

**Final Report: Project Semester**  
on  
**Enhancing Quality Assurance Through Real-Time Log Analytic  
Dashboard & AI-Powered Test Case Generation Agent**

Submitted by  
Kunal Rao  
10211021  
B.E Electrical and Computer Engineering

Under the Guidance of

Host Mentor  
Mr. Aashish Paliwal  
Module Lead  
Logic Fruit Technologies

Faculty Supervisor  
Dr. Alok Kumar Shukla  
Assistant Professor  
DEIE, TIET, Patiala



**THAPAR INSTITUTE**  
OF ENGINEERING & TECHNOLOGY  
(Deemed to be University)

**2025**

**Department of Electrical and Instrumentation Engineering**  
**Thapar Institute of Engineering and Technology, Patiala**  
*(Declared as Deemed-to-be-University u/s 3 of the UGC Act., 1956)*  
**Post Bag No. 32, Patiala – 147004**  
**Punjab (India)**

## To Whomsoever it may concern



This is to certify that **Kunal Rao**, a student of Thapar Institute of Engineering and Technology, Patiala, has undertaken a project titled “**Enhancing Quality Assurance Through Real-Time Log Analytic Dashboard & AI-Powered Test Case Generation Agent**” during his industrial training program at **Logic Fruit Technologies Pvt. Ltd.**

The organization has no objection to the submission of this report to TIET for the purpose of final presentation and viva, as part of the requirements for the award of the B.E. degree in Electrical and Computer Engineering.

The industrial training program commenced on 3rd March and is currently ongoing.

*Name of Industrial Mentor:* Mr. Aashish Paliwal

*Name of student:* Kunal Rao

Signature of Industrial Mentor:	Signature of Student:
	

Date: 2<sup>nd</sup> June, 2025

Place: Gurugram, Haryana

## Declaration

I hereby declare that the midway report titled, “**Enhancing Quality Assurance Through Real-Time Log Analytics Dashboard & AI-Powered Test Case Generation Agent**”, submitted as a partial requirement for the Project Semester (ULC891) course towards the Bachelor of Engineering degree in Electrical and Computer Engineering at Thapar Institute of Engineering and Technology, Patiala, is an accurate representation of the work I undertook. This work is carried out under the guidance and supervision of Mr. Aashish Paliwal, Module Lead at Logic Fruit Technologies (Host Mentor), and Dr. Alok Kumar Shukla, Assistant Professor at the Department of Electrical and Instrumentation Engineering, TIET, Patiala, India.

Place: Patiala

Kunal Rao

Date: 23<sup>rd</sup> April, 2025

102119021

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.



Host Mentor  
Mr. Aashish Paliwal  
Module Lead  
Logic Fruit Technologies, Gurugram



Faculty Supervisor  
Dr. Alok Kumar Shukla  
Assistant Professor  
DEIE, TIET, Patiala

## **Acknowledgement**

I wish to sincerely thank my host mentor Mr. Aashish Paliwal for his valuable guidance and continuous support during this project semester. I also express my gratitude to Dr. Alok Kumar Shukla, my faculty supervisor, for his consistent advice and helpful insights. My appreciation also extends to the team at Logic Fruit Technologies for their assistance, cooperation, and the resources provided, all of which significantly contributed to the progress of my project.

**Kunal Rao**

**102119021**

## Abstract

This report outlines the progress of the project titled **“Enhancing Quality Assurance Through Real-Time Log Analytics Dashboard & AI-Powered Test Case Generation Agent”** undertaken at Logic Fruit Technologies. The project aims to enhance quality assurance by implementing an automated monitoring system via a real-time log analytics dashboard and developing an AI-powered test case generation agent using LangChain and the Llama3.1 8B model.

At this stage, the real-time log analytics dashboard has been successfully developed, integrated with the generic test automation framework. The AI-based test case generation agent, named as **“TCG Agent”** has also been developed and is currently undergoing validation with various Software Requirements Documents (SRDs), only deployment remains.

The next steps involve completing the deployment of the test case generation agent, conducting extensive testing, debugging, performance optimization, and preparing for final delivery of the complete system.

## Table of Contents

<b>Declaration</b>	<b>i</b>
<b>Acknowledgement</b>	<b>ii</b>
<b>Abstract</b>	<b>iii</b>
<b>List of Figures</b>	<b>vii</b>
<b>List of Abbreviations</b>	<b>viii</b>
<b>1. Introduction</b>	<b>1</b>
1.1 Introduction	1
1.1.1 Company Profile	1
1.1.2 Role of Quality Assurance in Software Development	1-2
1.2 Real-Time Log Analytics Dashboard	3-4
1.3 AI-Powered Test Case Generation Agent	4-5
<b>2. Project Objective and Goal</b>	<b>6</b>
2.1 Goals	6
2.2 Objectives	6
2.2.1 Real-Time Log Analytics Dashboard	6
2.2.2 AI-Powered Test Case Generation Agent	6-7
<b>3. Literature Review</b>	<b>8</b>
3.1 Real-Time Log Analytics	8
3.2 Data Visualization and Dashboard design	8-9
3.3 Automated Test Case Generation	9-10
3.4 Frameworks: LangChain and Llama Model	10-11

3.5	Summary of Research Gap	11
<b>4.</b>	<b>Technical Challenges and Solutions</b>	
4.1	Challenge: Long and Complex SRDs	12
4.2	Challenge: Prompt Injection and Model Hallucination	12-13
4.3	Challenge: Dashboard Performance Lag Under High-Frequency Input	13
4.4	Challenge: Format Preservation in Exported Test Case Documents	13-14
<b>5.</b>	<b>Methodology</b>	15
5.1	Real-Time Log Analytics Dashboard	15
5.1.1	Data Collection and Aggregation	15
5.1.2	Data Processing and Parsing	15
5.1.3	Real-Time Analysis and Monitoring	15
5.1.4	Dashboard Design and Visualization	15
5.2	AI-Powered Test Case Generation Agent	16
5.2.1	Requirement Analysis and Contextual Understanding	16-17
5.2.2	Data Preparation and Model Fine-tuning	17
5.2.2.1	SRD to Test Case mapping	17
5.2.2.2	Conversion to Alpaca format	17
5.2.2.3	Dataset evaluation and refinement	18
5.2.3	Test Scenario Generation	18
5.2.4	Integration into QA Workflow	18-19
<b>6.</b>	<b>Conclusion and Future Steps</b>	20
6.1	Conclusion	20
6.2	Future Steps	20-21
<b>7.</b>	<b>References</b>	22-23

