1.1 State of Art Developments

The current internal process for analyzing device-level data related to system utility parameters and health metrics involves using the IDMS (Intelligent Driver Monitoring System) proprietary portal. This portal allows for manual identification and debugging of issues specific to a single device, which can then be extrapolated to multiple devices to find broader problems associated with package releases. The infrastructure supporting this project includes several advanced technologies and methodologies:

- Advances in Cloud Computing for Data Storage and Retrieval
    - AWS (Amazon Web Services) S3 Instances: Amazon Web Services (AWS) S3 instances are crucial for fetching raw data about devices. This scalable, high-availability cloud storage ensures secure and efficient data retrieval, enabling comprehensive analysis and monitoring of system utility usage across multiple devices.
- Use of AI and Machine Learning in Automating Data Analysis and Monitoring
    - Python Libraries and Tools: AI and machine learning techniques are implemented using robust Python libraries such as Pandas, NumPy, Scikit-learn, and TensorFlow. These tools automate data analysis, transforming raw data into actionable insights. Machine learning models detect patterns and anomalies in system utility usage, reducing the need for manual intervention.
    - Visualization Tools: Libraries like Matplotlib and Seaborn create comprehensive visualizations of the analyzed data, aiding in understanding complex patterns and trends for faster, more correct decision-making.
- Current Tools and Frameworks for System Utility Monitoring and Performance Analysis
    - Excel Sheets and Analytical Plots: Traditional tools like Excel are used for data organization and preliminary analysis, while advanced plotting tools generate detailed graphs and charts. These visual aids summarize system performance metrics and find potential issues.
- Industry Trends and Best Practices for Managing and Optimizing System Utilities Usage
    - Resource Monitoring: Continuous monitoring of key resources like RAM, GPU, and storage at both the process and service levels is essential for best performance and early detection of potential bottlenecks.
    - Proactive Maintenance: Best practices include threshold-based alerts and predictive analytics to prevent issues before they escalate, ensuring smooth and efficient device operation.
    - Data-Driven Decision Making: Using detailed performance analytics and machine learning insights allows for informed decisions about resource allocation, system upgrades, and overall optimization strategies.

1.2 Motivation

The decision to undertake this project is driven by several key factors aimed at enhancing the efficiency and accuracy of system utilities usage monitoring and analysis:

1. **Need for Efficient and Accurate Monitoring of System Utilities Usage**

- **Manual Check Process**: The current methodologies need navigating the IDMS portal and manually checking critical events such as RAM usage and process crash information. This process is time-consuming and prone to human error.
- **Lack of Reliable Service Monitoring**: There is no dependable way to ensure that all services are functioning according to requirements and expectations in staging builds, leading to bugs and issues appearing in production OTA versions.

2. **Challenges in Manual Analysis and Benefits of Automation**
    - **Lack of Data Summarization**: Currently, there is no method to summarize data for a given device over a specific time, making it difficult to analyse usage efficiently.
    - **Manual Testing Processes**: Unit and functionality testing involve navigating through log data manually by each team for their respective devices, a process that can take multiple days before a release.
    - **Tedious QA (Quality Assurance) Processes**: Quality Assurance (QA) processes are very time-consuming, involving multiple cycles of bug checking, informing developers, changing code, and retesting.
3. **Importance of Proactive Maintenance and Issue Prevention through Threshold Generation**
    - **Threshold Generation**: Implementing algorithms for customizable threshold generation will serve as critical indicators, enabling proactive maintenance and issue prevention. This approach helps in setting actionable limits for various system utility parameters, thus ensuring devices run within safe and efficient bounds.
4. **Enhancing Decision-Making Processes with Data-Driven Insights**
    - **Facilitating Data-Driven Analysis**: The current system lacks AI-assisted tools to help process and analyse substantial amounts of data, making it challenging to conclude the state of a system accurately.
    - **Automating Data Analysis**: By automating the study of log and other device data, the process of giving a go-ahead for a release version can be streamlined. This automation reduces the dependency on manual checks and allows for more precise and prompt decision-making.
    - **Comprehensive Framework for Teams**: Providing a comprehensive framework will help teams analyse their services and code more effectively, supporting continuous improvement and integration of new functionalities into the pipeline.

1.3 Problem Statement

The current system for monitoring device health and performance statistics is inefficient, relying heavily on manual data retrieval and analysis through the IDMS portal. This process is time-consuming, prone to errors, and lacks scalability, resulting in delayed detection and resolution of critical issues, which means

that releases to field and production devices take longer. The result is that customer device errors take a while to be solved reliably, reducing customer satisfaction. Additionally, there is no automated method for generating thresholds tailored to specific feature sets and use cases, nor a comprehensive approach for analyzing resource consumption and budget usage for new services which means that extensive manual testing must be performed to permit/disallow a new functionality on the device. This project aims to develop a peripheral tool that automates data retrieval, threshold generation, budget analysis, and service monitoring, using AI and machine learning to enhance efficiency, accuracy, and proactive issue prevention, that will be used by multiple development and QA teams to ensure prompt addition and release of features to customer devices.

1.4 Objective

The primary aim of this project is to develop an advanced tool for efficient monitoring, budgeting, and threshold management of system utilities usage across diverse services deployed on devices. The specific goals include:

1. **Develop an Automated Monitoring Tool**
    - Design and implement a comprehensive tool to check system utilities usage, including CPU, RAM, GPU, and storage, across different services and devices.
2. **Customizable Threshold Generation**
    - Implement algorithms to generate thresholds specific to distinct feature sets and use cases. These thresholds will serve as critical indicators for proactive maintenance and issue prevention.
3. **Reliable Budgeting Methodologies**
    - Establish reliable methodologies for estimating resource consumption and budgeting for new services and functionalities, helping informed decision-making aligned with organizational goals.
4. **Data-Driven Device Analysis**
    - Develop a framework to help a data-driven approach for analysing device health and performance, using AI and machine learning to generate actionable insights.
5. **Automation of Log and Data Analysis**
    - Automate the study and analysis of log files and other device data to support informed decisions about release versions, ensuring prompt and correct assessment of device readiness.
6. **Replacement of Manual Analysis**
    - Replace existing manual analysis processes with an automated AI-driven approach to enhance accuracy, reduce human error, and improve efficiency.

7. **Comprehensive Framework for Service and Code Analysis**
    - Develop a comprehensive framework to aid development and QA teams in analysing their services and code, improving overall system reliability and performance.

1.5 Scope

The scope of this project encompasses the development and implementation of a tool designed to check, budget, and manage system utilities usage across numerous services deployed on devices. The project includes specific functionalities and excludes certain aspects to keep focus and ensure feasibility within the given computation, time and procedural constraints.

Included:

1. **Threshold Generation**
    - Implement algorithms to generate customizable thresholds specific to distinct feature sets and use cases – production and staging devices.
2. **Budget Analysis**
    - Conduct comprehensive analyses to estimate resource consumption for new services or functionalities, helping informed decision-making aligned with resource constraints and package support.
3. **Service Usage Monitoring**
    - Develop robust monitoring mechanisms to track service usage and detect abnormalities indicative of potential device, service/functionality and package-related issues.
4. **Data Retrieval from Cloud**
    - Implement methods to retrieve cloud-uploaded data about device health and performance statistics from AWS S3 instances.
5. **Automated Log and Data Analysis**
    - Automate the study and analysis of log files and other device data to support release decisions and replace manual analysis with AI-driven approaches.
6. **Report Generation**
    - Develop tools to parse, analyse, and generate reports, including Excel sheets and graphical plots, to display data effectively for stakeholders.
7. **Integration and Continuous Improvement**
    - Ensure continuous improvement of the tool by integrating new functionalities and refining existing features based on feedback and evolving requirements.

Excluded:

1. **Detailed Hardware-Level Performance Optimization**
    - The project does not include optimization of hardware performance at the part level as components and boards are offered by third-party companies like NVIDIA and Qualcomm.
2. **External System Integration Beyond Defined APIs**
    - Integration with external systems outside the defined APIs and existing infrastructure is not within the scope of this project.
3. **User Interface Development for End Users**
    - The creation of an extensive user interface for end users, beyond essential dashboards and reporting tools, is not included.
4. **On-Device Monitoring Infrastructure**
    - The project does not cover the development of infrastructure for real-time monitoring and response, focusing instead on periodic analysis and reporting conducted by employees on their PCs and not a device-level monitoring service.

1.6 Methodology

The method outlines the structured approach and methods employed to achieve the project aims of developing an advanced tool for monitoring, budgeting, and threshold management of system utilities usage across diverse services deployed on devices. The process is divided into several key stages:

1. **Literature Review and Research on Existing Solutions**
    - Conduct a comprehensive literature review to understand current methodologies and tools used for system utilities watching and performance analysis.
    - Analyse industry trends and best practices to show gaps and opportunities for improvement in existing solutions.
2. **Design and Development of System Architecture**
    - Develop the system architecture using Data Flow Diagrams (DFDs) and Class Diagrams to visualize data processing and interactions between different components.
    - Define the overall structure, data flow, and key functionalities of the tool to ensure a clear and coherent design.
3. **Data Retrieval and Processing**
    - Implement methods to fetch data from the cloud (e.g., AWS S3) for given Device IDs and time periods.

- Develop Python scripts to parse, analyse, and populate Excel sheets and generate graphs/plots to display data effectively.

4. **Threshold Generation and Validation**
    - Collaborate with Quality Assurance (QA) and other relevant teams to discuss and gather information on threshold generation for devices in the field.
    - Create feature-set and usage-wise thresholds based on the collected data and confirm these thresholds with teams responsible for their respective services.
5. **Implementation and Testing**
    - Deploy new services and functionalities on devices, ensuring they are tested against the established thresholds during the testing phase.
    - Monitor the system utilities usage and conduct thorough testing to confirm the effectiveness of the deployed features.
6. **Report Generation and Analysis**
    - Gather an extensive list of staging and production devices with known feature sets and generate comprehensive reports.
    - Utilize the developed Python scripts to automate the generation of these reports, ensuring accuracy and consistency.
7. **Budget Analysis and Monitoring**
    - Conduct frequent generation of reports to analyse and check system utilities usage, easing discussions with service developers.
    - Use the reports to make informed decisions about resource allocation and budgeting for new services and functionalities.
8. **Iterative Improvement and Integration**
    - Implement an iterative development process, incorporating feedback from stakeholders and continuous integration to refine and enhance the tool.
    - Ensure that the tool evolves to meet changing requirements and integrates seamlessly into existing development and operational pipelines.

1.7 Organization of the Report

This report is structured to provide a comprehensive understanding of the project, from the first motivation and problem statement to the final implementation and testing. The report is organized as follows:

Chapter 1: Introduction

1. **State of Art Developments:** Overview of current advancements relevant to the project.
2. **Motivation:** Explanation of the driving factors behind choosing this project.
3. **Problem Statement:** Articulation of the specific problem addressed by the project.
4. **Objective:** Definition of the main goals and sub-goals of the project.
5. **Scope:** Outline of what is included and excluded in the project.
6. **Methodology:** Description of the approach and methods used to achieve the project goals.
7. **Organization of the Report:** Overview of the structure of the report.
8. **Summary:** Summary of the chapter.

Chapter 2: Overview of the Domain

1. **Introduction:** Introduction to the domain relevant to the project.
2. **Relevant Information:** Detailed information pertinent to the domain.
3. **Summary:** Recap of the key points discussed in the chapter.

Chapter 3: Software Requirements Specification of the Project

1. **Overall Description:**
    - Product Perspective
    - Product Functions
    - User Characteristics
    - Constraints and Dependencies
2. **Specific Requirements:**
    - Functional Requirements
    - Performance Requirements
    - Supportability
    - Software Requirements
    - Hardware Requirements
    - Design Constraints
    - Interfaces:
        - User Interfaces of the System
        - Software Interfaces of the System
    - Non-Functional Requirements

3. **Summary:** Overview of the requirements and constraints.

Chapter 4: High Level Design of the Project

1. **Design Considerations:**
    - General Constraints
    - Development Methods
2. **Architectural Strategies:**
    - Programming Language
    - User Interface Paradigm
    - Error Detection and Recovery
    - Data Storage Management
3. **System Architecture:** Description of the overall system architecture.
4. **Data Flow Diagrams / UML Diagrams:**
    - Level 0 / UML Diagram 1
    - Level 1 / UML Diagram 2
    - Level 2 / UML Diagram 3
5. **Summary:** Summary of the design and architecture.

