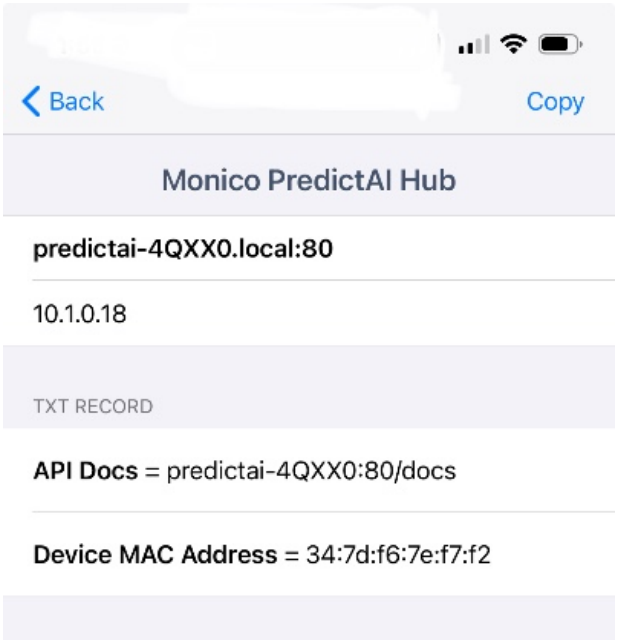


Figure 2: Resolved mDNS service example

2.2. RESTFUL API

The RESTful API facilitates instantaneous GET requests, allowing state updates and configuration changes in a single transaction. This minimizes latency and avoids synchronization issues. REST endpoints are designed exclusively for modifying server settings, while high-volume data streaming occurs over WebSockets for efficiency.

2.3. Websockets

WebSockets enable real-time data streaming, allowing seamless integration with dashboards and local storage solutions like NAS devices. This subsystem also broadcasts updates to connected clients, aiding in visualization and monitoring the number of active connections. Monitoring connections helps optimize bandwidth usage and ensures computational resources on the edge device are not overburdened.

3. Available Control Endpoints

The API provides a range of control endpoints to facilitate management and monitoring of the system's operations. These endpoints are integral to ensuring flexibility in configuration and real-time adaptability during runtime. To access the list of available endpoints, users can navigate to http://predictai-*****/docs, where the ***** represents the unique identifier generated on the first run. For networks without local DNS resolution enabled, users will need to use the server's IP address to access the documentation. While this introduces some inconvenience in dynamic IP environments, custom tools may be useful in resolving the IP.

3.1. active-tasks

This allows users to monitor the tasks currently running on the server. This functionality is particularly useful for diagnosing resource allocation and verifying that the system is functioning as intended. By providing insight into the server's active processes, users can better manage computational loads and troubleshoot any potential issues.

3.2. set-streaming

This determines whether data streaming to WebSocket connections is enabled. When streaming is turned off, clients can still establish WebSocket connections but will not receive any data. This allows for more controlled operation of the server, giving users the ability to pause data transmission without disrupting the overall infrastructure.

3.3. server-info

This endpoint is designed to deliver comprehensive server details, including the current network configuration and the state of various settings. This endpoint is instrumental in providing a snapshot of the system’s status at any given time, enabling users to verify its readiness and make informed adjustments to the configuration.


```
{
  "Server Hostname": "predictai-4QXX0",
  "IP Address": "10.1.0.18",
  "Server Port": "80",
  "mDNS Service": "Monico PredictAI Hub",
  "WebSocket Client Count": "0",
  "Data Live Streaming": "True",
  "Data Averaging Interval": "5",
  "AI Prediction State": "True"
}
```

Figure 3: API JSON response for “server-info” endpoint

3.4. set-averaging-interval-period

This endpoint offers the capability to configure the interval at which averaged data is sent to connected clients. By specifying a time period in seconds, users can adjust the frequency of updates to align with their specific requirements. This feature is particularly beneficial in environments where network bandwidth is a concern, as it allows for reduced data transmission rates without sacrificing the integrity of the information.

3.5. set-prediction-state

This endpoint is dedicated to managing the activation of the machine learning inference model. By toggling this endpoint, users can enable or disable predictive operations, thereby optimizing the system’s computational resources. Disabling predictions when they are not required ensures that the edge device can prioritize other tasks, such as data streaming or analytics without wasting compute resources.

4. Application Portability

The application has been designed with portability and adaptability at its core, ensuring it can be deployed across various operating systems and hardware platforms. Its compact size, occupying less than 1GB as shown in Figure 4, allows for operation on devices with a limited storage capacity. This lightweight architecture ensures that the majority of computational resources remain available for critical tasks such as data processing and machine learning inference.

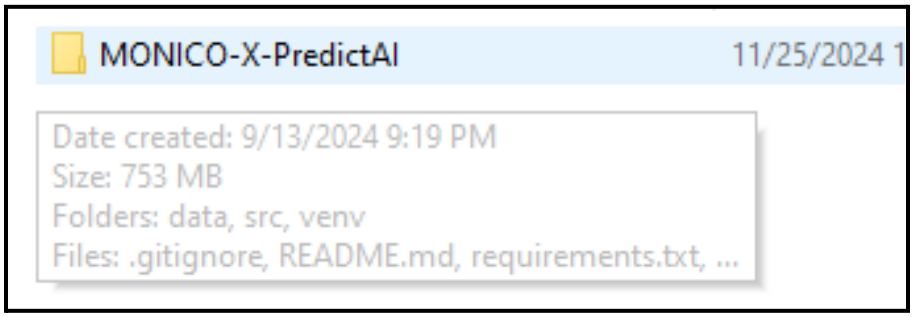


Figure 4: Complete application size shows less than 1GB

4.1. Compatibility

Compatibility is a key strength of the application. Built for use on the mCoreSDR device, it is also accessible to frontend developers aiming to create custom dashboard solutions. The networking and API features are based on universally supported technologies, allowing the system to function on a wide array of devices, from desktop computers to smartphones. The system's configuration data is easily modifiable through either over-the-air updates or direct

interaction with the API. The configuration file, server_config.json, stores all necessary settings as seen in Figure 5.

