INTERNSHIP REPORT

ON

Performance Monitoring and Analytics for Aero-Gas Turbine Engines

At

GTRE (DRDO)

Submitted by,

V SAIPRIYA DIPIKA - 20211CSE0178

Under the guidance of,

Dr. Sandeep Albert Mathias

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

At

PRESIDENCY UNIVERSITY

BENGALURU

APRIL 2025

PRESIDENCY UNIVERSITY

PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Internship report **"Performance Monitoring and Analytics for Aero-Gas Turbine Engines"** being submitted by V SAIPRIYA DIPIKA bearing roll number 20211CSE0178 in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Computer Science and Engineering is a Bonafide work carried out under my supervision.

DR. SANDEEP ALBERT MATHIAS
Assistant Professor
PSCS
Presidency University

Dr. ASIF MOHAMMED H.B
Assistant Professor (SG) & HoD
PSCS
Presidency University

Dr. MYDHILI NAIR
Associate Dean
PSCS
Presidency University

Dr. SAMEERUDDIN KHAN
Pro-Vice Chancellor - Engineering
Dean –PSCS / PSIS
Presidency University

PRESIDENCY UNIVERSITY

PRESIDENCY SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

I hereby declare that the work, which is being presented in the report entitled "**Performance Monitoring and Analytics for Aero-Gas Turbine Engines"** in partial fulfillment for the award of Degree of **Bachelor of Technology** in **Computer Science and Engineering**, is a record of my own investigations carried under the guidance of **Dr. Sandeep Albert Mathias, Assistant Professor, Presidency School of Computer Science and Engineering, Presidency University, Bengaluru.**

I have not submitted the matter presented in this report anywhere for the award of any other Degree.

V SAIPRIYA DIPIKA
20211CSE0178

INTERNSHIP COMPLETION CERTIFICATE

I am currently pursuing my internship. The duration of my internship is from 06/02/2025 – 02/05/2025

I will receive my certificate upon completion.

ABSTRACT

This report presents a detailed analysis of aero-gas turbine engine test run data, focusing on evaluating performance through extensive time-series datasets. Utilizing a custom Python application, the study processes thousands of data points per run, tracking module-specific parameters and fixed sensors to identify trends, stagnation, erratic readings, sensor malfunctions, and anomalies. With the help of the **Mistral-7B** language model, the analysis generates precise textual insights, enabling the detection of critical issues that could affect engine reliability. The application supports comparative analysis across multiple runs, handling up to 20,000 data points to uncover subtle performance shifts. Through clear visualizations and structured findings, the report provides actionable recommendations for maintenance and operational decisions, ensuring the safety and efficiency of aerospace systems.

ACKNOWLEDGEMENTS

First of all, we are indebted to **GOD ALMIGHTY** for giving us an opportunity to excel in our efforts to complete this project on time.

We express our sincere thanks to our respected dean **Dr. Md. Sameeruddin Khan**, Pro-VC - Engineering and Dean, Presidency School of Computer Science and Engineering & Presidency School of Information Science, Presidency University for getting us permission to undergo the project.

We express our heartfelt gratitude to our beloved Associate Dean **Dr. Mydhili Nair,** Presidency School of Computer Science and Engineering, Presidency University, and **Dr. Asif Mohammed H.B**, Head of the Department, Presidency School of Computer Science and Engineering, Presidency University, for rendering timely help in completing this project successfully.

We are greatly indebted to our guide **Dr. Sandeep Albert Mathias, Assistant Professor** and Reviewer **Dr. Vishwanath Y, Professor**, Presidency School of Computer Science and Engineering, Presidency University for their inspirational guidance, and valuable suggestions and for providing us a chance to express our technical capabilities in every respect for the completion of the internship work.

We would like to convey our gratitude and heartfelt thanks to the PIP4001 Internship/University Project Coordinator **Mr. Md Ziaur Rahman and Dr. Sampath A K,** department Project Coordinator **Dr. Abdul Khadar A** and Git hub coordinator **Mr. Muthuraj.**

We thank our family and friends for the strong support and inspiration they have provided us in bringing out this project.

V SAIPRIYA DIPIKA

LIST OF FIGURES

TABLE OF CONTENTS

Chapter 1
INTRODUCTION

The analysis of engine performance data is critical for ensuring the reliability and efficiency of aircraft systems. This report details the findings from a comprehensive evaluation of engine run data, focusing on the behavior of key parameters under various operating conditions. By examining large datasets, the study aims to uncover patterns and anomalies that could impact engine performance, providing AI generated actionable insights for maintenance and operational decisions.

The evaluation process centers on processing extensive time-series data collected from engine test runs, each containing thousands of data points. The analysis tracks both module-specific parameters and fixed sensors to assess trends, stability, and potential irregularities. Key metrics such as stagnation periods, erratic readings, and sensor malfunctions are scrutinized to determine the engine's health and operational readiness. This approach ensures that no critical detail is overlooked, offering a complete picture of performance across diverse test scenarios.

A significant aspect of this study is its ability to compare multiple test runs, enabling a deeper understanding of how engine behavior evolves over time or differs between conditions. By aligning data from different runs, the analysis highlights variations in parameter trends and identifies potential issues that may not be evident in a single dataset. This comparative perspective is essential for detecting subtle shifts that could indicate malfunctions, inefficiencies, or emerging faults, ultimately supporting proactive maintenance strategies.

The findings presented in this report are intended to guide engineers and decision-makers in optimizing engine performance and ensuring operational safety. Through detailed visualizations and structured analysis, the study translates complex data into clear, actionable recommendations. Whether confirming normal operation or flagging areas for inspection, the results aim to enhance confidence in engine reliability and contribute to the ongoing improvement of aircraft engines.

Chapter 2
LITERATURE SURVEY

1. Predicting Aircraft Engine Failures using Artificial Intelligence

(Bentaleb, Toumlal and Abouchabaka 2024)

This paper explores the use of AI for **early fault detection in aircraft engines**, analyzing **real-time sensor data** such as **temperature, pressure, and vibration**. The study highlights **pattern recognition techniques** to identify **anomalies and potential failures**, helping to optimize maintenance schedules and prevent unexpected breakdowns. It demonstrates the effectiveness of **machine learning models** in improving **operational efficiency and safety**. However, the approach lacks **interactive visualization tools** that allow users to define analysis parameters, an aspect that my project addresses by enabling **GUI-based data selection and generative AI for insights**.

Relevance to My Work:

- Uses **sensor-driven AI analysis**, similar to my project.
- Focuses on **predictive maintenance and trend detection** for reliability.

Research Gaps:

- No **interactive visualization or user-driven graph generation**.
- Lacks **generative AI for new insights**, limiting automation.