Chapter 5: Detailed Design of the Project

1. **Structure Chart:** Representation of the system structure.
2. **Functional Description of the Modules:**
    - Order Assignment Module
    - Order Assignment Configuration Module
    - Master File Generation Module
3. **Summary:** Overview of the detailed design.

Chapter 6: Implementation of the Project

1. **Programming Language Selection:** Justification for the choice of programming language.
2. **Platform Selection:** Explanation of the selected platform.
3. **Code Conventions:**
    - Naming Conventions
    - File Organization

- Declarations
- Comments

4. **Difficulties Encountered and Strategies Used to Tackle Them:**
    - Concurrency Control
5. **Summary:** Summary of the implementation phase.

Chapter 7: Software Testing of the Project

1. **Test Environment:** Description of the testing environment.
2. **Unit Testing:**
    - Order Assignment Module
    - Order Assignment Configuration Module
    - Order Assignment System
3. **Integration Testing:**
    - Integration Testing for Order Assignment
    - Integration Testing for Order Assignment Configuration
    - Integration Testing of Master File Generation
    - Integration Testing of Order Assignment System
4. **System Testing:** Comprehensive system testing.
5. **Summary:** Overview of the testing process and results.

Chapter 8: Experimental Results and Analysis of the Project

1. **Evaluation Metrics:** Metrics used to evaluate the project.
2. **Experimental Dataset:** Description of the dataset used.
3. **Performance Analysis:**
    - Order Assignment
    - Master File Generation
4. **Summary:** Summary of the experimental results and analysis.

Chapter 9: Conclusion and Future Enhancement

1. **Limitations of the Project:** Discussion of the project's limitations.
2. **Future Enhancements:** Potential future enhancements for the project.
3. **Summary:** Recap of the conclusions and future directions.

1.8 Summary

This chapter introduced the project, including the state-of-art developments, motivation, problem statement, goals, scope, method, and organization of the report. The following chapter will review the existing literature and related work in the field, providing a foundation for the design and implementation of our system.

2.1. Introduction

In today's rapidly evolving technological landscape, effective management and analysis of data play a pivotal role in ensuring the success and reliability of software and hardware products. Within this context, the establishment of edge infrastructure appears as a crucial initiative undertaken by companies to streamline the processing, summarization, and visualization of data originating from the devices they develop. For this company uniquely, debugging issues that plague production devices with respect to system services are of utmost importance to solve and release updates for. This section provides an overview of the significance of edge infrastructure and its role in helping efficient decision-making processes by providing quick insights and enabling effective debugging before the release of new software versions.

2.2. Relevant Information

Edge infrastructure represents a comprehensive approach to managing data generated by devices within a company's ecosystem. At its core, it involves fetching data from various sources such as cloud storage and databases, encompassing critical health metrics like CPU and GPU performance, process metrics, GPS data, CAN data, and alert information. Once retrieved, this data undergoes rigorous processing and organization, leveraging advanced techniques and tools like Python programming language, AWS API, SQL queries, and Matplotlib for visualization.

Key components of edge infrastructure include data summarization, visualization, and reporting. Data summarization involves condensing extensive datasets into concise reports, highlighting crucial insights and anomalies. Visualization plays a crucial role in presenting complex data in an intuitive and informative manner, enabling stakeholders to grasp trends and patterns effortlessly. Reporting ensures that the summarized and visualized data is accessible to relevant stakeholders, aiding in decision-making processes, debugging efforts, and continuous evaluation of software and hardware performance. This will be deployed for both staging (internal) devices for testing new packages and functionalities before handing them over to QA and for production (field) devices if a problem or potential threat is identified by respective teams on monitoring.

2.3. Overview of Netradyne Technologies Pvt. Ltd

Netradyne Technologies is a technology company focused on developing advanced AI and machine learning solutions for transportation safety. Founded in 2015, it specializes in driver and fleet safety systems, leveraging computer vision and edge computing. Their flagship product, Driveri, provides real-time insights and analytics to enhance road safety, improve driver behavior, and optimize fleet operations. Netradyne's technology aims to reduce accidents and increase operational efficiency in the commercial transportation industry. The company is headquartered in San Diego, California.

Fig 2.1 Flagship Product of Netradyne

2.4. Summary

In summary, edge infrastructure represents a strategic initiative aimed at enhancing the management and analysis of data generated by devices developed by the company. Through streamlined processes of data fetching, summarization, visualization, and reporting, edge infrastructure enables efficient decision-making, quick insights, and effective debugging. By leveraging advanced tools and methodologies, companies can ensure the reliability and performance of their software and hardware products, ultimately driving innovation and success in today's competitive market landscape.

3.1. Overall Description:

The software and hardware requirements for this project are simple and minimal.

3.1.1. Product Perspective: The project operates within the broader context of data analytics and software engineering, serving as a critical component of the company's infrastructure for managing and analysing data from devices. It interfaces with various data sources, databases, and cloud storage systems to fetch and process data.

3.1.2. Product Functions: The primary functions of the project include fetching data from cloud storage and databases, processing and summarizing the data, organizing it into comprehensible reports and visualizations, and providing a user interface for stakeholders to interact with the data. Additionally, the system must integrate with existing software and hardware systems within the company's ecosystem.

3.1.3. User Characteristics: Users of the system include developers who are service owners, quality assurance professionals, field analysts and decision-makers like Senior Directors within the company. These users possess varying levels of technical expertise and require a gradient of intuitive interfaces and functionalities tailored to their respective roles and responsibilities.

3.1.4. Constraints and Dependencies: Constraints and dependencies include the availability and reliability of data sources as they follow multiple different formats, compatibility with existing software and hardware systems to be used on employee PCs and EC2 instances, adherence to data privacy and security regulations as it contains sensitive device information.

3.2. Specific Requirements:

3.2.1. Functional Requirements:

- The system must fetch data from cloud storage and databases.
- It must process and summarize data, organizing it into reports and visualizations.
- The system should provide a user interface for stakeholders to interact with the data.
- Integration with existing software and hardware systems is required.
- Modular code to be easily extendable to new functionality, formats and parameters.
- It should be easily integrated into automation processes.

3.2.2. Performance Requirements:

- The system should be able to process large volumes of data efficiently.
- Reports and visualizations should be generated in a timely manner.
- The user interface should be responsive and intuitive.

3.2.3. Supportability:

- The system should be easy to maintain and update.

- Adequate documentation and support resources should be provided for users.

3.2.4. Software Requirements:

- The system should be developed using Python programming language.
- Utilize libraries such as Matplotlib for data visualization.
- It uses AWS SSM Keys and Identity Access Management (IAM) feature to connect with the Database.

3.2.5. Hardware Requirements:

- Adequate computing power and storage capacity to handle data processing and storage requirements – regular employee laptop.

3.2.6. Design Constraints:

- The system design should adhere to company standards and best practices.
- Considerations for scalability and extensibility should be considered during design.

3.2.7. Interfaces: 3.2.7.1. User Interfaces of the system:

- The user interface should be intuitive and user-friendly, allowing stakeholders to easily navigate and interact with the data.

3.2.7.2. Software Interfaces of the system:

- The system should integrate with existing software and hardware systems within the company's ecosystem.

3.2.8. Non-Functional Requirements:

- The system should adhere to data privacy and security regulations.
- Performance and reliability are critical non-functional requirements for the system.

3.3. Summary

In summary, the software requirements specification outlines the functional, performance, supportability, software, hardware, design, interface, and non-functional requirements of the project. These requirements serve as the foundation for the development and implementation of the system, ensuring its effectiveness, usability, and reliability within the company's infrastructure.

4.1. Design Considerations

4.1.1. General Constraints: The design of the project must adhere to several constraints, including:

- Resource Limitations: Limited computing power and storage capacity must be managed efficiently.
- Data Privacy and Security: Compliance with relevant data protection regulations and ensuring secure data handling practices.
- Compatibility: Ensuring compatibility with existing software and hardware systems within the company.
- Scalability: The design must accommodate future growth in data volume and user base and should hence be as modular as possible.
- Usability: The system must be intuitive and user-friendly to cater to users with varying levels of technical expertise and hands-on knowledge about specific parameters.

4.1.2. Development Methods: The development of the project will follow agile methodologies to allow for iterative progress and flexibility in responding to changing requirements. Key practices will include:

- Incremental Development: Breaking down the project into smaller, manageable components that can be developed and tested incrementally.
- Continuous Integration and Testing: Regularly integrating and testing components to identify and address issues early in the development cycle.
- Employee Feedback: Incorporating feedback from stakeholders and end-users throughout the development process to ensure the system meets their needs.

4.2. Architectural Strategies

4.2.1. Programming Language: The primary programming language for the project will be Python, chosen for its versatility and extensive library support. Python's robust ecosystem, including libraries for data processing (e.g., Pandas), visualization (e.g., Matplotlib), and interfacing with cloud services (e.g., Boto3 for AWS), makes it well-suited for this project.

4.2.2. User Interface Paradigm: The user interface will follow a web-based paradigm, ensuring accessibility across different devices and platforms. Key design principles will include:

- Simplicity and Clarity: Ensuring the interface is straightforward and easy to navigate.
- Responsiveness: Providing a responsive design that works well on desktop and cloud machine instances.
- Customization: Allowing users to customize views and reports based on their specific needs and preferences, allowing them to extract parameters that they require only.

4.2.3. Error Detection and Recovery: The system will incorporate robust error detection and recovery mechanisms to maintain reliability and user trust. Strategies will include:

- Logging and Monitoring: Implementing comprehensive logging and simple debug statements on success or encountering exceptions to detect and diagnose issues promptly.

4.2.4. Data Storage Management: Efficient data storage management is critical for handling large volumes of data generated by the devices. The approach will include:

- Database Management: Utilizing SQL databases for structured data storage and retrieval, ensuring efficient query performance and data integrity.
- Cloud Storage: Leveraging cloud storage solutions for scalable and cost-effective data storage, particularly for large datasets.
- Data Archiving: Implementing data archiving strategies to manage historical data and optimize storage usage, ensuring that active data remains easily accessible while older data is securely archived and stored in appropriate s3 buckets for easy retrieval subsequently.

4.3. System Architecture

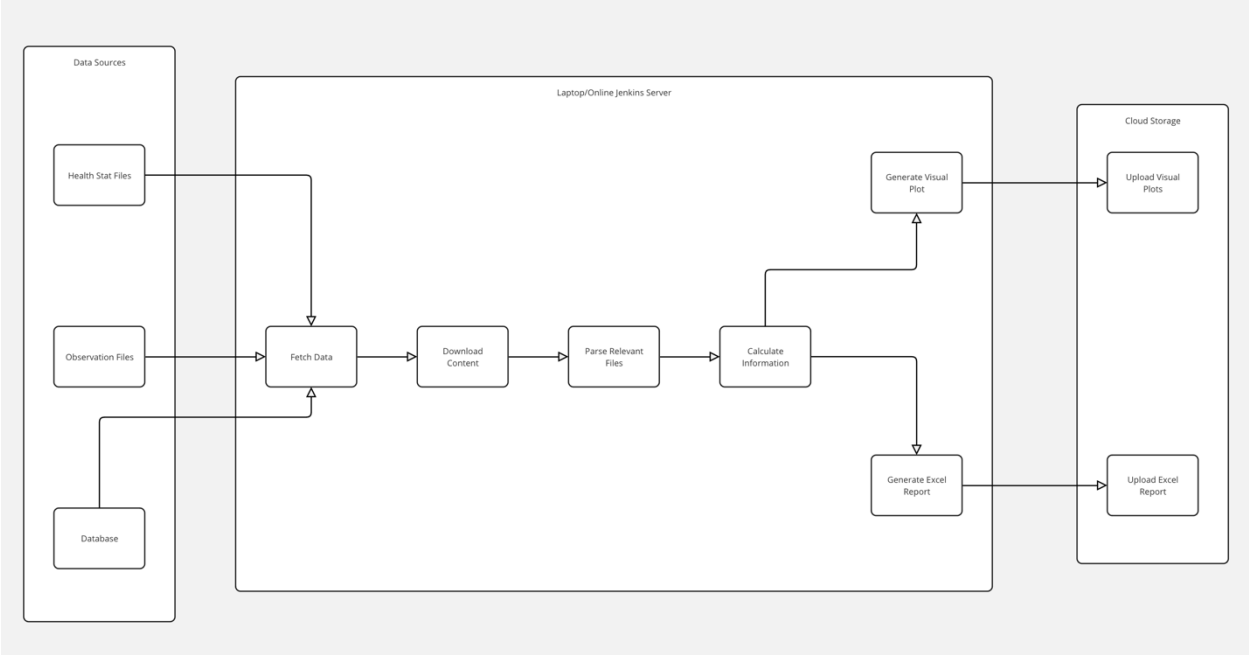


Fig 4.1 Architecture Diagram

The architecture diagram illustrates the workflow of a data processing and reporting system, which spans across multiple components and stages. Here's a detailed description:

1. **Data Sources**: The system starts by pulling data from three main sources:
    - **Health Stat Files**

- **Observation Files**
- **Database**

2. **Fetch Data**: The collected data from the above sources is fetched for further processing.
3. **Download Content**: Once the data is fetched, it is downloaded and made available for the next steps.
4. **Parse Relevant Files**: The downloaded content is parsed to extract relevant information necessary for analysis.
5. **Calculate Information**: The parsed data is processed to calculate the required information. This step includes various data transformation and calculation processes to prepare the data for reporting.
6. **Generate Visual Plot**: The calculated information is used to generate visual plots, which help in visualizing the data trends and insights.
7. **Generate Excel Report**: In parallel to visual plot generation, an Excel report is created using the calculated data.
8. **Cloud Storage**: The final outputs, including visual plots and Excel reports, are uploaded to cloud storage for accessibility and distribution:
   - **Upload Visual Plots**
   - **Upload Excel Report**

This entire workflow can be executed on a laptop or an online Jenkins server, which handles the automation and orchestration of the various stages of data processing and report generation. This architecture ensures a streamlined process from data collection to the final report distribution, leveraging both local and cloud resources.