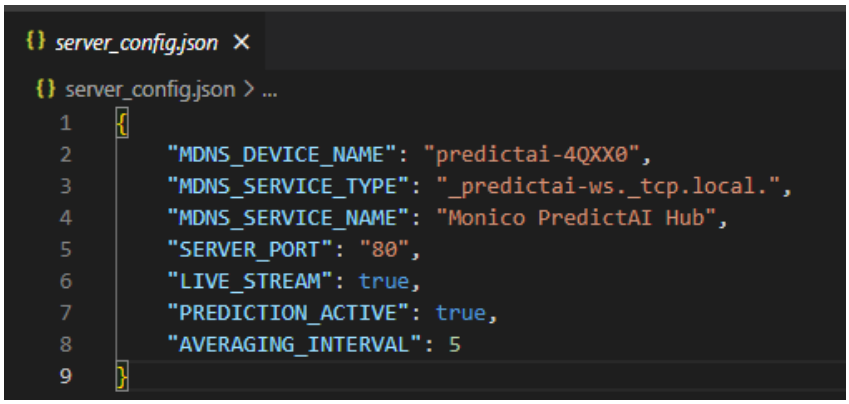


Figure 5: Example of server_config.json

4.2. Constraints

Despite its versatility, the application does have some constraints. For instance, mDNS services rely on local device discovery being enabled within the network. Without this, the system cannot resolve its hostname, requiring users to manually identify the server's IP address. This limitation can introduce additional complexity in certain strict network environments.

While the server itself does not demand specialized hardware for basic operation, machine learning tasks require dedicated computational resources to avoid performance bottlenecks. Edge devices equipped with modest processors may struggle to handle the additional workload of predictive models, leading to potential crashes or delays.

Additionally, the system is not suited for deployment on bare-metal or FreeRTOS systems. Python is not optimized for such environments, making implementation on these platforms impractical. Linux-based processors are the preferred choice, as they offer the necessary flexibility to run the application effectively.

5. Subsystem Validation

Validation metrics focused on the system's capacity to handle active client connections and its compact application size. External factors like internet and processor speed were excluded, as they vary by deployment environment. Key validation results show that the system successfully supported 35 active WebSocket connections simultaneously (see Figure 7). The complete project folder, including configurations, consumed less than 1GB of storage (Figure 8). Figures 6-8 highlight the different validation metrics and their results.

Application Size	Check complete application size once all packages are installed. Exclude all data files and tensorflow libraries	< 1GB
Max Client Count	Create 35 terminal sessions across different terminals and stream large amounts of random data to ensure throughput is successful	35

Figure 6: Validation Plan for the API and Networking Subsystem


```
{
  "Server Hostname": "predictai-4QXX0",
  "IP Address": "10.1.0.18",
  "Server Port": "80",
  "mDNS Service": "Monico PredictAI Hub",
  "WebSocket Client Count": "35",
  "Data Live Streaming": "True",
  "Data Averaging Interval": "5",
  "AI Prediction State": "True"
}
```


Figure 7: API endpoint showing the 35 active connections

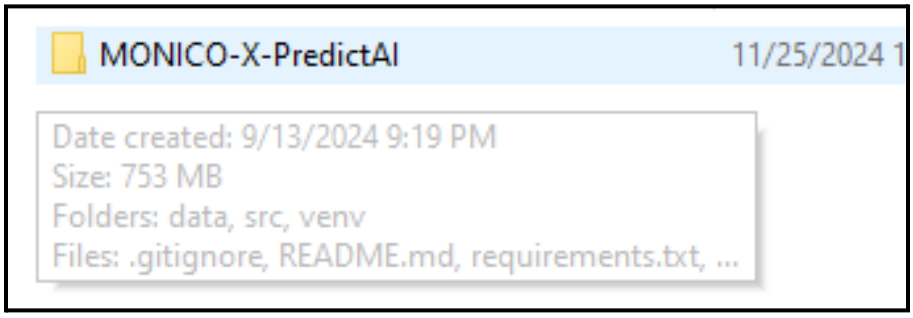


Figure 8: Complete project folder size

6. Conclusion

This project demonstrates the successful design and implementation of a robust API and networking subsystem for edge computing. The combination of mDNS resolution, RESTful endpoints, and WebSocket streaming enables seamless device communication, real-time data handling, and efficient resource management. The system's lightweight architecture and cross-platform compatibility make it highly adaptable for various deployment environments. Comprehensive testing confirms its reliability and scalability.

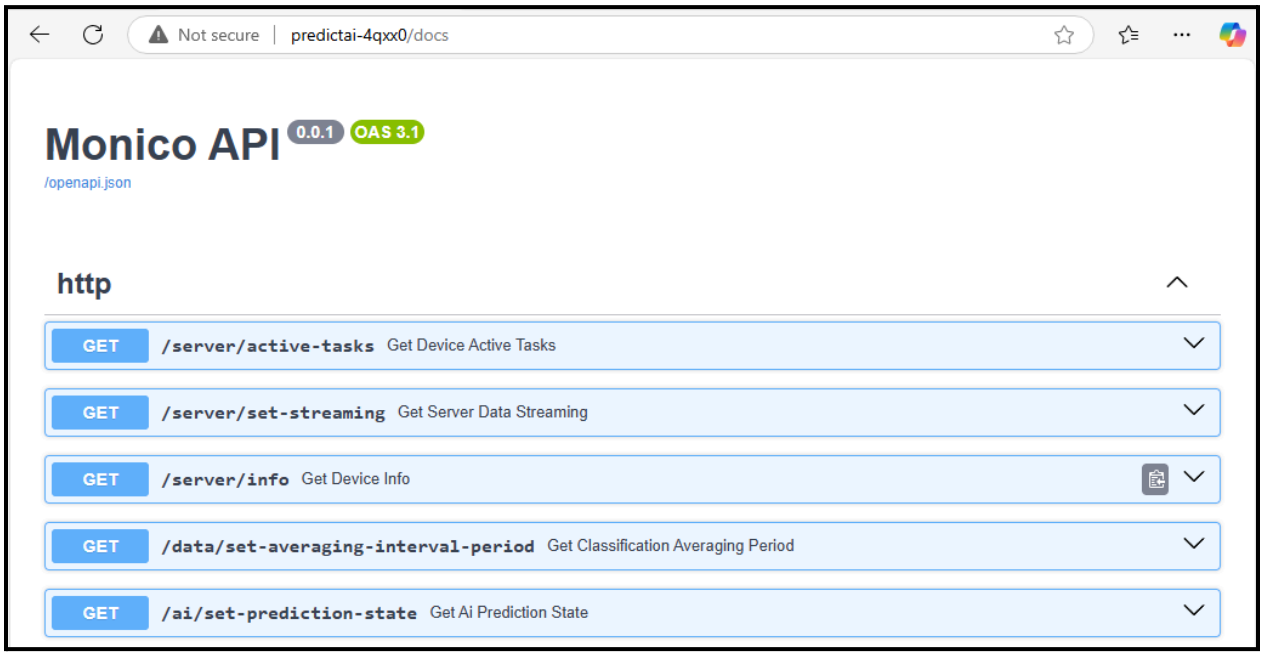


Figure 9: API endpoints currently available from the server

6. Future Plans

Although the system meets the required validation metrics, many ideas for potential improvements were found along the way. Some of these include"

1. Expand ML capabilities: Integrate additional machine learning models for real-time analytics and predictions.
2. Enhance security: Incorporate advanced encryption for data streams and API interactions, as well as user authentication.
3. Implement advanced discovery protocols: Explore alternatives to mDNS for environments where local DNS resolution is unavailable.