## **List of Figures**

1.2.1	Real-time QA workflow integration	3
1.2.2	Real-time Log analytics Dashboard	4
1.2.3	Test cycle analytics tab	4
4.1	Flowchart: Log analytics dashboard	12
4.2	Flowchart: Test Case Generation Agent	14



## **List of Abbreviations**

<b>QA</b>	Quality Assurance
<b>AI</b>	Artificial Intelligence
<b>UI</b>	User Interface
<b>CI/CD</b>	Continuous Integration / Continuous Deployment
<b>JSON</b>	JavaScript object Notation
<b>CSV</b>	Comma Seperated Values
<b>XML</b>	Extensible Markup Language
<b>API</b>	Application Programming Interface
<b>ML</b>	Machine Learing
<b>NLP</b>	Natural Language Processing
<b>LLM</b>	Large Language Model
<b>SRS</b>	Software Requirement Specification
<b>GPU</b>	Graphical Processing Unit
<b>KPI</b>	Key Performance Indicator

# **Chapter 1**

## **Introduction**

### **1.1 Introduction**

#### **1.1.1 Company Profile**

Logic Fruit Technologies Pvt. Ltd. is an engineering firm that focuses on developing practical solutions in areas like embedded systems, digital signal processing, and automated testing. The company works on specialized projects involving FPGA design, high-speed circuit boards, and custom protocols, primarily serving industries such as aerospace, defence, and telecom.

Over the years, Logic Fruit Technologies has built a reputation for delivering technically sound, reliable products to clients in both Indian and international markets. The company emphasizes hands-on problem-solving and encourages a strong learning environment, where engineers and interns are given the opportunity to engage with complex, real-world challenges.

As part of my industrial training, I was placed in the R&D department, where I contributed to projects aimed at improving software quality assurance. My work involved developing systems for log analysis and test case generation. The support from mentors and access to technical resources at the company played a key role in helping me complete this project successfully.

#### **1.1.2 Role of Quality Assurance in Software Development**

Software development is changing and there is a need for more advanced Quality Assurance (QA) to ensure complex systems are reliable, performant, and stable. Legacy QA approaches simply can't keep up with the fast development cycles and technical complexity of today's applications, no matter how fundamental they are. This has precipitated an explosion of 'next-gen scrappy QA techniques' that rely on automation/artificial intelligence/real-time analytics to deliver effective solutions to these pressing challenges.

A major improvement was the use of real-time log analytics dashboards. These dashboards provide teams instant, actionable insights on software performance and stability by visualizing testing data, trends and outcomes in real-time. Through the automation of test execution, combined with reporting dashboards, teams are able to expedite testing, pinpoint issues faster and make decisions based on empirical data without blocking development. Such strategy fosters cooperation and maintains good quality during the whole cycle of software development [1].

The scope of the software testing is already being redefined by AI and machine learning, too. AI Based Tools will be able to automatically create and sustain the test cases by examining the requirements, user stories and history. This not only decreases the amount of manual labor needed, but also provides a more thorough test coverage, even for edge cases that the humans might forget. Predictive analytics then also help teams to move the needle on identifying potential hotspots that can be tested and resolved early[2][4].

The intelligent Test Case Generation Agent makes a basic test framework for manual testers. By leveraging llama3.1-8B fine-tuned on custom dataset, this agent can continuously adapt test cases in real time to align with evolving application requirements. Such automation accelerates the creation and execution of tests, minimizes maintenance efforts through self-healing scripts, and delivers deeper test coverage than manual methods. This results in faster release cycles, reduced costs, and improved overall product quality [3][4].

Additionally, incorporating these technologies into continuous integration and delivery (CI/CD) pipelines facilitates ongoing testing. Automated and AI-based testing tools can initiate tests with each code modification, guaranteeing that issues are identified promptly and application stability is maintained throughout the software development process [1][2].

In Conclusion, the combination of real-time analytics and smart automation enables QA teams to concentrate on strategic activities, like exploratory testing and enhancing user experiences, while technology efficiently handles routine and repetitive tasks. In conclusion, the merging of real-time log analytics and AI-powered test case generation is revolutionizing software QA

## 1.2 Real-Time Log Analytics Dashboard

The fundamental to maintaining a robust system performance is log analytics. As systems grow in complexity, the sheer volume and diversity of log data generated from applications, infrastructure, and network devices can quickly overwhelm traditional manual inspection methods, making them inefficient and susceptible to oversight and even negligence. The Real-Time Log Analytics Dashboard that is developed, directly addresses these challenges by automating the aggregation, analysis, and visualization of log data, thereby transforming raw logs into actionable insight.

The main aim of real-time log analytics dashboards is to centralize and present log data in an intuitive, easy-to-read format. By capturing logs, metrics, and traces into a single unified view, teams can efficiently monitor critical metrics from one console. This centralization is particularly valuable for distributed systems and cloud-native architectures, where logs are generated from multiple sources and environments. The dashboard enables users to detect anomalies, observe trends, and make proactive, data-driven decisions, greatly improving operational efficiency and reducing response times to incidents.

Proactive monitoring is another core benefit of real-time log analytics. The dashboard continuously scans for application crashes, unexpected shutdowns, security and other operational events, providing immediate alerts when issues arise.