4.4. Process Overview

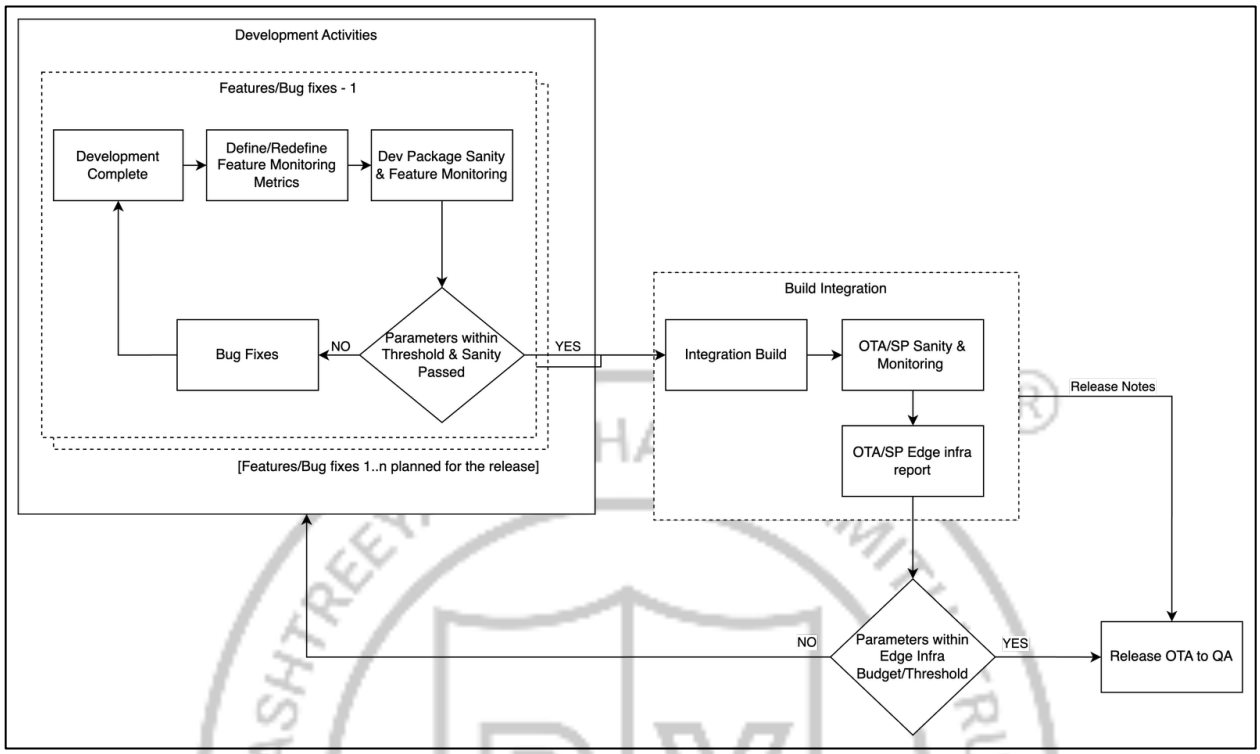


Fig 4.2 Process Overview Diagram

The diagram presents a detailed workflow for development activities and build integration within a software release process.

Development Activities

1. **Development Complete**: Marks the initial completion of development tasks.
2. **Define/Redefine Feature Monitoring Metrics**: Establishes or updates the metrics needed to monitor the new features.
3. **Dev Package Sanity & Feature Monitoring**: Conducts sanity checks and monitors features for correctness and performance.
4. **Decision Point - Parameters within Threshold & Sanity Passed**:
    - **YES**: If parameters are within the defined thresholds and sanity checks pass, the process moves to build integration.
    - **NO**: If the parameters do not meet the thresholds or sanity checks fail, bug fixes are performed, and the process returns to rechecking the metrics and sanity.

Build Integration

1. **Integration Build**: Compiles and integrates the new features and bug fixes into a build.

2. **OTA/SP Sanity & Monitoring**: Performs over-the-air/service pack sanity checks and monitors the integrated build.
3. **OTA/SP Edge Infra Report**: Generates a report on the edge infrastructure based on the OTA/SP monitoring.

Decision Point - Parameters within Edge Infra Budget/Threshold

- **YES**: If parameters meet the edge infrastructure budget and thresholds, the OTA (over-the-air update) is released to QA.
- **NO**: If not, further actions are required to adjust the build or fix issues before it can be released.

Release Notes

- Release notes are generated and provided as part of the build integration process to document the changes and updates included in the release.

4.4. Data Flow Diagrams / UML Diagrams

This section depicts the process of generating an Edge Infrastructure Report by integrating multiple data sources and producing outputs for reporting and visualization. The workflow is as follows:

Input Data Sources

1. **Health Stat Files**: Provides health-related information.
2. **Observation Files**: Supplies observational data, including GPS information.
3. **Database**: Contributes additional data, including alert information.

Data Integration

- **Health Info**: Extracted from Health Stat Files.
- **GPS Info**: Derived from Observation Files.
- **Alert Info**: Gathered from the Database.

These inputs are collectively fed into the central process of **Edge Infra Report Generation**.

Edge Infra Report Generation

- The core component that processes and integrates health, GPS, and alert information to generate comprehensive edge infrastructure reports.

Outputs

1. **Report Summary**: A condensed version of the report highlighting key findings and insights.
2. **Visualization of Raw Data**: Graphical or visual representation of the raw data to aid in better understanding and analysis.

The diagram illustrates a streamlined process where various data inputs are combined to generate insightful reports and visualizations, aiding in effective data analysis and decision-making.

4.4.1. Data Flow Diagram – Level 0 / UML diagram 1

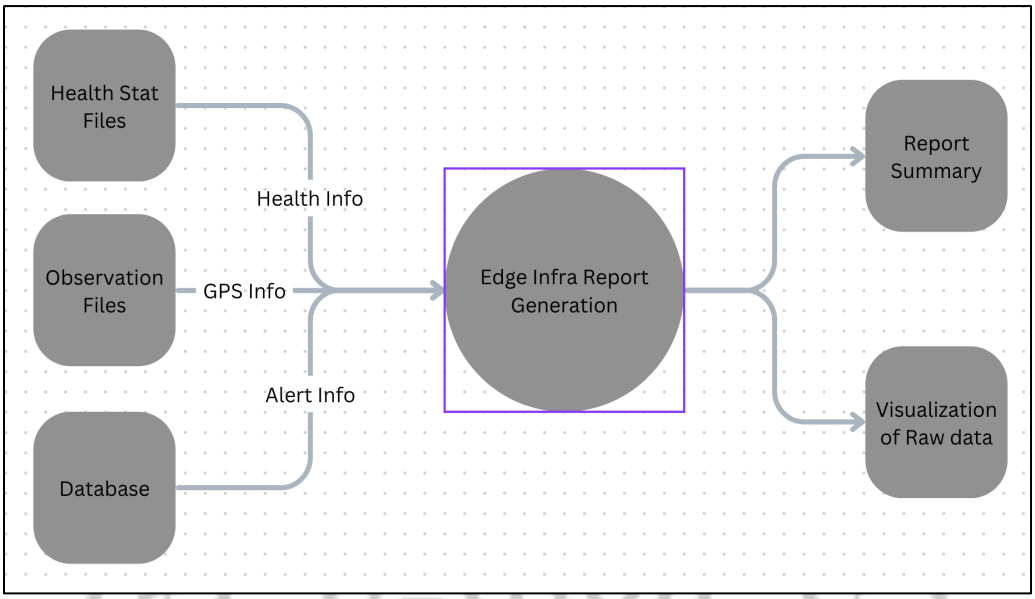


Fig 4.3 DFD Level 0

4.4.2. Data Flow Diagram – Level 1 / UML diagram 2

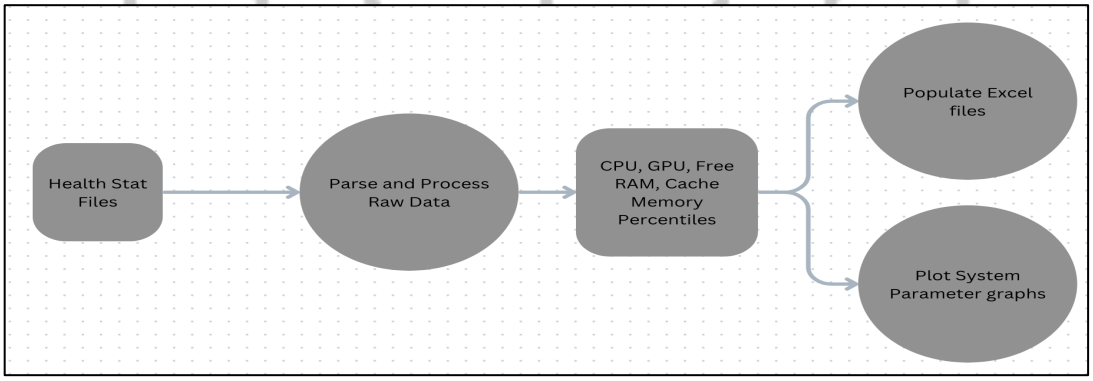


Fig 4.4 DFD Level 1 – Healthstat Pipeline (1.1)

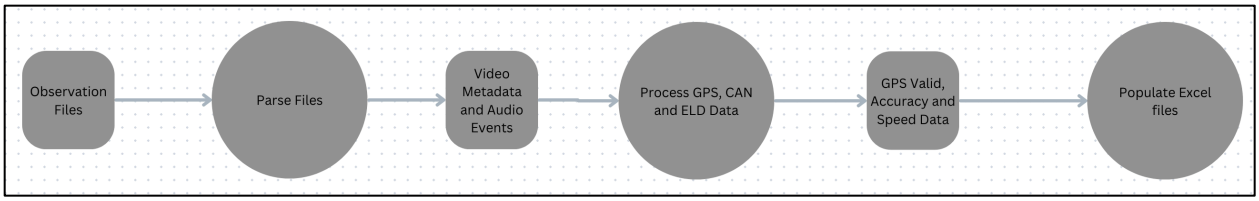


Fig 4.5 DFD Level 1 – Observation Pipeline (1.2)

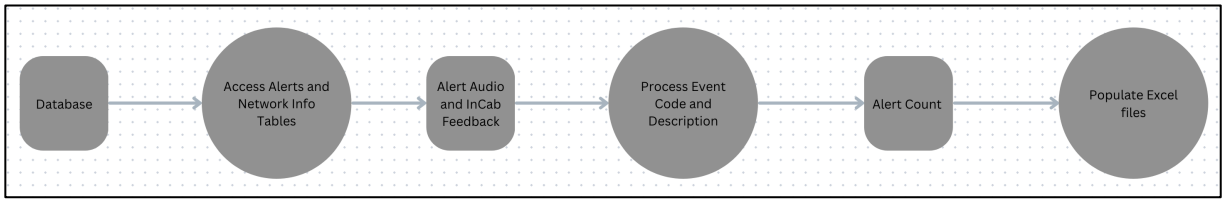


Fig 4.6 DFD Level 1 – Database Pipeline (1.3)