These improvements would help make the system more versatile, secure, and efficient for broader use cases outside of its current use case.

MONICO DATA ANALYSIS

Digvijay Alluri
Nicho Naugle
Michael Cubriel

Machine Learning Subsystem Report

COMPLETED BY: MICHAEL CUBRIEL

REVISION – 2

Table of Contents

1. Subsystem Introduction

The subsystem for this project focuses on developing a predictive model to forecast when Monico's natural gas compressor is likely to fail and identifying what might be causing that failure. Three years worth of raw sequential data totaling over 120 gigs was provided by Monico. There are over 170 features derived from the sensor data. To handle the sequential nature of the data, a Random Forest was used as the core model architecture. By leveraging the predicted time till failure and error code, the AI allows for more precise and proactive maintenance strategies, reducing overall down time for the system.

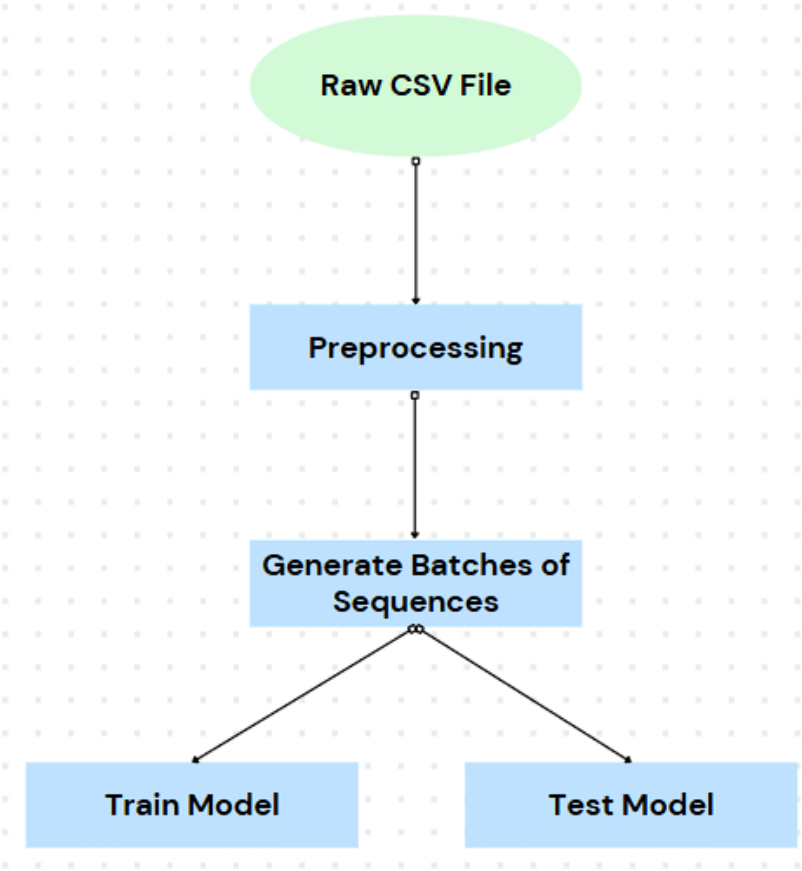


Figure 1: Block diagram of the Workflow for the Subsystem

2. Data

The dataset provided by Monico features operational data from years 2022, 2023, and 2024, with a total size exceeding 120 gigabytes spread across multiple csv files where each csv file contains the data for a day. Each row in the data contains over 170 measurements, giving a comprehensive look at the compressors performance and conditions.

2.1. Fault Relay

Among these features is a fault relay, represented by a binary indicator (0 or 1), which signals if the engine has stopped because of a serious problem. This fault relay will be used as the measurement of when the engine has “failed”. After combing through the data, it was found that the fault relay triggered an average of 65 times a year giving us an adequate amount of failures to train the model.

2.2. Error Codes

Each time the engine fails (fault relay equals 1) there are corresponding error codes that give insight into the potential issues. The error codes are structured into two main categories.

1. Event Codes: These are a combination of the Event ID (EID) and Warning Category Indicator (WCI). The WCI define the severity of the event:

    - WCI = 1: A warning; the engine continues running.
    - WCI = 2: A fault that shuts the engine down.
    - WCI = 3: Reserved but unused in the provided data.

2. Diagnostic Codes: Comprising a Component ID (CID) and Failure Mode Identifier (FMI). While the FMI’s indicate the type of failure the data set does not identify their severities. The significance of Event vs Diagnostic is that an Event is a condition of a process parameter whereas a Diagnostic is a condition of a component or sensor. Examples:

    FMI Codes and Descriptions:

    - **00**: Data valid but above normal operating range
    - **01**: Data valid but below normal operating range
    - **02**: Data erratic, intermittent, or incorrect
    - **03**: Voltage above normal or shorted high
    - **04**: Voltage below normal or shorted low
    - **05**: Current below normal or open circuit
    - **06**: Current above normal or grounded circuit

- **07**: Mechanical system not responding properly
- **08**: Abnormal frequency, pulse, or period
- **09**: Abnormal update
- **10**: Abnormal rate of change
- **11**: Failure mode not identifiable
- **12**: Bad device or component
- **13**: Out of calibration
- **14–15, 20**: Not used

3. Preprocessing

To prepare the data for training, a function called “time till next failure” handled the preprocessing of the data. First the function handles all of the missing values in the columns and replaces them with the appropriate place holder. Next columns that were found to have low or no correlation with the fault relay status were dropped to reduce dimensionality. Features that were also highly correlated with each other were also removed. Since the data was very large, timestamps were averaged every 15 seconds using resampling. After resampling, a min max scaler normalized the features to a range between 0 and 1 to improve the efficiency of model training.

Finally, an additional column was created that contained the next failure in hours for each timestamp. This was achieved by finding the next failure (fault relay equals 1) and finding the difference in hours between the two. Additionally, the error codes had to be encoded as integers so they could be fed into the model.