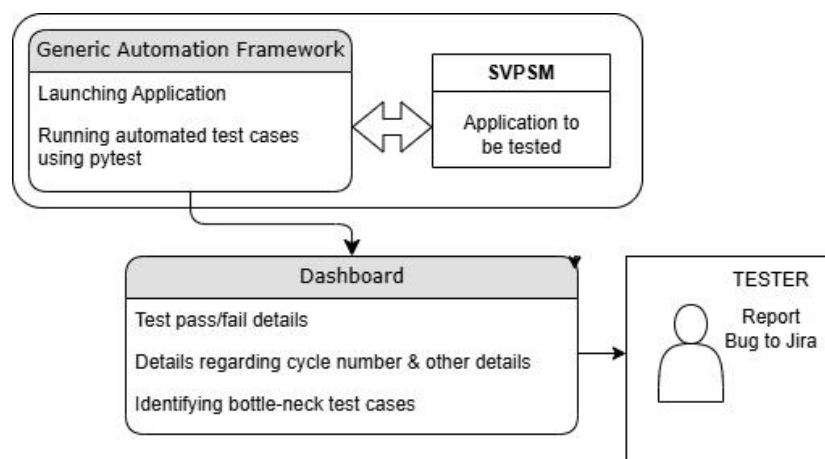


Figure 1.2.1: Real-Time QA workflow Integration

Moreover, real-time dashboards empower manual testers to conduct detailed testing. This capability is essential for understanding the root cause of incidents, identifying long-term trends, and ensuring compliance with security and operational standards. Advanced dashboards often incorporate artificial intelligence and machine learning to enhance anomaly detection, pattern recognition, and predictive analytics, further strengthening an organization’s ability to anticipate and mitigate risks.

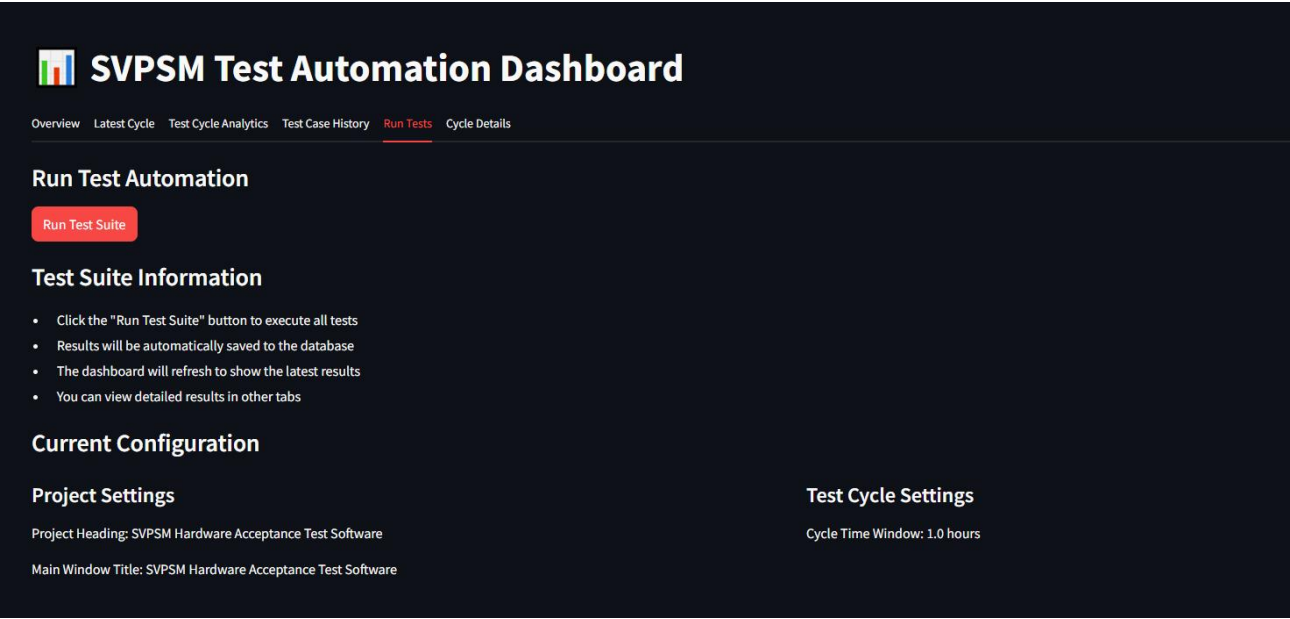


Figure 2.2.2: Real-Time Log Analytics Dashboard

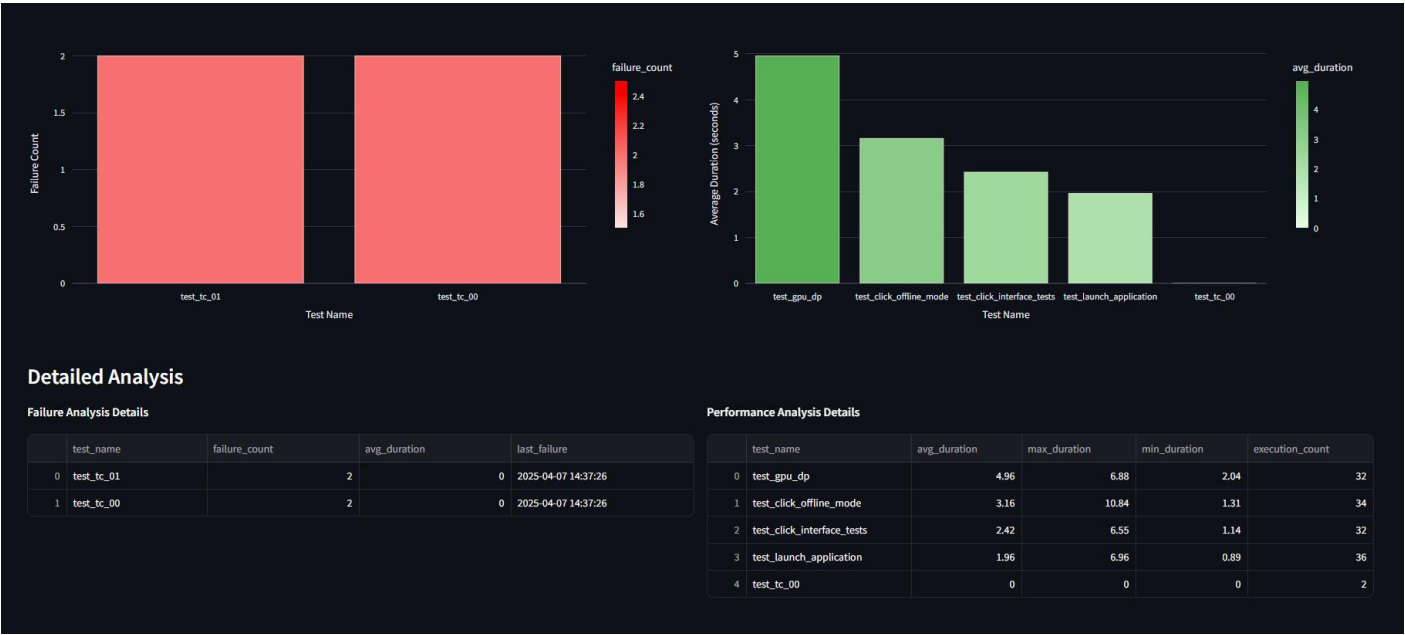


Figure 3.2.3: Test Cycle Analytics Tab

In summary, the Real-Time Log Analytics Dashboard transforms the way organizations approach system monitoring and incident response. By automating log data collection, analysis, and visualization, it delivers immediate insights, supports proactive operations, and enhances both performance and security across complex software infrastructures.