4.4.3. Data Flow Diagram – Level 2 / UML diagram 3

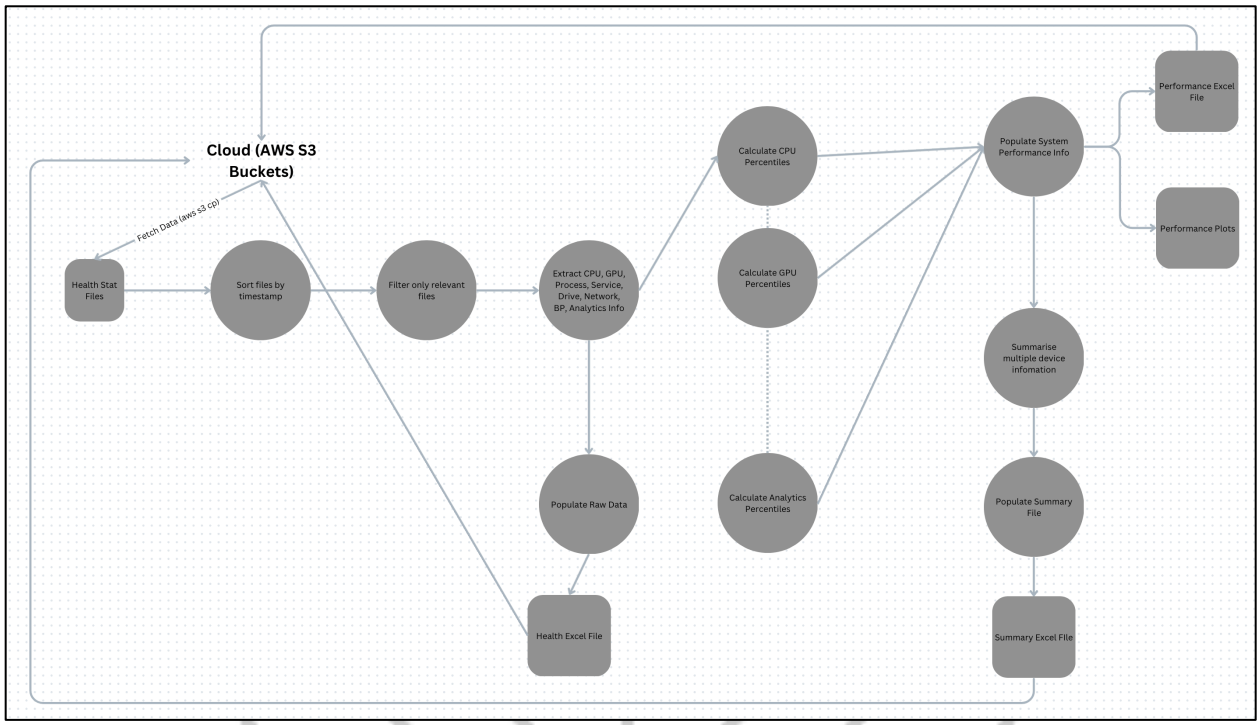


Fig 4.7 DFD Level 2 – Healthstat Pipeline (2.1)

4.5. Summary

Chapter 4 outlines the high-level design of the project, covering key aspects such as design considerations, architectural strategies, and system architecture. The project addresses constraints like resource limitations, data privacy, and scalability, using agile development methods. Python is chosen for its versatility, and a web-based UI ensures accessibility. Robust error detection and efficient data storage management are implemented. The system architecture integrates data sources, processing, visualization, and user interfaces. Data flow and UML diagrams visually represent the system's processes and interactions, ensuring a robust, scalable, and user-friendly design.

5.1 High-level Workflow

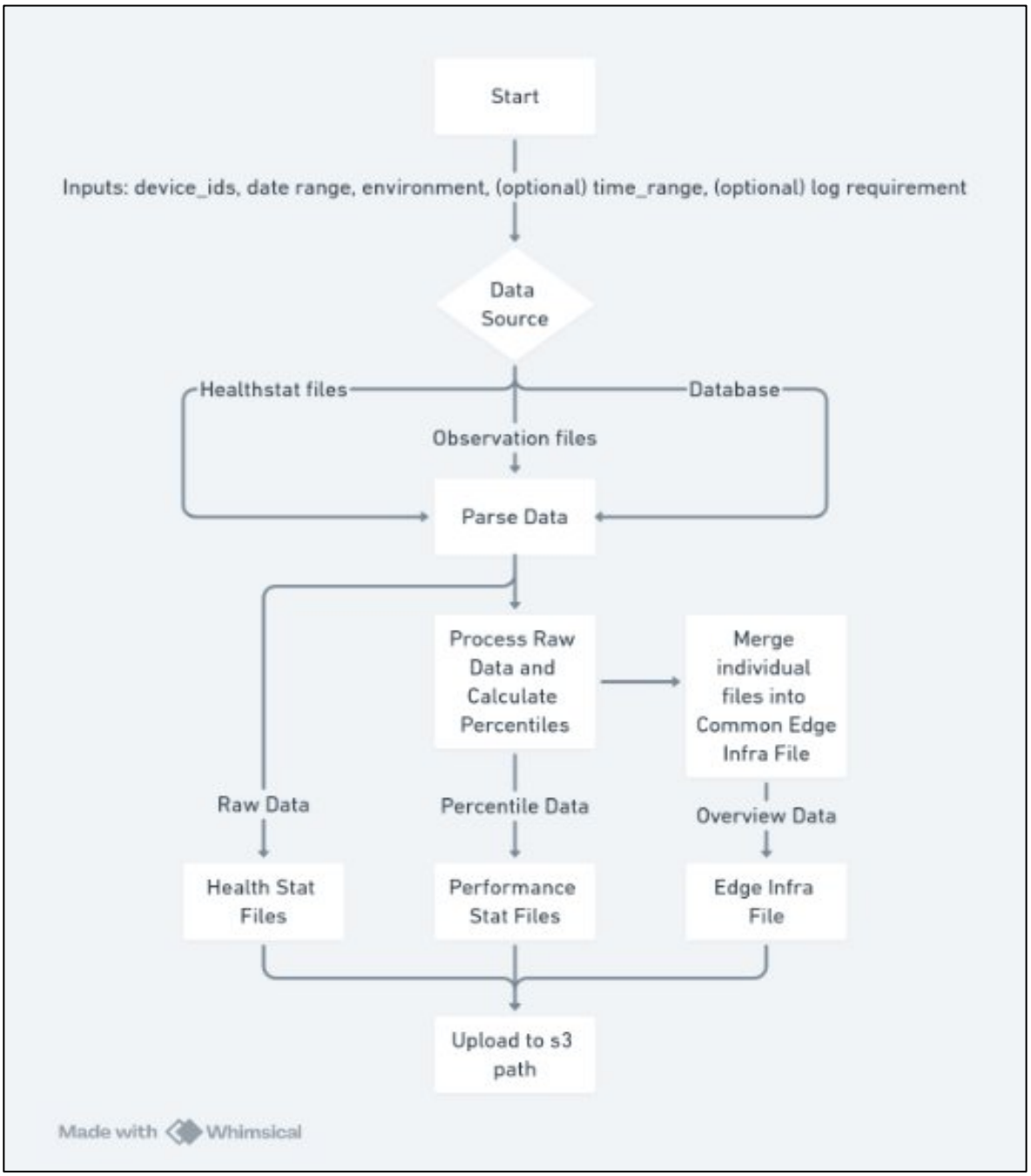


Fig 5.1 Project Workflow Diagram

The following section outlines the high-level workflow for the project, detailing the sequential steps involved in processing and summarizing device data. Each step plays a critical role in ensuring the efficient generation and visualization of summarized reports.

1. **Start:**

- The workflow begins with the initiation of the data processing pipeline. This step sets up the necessary environment and prerequisites for subsequent data parsing and processing activities.

2. **Parse Data:**
    - In this step, raw data is retrieved from various sources, including cloud storage and databases. The data encompasses health metrics (CPU, GPU, and process metrics), GPS, CAN data, and alert data. Parsing involves reading and organizing this raw data into a structured format suitable for further processing.
3. **Process Raw Data and Calculate Percentiles:**
    - Parsed data is processed to compute key statistical measures, such as percentiles. This involves cleaning and transforming the data, handling missing values, and calculating statistical summaries. The goal is to derive meaningful insights from the raw data that can inform subsequent analysis.
4. **Merge Individual Files into Common Edge Infra File:**
    - Processed data from various sources and individual files are merged into a comprehensive common file, referred to as the Edge Infra File. This consolidated file serves as the central repository for all relevant data, facilitating streamlined access and analysis.
5. **Generate Specific Outputs:**
    - Based on the consolidated data, specific outputs are generated to address different analysis needs. These outputs include:
        - **Health Stat Files:** Files containing detailed health statistics of devices, covering CPU, GPU, and process metrics.
        - **Performance Stat Files:** Files detailing the performance metrics of devices, aiding in the assessment of overall device performance.
        - **Edge Infra File:** The final consolidated file that integrates all processed data and serves as the primary reference for further analysis.
6. **Upload to S3 Path:**
    - The last step involves uploading the generated files to a designated S3 path on AWS. This ensures secure and scalable storage of the processed data, making it accessible for further analysis and reporting.

5.2 Module-level Workflow

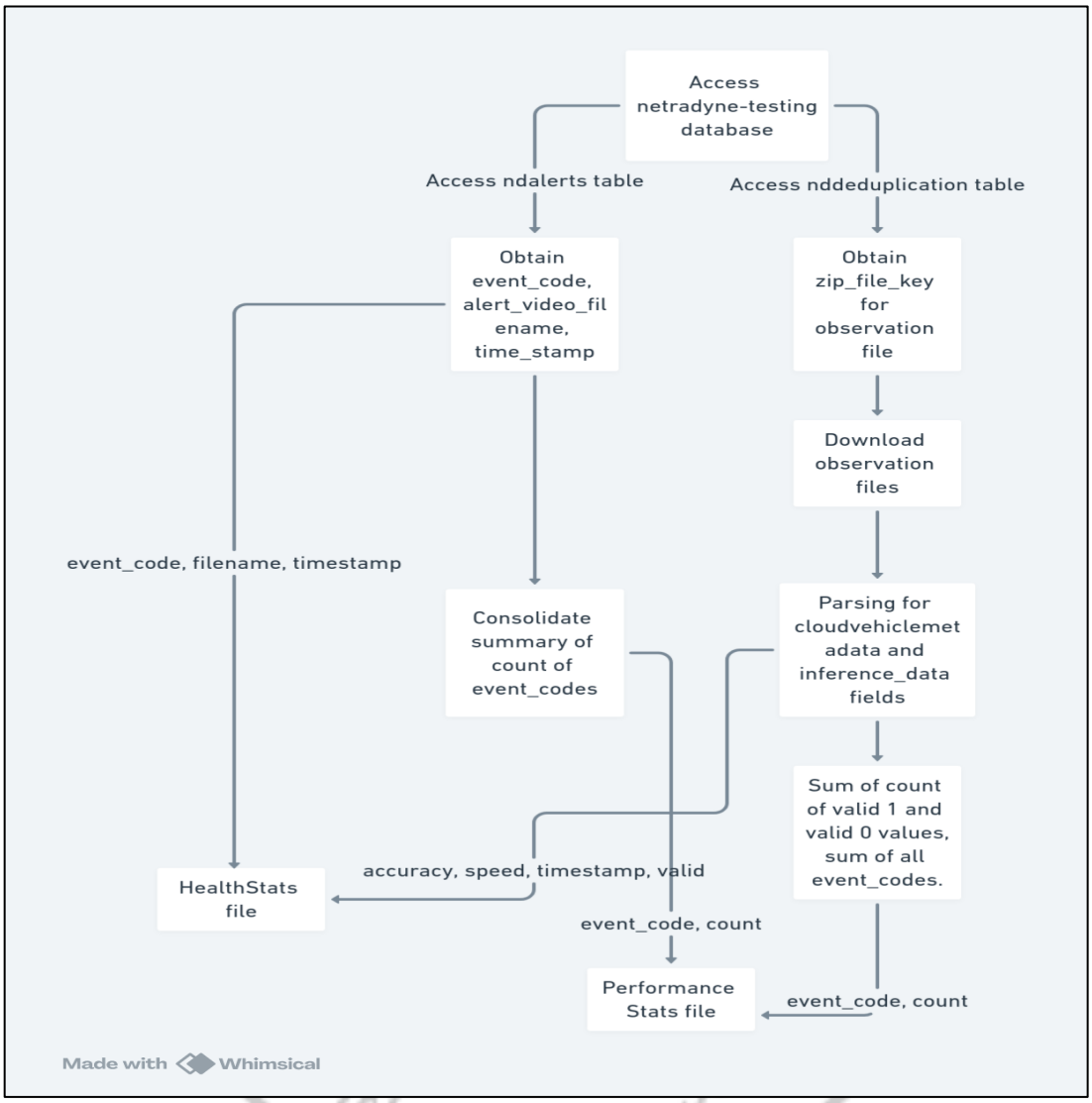


Fig 5.2 Database Flow Diagram

This diagram illustrates the workflow of a database pipeline designed for processing and analysing event data from the netradyne-testing database. The process involves two main tables: ndalerts and nddeduplication.