4. Random Forest Model

The Random Forest model was chosen because of its ability to handle high-dimensional data efficiently. Random Forest is an ensemble learning model that builds multiple decision trees during training and outputs the average prediction across all trees. This approach helps reduce overfitting and improves generalization, making it suitable for predicting time until failure based on a large number of sensor features. Although Random Forests models do not inherently model time dependencies I adapted it for time series data by ensuring that the data was split sequentially and disabling bootstrapping so that each tree was trained on temporally ordered data. Important hyperparameters include the number of trees, the maximum depth of each tree, and the minimum number of samples required to split a node. Random Forest models were also chosen for this application because they handle nonlinear interactions between features well and require minimal data preprocessing compared to other machine learning models

4.1. Model Interpretability with SHAP

To better understand the Random Forest model's predictions, SHAP (SHapley Additive exPlanations) values were used to interpret the feature contributions. SHAP assigns an

importance value to each feature for a particular prediction, allowing for a better understanding of how each sensor input affects the predicted time until failure. The SHAP waterfall plot summarizes the most influential features for a specific prediction, showing both the magnitude and direction (positive or negative) of each feature's impact. Features such as Fuel Quality, Battery Voltage, and Turbine Inlet Temperature were found to have significant influence on the model's output. This analysis provides critical insight for both model validation and real-world operational understanding of the compressor system

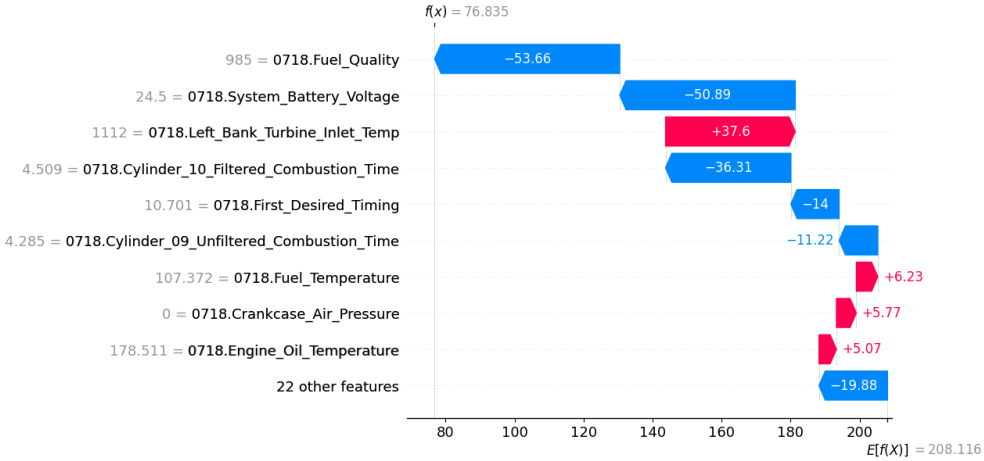


Figure 2: Shap Explanations

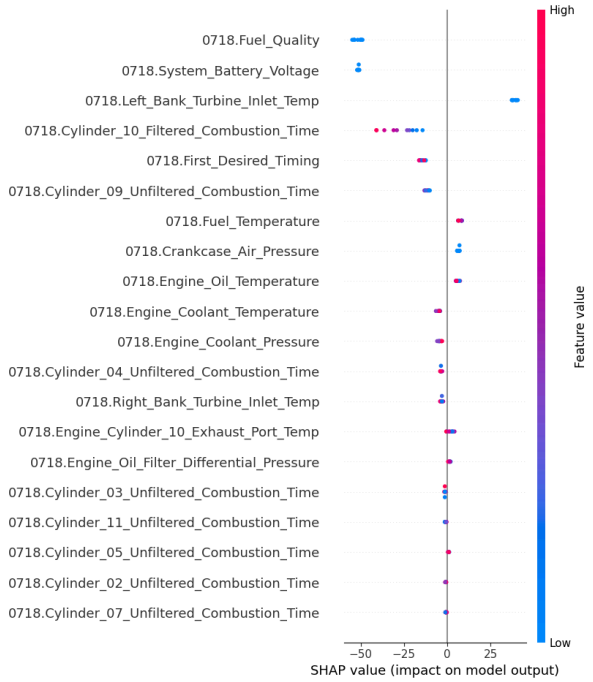


Figure 3: Shap explanations

5. Sequence Batch Generator

Since the data is so large, over 120 gigabytes, processing all of the data simultaneously would have exceeded memory constraints. To address this, a sequence batch generator was implemented. The generator splits the data into batches of sequences allowing the model to process the data iteratively during training as opposed to loading the entire data set at once.

The sequence batch generator works by dividing the data into smaller chunks or batches, each containing multiple sequences. Each sequence contains a fixed length and the associated features. For each sequence the target value (time till the next failure) is derived from the data points immediately following the sequence. This approach not only alleviates the memory issue but also optimized training by allowing for gradient updates after processing each batch.

```python
def sequence_batch_generator(df, selected_features, target, sequence_length, batch_size):
    # Calculate the number of batches based on the batch size
    num_batches = (len(df)) // (batch_size * sequence_length)
    # Loop through each batch index to generate batches of sequences
    for batch_idx in range(num_batches):
        sequences = []  # List to store individual sequences for the current batch
        targets = []    # List to store target values for the current batch

        # Generate sequences within each batch
        for i in range(batch_size):
            # Calculate start and end indices for each sequence within the batch
            start_idx = (batch_idx * batch_size + i) * sequence_length
            end_idx = start_idx + sequence_length

            # Ensure that the end index is within the bounds of the dataframe
            if end_idx < len(df):
                # Extract the sequence of feature values from start to end indices
                seq = df.iloc[start_idx:end_idx][selected_features].values
                # Extract the target label at the end of the sequence range
                label = df.iloc[end_idx][target]
                # Append the sequence and label to their respective lists
                sequences.append(seq)
                targets.append(label)

        # Convert lists to numpy arrays for model compatibility and yield them as a batch
        yield np.array(sequences), np.array(targets)
```

Figure 4: Sequence Batch Generator Function

6. Training & Testing

To train the model a function was developed that handles the large data set and uses the preprocessing and batch generator function to effectively train the model. The training function separates the data into training and validation sets and trains on one file at a time. After finishing training it outputs the training and validation loss plotted over epochs to assess model convergence.

For evaluating the model, a function was made to process a separate test data set that the model has not seen before. It uses the same logic to create the batches of sequences as the incremental training function and makes a prediction for each sequence. The function then calculates the mean absolute error, which gives the average size of error in a given set of predictions no matter if positive or negative. Additionally, a visualization of the true values and the models predictions over time is generated to provide a more clear picture on how the model is adjusting to the data.

```
Starting training phase
Training on file: /content/drive/MyDrive/MonicoData/2022_Files/2022/Export_20220101T000000_20220101T235959.csv
Training on file: /content/drive/MyDrive/MonicoData/2022_Files/2022/Export_20220102T000000_20220102T235959.csv
Training on file: /content/drive/MyDrive/MonicoData/2022_Files/2022/Export_20220103T000000_20220103T235959.csv
Training on file: /content/drive/MyDrive/MonicoData/2022_Files/2022/Export_20220104T000000_20220104T235959.csv
Training on file: /content/drive/MyDrive/MonicoData/2022_Files/2022/Export_20220105T000000_20220105T235959.csv
Training on file: /content/drive/MyDrive/MonicoData/2022_Files/2022/Export_20220106T000000_20220106T235959.csv
Training on file: /content/drive/MyDrive/MonicoData/2022_Files/2022/Export_20220107T000000_20220107T235959.csv
Training on file: /content/drive/MyDrive/MonicoData/2022_Files/2022/Export_20220108T000000_20220108T235959.csv
Training on file: /content/drive/MyDrive/MonicoData/2022_Files/2022/Export_20220109T000000_20220109T235959.csv
Training on file: /content/drive/MyDrive/MonicoData/2022_Files/2022/Export_20220110T000000_20220110T235959.csv
```

Figure 5: Training Function Output

7. Subsystem Validation

The validation metric chosen for the model focused on its mean absolute error (MAE) in the time till failure and the accuracy on the type of failure (Error code). For the MAE, If the failure was within a couple days time, then the model could reliably predict that it was going to fail but as you stray further away from the date of the next failure, the model has trouble accurately predicting the next failure.