### 1.3 AI-Powered Test Case Generation Agent

Automated test case generation marks a transformative improvement in the way Quality Assurance (QA) processes are handled. In conventional workflows, test cases are manually written—a time-consuming task that demands significant human effort and is prone to inconsistency and oversight. These limitations often result in inadequate test coverage, allowing bugs to go undetected and increasing the time and cost required for later fixes. In contrast, automated test generation ensures more consistent and thorough validation of software behavior across a wide range of scenarios.

In this project, I have successfully developed a Test Case Generation Agent that utilizes the LangChain framework in combination with the LLaMA 3.1 8B language model, which is fine-tuned on custom dataset. LangChain enables the orchestration of multiple AI-driven tasks, maintaining context and coherence throughout the generation pipeline. Paired with LLaMA 3.1 8B's strong natural language capabilities, this setup allows the agent to generate well-structured, detailed, and context-aware test scenarios based on software requirements.

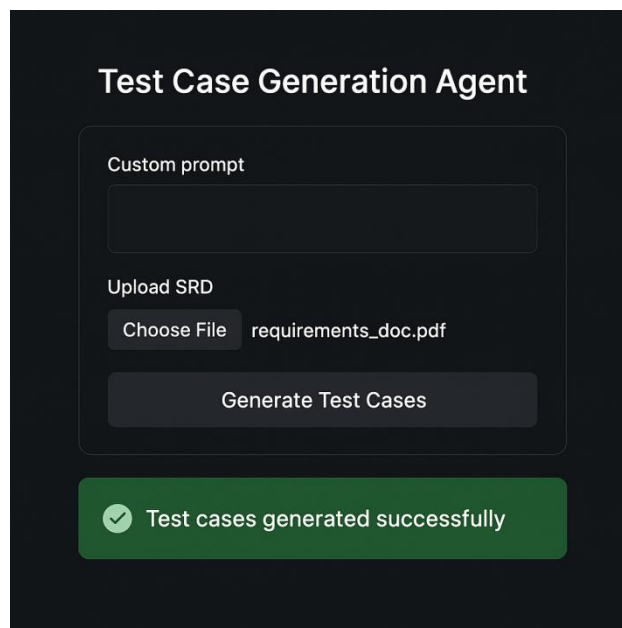


Figure 1.3.1: Test case generation agent user interface

With the development phase completed, the agent is now undergoing validation using a variety of Software Requirement Documents (SRDs) to ensure its effectiveness and adaptability. Once testing is finalized, it will be deployed on the company's server or on a standalone system configured for that purpose. This approach not only reduces manual intervention but also minimizes human error, ensures better test coverage, and allows the testing process to evolve alongside software changes. Overall, this agent stands as a significant leap forward in automated software testing, improving both the speed and reliability of QA workflows.

## **Chapter 2**

### **Project Objective and Goal**

This project, titled "Enhancing Quality Assurance Through Real-Time Log Analytics Dashboard & Test Case Generation Agent," aims to strengthen software Quality Assurance (QA) by introducing innovative solutions. To ensure clarity, effectiveness, and comprehensive coverage, the objectives of this project have been structured into two main modules: the Real-Time Log Analytics Dashboard and the Intelligent Test Case Generation Agent.

#### **2.1 Goals**

The primary goal of this project is to significantly improve software reliability, maintainability, and quality assurance efficiency at Logic Fruit Technologies by developing:

- A robust, scalable, and intuitive Real-Time Log Analytics Dashboard capable of aggregating and visualizing diverse log data.
- A sophisticated Test Case Generation Agent leveraging advanced AI language modeling frameworks—LangChain combined with the Llama3.1 8B model, fine-tuned on custom dataset—to automate and optimize test scenario creation.

These goals aim to substantially reduce manual intervention, minimize errors, and enhance system transparency, thereby promoting proactive management and preventive measures in software quality assurance.

#### **2.2 Objectives**

The detailed objectives supporting the realization of these overarching goals are outlined as follows:

##### **2.2.1 Real-Time Log Analytics Dashboard**

- I. Creating a user-centric, interactive dashboard designed to provide intuitive visual representations of aggregated log data. .
- II. Implementing dynamic monitoring capabilities that facilitate the real-time tracking of software systems.



- III. Ensuring the log analytics dashboard is capable of processing large data volumes swiftly without any significant degradation in performance.

### **2.2.2 Test Case Generation Agent**

- I. Mapping SRDs and ATPs to create a dataset for fine-tuning Llama3.1-8B
- II. Developing a fully automated Test Case Generation Agent utilizing LangChain framework with the Llama3.1 8B advanced language model.
- III. Creating UI for the test case agent
- IV. Ensuring the generated test cases are comprehensive, covering all relevant functional requirements and edge cases effectively.

Through these clearly defined objectives, the project addresses significant challenges in modern software QA processes. Each objective is systematically aligned with the overall goal of enhancing software reliability, efficiency, and responsiveness, ultimately leading to improved productivity, reduced downtime, and superior end-user satisfaction.

## **Chapter 3**

### **Literature Review**

This chapter provides a comprehensive analysis of existing literature that informs the design and development of the two major components of this project: the Real-Time Log Analytics Dashboard and the AI-powered Test Case Generation Agent. These components, although distinct in function, are closely interconnected in the larger context of modernizing software quality assurance practices. The review explores four core themes: the growing role of real-time log analytics in systems monitoring, the critical role of visualization in interpreting complex data, the evolution of test case automation, and the emergence of advanced frameworks and models such as LangChain and LLaMA 3.1 8B in intelligent software engineering.