1. **Accessing Databases:**
    - **From the ndalerts table:**
        - Extract event_code, alert_video_filename, and timestamp.
    - **From the nddeduplication table:**
        - Obtain zip_file_key for observation files.

- Download observation files using the zip_file_key.
- Parse observation files to extract vehiclemetadata and inference_data fields.
- Calculate the sum of valid (1) and invalid (0) values and total event counts.

2. **Data Processing and Summarization:**
    - **For the ndalerts table data:**
        - Consolidate the extracted data into a summary that counts occurrences of each event_code.
    - **For the observation files data:**
        - Calculate performance statistics by counting valid and invalid values for each event_code.
3. **File Generation:**
    - The processed data is compiled into two main output files:
        - **HealthStats File:** Contains event_code, filename, and timestamp.
        - **Performance Stats File:** Includes accuracy, speed, timestamp, valid, event_code, and count.

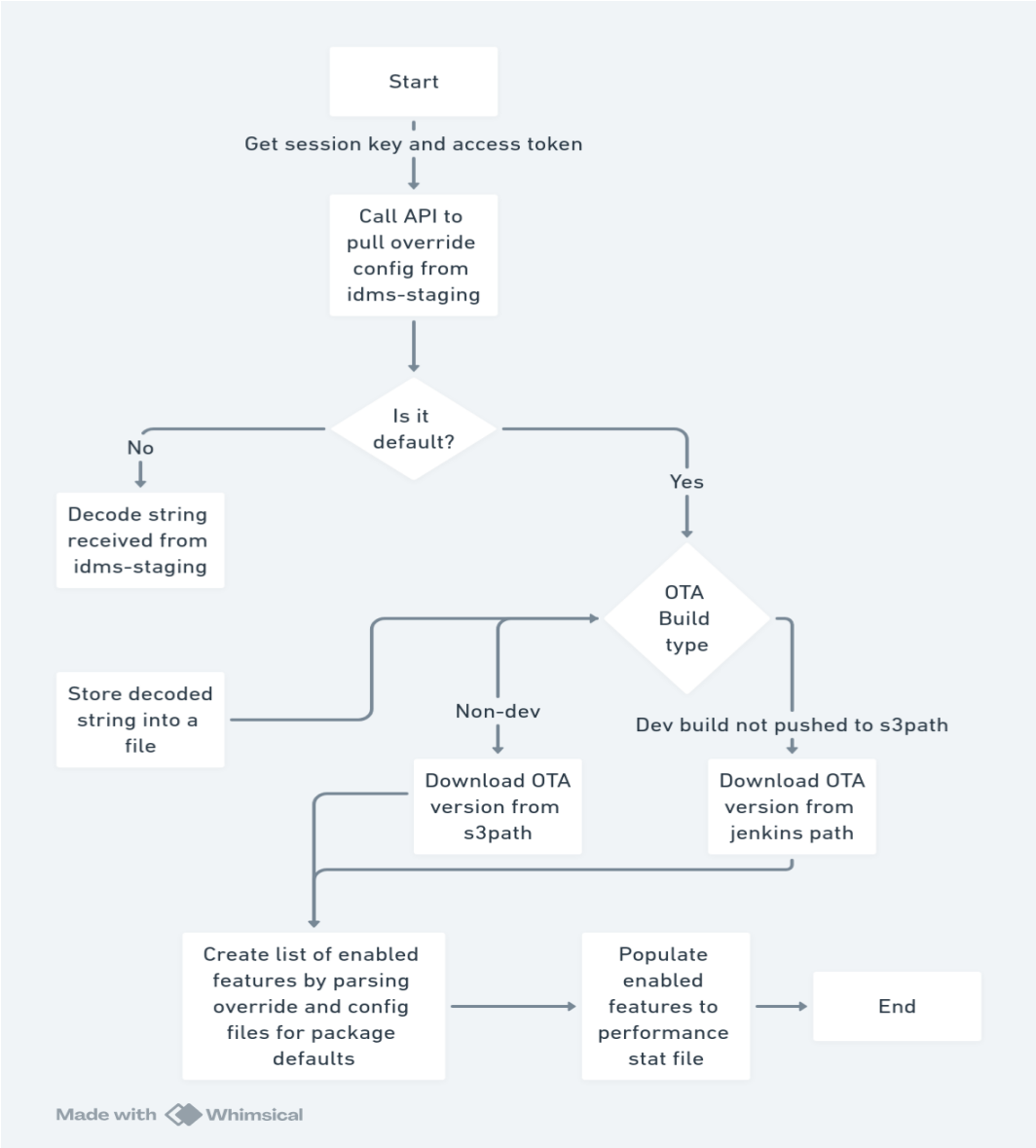


Fig 5.3 Feature Extraction Flow Diagram

This diagram depicts the workflow of a feature extraction pipeline designed to process configuration data and generate performance statistics. Here's a detailed description of the flow:

1. **Initialization:**
    - **Start:** Initiates the process.
    - **Get Session Key and Access Token:** Retrieves necessary credentials to access the API.
2. **Fetching Configuration Data:**
    - **Call API to Pull Override Config from idms-staging:** Uses the session key and access token to fetch override configuration data from the idms-staging environment.

3. **Processing Configuration Data:**
    - **Is it Default?**
        - **No:**
            - **Decode String Received from idms-staging:** Decodes the fetched configuration string.
            - **Store Decoded String into a File:** Saves the decoded configuration data into a file for further processing.
        - **Yes:**
            - **OTA Build Type:**
                - **Non-dev:**
                    - **Download OTA Version from s3path:** Retrieves the Over-the-Air (OTA) version from the specified S3 path.
                - **Dev Build Not Pushed to s3path:**
                    - **Download OTA Version from Jenkins path:** Fetches the OTA version from the Jenkins path for development builds.
4. **Feature Extraction:**
    - **Create List of Enabled Features:**
        - **Parsing Override and Config Files for Package Defaults:** Analyses the override and configuration files to compile a list of enabled features based on package defaults.
5. **Final Output:**
    - **Populate Enabled Features to Performance Stat File:** Updates the performance statistics file with the list of enabled features.

5.3 Summary

This chat provides detailed descriptions of two distinct data processing workflows illustrated through flowcharts. The first workflow pertains to a database pipeline for the netradyne-testing database, detailing steps to extract, process, and consolidate event data from
the ndalerts and nddeduplication tables into summary statistics files. The second workflow describes a feature extraction pipeline, outlining steps to fetch, decode, and process configuration data from idms-staging to generate a list of enabled features, which is then incorporated into performance statistics. Both workflows emphasize systematic data extraction, processing, and final output generation to maintain accurate and comprehensive performance metrics.

6.1. Programming Language Selection

The choice of programming language for the project is a crucial decision that impacts various aspects of development, including performance, flexibility, and compatibility with existing systems. After careful consideration of the project requirements and objectives, the following factors have influenced the selection of the programming language:

1. **Versatility:** The selected programming language should be versatile enough to support a wide range of data processing and visualization tasks efficiently.
2. **Library Support:** Availability of comprehensive libraries and frameworks for data manipulation, visualization, and interaction with cloud services is essential to streamline development and enhance productivity.
3. **Community Support:** A vibrant and active community around the programming language ensures access to resources, documentation, and community-driven solutions to familiar challenges.

Based on these considerations, **Python** has been chosen as the primary programming language for the project. Python offers the following advantages:

- **Rich Ecosystem:** Python boasts an extensive ecosystem of libraries and frameworks, including Pandas for data manipulation, Matplotlib for data visualization, and Boto3 for interaction with AWS services, making it well-suited for data processing and visualization tasks.
- **Compatibility:** Python's compatibility with various operating systems and platforms ensures seamless integration with existing systems and infrastructure.

6.2. Code Conventions

Maintaining consistent code conventions is essential for ensuring readability, maintainability, and collaboration within the project. The following subsections outline the code conventions adopted for the project:

6.2.1. Naming Conventions

- **Variables and Functions:** Use descriptive names that accurately convey the purpose of the variable or function. Follow the camelCase naming convention for variables and functions (e.g., processData, calculatePercentile).
- **Constants:** Constants should be in all uppercase with underscores separating words (e.g., MAX_ITERATIONS, DEFAULT_THRESHOLD).
- **Classes:** Class names should be in PascalCase (also known as UpperCamelCase) to distinguish them from variables and functions (e.g., DataProcessor, VisualizationManager).

6.2.2. File Organization

- **Module Structure:** Organize code into logical modules based on functionality. Each module should focus on a specific aspect of the project, such as data processing, visualization, or user interface.
- **File Naming:** Follow a consistent naming convention for files. Use descriptive names that reflect the contents of the file. Separate words with underscores if needed (e.g., data_processor.py, visualization_utils.py).

6.2.3. Declarations

- **Variable Declaration:** Declare variables at the beginning of each function or block to enhance readability and maintainability.

6.2.4. Comments

- **Purpose of Comments:** Comments should provide additional context or explanation where necessary, particularly for complex algorithms or non-obvious code segments.
- **Comment Style:** Follow the Python docstring conventions for documenting modules, classes, functions, and methods. Use inline comments sparingly and ensure they are concise and relevant.

6.3. Difficulties Encountered and Strategies Used to Tackle Them

Throughout the project's development, several challenges and difficulties were encountered. The following section highlights these challenges and outlines the strategies employed to address them effectively:

1. Data Parsing Complexity:
    - **Challenge:** Parsing raw data from various sources proved to be complex due to inconsistencies in data formats and structures across packages and versions.
    - **Strategy:** Implemented robust data parsing algorithms capable of handling diverse data formats. Utilized libraries and frameworks with built-in data parsing functionalities to streamline the process. Added exception handling within try-except blocks to avoid one parameter affecting another.
2. Performance Optimization:
    - **Challenge:** Processing large volumes of data within acceptable time frames presented performance optimization challenges.
    - **Strategy:** Employed optimization techniques such as parallel processing, caching, and algorithmic optimizations to enhance performance. Leveraged profiling tools to identify bottlenecks and optimize critical code segments. Implemented multi-threading using python libraries to create threads for each device and each parameter.
3. Integration with External Systems:

- **Challenge:** Integrating the project with external systems, including cloud services and databases, posed integration challenges. Requesting permission for credentials to confidential company databases and cloud services.
- **Strategy:** Collaborated closely with system administrators and stakeholders to ensure seamless integration. Utilized well-documented APIs and SDKs provided by external systems. Conducted thorough testing and validation to verify integration functionality. Stored credentials in encrypted format and used SSM keys for increased security.

4. User Interface Complexity:
   - **Challenge:** Designing a user-friendly and intuitive interface for interacting with complex data presented usability challenges.
   - **Strategy:** Conducted meetings and user feedback sessions for employees, to understand user requirements and preferences. Implemented an iterative design process, incorporating user feedback and making incremental improvements to the interface.
5. Error Handling and Recovery:
   - **Challenge:** Ensuring robust error handling and recovery mechanisms to handle unexpected errors and failures.
   - **Strategy:** Implemented comprehensive error handling strategies, including logging, exception handling, and automated recovery mechanisms. Conducted thorough testing to simulate and validate error scenarios, ensuring the reliability and resilience of the system.
6. Documentation and Knowledge Sharing:
   - **Challenge:** Maintaining up-to-date documentation and facilitating knowledge sharing among team members.
   - **Strategy:** Established clear documentation standards and guidelines. Encouraged regular documentation updates and knowledge sharing sessions. Leveraged collaboration tools and version control systems to centralize documentation and facilitate collaboration.
7. Timeline Constraints:
   - **Challenge:** Adhering to strict and urgent timelines to integrate this within threshold generation process.
   - **Strategy:** Created a priority list of parameters required and followed that order of implementation and simultaneously generated reports with implemented functionality.