---

2. A Machine-Learning Approach to Assess Aircraft Engine System Performance

(Tong 2020)

This paper investigates **aircraft engine performance assessment using ML models**, focusing on **data-driven anomaly detection and predictive maintenance strategies**. The study emphasizes the role of **sensor-based diagnostics** in optimizing maintenance schedules and improving efficiency. It provides a structured approach to **data collection, preprocessing, and modelling**, ensuring accurate **fault detection and performance tracking**. While effective in engine health monitoring, it does not include **interactive analysis** where users can select parameters dynamically or generate insights through **AI-driven automated interpretation**, both of which are core components of my project.

Relevance to My Work:

- Uses **ML for performance monitoring**.
- Aligns with my project's focus on **sensor-based trend analysis**.

Research Gaps:

- Lacks **GUI-based visualization** for user-defined analysis.
- Does not support **natural language queries for retrieving insights**.

---

3. **Enhancing Predictive Maintenance in the Industrial Sector: A Comparative Analysis of ML Models**

(Levin 2024)

This study evaluates **multiple machine learning models** for predictive maintenance across different industries, analyzing their effectiveness in **fault detection, reducing downtime, and optimizing efficiency**. By comparing various ML approaches, it identifies the most **reliable models** for maintenance forecasting. The paper provides **valuable insights into model performance**, but it does not explore **aerospace-specific applications**.

Relevance to My Work:

- Helps in selecting **effective ML models** for predictive maintenance.
- Provides insight into **performance evaluation of different AI techniques**.

Research Gaps:

- Doesn't explore **aerospace-specific applications**.

4. **Predicting Machine Failures from Multivariate Time Series: An Industrial Case Study**

(Vago, et al. 2024)

This paper focuses on **time-series-based failure prediction**, analyzing **sensor data across multiple time points** to detect **patterns and anomalies**. The study demonstrates how ML techniques can **improve failure prediction accuracy** and optimize **maintenance planning**. It presents a strong foundation for **trend analysis and forecasting**, but does not incorporate **interactive querying** or allow **user-defined parameters for visualization**. My project builds on this by enabling users to **dynamically select data for analysis** and by incorporating **AI-driven responses to user queries**, improving overall system usability.

Relevance to My Work:

- Uses **time-series data analysis** to predict failures.
- Aligns with my project's **focus on performance monitoring**.

Research Gaps:

- No **interactive query-based result retrieval**.
- Lacks **generative AI-based insights** for deeper automation.

---

5. **Advanced ML for Predictive Maintenance: A Case Study on Remaining Useful Life Prediction**

(Meddaoui, Hachmoud and Hain 2024)

This research applies **ML models to estimate the remaining useful life (RUL) of machines**, helping in **scheduling proactive maintenance**. The study highlights **deep learning techniques** that improve **predictive accuracy**, ensuring **early failure detection**. However, the research focuses **only on model efficiency**, lacking **interactive visualization, flexible querying, and AI-driven insight generation**. My project enhances this by **providing a user-friendly interface** where users can define visualization parameters and interact with the AI model using **natural language queries**, allowing for more adaptive and user-controlled analysis.

Relevance to My Work:

- Focuses on **predictive maintenance and lifespan estimation**.
- Similar approach to **analyzing machine performance trends**.

Research Gaps:

- No **interactive visualization or user-defined analysis**.
- Does not use **generative AI for automated conclusions**.

---

6. Machine Learning-Based Fault-Oriented Predictive Maintenance in Industry 4.0

(Justus and Kanagachidambaresan 2024)

This study presents an **ML-based fault classification framework** for **Industry 4.0 applications**, focusing on **automated fault detection and system optimization**. It demonstrates how **fault-oriented models** can improve predictive maintenance but lacks **aerospace-specific applications** and does not explore **flexible user-driven analysis**. My project builds on this by enabling **interactive analysis, dynamic visualizations, and NLP-driven querying**, allowing users to **extract meaningful insights** beyond automated fault classification.

Relevance to My Work:

- Uses **ML for fault detection and predictive maintenance**.
- Aligns with my project's **performance monitoring goals**.

Research Gaps:

- No **interactive visualization or user input** support.
- Lacks **natural language-based querying** for deeper insights.

Chapter 3
PROPOSED METHODOLOGY

This approach integrated data handling, user interaction, AI-driven analysis, and optimization techniques, leveraging a custom Python application and the Mistral-7B language model for offline operation. Below is a detailed outline of the methodology employed:

1. Data Collection and Preprocessing

- **Data Retrieval**: Sensor data from aero-gas turbine engine test runs, each containing up to 10,000 time-series data points, was extracted from a MySQL database using SQL queries. The data included module-specific parameters and fixed sensor readings, identified by run IDs and timestamps.
- **Data Cleaning**: Raw data was processed to address missing values (filled or flagged), outliers (detected via z-scores > 3), and inconsistencies (e.g., timestamp misalignments). This ensured data integrity for analysis.
- **Preprocessing**: Time-series data was standardized by aligning timestamps to a common base date and normalizing parameter values. Data was segmented into relevant channels (temperature or pressure) based on user-selected modules, preparing it for statistical analysis and model input.

2. System Architecture and User Interface Development

- **Application Framework:** A Flask-based web application was developed to serve as the backbone for data processing, visualization, and user interaction. The application interfaced with the MySQL database and the Mistral-7B model, ensuring seamless data flow.
- **User Interface:** A graphical interface allowed users to select test runs, engine modules (e.g., temperature or pressure analysis), and parameters for custom visualizations. The interface supported dynamic plot generation for up to 12 parameters (10 module-specific, 2 fixed sensors).
- **Chatbot Integration:** A natural language processing (NLP) chatbot, powered by Mistral-7B, was embedded to handle user queries. Users could request specific insights

(e.g., "explain the trend of S2_234_Pa" or "compare with previous run") via text input, with responses delivered in real-time.

3. AI-Driven Analysis and Insight Generation

- **Statistical Processing:** The application used pandas and NumPy for initial data analysis, computing trends (upward, downward, stable, erratic), stagnation periods (consecutive points with near-zero differences), and anomalies (outliers and negative values). These metrics were summarized for each parameter to capture the full dataset's behavior.
- **Mistral-7B Analysis:** The Mistral-7B language model, hosted locally and optimized with INT8 quantization, analyzed summarized data and a sampled subset (200 rows for single runs, 100 per run for comparisons) to generate textual insights. The model identified stagnation, erratic readings, sensor malfunctions, and anomalies, formatted as concise reports (e.g., "Stagnation: S2_234_Pa flat 12:03:00-12:04:00").
- **Comparative Analysis:** For comparisons, the system aligned data from two test runs (up to 20,000 data points), computed parameter-wise differences, and used Mistral-7B to highlight variations in trends and anomalies, ensuring a comprehensive evaluation of engine behavior over time.