#### **3.1 Real-Time Log Analytics**

The increasing complexity of digital infrastructure has led to an explosion in the volume, velocity, and variety of system-generated logs. Logs capture vital system activities such as errors, transactions, requests, resource usage, and performance metrics, which are indispensable for operational insights. However, static log inspection is inefficient in fast-paced environments. Real-time log analytics addresses this challenge by enabling continuous monitoring and real-time anomaly detection.

Liu et al. (2019) report that the implementation of real-time log analysis significantly reduces downtime by enabling teams to detect and resolve issues before they cause critical failures. This proactive capability transforms reactive IT support into intelligent, preventive monitoring. For mission-critical industries such as finance, healthcare, and aviation, such real-time capabilities can mean the difference between service continuity and catastrophic disruption.

Moreover, Vu et al. (2018) emphasize the importance of aggregating logs from disparate sources into a centralized, normalized stream for better visibility. Their study demonstrates how visual dashboards allow IT administrators to track key performance indicators (KPIs), view real-time system health, and set alert thresholds for anomalies. These dashboards are foundational to modern observability platforms and DevOps pipelines.

In this project, real-time log analytics is realized through a custom-built dashboard hosted on Streamlit. This dashboard receives logs from automated testing activities, aggregates them, and presents visual summaries to support debugging, reporting, and decision-making during the test lifecycle.

### **3.2 Data Visualization and Dashboard Design**

Data visualization is the science and art of transforming abstract information into concrete visual representations that enhance understanding. In software quality assurance, where thousands of data points are generated per test cycle, visualization helps teams make quick and informed decisions. A well-designed dashboard not only presents data but also tells a story that is meaningful to its users.

Few (2013) posits that effective dashboard design requires clarity, focus, and accessibility. He notes that dashboards must reduce cognitive friction by organizing data according to priority and context. Data overload can paralyze decision-making; hence, good design practices such as grouping related metrics, using color judiciously, and eliminating unnecessary visual noise are essential.

Knaflitz (2015) builds on this by promoting visual storytelling as a method of enhancing engagement and comprehension. Her work emphasizes the role of layout, annotation, and interactivity in guiding users toward insights. These principles directly influenced the design of the project dashboard, which features time-series visualizations, event filters, and test case summaries.

Incorporating best practices from these studies, the dashboard developed in this project bridges the gap between raw log data and meaningful QA metrics. It empowers users to not only monitor but also interpret the outcomes of automated test runs in real time.

### **3.3 Automated Test Case Generation**

Manual test case development, while precise, is time-consuming and often limited in scope. As software evolves rapidly through agile development cycles, the need for automation in test generation has become critical. Automated test case generation reduces manual overhead, improves coverage, and allows for continuous testing.

Anand et al. (2013) argue that automation supports scalability and consistency in software testing, particularly in regression suites. Their work highlights that automated test generation from requirements improves traceability and reduces human error. However, conventional automation tools still rely heavily on structured inputs and predefined rules.

With the advent of AI and machine learning, especially natural language processing (NLP), it has become possible to derive test cases directly from unstructured documents such as SRDs (Software Requirement Documents). Sanguesa et al. (2021) explore how transformer-based models, trained on curated datasets, can understand requirements and produce semantically aligned test cases. These models offer dynamic adaptation, enabling updates to test cases as the underlying requirements evolve.

In this project, the Test Case Generation Agent was built using the LLaMA 3.1 8B model, which was fine-tuned on a custom dataset derived from actual SRDs and test case mappings. The system interprets input requirement segments and generates output in the form of structured test cases formatted for direct integration into QA workflows. This capability accelerates the testing process and enhances its alignment with business logic.

### **3.4 Frameworks: LangChain and LLaMA 3.1 8B**

Modern NLP applications often require more than just a single model prediction. They depend on structured reasoning, task decomposition, and contextual chaining—needs that are addressed by frameworks such as LangChain. LangChain is an open-source orchestration framework designed for large language models (LLMs). It enables chaining multiple steps of reasoning and integrates external data sources, APIs, and custom logic within a coherent task pipeline.

LangChain plays a vital role in this project by acting as the intermediary that connects user instructions, model prompts, and test case formatting. It allows for prompt engineering strategies that are contextually aware and dynamically adaptable.

On the modeling side, LLaMA 3.1 8B represents a leap forward in model size, architecture, and accuracy. Designed by Meta AI, LLaMA models are optimized for low-resource environments while maintaining strong performance in language understanding and

generation. The 3.1 8B variant, when fine-tuned, becomes capable of producing highly structured and relevant text outputs.

Together, these technologies enable a fluid, intelligent system for automated test generation. By leveraging LangChain's orchestration and LLaMA's language modeling, the project delivers outputs that are not only accurate but also production-ready.

### **3.5 Summary of Research Gaps and Project Alignment**

Although significant progress has been made in both system monitoring and AI-driven automation, integration challenges remain. Many existing dashboards are limited by rigid architectures that do not support real-time updates or automated feedback loops. Similarly, while large language models have been used in areas like summarization or chatbot development, their potential in QA automation is largely untapped.

This project contributes a novel integration of these domains. It combines the monitoring power of real-time analytics with the adaptability of AI-generated test scenarios, forming a unified QA solution. The literature reviewed supports the feasibility and importance of this direction, while also highlighting the innovation this project introduces.

By addressing gaps in real-time observability, test generation efficiency, and model-based contextual reasoning, the project provides a scalable blueprint for future QA systems driven by intelligent automation.

In essence, this chapter affirms that the proposed solution is not only technically viable but also timely and relevant to the current needs of the software industry. It sets the stage for a transformation in how quality assurance is approached in the age of AI and continuous delivery.

## Chapter 4

### Technical Challenges and Solutions

#### 4.1 Challenge: Long and Complex SRDs

##### Problem:

Software Requirement Documents (SRDs) often span dozens of pages and are authored by different stakeholders, leading to a wide variation in formatting, structure, and language. Some SRDs follow a strict template with well-defined sections and bullet points, while others are more freeform, mixing functional and non-functional requirements without clear boundaries. This lack of consistency created a major hurdle in preparing the documents as clean, structured input for training the LLaMA 3.1 model. Long paragraphs, nested sub-points, and inconsistent headings made it difficult to extract meaningful and contextually bounded requirement snippets.