6.4. Summary

In summary, Python was selected as the primary programming language for the project due to its versatility, rich ecosystem, ease of learning, and strong community support. Python's extensive libraries

and frameworks, such as Pandas for data manipulation and Matplotlib for data visualization, contributed to efficient data processing and visualization tasks. Adhering to consistent code conventions, including camelCase for variables and functions, PascalCase for classes, and all-uppercase for constants, ensured readability and maintainability of the codebase. The project followed a modular file organization structure, with logical separation of code into modules based on functionality. Declarations were organized systematically within classes and modules, accompanied by descriptive comments following Python docstring conventions. By consistently applying these language selection and code convention principles, the project maintained a clean, organized, and understandable codebase, facilitating collaboration and enhancing overall project quality.

7.1 Testing Workflow

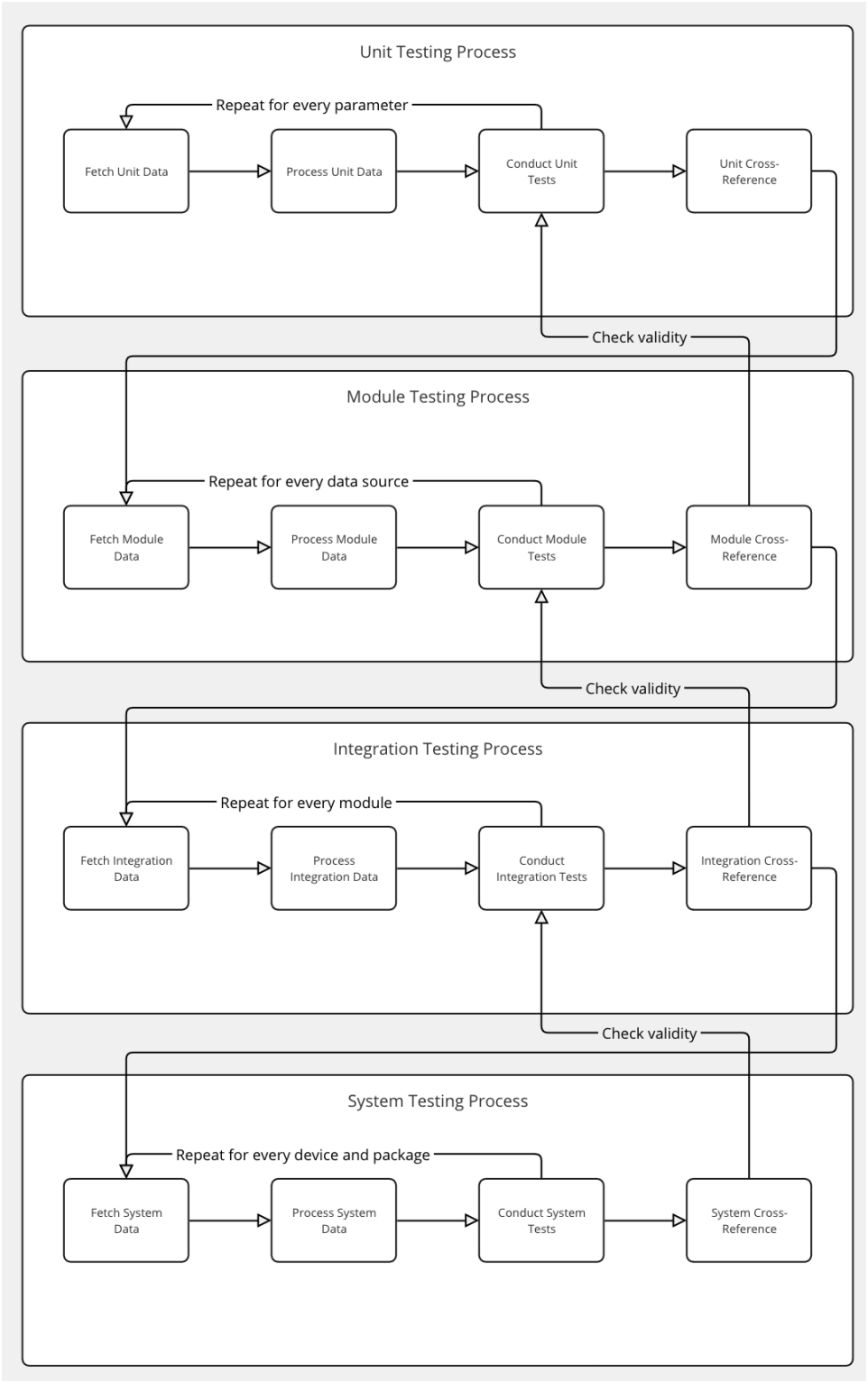


Fig 7.1 Testing Workflow Diagram

This detailed workflow diagram breaks down the testing process into four main phases:

1. **Unit Testing Process:** Involves fetching, processing, and cross-referencing data for individual parameters within the same module.
2. **Module Testing Process:** Tests each module independently, ensuring its functionality and cross-referencing data with previous testing processes.
3. **Integration Testing Process:** Integrates modules and tests their interaction, validating data flow and processing across modules.
4. **System Testing Process:** Performs system-level testing with real-world data and scenarios, verifying overall functionality and cross-referencing system-level data with previous testing processes.

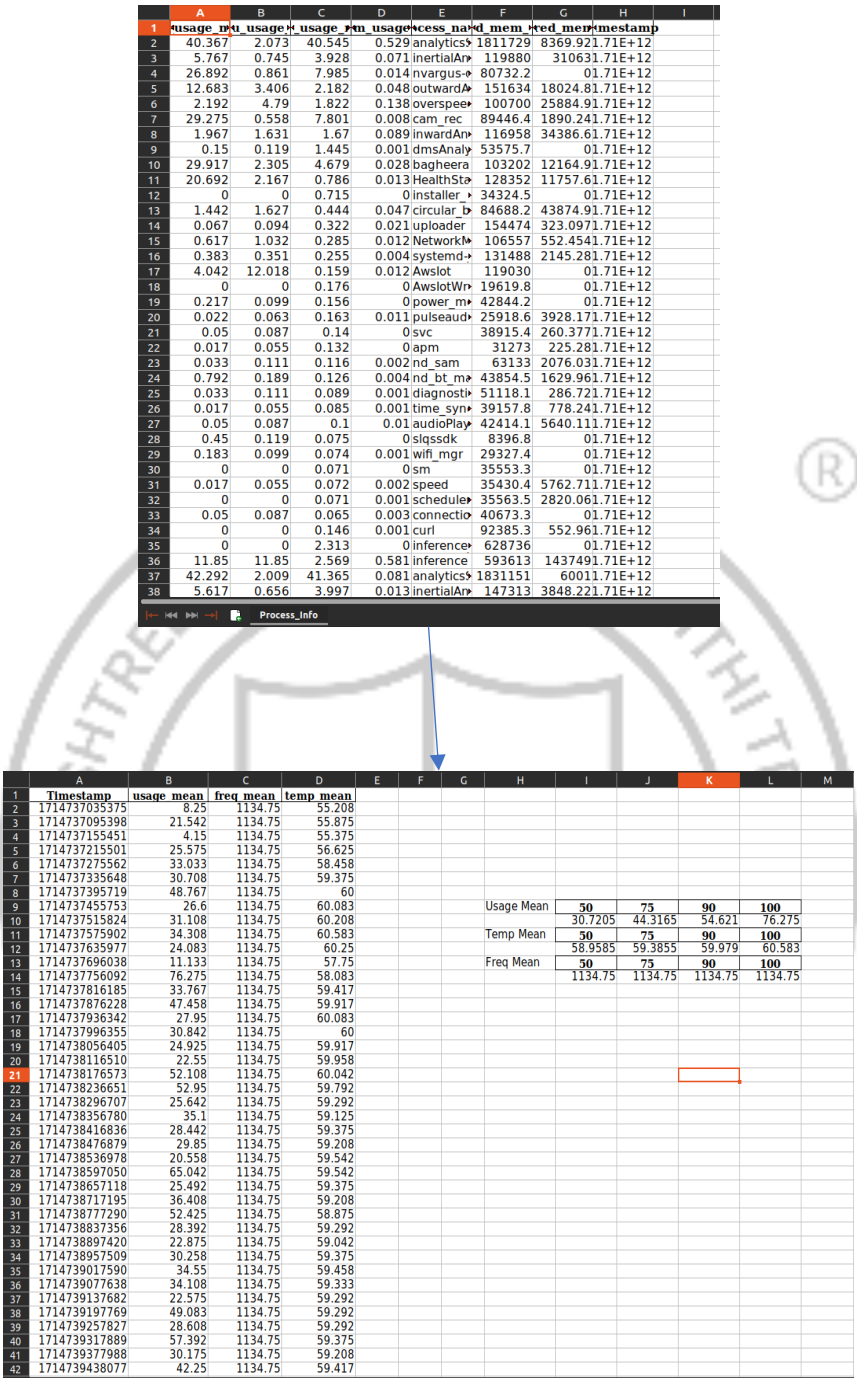
Each phase involves cross-referencing data with previous testing processes to ensure accuracy and reliability at every stage of the testing process.

7.2 Unit Testing

7.2.1 Unit Testing of the Healthstat Module

Description: Conduct unit testing for the Healthstat module to ensure its functionality and accuracy in processing health statistics data.

- Fetch raw health statistics data.
- Process raw data to calculate health statistics.
- Populate processed data.
- Crosscheck processed data with expected results.
- Repeat for multiple parameters within the Healthstat module.



	A	B	C	D	E	F	G	H	I
1	usage_m	u_usage	u_usage_m	m_usage	ccess_na	d_mem_	red_mem	mestamp	
2	40.367	2.073	40.545	0.529	analytics	1811729	8369.92	1.71E+12	
3	5.767	0.745	3.928	0.071	inertialAn	119880	310631	1.71E+12	
4	26.892	0.861	7.985	0.014	nvargus-d	80732.2	01	1.71E+12	
5	12.683	3.406	2.182	0.048	outwardA	151634	18024.8	1.71E+12	
6	2.192	4.79	1.822	0.138	overspee	100700	25884.9	1.71E+12	
7	29.275	0.558	7.801	0.008	cam_rec	89446.4	1890.24	1.71E+12	
8	1.967	1.631	1.67	0.089	inwardAn	116958	34386.6	1.71E+12	
9	0.15	0.119	1.445	0.001	dmsAnaly	53575.7	01	1.71E+12	
10	29.917	2.305	4.679	0.028	bagheera	103202	12164.9	1.71E+12	
11	20.692	2.167	0.786	0.013	HealthSta	128352	11757.6	1.71E+12	
12	0	0	0.715	0	installer_	34324.5	01	1.71E+12	
13	1.442	1.627	0.444	0.047	circular_b	84688.2	43874.9	1.71E+12	
14	0.067	0.094	0.322	0.021	uploader	154474	323.097	1.71E+12	
15	0.617	1.032	0.285	0.012	NetworkM	106557	552.454	1.71E+12	
16	0.383	0.351	0.255	0.004	systemd-	131488	2145.28	1.71E+12	
17	4.042	12.018	0.159	0.012	Awslot	119030	01	1.71E+12	
18	0	0	0.176	0	AwslotWr	19619.8	01	1.71E+12	
19	0.217	0.099	0.156	0	power_m	42844.2	01	1.71E+12	
20	0.022	0.063	0.163	0.011	pulseaud	25918.6	3928.17	1.71E+12	
21	0.05	0.087	0.14	0	svc	38915.4	260.377	1.71E+12	
22	0.017	0.055	0.132	0	apm	31273	225.28	1.71E+12	
23	0.033	0.111	0.116	0.002	nd_sam	63133	2076.03	1.71E+12	
24	0.792	0.189	0.126	0.004	nd_bt_ma	43854.5	1629.96	1.71E+12	
25	0.033	0.111	0.089	0.001	diagnosti	51118.1	286.72	1.71E+12	
26	0.017	0.055	0.085	0.001	time_syn	39157.8	778.24	1.71E+12	
27	0.05	0.087	0.1	0.01	audioPlay	42414.1	5640.11	1.71E+12	
28	0.45	0.119	0.075	0	slqssdk	8396.8	01	1.71E+12	
29	0.183	0.099	0.074	0.001	wifi_mgr	29327.4	01	1.71E+12	
30	0	0	0.071	0	sm	35553.3	01	1.71E+12	
31	0.017	0.055	0.072	0.002	speed	35430.4	5762.71	1.71E+12	
32	0	0	0.071	0.001	schedule	35563.5	2820.06	1.71E+12	
33	0.05	0.087	0.065	0.003	connectio	40673.3	01	1.71E+12	
34	0	0	0.146	0.001	curl	92385.3	552.961	1.71E+12	
35	0	0	2.313	0	inference	628736	01	1.71E+12	
36	11.85	11.85	2.569	0.581	inference	593613	1437491	1.71E+12	
37	42.292	2.009	41.365	0.081	analytics	1831151	60011	1.71E+12	
38	5.617	0.656	3.997	0.013	inertialAn	147313	3848.22	1.71E+12	