4. Visualization and Reporting

- **Plot Generation:** Matplotlib was employed to create high-quality time-series plots, down sampling data to ~1,000 points (every 10th point) for visual clarity while preserving trends. Plots displayed up to 12 parameters on dual axes (module parameters and fixed sensors), saved as PNG files at 150 DPI.
- **Insight Delivery:** Analysis results, including Mistral-7B’s textual summaries and visualizations, were presented via the web interface. The chatbot delivered query-specific responses, such as trend explanations or comparative insights, accompanied by relevant plots.
- **Report Structure:** Findings were structured to include stagnation periods, erratic readings, sensor issues, anomalies, and an overall trend assessment (e.g., "normal" or "inspect engine"), providing clear recommendations for maintenance or operational clearance.

5. Optimization and Scalability

- **Performance Optimization:** Accelerated functions were used for outlier and stagnation detection, reducing computation time. Data sampling ensured model input fit within Mistral-7B’s token limit (~4,000 tokens) without compromising full-dataset analysis.
- **Scalability:** The system was designed to handle large datasets (up to 20,000 data points for comparisons) by leveraging efficient data structures and caching summaries to avoid redundant processing.
- **Continuous Improvement:** The framework allowed incorporation of new test run data, enabling the model to refine its understanding of engine behavior over time, though continuous learning was not fully implemented in this phase.

Chapter 4
OBJECTIVES

1. **Data Collection, Preparation, Cleaning, and Preprocessing**

- Retrieve sensor data from a MySQL database for aero-gas turbine engine test runs.
- Clean data by addressing missing values, outliers, and inconsistencies.
- Preprocess time-series data for compatibility with the Mistral-7B model.

2. **User Interface and API Design**

- Provide a web interface to select engine parameters and generate custom visualizations.
- Enable natural language queries via a chatbot for insights and specific visualizations.
- Develop an API for efficient data retrieval and result delivery.

3. **AI Model and Chatbot Integration**

- Use Mistral-7B to detect anomalies, compare trends, and generate engine performance insights.
- Integrate a chatbot to deliver tailored responses based on user queries.
- Optimize the model for accurate offline analysis of large datasets.

4. **Scaling and Optimization for Performance Monitoring**

- Process up to 20,000 data points for single and comparative test run analyses.
- Apply optimization techniques to improve speed and accuracy.
- Support continuous learning from new data to enhance model performance.

Chapter 5
SYSTEM DESIGN & IMPLEMENTATION

The implementation of the Aero-Gas Turbine Engine Data Analysis Project comprises a Flask-based web application that integrates data processing, AI-driven analysis, and user interaction. The system is designed to handle large-scale aero-gas turbine engine test run data, providing insights through visualizations and a chatbot powered by the Mistral-7B language model. Below, we outline the system design for `index.html` and `app.py`, followed by the algorithm governing `app.py`'s core functionality.

System Design:
index.html

The `index.html` file serves as the front-end interface, providing a user-friendly platform for interacting with the analysis system. Designed using HTML, CSS, and JavaScript with Bootstrap for styling, it enables users to select test runs and engine modules, view visualizations, and query insights via a chatbot.

- **Structure**:
  - **Header**: Displays the project title and navigation options.
  - **Input Form**: Contains fields for entering a run ID (e.g., V12B34R1233) and selecting a module (e.g., "Compressor: temperature analysis") from a dropdown populated with available modules.
  - **Visualization Area**: Displays time-series plots for selected parameters (up to 12, including 10 module-specific and 2 fixed sensors) as PNG images, with dual axes for module and fixed sensor data.
  - **Chatbot Interface**: Includes a text input for natural language queries (e.g., "compare with previous run" or "explain the trend of S2_234_Pa") and a response area for textual insights and additional plots.
  - **Results Section**: Shows analysis outputs, including Mistral-7B-generated textual summaries (e.g., stagnation, erratic readings, anomalies) and the database table name.

- **Interactivity**:
  - **Form Submission:** JavaScript handles form submissions, sending run ID and module selections to the /generate_plot endpoint via AJAX (POST request) to retrieve plots and analysis results.
  - **Chatbot Queries:** JavaScript captures user queries, sends them to the /chat endpoint, and displays responses (text and optional plots) dynamically**.**
  - **Dynamic Updates:** The interface updates visualizations and text outputs without page reloads, using jQuery for DOM manipulation.
- **Styling**:
  - Bootstrap ensures a responsive, clean layout with consistent styling for buttons, forms, and output sections.
  - Custom CSS adjusts plot sizing, text formatting, and chatbot response alignment for clarity.

app.py

The `app.py` file is the back-end core, implemented in Python using Flask to manage data retrieval, processing, analysis, visualization, and user interactions. It integrates with a MySQL database, statistical libraries, and the Mistral-7B model for offline operation.

- **Structure**:
  - **Imports and Setup**: Includes Flask for web routing, MySQL Connector for database access, pandas/NumPy for data processing, Matplotlib for plotting, and transformers/bitsandbytes for Mistral-7B.
  - **Configuration:** Defines database credentials, fixed sensors (IDs 3 and 9), temporary plot directory, and loads module/channel mappings from JSON files.
  - **Global State:** Maintains current_run_data (run ID, module, table name, DataFrame, summary cache) for session persistence.
  - **Model Loading:** Initializes Mistral-7B with INT8 quantization for efficient offline inference.

- **Endpoints**:
  - `/`: Renders index.html with a list of module options (e.g., "Compressor (temp analysis)", "Turbine (pressure analysis)") derived from module-info.json.
  - **`/generate_plot:`** Accepts run ID and module, retrieves data, generates plots and Mistral-7B analysis, and returns JSON with plot image (base64), textual insights, and table name.
  - **`/chat:`** Processes natural language queries (e.g., trend analysis, comparisons), invokes Mistral-7B for responses, and returns JSON with text and optional plots.

- **Key Functions**:
  - **`get_db_connection`**: Establishes a MySQL connection.
  - **`extract_table_name`**: Parses run ID to derive table name (e.g., V12B34 from V12B34R1233).
  - **`generate_query_data`**: Queries database for run-specific data, constructs a pandas DataFrame with timestamps and parameters.
  - **`summarize_data`**: Computes trends, outliers, stagnation, missing data, and anomalies for all data points.
  - **`generate_plot_image`**: Creates time-series plots (downsampled to ~1,000 points) for up to 12 parameters, saved as base64-encoded PNGs.
  - **`analyze_with_Mistral`**: Uses Mistral-7B to generate textual insights from summaries and sampled data (200 rows for single runs).
  - **`chat_with_Mistral`**: Handles chatbot queries for trend explanations or comparisons (100 rows per run), invoking Mistral-7B for responses.
- **Dependencies**:
  - Python 3.12, Flask, MySQL Connector, pandas, NumPy, Matplotlib, transformers, bitsandbytes.
  - JSON configuration files (`module-info.json`, `channel-mapping.json`) for module and channel mappings.