Monico, when presenting the project, mentioned that similar data was given to a company Smart Signal for predictive maintenance but their system could only identify anomalies and could not correlate those with specific failures.

```
Testing complete! Metrics on test set:
Mean Absolute Error (MAE): 48
Training complete.


Testing complete! Metrics on test set:
Accuracy: 18%
Training complete.
 Disconnecting in 10 seconds...
```

Figure 6: Example Test Results

8. Conclusion

The subsystem developed for predicting failures and error codes in Monicos natural gas compressor shows promise. By developing a pipeline for preprocessing, training, and testing the system was able to process and learn from 120 gigabytes of complex time series

data. While the system did not fully meet the requirements, it highlights the complexity of the task and the need for further refinement in model architecture and other hyperparameters. Most importantly, it lays a foundation for actionable predictive maintenance.

9. Future Plans

This task could have been approached many different ways with AI/ML. In the future, it would be worthwhile to take a different approach as opposed to predicting the next failure and achieve better results. More computing power would also allow different types of models to be explored.

MONICO DATA ANALYSIS

Digvijay Alluri
Nicho Naugle
Michael Cubriel

SYSTEM VALIDATION AND INTEGRATION RESULTS

REVISION 1

SYSTEM VALIDATION AND INTEGRATION RESULTS

FOR

Monico Data Analysis

PREPARED BY:

_____

Author Date

APPROVED BY:

_____

Project Leader Date

_____

John Lusher II, P.E. Date

_____

T/A Date

Change Record

Rev.	Date	Originator	Approvals	Description
1.0	4/27/2025	Whole Team	Whole Team	Draft Release

Table of Contents

List of Figures

1. System Validation

1.1. Real-Time Data Ingestion

Sensor data ingestion was successfully implemented and verified. The system ingests real-time sensor data from heavy machinery at one-minute intervals, handling between 186 to 250 variables per machine. Extensive testing confirmed that the data pipeline consistently captured all expected variables without loss or delay. Validation runs ensured that variable names, timestamps, and corresponding sensor values were accurately recorded, enabling reliable downstream processing for analysis and predictive maintenance.

1.2. Data Classification

The system was designed to create single-node averages from the ingested sensor data at one-minute intervals, or at a user-specified frequency. This functionality was thoroughly tested and verified to ensure accurate computation of averages across all selected variables. The averaged data is automatically saved in CSV format, maintaining proper structure and consistency for seamless integration into model training workflows. Validation confirmed that the output files correctly reflected the requested averaging frequency and preserved the integrity of the original sensor data.

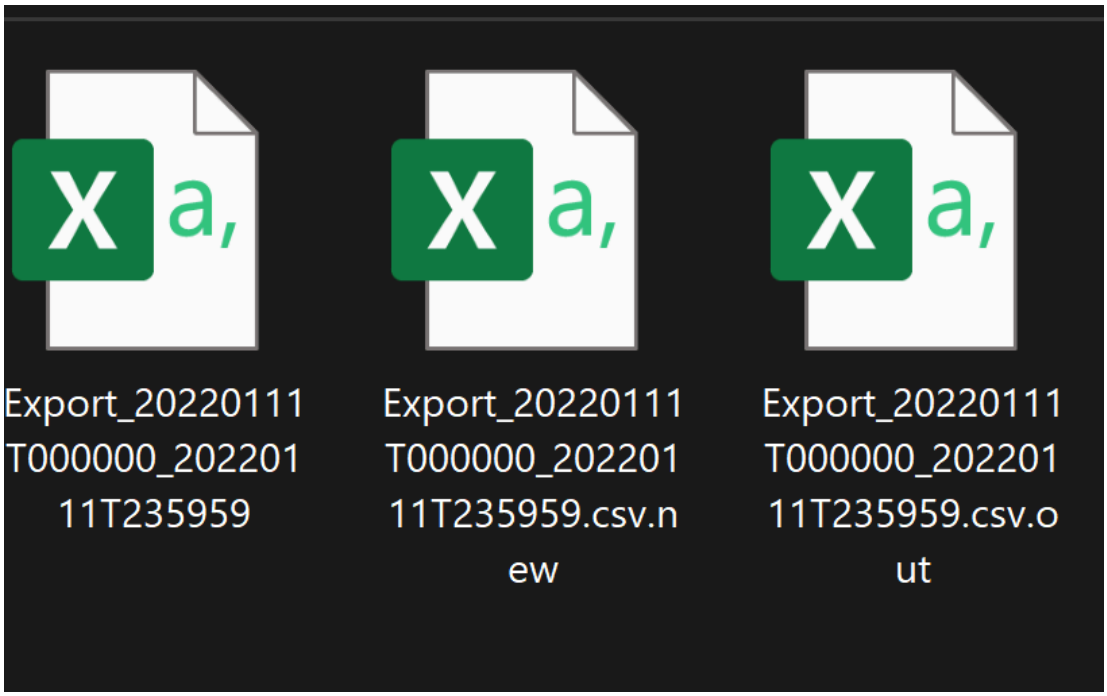


Figure 1: CSV files generated

A	B	C	D	E	F	G	H
Timestamp	0718.1st_S	0718.1st_S	0718.Acce	0718.Actu	0718.Actu	0718.Actu	0718.Afte
2022-01-11T00:00:00	351	349.267	10	0	10	13.4635	
2022-01-11T00:05:00	351	349.407	10	0	10	13.4634	
2022-01-11T00:10:00	351	349.548	10	0	10	13.4632	
2022-01-11T00:15:00	351	349.688	10	0	10	13.463	
2022-01-11T00:20:00	351	349.828	10	0	10	13.4629	
2022-01-11T00:25:00	351	349.968	10	0	10	13.4627	
2022-01-11T00:30:00	351	350.109	10	0	10	13.4625	
2022-01-11T00:35:00	351	350.249	10	0	10	13.4623	
2022-01-11T00:40:00	351	350.389	10	0	10	13.4622	
2022-01-11T00:45:00	351	350.529	10	0	10	13.462	
2022-01-11T00:50:00	351	350.669	10	0	10	13.4618	
2022-01-11T00:55:00	351	350.81	10	0	10	13.4617	

Figure 2: Averaging data

1.3. Random Forest Model

The Random Forest model was validated by splitting the dataset into training and testing sets, with 80% used for training and 20% reserved for testing. Performance was evaluated using Mean Absolute Error (MAE) and R^2 score. The MAE measures the average difference between the predicted and actual time until failure, while the R^2 score assesses how well the model explains the variability of the target variable. Through cross-validation we confirmed that the best performing model on the data set scored around a 73 MAE and 0.51 R^2 score. Feature importance and SHAP analysis further validated that the model was making predictions based on relevant compressor parameters.