Process_Info

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Timestamp	usage_mean	freq_mean	temp_mean									
2	1714737035375	8.25	1134.75	55.208									
3	1714737095398	21.542	1134.75	55.875									
4	1714737155451	4.15	1134.75	55.375									
5	1714737215501	25.575	1134.75	56.625									
6	1714737275562	33.033	1134.75	58.458									
7	1714737335648	30.708	1134.75	59.375									
8	1714737395719	48.767	1134.75	60									
9	1714737455753	26.6	1134.75	60.083				Usage Mean	50	75	90	100	
10	1714737515824	31.108	1134.75	60.208					30.7205	44.3165	54.621	76.275	
11	1714737575902	34.308	1134.75	60.583				Temp Mean	50	75	90	100	
12	1714737635977	24.083	1134.75	60.25					58.9585	59.3855	59.979	60.583	
13	1714737696038	11.133	1134.75	57.75				Freq Mean	50	75	90	100	
14	1714737756092	76.275	1134.75	58.083					1134.75	1134.75	1134.75	1134.75	
15	1714737816185	33.767	1134.75	59.417									
16	1714737876228	47.458	1134.75	59.917									
17	1714737936342	27.95	1134.75	60.083									
18	1714737996355	30.842	1134.75	60									
19	1714738056405	24.925	1134.75	59.917									
20	1714738116510	22.55	1134.75	59.958									
21	1714738176573	52.108	1134.75	60.042									
22	1714738236651	52.95	1134.75	59.792									
23	1714738296707	25.642	1134.75	59.292									
24	1714738356780	35.1	1134.75	59.125									
25	1714738416836	28.442	1134.75	59.375									
26	1714738476879	29.85	1134.75	59.208									
27	1714738536978	20.558	1134.75	59.542									
28	1714738597050	65.042	1134.75	59.542									
29	1714738657118	25.492	1134.75	59.375									
30	1714738717195	36.408	1134.75	59.208									
31	1714738777290	52.425	1134.75	58.875									
32	1714738837356	28.392	1134.75	59.292									
33	1714738897420	22.875	1134.75	59.042									
34	1714738957509	30.258	1134.75	59.375									
35	1714739017590	34.55	1134.75	59.458									
36	1714739077638	34.108	1134.75	59.333									
37	1714739137682	22.575	1134.75	59.292									
38	1714739197769	49.083	1134.75	59.292									
39	1714739257827	28.608	1134.75	59.292									
40	1714739317889	57.392	1134.75	59.375									
41	1714739377988	30.175	1134.75	59.208									
42	1714739438077	42.25	1134.75	59.417									

Fig 7.2 Unit Testing – Process Information

7.2.2 Unit Testing for Database Module

Description: Conduct unit testing for the Database module to validate its functionality in interacting with the database and storing/retrieving data.

- Fetch sample data from the database.

- Perform database operations (e.g., insert, update, delete).
- Verify data integrity and consistency.
- Crosscheck database records with expected results.
- Repeat for different database operations and scenarios.

7.2.3 Unit Testing for Observation Module

Description: Conduct unit testing for the Observation Files module to ensure its functionality in processing observation data files.

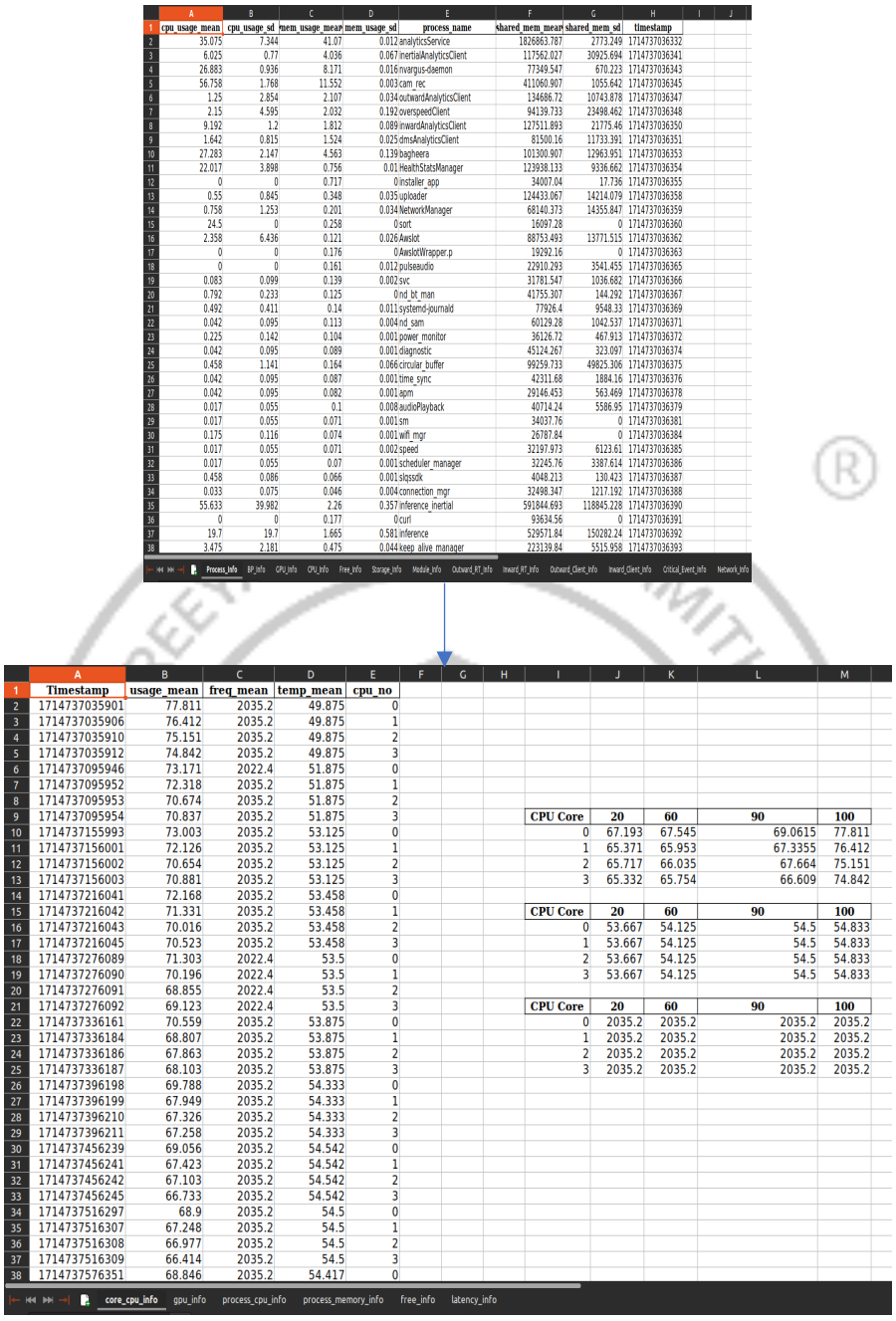
- Fetch sample observation data files.
- Parse and process observation data.
- Populate processed observation data.
- Crosscheck processed data with expected results.
- Repeat for multiple observation data files and parameters.

7.3 Integration Testing

7.3.1 Integration Testing of Healthstat and Database Modules

Description: Conduct integration testing to validate the interaction between the Healthstat and Database modules.

- Fetch sample health statistics data.
- Process health statistics data using the Healthstat module.
- Store processed data in the database using the Database module.
- Retrieve data from the database.
- Crosscheck retrieved data with processed health statistics data.
- Validate data consistency and integrity between the modules.



cpu_usage_mean	cpu_usage_sd	mem_usage_mean	mem_usage_sd	process_name	shared_mem_mean	shared_mem_sd	timestamp
35.075	7.344	41.07	0.012	analyticsService	1826863.787	2773.249	1714737036332
6.025	0.77	4.036	0.067	inertialAnalyticsClient	117562.027	30925.694	1714737036341
26.883	0.936	8.171	0.016	nvargus-daemon	77349.547	670.223	1714737036343
56.758	1.768	11.552	0.003	cam_rec	411060.907	1055.642	1714737036345
1.25	2.854	2.107	0.034	outwardAnalyticsClient	134686.72	10743.878	1714737036347
2.15	4.595	2.032	0.192	overspeedClient	94139.733	23498.462	1714737036348
9.192	1.2	1.812	0.089	inwardAnalyticsClient	127511.893	21775.46	1714737036350
1.642	0.815	1.524	0.025	dmsAnalyticsClient	81500.16	11733.391	1714737036351
27.283	2.147	4.563	0.139	bagheera	101300.907	12963.951	1714737036353
22.017	3.898	0.756	0.01	HealthStatsManager	123938.133	9336.662	1714737036354
0	0	0.717	0	installer_app	34007.04	17.736	1714737036355
0.55	0.845	0.348	0.035	uploader	124433.067	14214.079	1714737036358
0.758	1.253	0.201	0.034	NetworkManager	68140.373	14355.847	1714737036359
24.5	0	0.258	0	sort	16097.28	0	1714737036360
2.358	6.436	0.121	0.026	Awslot	88753.493	13771.515	1714737036362
0	0	0.176	0	AwsIotWrapper.p	19292.16	0	1714737036363
0	0	0.161	0.012	pulseaudio	22910.293	3541.455	1714737036365
0.083	0.099	0.139	0.002	svc	31781.547	1036.682	1714737036366
0.792	0.233	0.125	0	nd_bt_man	41755.307	144.292	1714737036367
0.492	0.411	0.14	0.011	systemd-journald	77926.4	9548.33	1714737036369
0.042	0.095	0.113	0.004	nd_sam	60129.28	1042.537	1714737036371
0.225	0.142	0.104	0.001	power_monitor	36126.72	467.913	1714737036372
0.042	0.095	0.089	0.001	diagnostic	45124.267	323.097	1714737036374
0.458	1.141	0.164	0.066	circular_buffer	99259.733	49825.306	1714737036375
0.042	0.095	0.087	0.001	time_sync	42311.68	1884.16	1714737036376
0.042	0.095	0.082	0.001	apm	29146.453	563.469	1714737036378
0.017	0.055	0.1	0.008	audioPlayback	40714.24	5586.95	1714737036379
0.017	0.055	0.071	0.001	sm	34037.76	0	1714737036381
0.175	0.116	0.074	0.001	wifi_mgr	26787.84	0	1714737036384
0.017	0.055	0.071	0.002	speed	32197.973	6123.61	1714737036385
0.017	0.055	0.07	0.001	scheduler_manager	32245.76	3387.614	1714737036386
0.458	0.086	0.066	0.001	slqssdk	4048.213	130.423	1714737036387
0.033	0.075	0.046	0.004	connection_mgr	32498.347	1217.192	1714737036388
55.633	39.982	2.26	0.357	inference_inertial	591844.693	118845.228	1714737036390
0	0	0.177	0	curl	93634.56	0	1714737036391
19.7	19.7	1.665	0.581	inference	529571.84	150282.24	1714737036392
3.475	2.181	0.475	0.044	keep_alive_manager	223139.84	5515.958	1714737036393