Algorithm for app.py

The algorithm outlines the core logic of `app.py`, detailing how it processes requests, analyzes data, and delivers results. It operates in two primary modes: **plot generation and chatbot interaction.**

1. **Initialization**:
   - Load Mistral-7B model with INT8 quantization for offline inference.
   - Initialize Flask app, database credentials, and temporary plot directory.
   - Load module and channel mappings from JSON files.
   - Set fixed sensor IDs and initialize `current_run_data`.
2. **Root Endpoint (`/`)**:
   - Generate module options by combining module names from `module-info.json` with analysis types (temperature, pressure).
   - Render `index.html` with module options for user selection.
3. **Plot Generation (`/generate_plot`)**:
   - **Input**: Receive JSON with `run_id` (e.g., V12B34R1233) and `selected_module` (e.g., "Compressor: temperature analysis)") via POST.
   - **Validation**: Check for valid run ID and module; return error if invalid.
   - **Data Retrieval**:
     - Extract table name from run ID (e.g., V12B34).
     - Query MySQL database for run-specific data (Timestamp, run_date, module parameters, fixed sensors).
     - Construct pandas DataFrame with up to 10,000 rows.
   - **Preprocessing**:
     - Clean data (handle missing values, outliers).
     - Align timestamps to base date.
   - **Analysis**:
     - Summarize full dataset: compute trends (upward, downward, stable, erratic), outliers (z-score > 3), stagnation (near-zero diffs), missing data, and anomalies (negative values, outliers).
     - Sample 200 rows (prioritizing outliers/anomalies) for Mistral-7B input.

        - Invoke Mistral-7B with summary and sampled data to generate textual insights (stagnation, erratic readings, malfunctions, anomalies, overall trend).
    - **Visualization**:
        - Downsample data to ~1,000 points (every 10th point).
        - Generate Matplotlib plot for up to 12 parameters (10 module-specific, 2 fixed sensors) on dual axes.
        - Save plot as PNG (150 DPI), encode as base64.
    - **Output**: Return JSON with base64 plot, Mistral-7B insights, and table name.
    - **State Update**: Store run ID, module, table name, DataFrame, and summary in `current_run_data`.
4. **Chatbot Interaction (`/chat`)**:
    - **Input**: Receive JSON with `query` (e.g., "compare with previous run", "explain the trend of S2_234_Pa") via POST.
    - **Query Parsing**:
        - Identify query type: trend explanation, comparison (previous or specific run), or invalid.
    - **Trend Explanation**:
        - If query matches "explain the trend of ":
            - Validate column in `current_run_data["df"]`.
            - Compute trend (erratic, upward, downward, stable) and alignment with fixed sensors.
            - Generate plot for single column.
            - Return JSON with textual explanation and base64 plot.
    - **Comparison**:
        - If query involves comparison:
            - Determine comparison run ID (previous or user-specified).
            - Retrieve comparison data (up to 10,000 rows) from database.
            - Summarize both datasets (current and comparison).
            - Sample 100 rows per run (total 200), prioritizing outliers/anomalies.
            - Invoke Mistral-7B with summaries and sampled data to compare trends, stagnation, anomalies, etc.

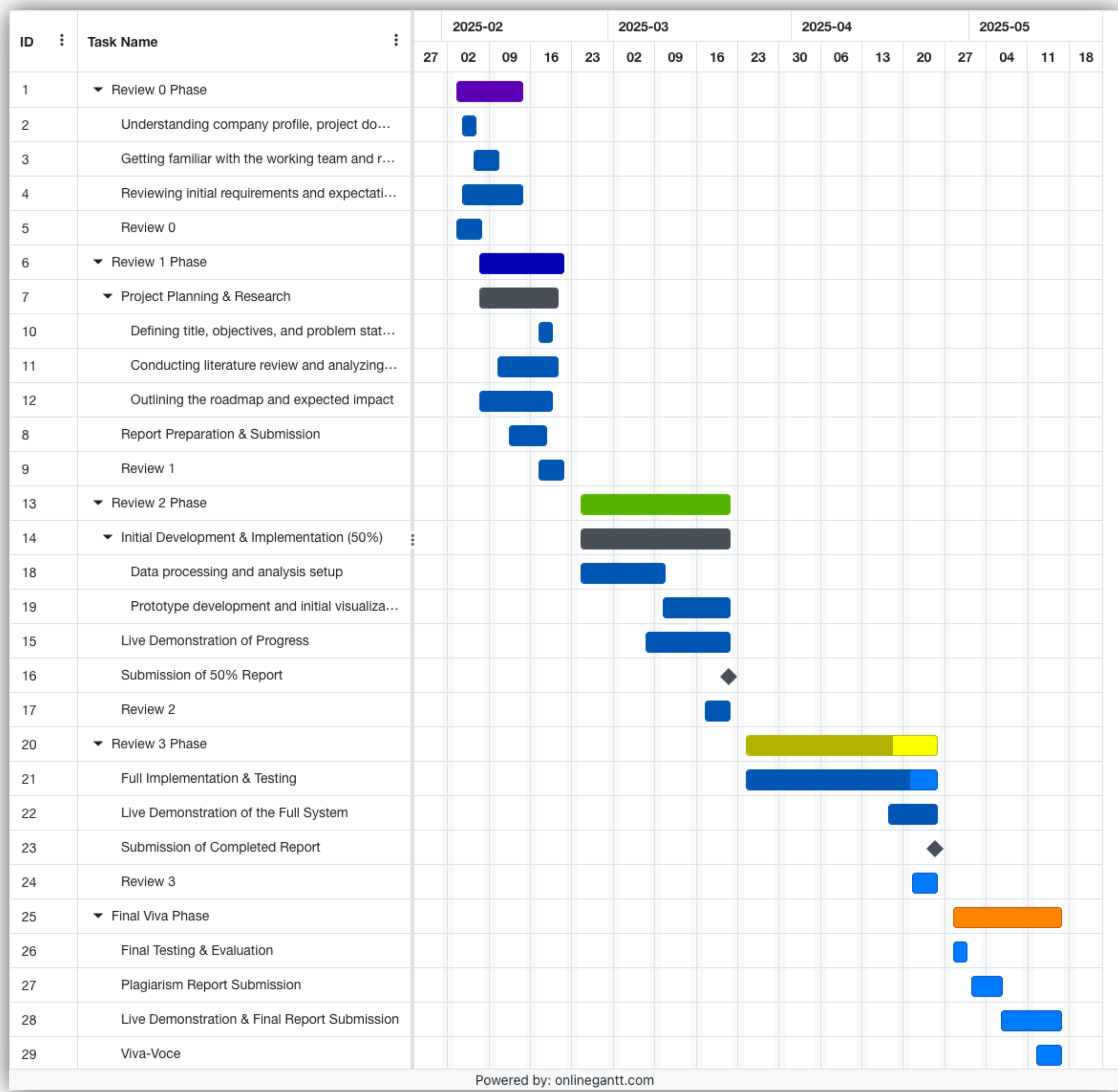
        - Generate comparison plot.
        - Return JSON with textual comparison and base64 plot.
    - **Invalid Query**: Return error message if query is unrecognized.
    - **Error Handling**: Return errors for missing data, invalid run IDs, or model failures.