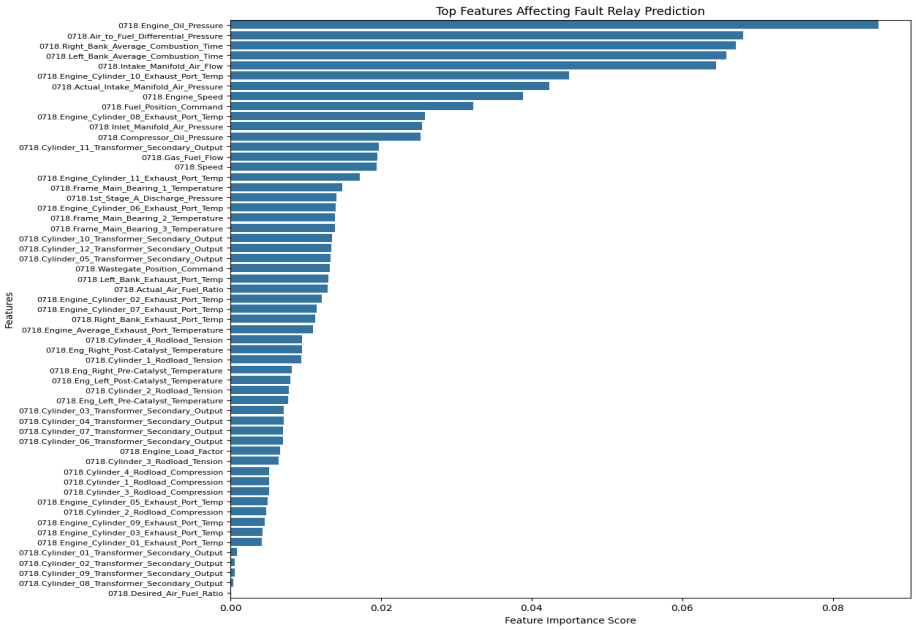


Figure 3: Random Forest Feature Importance

```
Mean Absolute Error (MAE): 73.12912560906295
R² Score: 0.508645349169109 46
```

Figure 4: Best Model Performance Results

1.4. Edge Device Compatibility

The system was successfully integrated with the mCoreSDR device from Monico, achieving forecast operations with a latency of less than one second. All processing, from data ingestion to failure prediction, was performed locally on the device without relying on any cloud services. Testing under various load conditions confirmed that the system consistently met the sub-second latency requirement, ensuring real-time performance and immediate availability of predictive insights directly at the edge.

1.5. Predictive Maintenance

Validation of the predictive maintenance system involved ensuring that the model could accurately forecast compressor failures with sufficient lead time for preventive actions. By evaluating the model on historical data, we verified that it could predict failures within a

reasonable time window, giving operators actionable insights. The model's output was cross-referenced with actual recorded failures to confirm consistency..

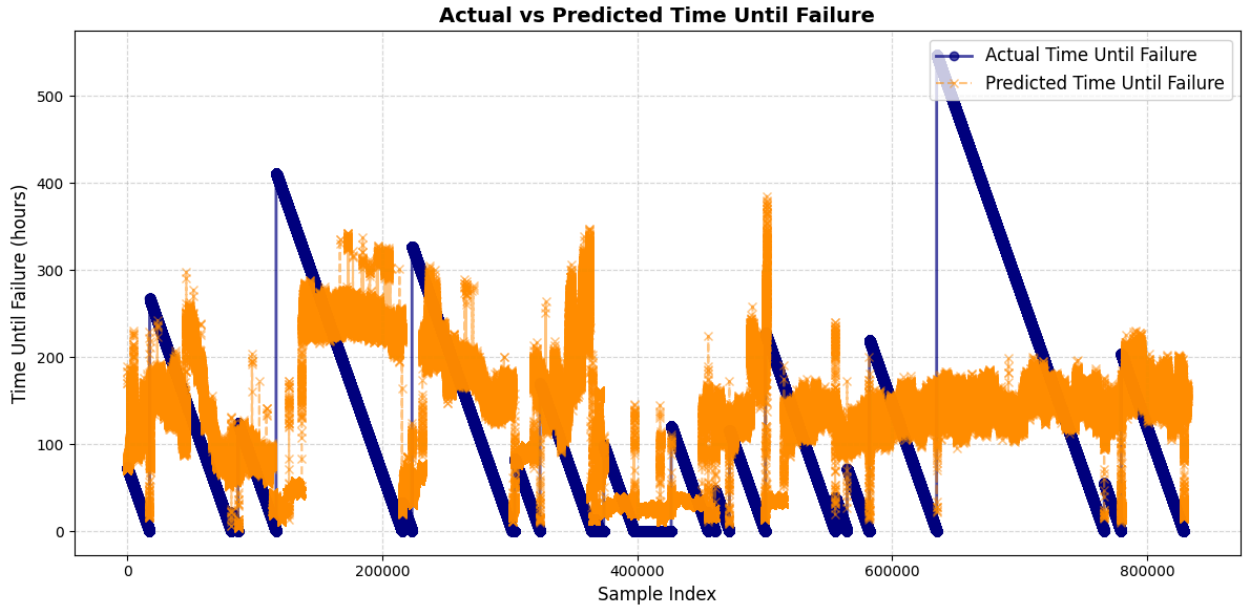


Figure 5: Predicted Vs Actual Time Till Failure Plotted

1.6. Scalability

The system was verified to handle real-time data ingestion and processing from up to 35 machines simultaneously. Dynamic machine management features were implemented, allowing machines to be added or removed from monitoring without interrupting system operations. Extensive testing confirmed stable performance, accurate data handling, and uninterrupted processing even as the number of active machines fluctuated. The system maintained reliable performance across all scenarios, meeting the scalability and flexibility requirements.

1.7. User Interface Integration

The system was verified to handle real-time data updates via websockets and can easily be integrated into an existing user dashboard. Moreover, all HTTP endpoints available on the REST API functioned as expected, allowing for customized streaming intervals including the one minute interval used for validation.