##### Solution:

To address this, a custom text chunking mechanism was developed. Instead of relying on fixed-size token windows, the system uses **semantic chunking** — splitting content based on logical sections such as numbered headings, stepwise procedures, acceptance criteria markers, and domain-specific keywords like “The system shall...” or “Expected Result.” Regular expressions and rule-based parsing were used to break long documents into independently meaningful units while preserving contextual coherence. This preprocessing step significantly improved both the **fine-tuning phase** and **inference-time generation**, allowing the model to better understand the intent of each segment and generate aligned test cases.

#### 4.2 Challenge: Prompt Injection and Model Hallucination

##### Problem:

While using LLaMA 3.1 for generating test cases, an issue was observed where the model would sometimes output unrelated or overly generic test scenarios. These hallucinations stemmed from unclear prompts, overlapping instruction semantics, or irrelevant context carried forward from previous generations. In some instances, subtle prompt injection occurred when SRDs contained ambiguous or contradictory language that the model misinterpreted as meta-instructions rather than content.

**Solution:**

To mitigate this, the prompting pipeline was redesigned to include **structured templates** that explicitly define the expected output format. Each prompt follows a strict instruction-input-output format, using tags such as Requirement:, Generate:, and Note: to isolate and define each component. Additionally, a **few-shot prompting** approach was adopted — inserting 2–3 hand-crafted examples before the actual SRD snippet — which significantly anchored the model’s response behavior. To further reduce hallucinations, a post-processing validation script was implemented. This script checks for irrelevant keywords, excessive vagueness, and mismatch between test case conditions and the original requirement. These safeguards collectively improved the model’s reliability and alignment with intended outputs.

### 4.3 Challenge: Dashboard Performance Lag Under High-Frequency Input

**Problem:**

The real-time log analytics dashboard, built using Streamlit, initially exhibited sluggish performance when handling a continuous inflow of log data from parallel automated test runs. Users experienced latency in chart updates, delayed filter responses, and inconsistent rendering of visualizations. This issue became more noticeable when dealing with long-duration test sessions or high-velocity log streams generated from concurrent environments.

**Solution:**

Several optimization strategies were applied to enhance performance. First, **log input sampling** was introduced to process only a representative subset of logs during high-traffic intervals. Secondly, **caching mechanisms** using Streamlit’s native `@st.cache_data` functions were integrated to avoid redundant recomputation of visual summaries. Additionally, the UI update frequency was throttled to balance responsiveness with computational load — limiting full visual refreshes to once every 1–2 seconds during peak activity. These enhancements ensured that the dashboard retained its real-time capabilities without overwhelming the frontend rendering engine.

### 4.4 Challenge: Format Preservation in Exported Test Case Documents

**Problem:**

During initial deployment of the Test Case Generation Agent, exported .docx files often displayed formatting inconsistencies. Issues included improper indentation, broken

numbering of test steps, missing section headers, and inconsistent font sizes. This reduced the professional appearance of the test cases and made them harder for QA teams to interpret or import into TestRail or Jira.

**Solution:**

To ensure well-structured and visually consistent output, the system transitioned from basic string-to-text export to a more advanced templating method using the python-docx library. A predefined document template was created with styles for titles, bullet points, and enumerated test steps. The agent was configured to inject content into this template using programmatic sectioning — each test case was added as a new paragraph block with appropriate styling, indentation, and hierarchical numbering. The result was a polished and structured Word document that could be used directly in real-world QA workflows without further editing.



## **Chapter 5**

### **Methodology**

This chapter describes the methodologies adopted for the development of two primary components: the Real-Time Log Analytics Dashboard and the Test Case Generation Agent. Each methodology has been systematically designed to address specific challenges and optimize software QA processes.

#### **5.1. Methodology for Real-Time Log Analytics Dashboard**

##### **5.1.1 Data Collection and Aggregation**

Initially, the project focuses on collecting log data from diverse and distributed sources such as application servers, network devices, databases, and cloud infrastructures. Given the heterogeneous nature of logs (varying formats like text logs, JSON, CSV, XML, and structured logs), it is essential to standardize and aggregate them efficiently.

##### **5.1.2 Data Processing and Parsing**

Once the logs are aggregated, they undergo parsing and processing to extract useful fields. Parsing involves converting unstructured log data into structured and analyzable formats, enabling efficient indexing, searching, and analysis.

##### **5.1.3 Real-Time Analysis and Monitoring**

Real-time analysis involves setting up automated mechanisms to identify and classify anomalies, performance bottlenecks, and security threats as they occur. Automated alerts based on predefined thresholds and machine learning-based anomaly detection are established at this stage.

##### **5.1.4 Dashboard Design and Visualization**

The dashboard design prioritizes user-centric visualizations to enhance user understanding and response speed. It provides a clear, concise, and intuitive interface enabling stakeholders to easily interpret log data and swiftly react to issues.