5. **Optimization**:
    - Split the model across CPU and GPU for computations such as outlier and stagnation detection to reduce computation time.
    - Cache summaries in `current_run_data` to avoid redundant processing.
    - Down sample plotting data to maintain visual clarity and performance.
    - Sample model input to fit Mistral-7B's token limit (~4,000 tokens).
6. **Output Delivery**:
    - Serve results via Flask endpoints, updating the web interface dynamically.
    - Ensure plots and insights are formatted for clarity (e.g., timestamps in HH:MM:SS, structured textual reports).

Chapter 6

TIMELINE FOR EXECUTION OF PROJECT

(GANTT CHART)



GITHUB LINK

https://github.com/saipriya-dipika/AI-ML-Internship

Chapter 7
OUTCOMES

- **Data Analysis**: Successfully processed and analyzed up to 10,000 data points per test run, identifying trends, stagnation periods, erratic readings, sensor malfunctions, and anomalies in engine performance.
- **Scalable Comparative Insights**: Enabled comparison of multiple test runs (up to 20,000 data points), detecting subtle performance shifts and potential faults across different operating conditions.
- **AI-Driven Insights**: Utilized the Mistral-7B language model to generate precise, textual summaries of engine health
- **User-Friendly Interface**: Delivered a Flask-based web interface for dynamic selection of engine modules and parameters, providing high-quality visualizations of up to 12 parameters (10 module-specific, 2 fixed sensors).
- **Interactive Chatbot**: Implemented a natural language chatbot powered by Mistral-7B, allowing users to query specific trends, comparisons, or anomalies with tailored responses and visualizations.
- **Optimized Performance**: Utilized memory optimization techniques for fast outlier and stagnation detection, ensuring efficient processing of large datasets while maintaining analytical accuracy.
- **Offline Operation**: Achieved fully offline functionality, integrating local Mistral-7B inference and MySQL database access, suitable for secure aerospace environments.
- **Actionable Recommendations**: Provided structured outputs (e.g., "normal" or "inspect engine") with detailed justifications, supporting engineers in optimizing engine reliability and safety.
- **Enhanced Visualization**: Generated clear, dual-axis time-series plots (down sampled to ~1,000 points for clarity) to visualize parameter trends and anomalies, aiding decision-making.
- **Continuous Improvement Framework**: Established a system capable of incorporating new test run data, laying the foundation for ongoing refinement of engine performance analysis.

Chapter 8
RESULTS AND DISCUSSIONS

Results

- **Data Analysis**: Processed up to 10,000 data points per test run, identifying trends, stagnation, erratic readings, sensor issues, and anomalies for aero-gas turbine engines.
- **Comparative Insights**: Analyzed up to 20,000 data points across multiple runs, detecting performance shifts and potential faults.
- **AI Insights**: Mistral-7B generated clear reports, e.g., "Anomalies: S3_567_Psi spikes to 5.2 at 12:02:15," with maintenance recommendations.
- **Visualizations**: Produced dual-axis plots for up to 12 parameters, down sampled to ~1,000 points for clarity.
- **User Interface**: Web interface and chatbot enabled easy module selection, plot generation, and natural language queries.
- **Performance**: Optimized functions and data sampling ensured fast processing of large datasets offline.

Discussion

The system effectively analyzed engine performance, meeting the need for quick and scalable and insights. Mistral-7B’s reports were clear, aiding quick maintenance decisions, though token limits required careful data sampling. The web interface and chatbot made analysis accessible, but the chatbot’s query flexibility could improve. Optimizations sped up processing, and offline operation. Minor challenges included plot down sampling. The system has potential for future enhancements in query handling and adaptive sampling.

Chapter 9
CONCLUSION

This project built a solid tool for analyzing aero-gas turbine engine data, handling up to 10,000 data points per test run and 20,000 when comparing runs. Using a web interface, fast data processing, and the Mistral-7B model, it delivered clear findings on engine performance, spotting issues like anomalies or sensor problems to help with maintenance and keep operations safe. The interface and chatbot made it easy to use, though the chatbot could handle more varied questions. Plots worked well, and were plotted in high quality. Overall, the system did what it intended, giving a quick and user-friendly way to check engine health. With some tweaks, like better plot details or smarter query handling, it could be even more useful in the future.

REFERENCES

1. Bentaleb, Asmae, Kaoutar Toumlal, and Jaafar Abouchabaka. 2024. "Predicting Aircraft Engine Failures using Artificial Intelligence." *International Journal of Advanced Computer Science & Applications.* 944.
2. Tong, Michael T. 2020. "A Machine-Learning Approach to Assess Aircraft Engine System Performance." *ASME Turbo Expo 2020: Turbomachinery Technical Conference and Exposition* . Research Gate. V001T01A022.
3. Levin, Semen. 2024. "Enhancing predictive maintenance in the industrial sector: A comparative analysis of machine learning models." *AIP Conference Proceedings* (AIP Publishing) 3243.
4. Vago, Pinciroli, Nicol{\`o} Oreste, Francesca Forbicini, and Piero Fraternali. 2024. "Predicting Machine Failures from Multivariate Time Series: An Industrial Case Study." *Machines* (MDPI) 12: 357.
5. Meddaoui, Anwar, Adil Hachmoud, and Mustapha Hain. 2024. "Advanced ML for predictive maintenance: a case study on remaining useful life prediction and reliability enhancement." *The International Journal of Advanced Manufacturing Technology* (Springer) 132: 323-335.
6. Justus, Vivek, and G. R. Kanagachidambaresan. 2024. "Machine learning based fault-oriented predictive maintenance in industry 4.0." *International Journal of System Assurance Engineering and Management* (Springer) 15: 462-474.