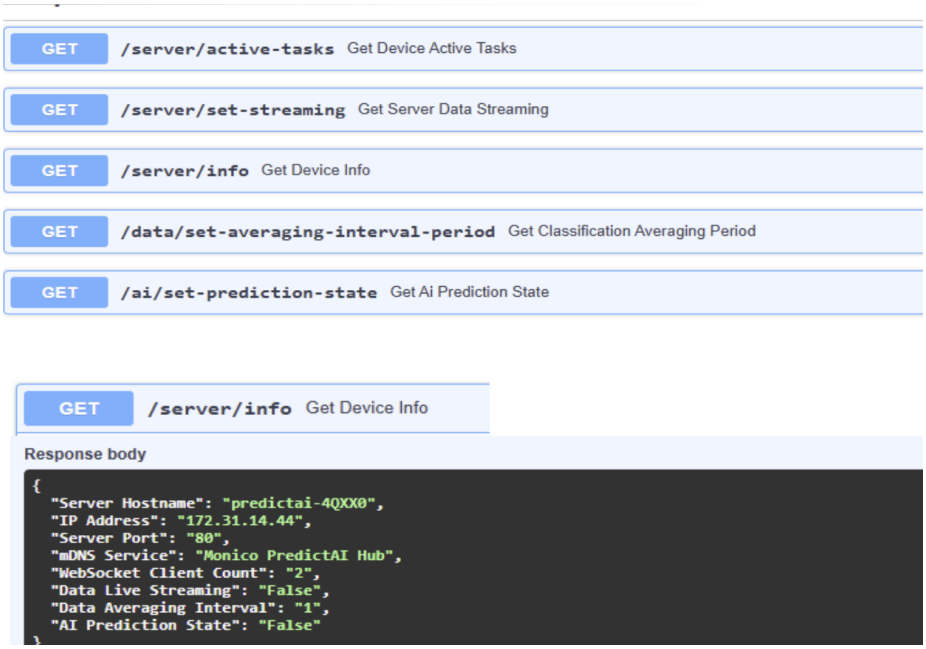


Figure 6: REST API showing server hostname and multiple clients connected

1.8. Data sampling frequency

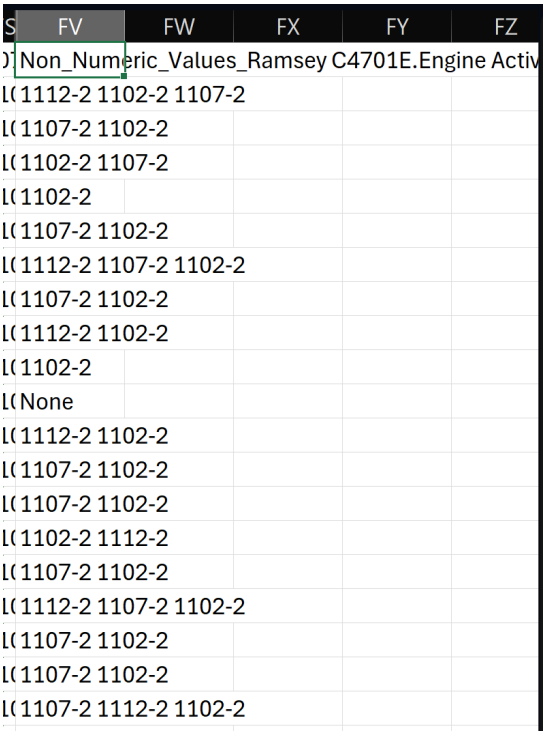
The system was verified to sample data at one minute intervals, for both websockets streaming as well as inference intervals. If the interval was changed then the model would receive the averaged data at that interval.

1.9. Failure prediction

The system was verified to successfully load the machine learning statistical regression model, with averaged data used as inputs. Outputs were successfully returned to the client over websockets.

1.10. Data streams

The system successfully processes between 186 to 250 variables per machine to enable real-time failure prediction and monitoring. Through comprehensive testing, it was confirmed that all variables are consistently handled without bottlenecks or data loss. The processing pipeline accurately cleans, structures, and analyzes incoming data streams, ensuring timely identification of anomalies and providing reliable inputs for the predictive maintenance model. System validation demonstrated that the architecture can sustain real-time operations across multiple machines simultaneously.



FV	FW	FX	FY	FZ
Non_Numeric_Values_Ramsey	C4701E.Engine Activ			
1112-2 1102-2 1107-2				
1107-2 1102-2				
1102-2 1107-2				
1102-2				
1107-2 1102-2				
1112-2 1107-2 1102-2				
1107-2 1102-2				
1112-2 1102-2				
1102-2				
None				
1112-2 1102-2				
1107-2 1102-2				
1107-2 1102-2				
1102-2 1112-2				
1107-2 1102-2				
1112-2 1107-2 1102-2				
1107-2 1102-2				
1107-2 1102-2				
1107-2 1112-2 1102-2				

Figure 7: Active Error Codes

1.11. Predictive window

The predictive window measures how early the model can forecast a failure. We validated it by comparing predicted times until failure against actual failures in the test set. The model provided sufficient lead time for maintenance actions, demonstrating that it could reliably alert operators before critical failures occurred.

1.12. Search Probability of Detection

Probability of detection is how reliably the model can identify failures before they occur. To validate this we assessed how often the model predicted a low time until failure shortly before an actual event. A high probability of prediction indicates that the model detects imminent failures

1.13. Operational Monitoring Range

The system was successfully tested to support real-time monitoring of machinery across a wide data sampling range, from 1 data point per minute up to 60 data points per minute. Validation confirmed that the data pipeline could handle varying sampling rates without loss,

delays, or processing errors. The system dynamically adjusted to different data ingestion rates while maintaining accurate analysis and predictive capabilities, ensuring robust performance across all supported sampling frequencies.

1.14. External Commands

All system code, documentation, configuration files, and supporting resources have been consolidated and maintained within a dedicated GitHub repository. The repository includes complete records of the system architecture, functionality, test results, and user instructions. This organized and version-controlled repository will be formally handed over to Monico, ensuring they have full access to the project deliverables for future deployment, maintenance, and further development.

1.15. Data Output

The system successfully spins up a REST API interface at the local hostname resolved via mDNS. From here all endpoints successfully return data and manage the streaming settings. It can be retrieved by connecting to the same local network and resolving the REST API interface through predictai-xxxxx.local/docs, where xxxxx is the randomly generated sequence used as the local hostname. Please reference section 1.7 for more information.