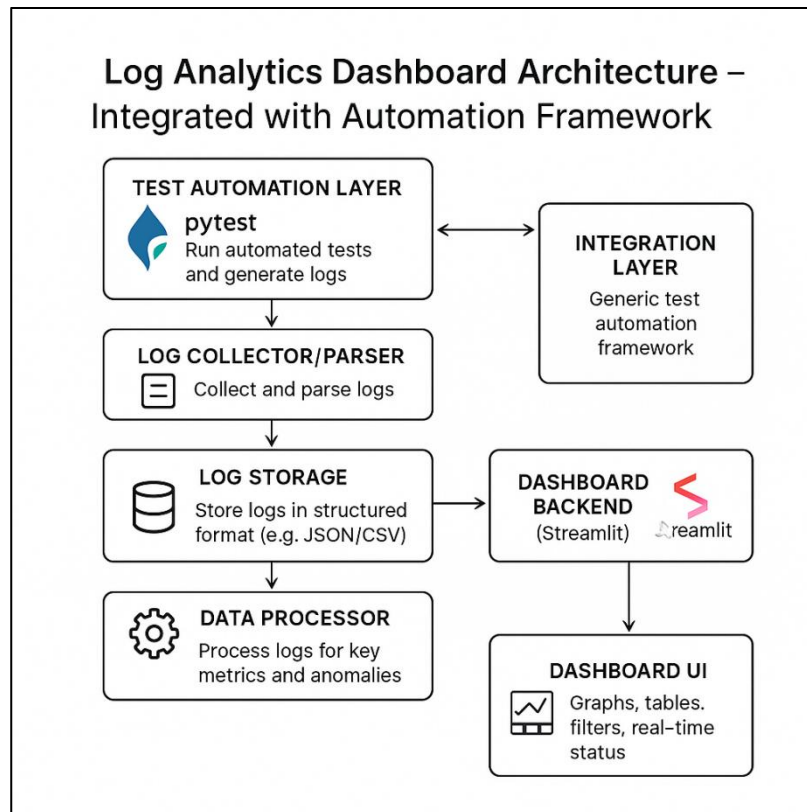


Figure 4.1 Flowchart: Log-Analytic Dashboard

## 5.2 Methodology for Development of Test Case Generation Agent

This section outlines the approach taken in the actual development and validation of the AI-powered Test Case Generation Agent. The system has been successfully built and is now undergoing rigorous testing across a range of Software Requirement Documents (SRDs). The methodology encompasses requirement analysis, model fine-tuning, scenario generation, and integration into existing QA workflows.

### 5.2.1 Requirement Analysis and Contextual Understanding

The initial phase focused on a deep contextual understanding of the software requirements. This involved parsing SRDs, identifying domain-specific vocabulary, and pinpointing critical and edge-case testing scenarios. Establishing a clear semantic foundation was essential for the language model to generate relevant and actionable test cases.

#### Tools and Techniques:

- **Requirement Documentation:** Analysis of actual SRDs and user stories used in real-world QA workflows.
- **Domain-Specific Knowledge Extraction:** Techniques and rule-based scripts were used to identify functional priorities and test boundaries across different domains.

## 5.2.2 Data Preparation and Model Fine-Tuning

A key aspect of this project was the creation of a high-quality, domain-specific dataset to fine-tune the LLaMA 3.1 8B language model. The dataset served as the foundation for training the Test Case Generation Agent to generate accurate, context-aware test scenarios based on varied software requirement documents (SRDs). Given the lack of publicly available datasets that map SRDs to structured test cases, a custom dataset was created in-house.

### 5.2.2.1 SRD to Test Case Mapping

The initial step involved manual and semi-automated mapping of SRD content to corresponding test cases. Multiple SRDs were collected from previous QA projects. Each document was carefully analyzed to identify individual functional requirements, edge conditions, and acceptance criteria. These were then mapped to real test cases either written manually by QA professionals or extracted from existing repositories.

Each test case was paired with its originating requirement to maintain semantic and contextual linkage. This ensured that the model would learn the logical relationship between requirements and the tests designed to validate them.

#### Tools and Techniques:

- Manual curation and expert QA input
- Regex-based parsing scripts for requirement segmentation

### 5.2.2.2 Conversion to Alpaca Format

Once the mappings were finalized, the next phase involved transforming the dataset into Alpaca format—a widely used format for fine-tuning instruction-following language models. The format typically includes fields such as:

***instruction:*** Describes the user request (e.g., “Generate test cases for the following SRD segment.”)

***input:*** The corresponding SRD snippet or requirement

***output:*** The expected test cases (structured in clear, enumerated format)

This structure allowed the LLaMA 3.1 8B model to understand the relationship between user instructions and expected responses, making the model more efficient at generalizing new SRDs and generating relevant test cases during inference.

### **Tools and Techniques:**

- Python scripts for JSON conversion
- Validation and format checking for Alpaca compatibility
- Use of templates to standardize formatting across all data points

#### **5.2.2.3 Dataset Evaluation and Refinement**

The dataset underwent multiple rounds of review to ensure quality and consistency. Incorrect mappings, ambiguous inputs, and unclear outputs were flagged and corrected. This iterative refinement process improved model training stability and output reliability.

As a result, the final dataset not only reflected the structure of real-world QA tasks but also optimized the LLaMA model’s ability to understand and respond to various requirement inputs with contextually appropriate test scenarios.

This custom dataset played a pivotal role in training an effective Test Case Generation Agent and serves as a valuable asset for future research and improvement of AI-assisted QA workflows.

#### **5.2.4 Integration into QA Workflow**

Following development, the agent was integrated with Logic Fruit Technologies' QA framework. The generated test cases are validated against different SRDs and then plugged into automated execution pipelines, ensuring seamless transition from generation to execution.

## Tools and Techniques:

- **Automation Framework Integration:** Test cases are aligned with PyTest and other in-house frameworks for automated execution.
- **CI/CD Pipelines:** Future integration with tools like Jenkins or GitLab CI is planned to enable fully automated QA workflows, including test generation, execution, and reporting.

This structured methodology ensures that the Test Case Generation Agent not only meets technical specifications but is also production-ready and adaptable to evolving QA needs.