APPENDIX-A

PSUEDOCODE

app.py:

```
# Global Variables
CONFIGURE database credentials (host, user, password, database)
SET fixed_sensors = ["x", "y"]
INITIALIZE temp_plot_directory
LOAD module_info from "module-info.json"
LOAD channel_mapping from "channel-mapping.json"
INITIALIZE current_run_data = {run_id: null, selected_module: null, table_name: null, df:
null, summary_cache: {}}
LOAD Mistral-7B model with INT8 quantization for offline use
INITIALIZE Flask app

# Helper Function: Connect to Database
FUNCTION get_db_connection()
  TRY
    CONNECT to MySQL database using credentials
    RETURN connection
  CATCH error
    LOG error
    THROW exception
  ENDTRY
ENDFUNCTION

# Helper Function: Extract Table Name from Run ID
FUNCTION extract_table_name(run_id)
  PARSE run_id to extract table prefix (e.g., "V12B34" from "V12B34R1233")
  IF valid
    RETURN table_name
  ELSE
    RETURN null
  ENDIF
```

```
ENDFUNCTION

# Helper Function: Get Previous Run ID
FUNCTION get_previous_run_id(current_run_id)
    PARSE current_run_id to extract table_prefix and run_number
    DECREMENT run_number by 1
    RETURN formatted run_id (e.g., "V12B34R1232")
ENDFUNCTION

# Helper Function: Detect Outliers
FUNCTION detect_outliers(values)
    IF length(values) < 2
        RETURN empty list
    ENDIF
    COMPUTE mean and standard_deviation of values
    IF standard_deviation = 0
        RETURN empty list
    ENDIF
    COMPUTE z_scores = abs((values - mean) / standard_deviation)
    RETURN indices where z_scores > 3
ENDFUNCTION

# Helper Function: Detect Stagnation
FUNCTION detect_stagnation(diffs, threshold=1e-5)
    RETURN indices where abs(diffs) < threshold
ENDFUNCTION

# Helper Function: Summarize Data
FUNCTION summarize_data(dataframe)
    INITIALIZE summaries = {}
    FOR each column in dataframe (excluding Timestamp)
        IF column has no valid data
```

```
            SET summaries[column] = {trend: "no data", outliers: [], stagnation: [], missing: [],
anomalies: [], stats: {min: null, max: null, mean: null}}
            CONTINUE
        ENDIF
        CONVERT column values to float array
        COMPUTE differences = diff(values)
        COMPUTE mean_diff and std_diff of differences
        DETERMINE trend:
            IF std_diff > 2 * abs(mean_diff)
                SET trend = "erratic"
            ELSE IF mean_diff > 0
                SET trend = "upward"
            ELSE IF mean_diff < 0
                SET trend = "downward"
            ELSE
                SET trend = "stable"
            ENDIF
        COMPUTE outlier_indices = detect_outliers(values)
        COMPUTE stagnant_indices = detect_stagnation(differences)
        IDENTIFY stagnant_periods (start, end indices and timestamps)
        IDENTIFY missing_data_ranges (start, end indices and timestamps)
        COMPUTE anomalies (outliers and negative values with indices, values, timestamps)
        COMPUTE stats = {min: min(values), max: max(values), mean: mean(values)}
        SET summaries[column] = {trend, outliers, stagnation, missing, anomalies, stats}
    ENDFOR
    RETURN summaries
ENDFUNCTION

# Helper Function: Fetch Data from Database
FUNCTION generate_query_data(run_id, module_name, channel_mapping, module_info)
    EXTRACT table_name from run_id
    IF table_name invalid
        LOG error
```

```
      RETURN null
   ENDIF
   CONNECT to database
   TRY
      GET all columns from table_name (excluding Timestamp, run_id, run_date)
      PARSE module_name to get module and analysis_type (temp or pressure)
      SELECT relevant_columns based on module_info and channel_mapping (temperature
or pressure channels)
      ADD fixed_sensor_columns (sensors 3 and 9)
      QUERY database: SELECT Timestamp, run_date, relevant_columns WHERE run_id =
run_id
      IF no rows returned
         LOG error
         RETURN null
      ENDIF
      CONSTRUCT DataFrame with Timestamp and parameter columns
      ALIGN timestamps to base_date
      RETURN DataFrame
   CATCH error
      LOG error
      RETURN null
   FINALLY
      CLOSE database connection
   ENDTRY
ENDFUNCTION

# Helper Function: Generate Plot
FUNCTION generate_plot_image(dataframe, run_id, selected_module, single_col=null)
   PARSE selected_module to get module_name and analysis_type
   IF single_col specified
      SET columns_to_plot = [single_col]
   ELSE
```

```
        SELECT module_columns based on module_name and analysis_type (temperature or
pressure)
        ADD fixed_sensor_columns (sensors 3 and 9)
    ENDIF
    DOWNSAMPLE dataframe to ~1,000 points (every 10th point) for plotting
    IF dataframe empty
        CREATE placeholder plot with test data
    ELSE
        CREATE dual-axis plot
        PLOT module_columns on primary axis with distinct colors
        PLOT fixed_sensor_columns on secondary axis with distinct style
        SET labels, title, grid, and timestamp formatting
    ENDIF
    SAVE plot as PNG (150 DPI) to temp_directory
    CONVERT plot to base64 string
    DELETE temporary plot file
    RETURN base64 string
ENDFUNCTION

# Helper Function: Analyze Data with Mistral-7B
FUNCTION analyze_with_mistral(dataframe, selected_module)
    IF dataframe empty
        RETURN "Error: No data to analyze"
    ENDIF
    GET timestamps from dataframe
    GET data_columns (excluding Timestamp)
    COMPUTE summaries = summarize_data(dataframe)
    SET max_rows = 200
    COLLECT critical_indices (outliers and anomalies from summaries)
    IF length(timestamps) <= max_rows