Timestamp	usage_mean	freq_mean	temp_mean	cpu_no
1714737035901	77.811	2035.2	49.875	0
1714737035906	76.412	2035.2	49.875	1
1714737035910	75.151	2035.2	49.875	2
1714737035912	74.842	2035.2	49.875	3
1714737095946	73.171	2022.4	51.875	0
1714737095952	72.318	2035.2	51.875	1
1714737095953	70.674	2035.2	51.875	2
1714737095954	70.837	2035.2	51.875	3
1714737155993	73.003	2035.2	53.125	0
1714737156001	72.126	2035.2	53.125	1
1714737156002	70.654	2035.2	53.125	2
1714737156003	70.881	2035.2	53.125	3
1714737216041	72.168	2035.2	53.458	0
1714737216042	71.331	2035.2	53.458	1
1714737216043	70.016	2035.2	53.458	2
1714737216045	70.523	2035.2	53.458	3
1714737276089	71.303	2022.4	53.5	0
1714737276090	70.196	2022.4	53.5	1
1714737276091	68.855	2022.4	53.5	2
1714737276092	69.123	2022.4	53.5	3
1714737336161	70.559	2035.2	53.875	0
1714737336184	68.807	2035.2	53.875	1
1714737336186	67.863	2035.2	53.875	2
1714737336187	68.103	2035.2	53.875	3
1714737396198	69.788	2035.2	54.333	0
1714737396199	67.949	2035.2	54.333	1
1714737396210	67.326	2035.2	54.333	2
1714737396211	67.258	2035.2	54.333	3
1714737456239	69.056	2035.2	54.542	0
1714737456241	67.423	2035.2	54.542	1
1714737456242	67.103	2035.2	54.542	2
1714737456245	66.733	2035.2	54.542	3
1714737516297	68.9	2035.2	54.5	0
1714737516307	67.248	2035.2	54.5	1
1714737516308	66.977	2035.2	54.5	2
1714737516309	66.414	2035.2	54.5	3
1714737576351	68.846	2035.2	54.417	0

CPU Core	20	60	90	100
0	67.193	67.545	69.0615	77.811
1	65.371	65.953	67.3355	76.412
2	65.717	66.035	67.664	75.151
3	65.332	65.754	66.609	74.842

CPU Core	20	60	90	100
0	53.667	54.125	54.5	54.833
1	53.667	54.125	54.5	54.833
2	53.667	54.125	54.5	54.833
3	53.667	54.125	54.5	54.833

CPU Core	20	60	90	100
0	2035.2	2035.2	2035.2	2035.2
1	2035.2	2035.2	2035.2	2035.2
2	2035.2	2035.2	2035.2	2035.2
3	2035.2	2035.2	2035.2	2035.2

Fig 7.3 Integration Testing Process – Healthstat and DB Module

7.3.2 Integration Testing of Database and Observation Files Modules

Description: Conduct integration testing to validate the interaction between the Database and Observation Files modules.

- Fetch sample observation data files.
- Parse and process observation data using the Observation Files module.

- Store processed observation data in the database using the Database module.
- Retrieve observation data from the database.
- Crosscheck retrieved data with processed observation data.
- Validate data consistency and integrity between the modules.

7.3.3 Integration Testing of Healthstat and Observation Files Modules

Description: Conduct integration testing to validate the interaction between the Healthstat and Observation Files modules.

- Fetch sample health statistics data.
- Process health statistics data using the Healthstat module.
- Fetch sample observation data files.
- Parse and process observation data using the Observation Files module.
- Integrate processed health statistics data and observation data.
- Crosscheck integrated data with expected results.
- Validate data consistency and integrity between the modules.

7.4 System Testing

System testing serves as the conclusive phase where the entire software system, comprising the Healthstat, Database, and Observation Files modules, undergoes rigorous evaluation to ensure its functionality, performance, and reliability in real-world scenarios. This encompasses testing the end-to-end workflow, validating data processing and interaction between modules, and assessing the system's ability to handle diverse data and user loads. Performance, scalability, security, and usability aspects are scrutinized to guarantee the system's readiness for deployment. System testing results, meticulously documented, inform necessary refinements or enhancements to deliver a robust and user-friendly solution. By validating the software system comprehensively, we ensure its alignment with project objectives and user expectations, thus enhancing its value proposition and contributing to project success. This tool has been tested with multiple devices as input simultaneously with over 50 devices per product line at a time with varied use-cases, environments and data availability through network connectivity and other parameters. It has been testes thoroughly considering user feedback where employees cross-referenced the data from the IDMS portal and gave inputs on whether the reports match the data. This has been performed over multiple days on a lab device as well as an experiment machine running on an EC2 instance for studying computational and storage requirements for the tool.

8.1. Evaluation Metrics

The chosen evaluation metrics for the experimentation phase were speed, reliability, and modularity. Speed was prioritized due to the necessity of generating reports for hundreds of staging devices to test new packages, enabling teams to study data efficiently. Reliability ensured the accuracy and consistency of data processing, while modularity focused on the trade-off between redundant code and code setup to facilitate easy addition of parameters/changes and accommodate format changes in raw data.

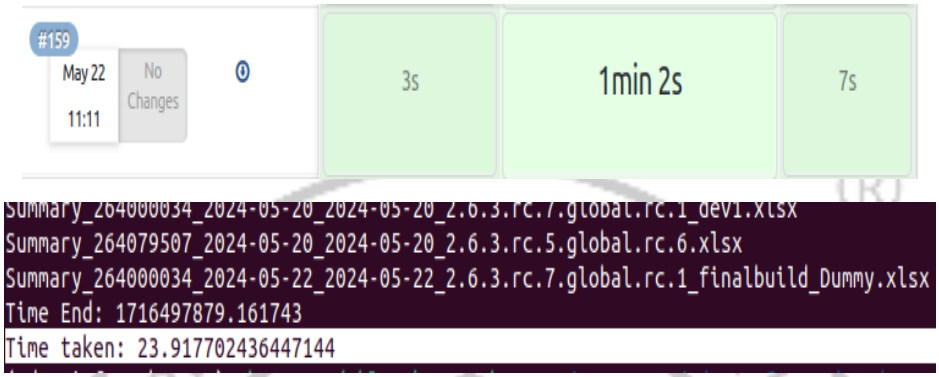


Fig 8.1 Evaluation Metric - Speed

8.2. Experimental Dataset

The experimental dataset consisted of 50 devices from each of the four products: D-210, D-230, D-430, and D-450. This dataset was selected to represent a diverse range of devices and provide sufficient data for performance analysis across different device types and configurations. Each device has its own line of OS versions and OTA versions and vary in that some are staging devices in the lab, some in the field and some production devices with a wide array of usage characteristics, data availability and system utilities used.

Device ID	Tenant Name	Device Version	Date start	Date end
103012400003	Peckham Materials Corp.	5.5.29.rc.11	15/03/24	15/04/24
103012400026	PGT Trucking	5.5.29.rc.11	15/03/24	15/04/24
103012400034	Edco Waste & Recycling Services	5.5.29.rc.11	15/03/24	15/04/24
103012400035	PacTrack Inc.	5.5.29.rc.11	15/03/24	15/04/24
103022400008	PGT Trucking	5.5.29.rc.11	15/03/24	15/04/24
103022400016	PacTrack Inc.	5.5.29.rc.11	15/03/24	15/04/24
103022400018	Atlas Disposal Industries LLC	5.5.29.rc.11	15/03/24	15/04/24
103022400020	PacTrack Inc.	5.5.29.rc.11	15/03/24	15/04/24
103022400021	Atlas Disposal Industries LLC	5.5.29.rc.11	15/03/24	15/04/24
103022400023	PacTrack Inc.	5.5.29.rc.11	15/03/24	15/04/24
264003354	In-Line Fence & Railing Co., Inc.	2.6.3.rc.7	15/03/24	15/04/24
264027392	Heavy Transport Solutions Inc,528	2.6.3.rc.7	15/03/24	15/04/24
264042976	Brazos Forest Products	2.6.3.rc.7	15/03/24	15/04/24
264001006	Arizona Foundation Solutions	2.6.3.rc.7	15/03/24	15/04/24
264034092	Dennis Taylor and Company Inc.	2.6.3.rc.7	15/03/24	15/04/24
264084995	Carson Freight LLC,407	2.6.3.rc.7	15/03/24	15/04/24
264002335	Murrow's Transfer, Inc.	2.6.3.rc.7	15/03/24	15/04/24
264081414	Manufacturers Reserve Supply	2.6.3.rc.7	15/03/24	15/04/24
264005574	Carson Freight LLC,407	2.6.3.rc.7	15/03/24	15/04/24
3633092923	Amazon Middle Mile / TOM	3.6.1.rc.4	15/03/24	15/04/24
3633093494	Amazon Middle Mile / TOM	3.6.1.rc.4	15/03/24	15/04/24
3633093410	Amazon Middle Mile / TOM	3.6.1.rc.4	15/03/24	15/04/24
3633021437	Amazon Middle Mile / TOM	3.6.1.rc.4	15/03/24	15/04/24
3633093326	Amazon Middle Mile / TOM	3.6.1.rc.4	15/03/24	15/04/24
3633093354	Amazon Middle Mile / TOM	3.6.1.rc.4	15/03/24	15/04/24
3633093325	Amazon Middle Mile / TOM	3.6.1.rc.4	15/03/24	15/04/24
3633093361	Amazon Middle Mile / TOM	3.6.1.rc.4	15/03/24	15/04/24

Fig 8.2 Example Dataset for Performance Analysis

8.3. Performance Analysis

The performance analysis involved subjecting the code to extended device lists and time intervals to evaluate parallelism and performance locally and on Jenkins (cloud resource instance). The analysis revealed that the optimized code was approximately 50% faster for dataset sizes of 10 or more devices over a period of more than 4-5 days. Additionally, it demonstrated a performance improvement of about 40% for smaller datasets compared to the previous framework that had been developed. The code runs faster on Jenkins as it uses an AWS cloud instance and hence downloading of files from their respective S3 paths is quicker. The time to complete the task purely depends on the system uptime, alerts and critical events generated and amount of data available as all processes from parsing to processing and populating depend on the amount of data and extent of parallelisation.

8.4. Summary

In summary, the experimentation phase focused on evaluating the software's performance using the chosen metrics of speed, reliability, and modularity. The experimental dataset comprised 50 devices from each of the four products, providing a comprehensive sample for analysis. The performance analysis revealed significant improvements in code efficiency, with the optimized code demonstrating faster processing times and better scalability compared to the previous framework. These findings underscore the effectiveness of the optimization efforts and highlight the importance of performance testing in ensuring the software's effectiveness and reliability in real-world scenarios.

9.1. Limitations of the Project

The current GUI development phase presents some limitations:

- **Checkbox Limitation:** The checkbox feature for extracting relevant information and populating files may be restrictive, as it limits the user to predefined options.
- **Manual Device ID Entry:** While providing an option to enter Device IDs manually offers flexibility, it may increase the likelihood of errors if not validated properly.
- **Data Visualisation Limitations:** While the inclusion of graphs for raw data facilitates anomaly detection, peaks, and trends, it may not cover all possible data analysis needs. Additionally, interactive plots may require additional development effort and technical expertise.

9.2. Future Enhancements

To address the project's limitations and further enhance its capabilities, several future enhancements can be considered:

- **Dynamic Checkbox Options:** Implement dynamic checkbox options to allow users to select relevant metrics and files based on their specific requirements, enhancing flexibility and usability.
- **Excel Sheet Integration:** Improve the Excel Sheet integration feature by providing automated data validation and error handling to minimize manual entry errors.
- **Advanced Data Visualisation:** Explore advanced data visualisation techniques, such as interactive dashboards and real-time updates, to enhance user understanding and analysis of data.
- **Integration of Additional Data Sources:** Expand the project's capabilities by integrating data from additional sources, such as ELD data, to provide more comprehensive insights and analysis.
- **Enhanced Information Highlighting:** Incorporate features to highlight potential issues more effectively, such as alerts, notifications, or automated anomaly detection algorithms, to improve data analysis and decision-making.

9.3. Summary

In summary, while the current GUI development phase introduces useful features for data extraction and visualisation, it also has limitations regarding checkbox options, manual entry of Device IDs, and data visualisation capabilities. However, the project's future enhancements aim to address these limitations and further improve its functionality and usability. By implementing dynamic checkbox options, enhancing Excel Sheet integration, exploring advanced data visualisation techniques, integrating additional data sources, and enhancing information highlighting features, the project will evolve to meet evolving user needs and provide more comprehensive and actionable insights from the data.