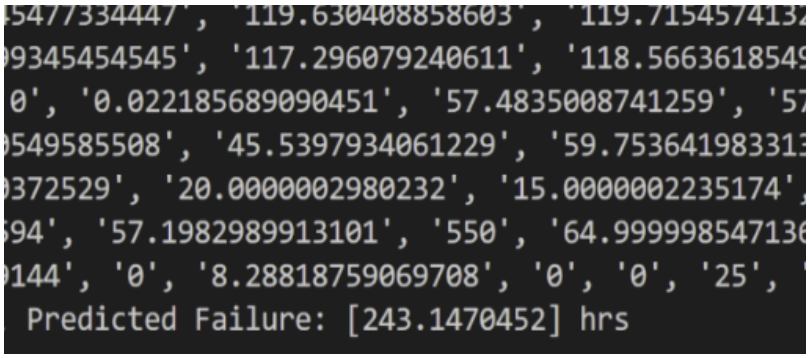


Figure 8: Streamed inference on websocket client interface

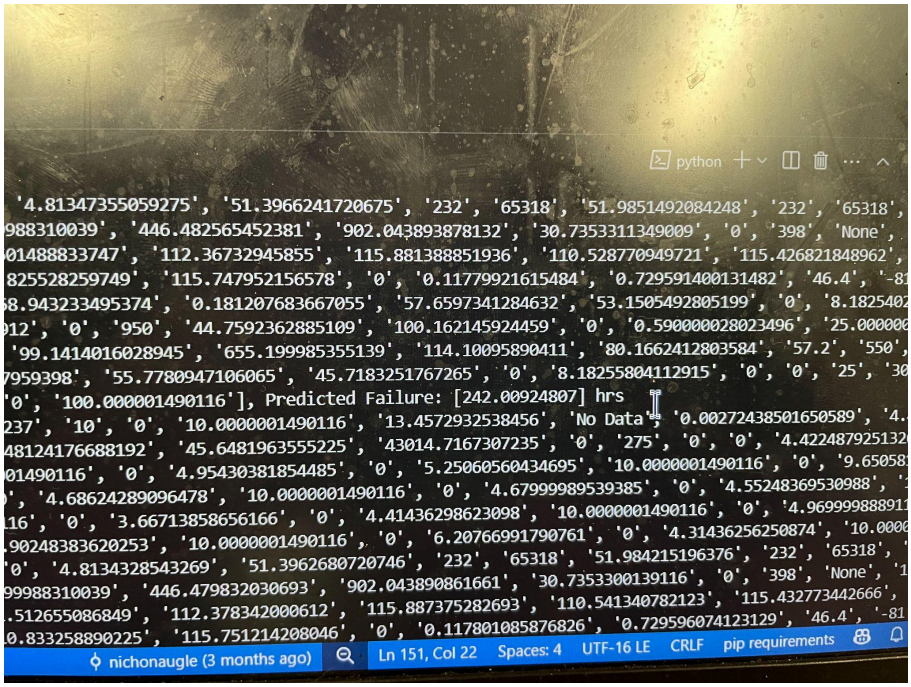


Figure 9: Streamed inference on websocket client interface 2

1.16. Diagnostic Output

A real-time dashboard was developed to provide a clear and intuitive interface for monitoring machine performance, system status, and predictive analytics results. The dashboard aggregates key metrics, trends, and alerts, offering maintenance teams an accessible overview of operational health. It was fully tested for responsiveness, data accuracy, and usability. This completed dashboard will be handed over to Monico along with the rest of the system deliverables to support immediate deployment and future enhancements.

1.17. Built In Test

The system includes a client interface which will scan the local network for predictai services and then connect to the resolved service over websockets to stream the data. A fake data streaming simulator is also included, using pre gathered CSV data stored in the project. As a result, the server is able to pull the data from the csv, average according to the defined interval, and return the inference from the ml model to the clients over the websocket interface. This allows for a complete test of the pipeline from data acquisition, preprocessing, ML inference, and local networking.

1.19. BIT Critical Fault Detection

The Built-In Test (BIT) system was used to simulate real-world streaming of sensor data into the AI model. This allowed us to validate the model's ability to predict failures under live conditions, where data arrives sequentially rather than in a static batch. During BIT simulations, the Random Forest model received continuous feature inputs and produced updated time-to-failure predictions.

1.20. BIT False Alarms

During BIT (Built-In Test) simulations, we also monitored the rate of false alarms, where the model incorrectly predicted an imminent failure despite no actual fault occurring. Managing the false alarm rate is critical to ensure that operators trust the AI predictions and are not overwhelmed by unnecessary maintenance alerts. The model maintained a low false alarm rate during testing, indicating that it could balance sensitivity to real faults without generating excessive false positives.

1.21. BIT Log

The Built-In Test (BIT) functionality was successfully implemented and verified. After each diagnostic test, the results are automatically saved to an Excel file in a structured and readable format. These files are stored within the Monico Data Analysis System, making them easily retrievable by maintenance personnel for review and action. Testing confirmed that all results are accurately logged, securely stored, and accessible for clearing once maintenance tasks are completed, ensuring full compliance with system requirements.

1.22. Isolation and Recovery

In our project, isolation was handled by the Random Forest model predicting when a compressor fault was likely based on incoming sensor data. The predictions were accessed through an API, allowing real-time requests and responses between the data stream and the AI model. Recovery was supported by allowing the system to recognize when operating conditions returned to normal, avoiding unnecessary maintenance actions. Testing showed that the API successfully delivered timely predictions, and the system could isolate true faults while minimizing false alarms during recovery periods.

2. Integration Results

The integration of the core system components—data preprocessing, the machine learning model, and the API interface—proceeded smoothly and efficiently. A key factor in this seamless integration was the design of the API interface, which automatically scans for and loads the TensorFlow model from a predefined location within the project structure. This eliminated complex configuration steps and ensured the model was readily available for inference. Furthermore, the data preprocessing module effectively handled the real-time sensor data, calculating averages based on the user-defined interval specified through the

API. This averaged data was then straightforwardly populated into an array, ready for input into the machine learning model (as validated in sections 1.2 and 1.9). The REST API and websocket interfaces (validated in 1.7, 1.8, and 1.15) provided reliable mechanisms for controlling data flow and retrieving results, confirming the cohesive operation of the entire pipeline from data ingestion to prediction output. The built-in test (BIT) capability (1.17) further demonstrated the successful end-to-end integration by simulating the complete workflow locally.

3. Conclusion

The system validation confirms the successful development and integration of a robust predictive maintenance solution capable of operating directly on edge devices. Key functionalities, including real-time data ingestion from multiple machines (1.1, 1.6, 1.10), accurate data classification and averaging (1.2), and efficient TensorFlow model execution (1.9), were all verified. The system demonstrated exceptional performance on generic hardware, achieving sub-second latency for failure predictions without reliance on cloud services ensuring viability of CPU inference (1.4). Integration with user interfaces via websockets and a comprehensive REST API proved successful (1.7, 1.15), and the system's ability to handle variable data sampling rates (1.8, 1.13) and scale to accommodate numerous machines (1.6) meets critical operational requirements. With comprehensive testing completed and all components functioning as designed, the system is prepared for handover and deployment, providing MONICO a powerful tool for real-time operational monitoring and predictive maintenance.