        SET sampled_indices = all indices
    ELSE
        SET sampled_indices = critical_indices
```

```
        IF length(sampled_indices) > max_rows
            TRIM sampled_indices to max_rows
        ELSE
            ADD regular_indices (evenly spaced, excluding critical_indices) to reach max_rows
        ENDIF
        SORT sampled_indices
    ENDIF
    EXTRACT sampled_timestamps and sampled_data for sampled_indices
    FORMAT sampled_data as CSV table (Timestamp, columns)
    IDENTIFY fixed_columns (sensors 3 and 9)
    IDENTIFY module_columns (excluding fixed_columns)
    IF fixed_columns missing
        RETURN "Error: Fixed sensors not found"
    ENDIF
    CONSTRUCT prompt:
        - Include module_name, total rows, sampled table
        - Specify fixed and module parameters
        - Include full summaries (JSON)
        - Request analysis for stagnation, erratic readings, malfunctions, anomalies
        - Specify output format with example
    TRY
        RUN Mistral-7B with prompt (max_tokens=800, temperature=0.7, top_p=0.9)
        EXTRACT response (excluding prompt)
        RETURN response or "No analysis generated"
    CATCH error
        LOG error
        RETURN "Error: Analysis failed"
    ENDTRY
ENDFUNCTION

# Helper Function: Handle Chatbot Queries
FUNCTION chat_with_mistral(query)
    IF current_run_data incomplete
```

```
        RETURN {response: "No current run data", comp_image: null}
    ENDIF
    PARSE query
    IF query contains "compare"
        DETERMINE comparison_run_id (previous or specified)
        FETCH comparison_data = generate_query_data(comparison_run_id, current_module)
        IF comparison_data null
            RETURN {response: "Comparison data not available", comp_image: null}
        ENDIF
        COMPUTE current_summaries = summarize_data(current_dataframe)
        COMPUTE comparison_summaries = summarize_data(comparison_dataframe)
        SAMPLE 100 rows per run (prioritizing outliers/anomalies)
        FORMAT sampled data as CSV tables
        IDENTIFY fixed and module parameters for both runs
        CONSTRUCT comparison prompt:
            - Include run IDs, row counts, sampled tables
            - Specify fixed and module parameters
            - Include summaries
            - Request comparison of trends, stagnation, anomalies
        RUN Mistral-7B with prompt
        GENERATE comparison_plot = generate_plot_image(comparison_dataframe,
comparison_run_id, current_module)
        RETURN {response: comparison_analysis, comp_image: comparison_plot}
    ELSE IF query matches "explain the trend of <column>"
        IF column exists in current_dataframe
            COMPUTE trend = analyze_trend(timestamps, column_values)
            CHECK alignment with fixed sensors
            GENERATE single_column_plot = generate_plot_image(current_dataframe,
current_run_id, current_module, column)
            RETURN {response: trend_explanation, comp_image: single_column_plot}
        ELSE
            RETURN {response: "Column not found", comp_image: null}
        ENDIF
```

```
    ELSE
        RETURN {response: "Query not recognized", comp_image: null}
    ENDIF
ENDFUNCTION

# Helper Function: Analyze Trend
FUNCTION analyze_trend(timestamps, values)
    IF values empty or all null
        RETURN "No data available"
    ENDIF
    IF length(values) < 2
        RETURN "Insufficient data"
    ENDIF
    COMPUTE differences = diff(values)
    COMPUTE mean_diff and std_diff
    GET start_time and end_time from timestamps
    IF std_diff > 2 * abs(mean_diff)
        RETURN "erratic from start_time to end_time"
    ELSE IF mean_diff > 0
        RETURN "upward from start_time to end_time"
    ELSE IF mean_diff < 0
        RETURN "downward from start_time to end_time"
    ELSE
        RETURN "stable from start_time to end_time"
    ENDIF
ENDFUNCTION

# Route: Root Endpoint
ROUTE "/" (GET)
    GENERATE module_options from module_info (combine with temp/pressure analysis)
    RENDER index.html with module_options
ENDROUTE
```

```
# Route: Generate Plot
ROUTE "/generate_plot" (POST)
   GET run_id and selected_module from JSON request
   IF run_id or selected_module missing
      RETURN JSON {message: "Provide run ID and module", image: null, comments: null,
table: null}
   ENDIF
   EXTRACT table_name from run_id
   IF table_name invalid
      RETURN JSON {message: "Invalid run ID", image: null, comments: null, table: null}
   ENDIF
   FETCH dataframe = generate_query_data(run_id, selected_module, channel_mapping,
module_info)
   IF dataframe null
      RETURN JSON {message: "No data for run", image: null, comments: null, table: null}
   ENDIF
   UPDATE current_run_data with run_id, selected_module, table_name, dataframe,
summarize_data(dataframe)
   GENERATE plot_image = generate_plot_image(dataframe, run_id, selected_module)
   GENERATE comments = analyze_with_mistral(dataframe, selected_module)
   RETURN JSON {image: plot_image, comments: comments, table: table_name}
ENDROUTE

# Route: Chatbot Interaction
ROUTE "/chat" (POST)
   GET query from JSON request
   IF query empty
      RETURN JSON {response: "Enter a query", comp_image: null}
   ENDIF
   COMPUTE result = chat_with_mistral (query)
   RETURN JSON {response: result.response, comp_image: result.comp_image}
ENDROUTE
```

```
# Main Execution
INITIALIZE logging (DEBUG level, output to file and console)
TRY
   START Flask server (port=5000, debug mode)
CATCH error
   LOG error
   THROW exception
ENDTRY
```