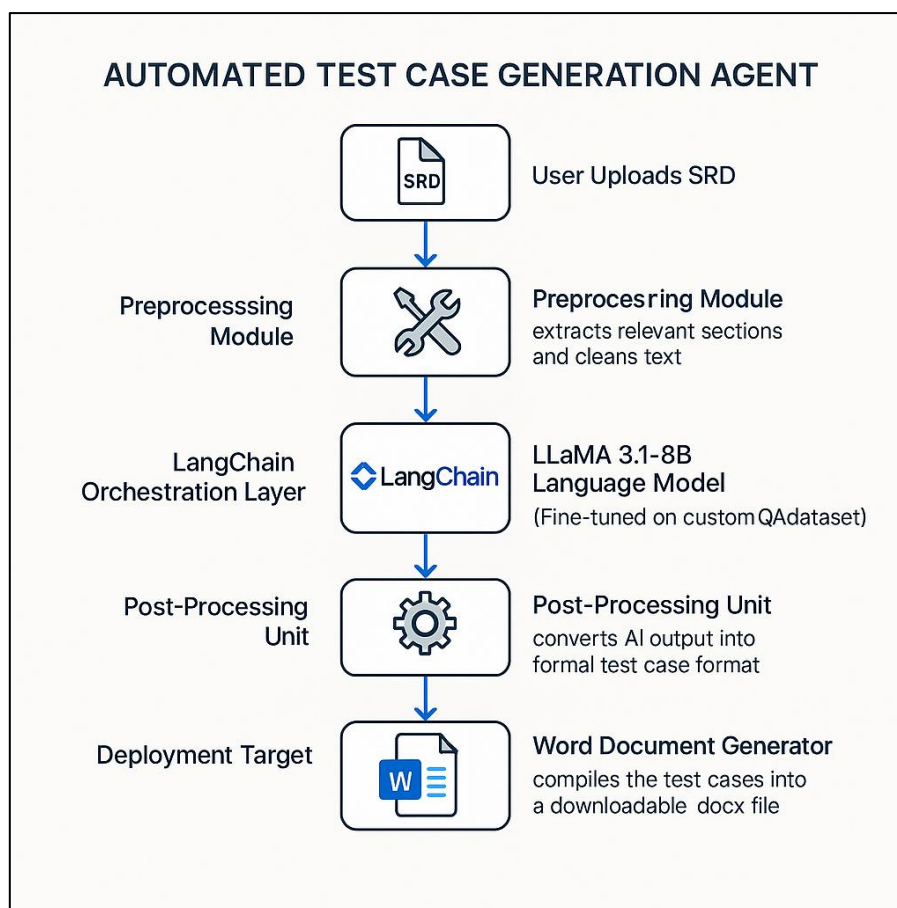


Figure 4.2 Flowchart: Test Case Generation Agent

## **Chapter 6**

### **Conclusion and Future Work**

As the Real-Time Log Analytics Dashboard and AI-powered Test Case Generation Agent near full implementation, this chapter revisits the overall objectives, assesses progress made so far, and outlines the roadmap for completing and evolving the project. Recent developments include the completion and successful integration of the dashboard with the company's generic test automation framework and the development of the test case generation agent, which is currently undergoing rigorous validation using multiple SRDs.

#### **6.1 Conclusions**

The project titled “Enhancing Quality Assurance Through Real-Time Log Analytics Dashboard & Test Case Generation Agent” marks a transformative effort in automating and enhancing software quality assurance workflows. As software ecosystems grow more dynamic and complex, traditional QA methods are increasingly strained. This project addresses those limitations through a forward-looking, intelligent automation strategy.

The Real-Time Log Analytics Dashboard has been fully developed, tested, and deployed. It is now integrated with the organization's generic test automation framework, enabling continuous and real-time monitoring of log data generated during testing processes. The dashboard empowers QA teams with faster diagnostics, better visibility into test execution, and collaborative insights across development and testing units.

Parallely, the Test Case Generation Agent has been successfully built using LangChain and the fine-tuned LLaMA 3.1 8B model. This agent automates the conversion of SRDs into structured test cases formatted as Word documents. The system is now undergoing validation through tests on diverse SRDs to ensure the accuracy, completeness, and contextual relevance of its output.

Together, these innovations bridge major gaps in current QA workflows, offering automation, intelligence, and real-time observability. They lay the groundwork for more advanced capabilities such as predictive failure analysis, automated bug classification, and autonomous test optimization. Beyond academic and organizational benchmarks, the project stands as a scalable foundation for next-generation QA practices.

## 6.2 Future Work

With both components nearing operational readiness, the remaining scope focuses on fine-tuning and full-scale deployment of the Test Case Generation Agent. Ongoing testing with various SRDs will continue to refine its contextual understanding and scenario generation logic.

Key areas of upcoming work include optimizing prompt engineering strategies for LangChain, enriching the training dataset with more domain-specific requirements, and improving output formatting and relevance. Feedback from QA professionals will be used to iteratively enhance the agent's reliability and usability.

Further integration with CI/CD pipelines is planned to support automated test generation and execution within continuous delivery environments. As part of long-term improvements, the project may explore adaptive learning techniques, self-healing test scripts, and expanded support for multilingual and cross-platform testing scenarios.

## References

- [1] Kulkarni, A. (2024). *Revolutionizing Quality Assurance with Automated Test Case Generation*. Retrieved from [LinkedIn](#)
- [2] HeadSpin. (2024). *How AI Automation is Revolutionizing QA Testing*. Retrieved from [HeadSpin Blog](#)
- [3] TechWell. (2024). *The AI Revolution in Software Quality Assurance: A New Era of Quality Engineering and Testing*. Retrieved from [TechWell Insights](#)
- [4] Radha. (2024). *AI is Transforming Software Testing: A New Era of Quality Assurance*. Retrieved from [Dev.to](#)
- [5] Devlane. (2024). *Test Automation Evolution: Trends and Innovations in QA Engineering*. Retrieved from [Devlane Blog](#)
- [6] DigitalOcean. (2024). *AI Testing Tools for Modern QA Teams*. Retrieved from [DigitalOcean Resources](#)
- [7] BrowserStack. (2024). *Quality Assurance Tools and Best Practices*. Retrieved from [BrowserStack Guide](#)
- [8] QATestLab. (2024). *Test Automation Insights and Trends*. Retrieved from [QATestLab Blog](#)
- [9] Liu, Y., Zhu, Y., & Cui, Y. (2019). *Challenges and opportunities towards fast-charging battery materials*. *Nature Energy*, 4, 540–550.
- [10] Vu, V.-B., Tran, D.-H., & Choi, W. (2018). *Implementation of constant current and constant voltage charge of inductive power transfer systems*. *IEEE Transactions on Power Electronics*, 33(9), 7398–7410.
- [11] Few, S. (2013). *Information Dashboard Design: Displaying Data for At-a-Glance Monitoring* (2nd ed.). Analytics Press.
- [12] Knaflic, C. N. (2015). *Storytelling with Data: A Data Visualization Guide for Business Professionals*. Wiley.
- [13] Anand, S., Burke, E. K., Chen, T. Y., Clark, J., Cohen, M. B., Grieskamp, W., ... & Zhu, H. (2013). *An orchestrated survey of methodologies for automated software test case generation*. *Journal of Systems and Software*, 86(8), 1978–2001.
- [14] Sanguesa, J. A., Torres-Sanz, V., Garrido, P., Martinez, F. J., & Marquez-Barja, J. M. (2021). *A Review on Electric Vehicles: Technologies and Challenges*. *Smart Cities*, 4(1), 372–404.



[15] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). *Language Models are Few-Shot Learners*. Advances in Neural Information Processing Systems, 33, 1877–1901.