APPENDIX-B

SCREENSHOTS

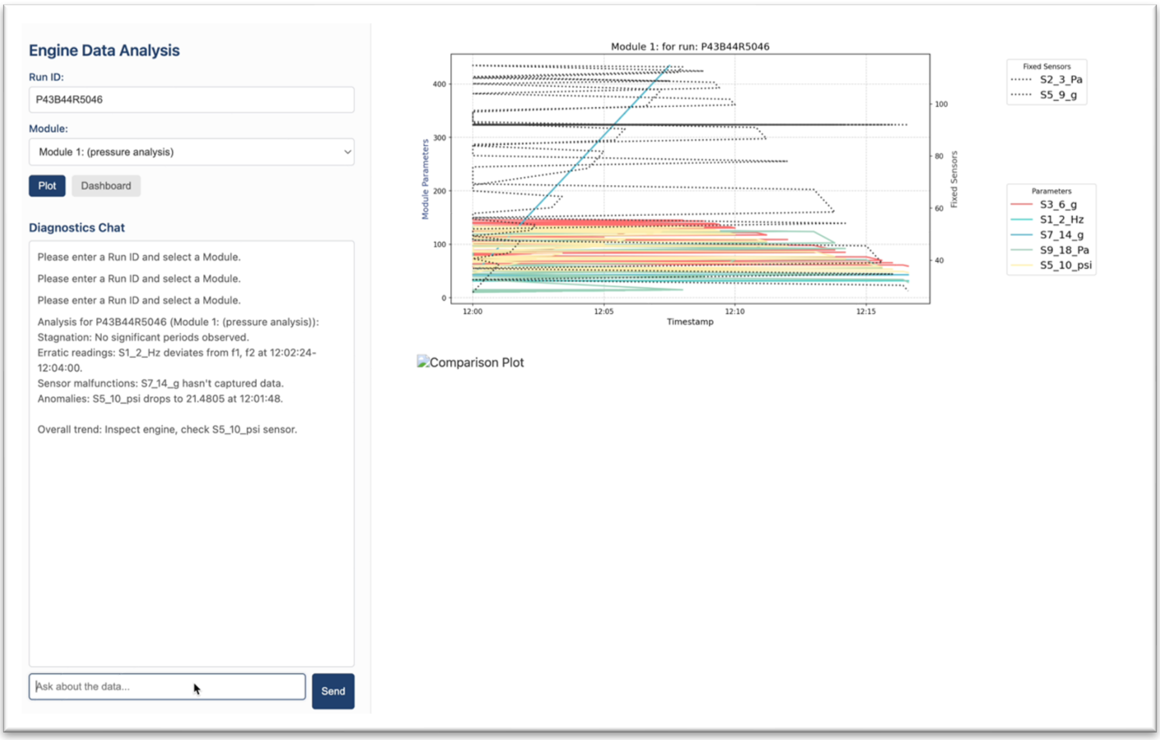


Figure 1: Run Analysis

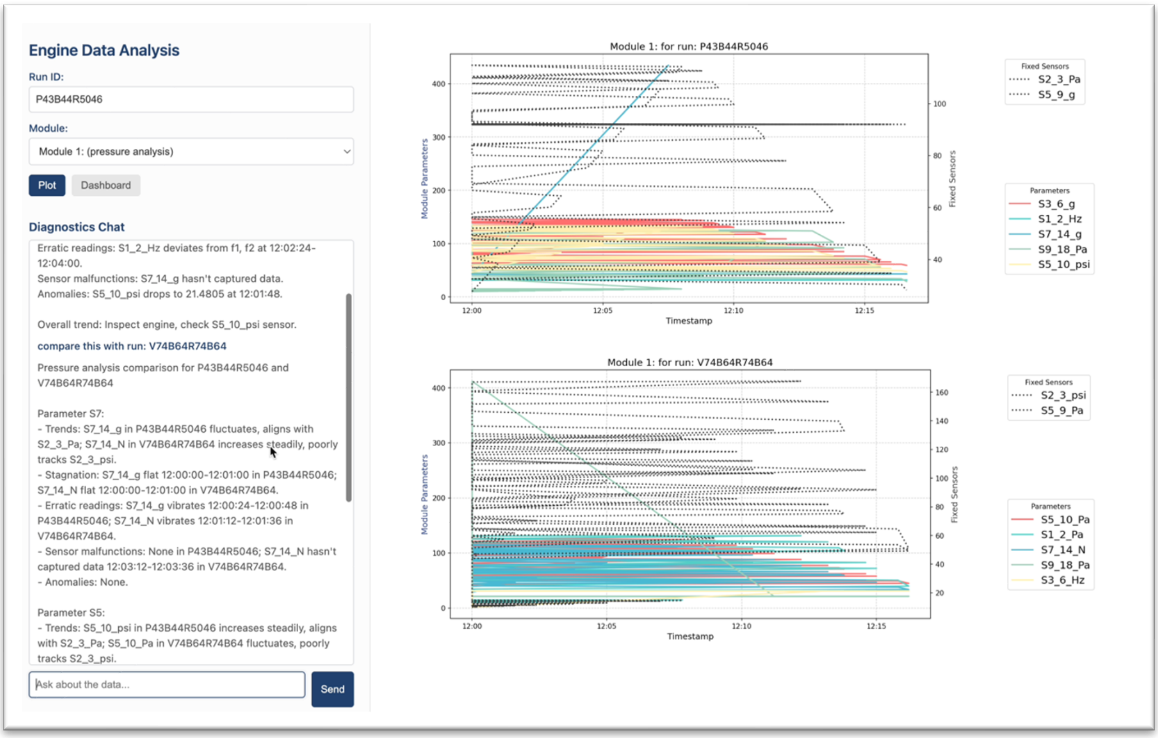


Figure 2: Comparison query

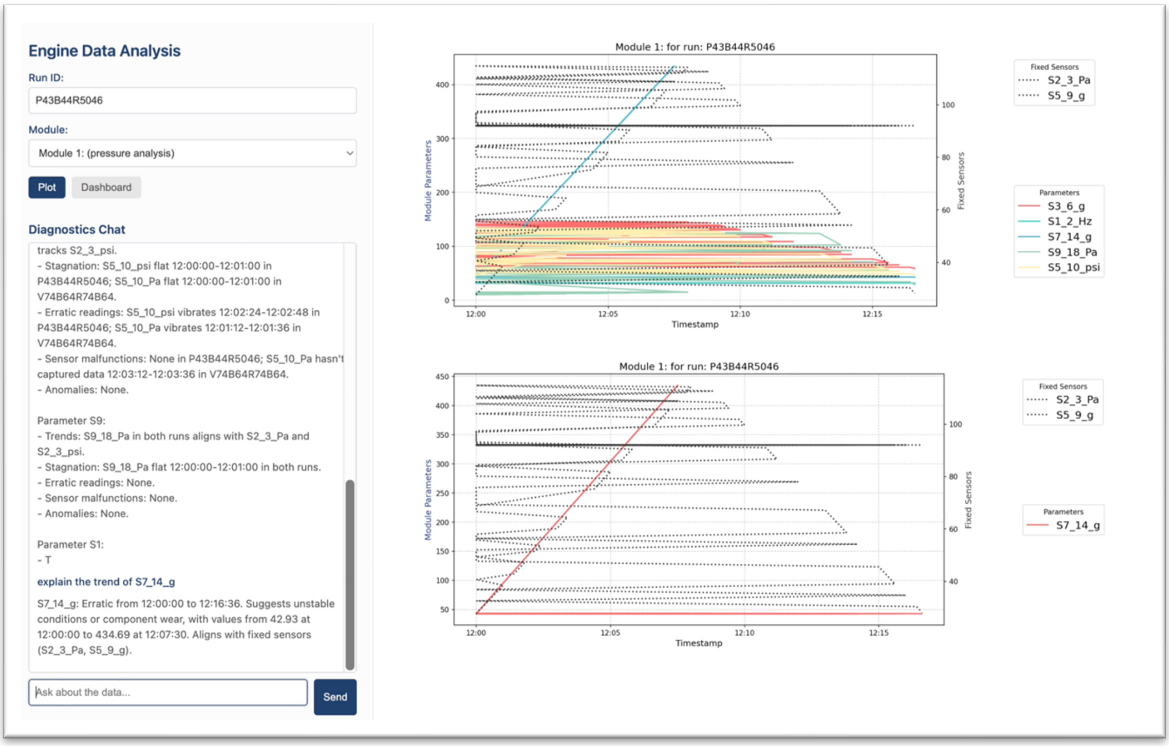


Figure 3: Single trend analysis

APPENDIX-C

ENCLOSURES

2. Include certificate(s) of any Achievement/Award won in any project-related event.

3. Similarity Index / Plagiarism Check report clearly showing the Percentage (%). No need for a page-wise explanation.

4. Details of mapping the project with the Sustainable Development Goals (SDGs).

SUSTAINABLE DEVELOPMENT GOALS

SDG 9: Industry, Innovation, and Infrastructure

The Aero-Gas Turbine Engine Data Analysis Project aligns with **SDG 9: Industry, Innovation, and Infrastructure** by implementing technology to analyse engine performance, ensuring user – friendly and accessible application for faster analysis

- **Capability**: Analyzes up to 10,000 data points per test run to catch issues like sensor failures, stagnation, or erratic readings.
- **Performance**: Compares multiple runs to spot trends or early faults, helping maintain engines efficiently.
- **Technology and Infrastructure**: Uses Mistral-7B to generate clear reports (e.g., "S3_567_Psi spikes to 5.2 at 12:02:15"), a web interface for easy data access, and a chatbot for quick queries.
- **Faster Processing**: Employs memory optimization to speed up data analysis, handling large datasets without delays, improving aerospace workflows.

Conclusion

The project supports SDG 9 by improving engine reliability, promoting sustainable aerospace practices, and introducing innovative data analysis tools, contributing to safer and more efficient air transport systems.