**Threat Track AI**

|  |
| --- |
|  |

**By:**

**Danyal khan**

**40856**

**Muhammad Javaid**

**40855**

**Khawaja Bilal Ahmad**

**41276**

**Supervised by:**

**Dr Mansoor Alam**

**Faculty of Computing**

**Riphah International University, Islamabad**

**Spring 2025**

**A Dissertation Submitted To**

**Faculty of Computing,**

**Riphah International University, Islamabad**

**As a Partial Fulfillment of the Requirement for the Award of the Degree of**

**Bachelors of Science in Computer Science**

**Faculty of Computing**

**Riphah International University, Islamabad**

Date: [date of final presentation]

**Final Approval**

This is to certify that we have read the report submitted by **Danyal Khan *(40856)***, **Khawaja Bilal Ahmad (41276), Muhammad Javaid (40855)** for the partial fulfillment of the requirements for the degree of the Bachelors of Science in Computer Science (BSCS). It is our judgment that this report is of sufficient standard to warrant its acceptance by Riphah International University, Islamabad for the degree of Bachelors of Science in Computer Science (BSCS).

**Committee:**

|  |  |
| --- | --- |
| **1** | [Name Supervisor]  (Supervisor) |
|  |  |
| **2** | [Name of HOD/chairman]  (Head of Department/chairman) |

**Declaration**

We hereby declare that this document “**Threat Track AI**” neither as a whole nor as a part has been copied out from any source. It is further declared that we have done this project with the accompanied report entirely on the basis of our personal efforts, under the proficient guidance of our teachers, especially our supervisor **Dr Mansoor Alam**. If any part of the system is proved to be copied out from any source or found to be reproduction of any project from anywhere else, we shall stand by the consequences.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Danyal Khan**

**40856**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Khawaja Bilal Ahmad**

**41276**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Muhammad Javaid**

**40855**

**Dedication**

Our project is dedicated to our parents, seniors, friends, and our supervisor “Dr Mansoor Alam” who has been our continual source of inspiration and whose support has helped this project succeed. This project would not have been possible without their support.

This work is the reflection of your faith in me.

Thank you for believing.

**Acknowledgement**

First of all we are obliged to Allah Almighty the Merciful, the Beneficent and the source of all Knowledge, for granting us the courage and knowledge to complete this Project.

I am also thankful to all my teachers for their efforts during my degree, and to my friends and classmates for their help and cooperation.

Lastly, I am deeply grateful to my parents and family for their continuous support, prayers, and motivation.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Danyal Khan**

**40856**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Khawaja Bilal Ahmad**

**41276**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Muhammad Javaid**

**40855**

**Abstract**

Threat Track AI is a smart, locally deployable project built to make log analysis and threat detection faster and easier for IT teams. By using advanced AI models like LLMs and Agentic AI, it automatically scans and organizes system logs, spots important alerts, and suggests practical solutions in real time. Unlike traditional systems that often overwhelm with too many alerts and require manual responses, this solution streamlines the process. It brings together log collection, anomaly detection and then classification, also provide a solution, and a real-time dashboard in one flexible and scalable platform. The goal is to help organizations detect and respond to threats quickly, reduce operational stress, and keep their systems running securely and smoothly.

Table of Contents

[Table of Contents i](#_Toc113957278)

[List of Tables iii](#_Toc113957279)

[List of Figures iv](#_Toc113957280)

[Abstract 1](#_Toc113957281)

[Chapter 1: Introduction 2](#_Toc113957282)

[1.1 Goals and Objectives 2](#_Toc113957283)

[1.2 Scope of the Project 2](#_Toc113957284)

[Chapter 2: Literature Review 3](#_Toc113957285)

[2.1 Introduction 3](#_Toc113957286)

[2.2 Background and Problem Elaboration 3](#_Toc113957287)

[2.3 Detailed Literature Review 3](#_Toc113957288)

[2.3.1 Definitions 3](#_Toc113957289)

[2.3.2 Related Research Work 1 3](#_Toc113957290)

[2.3.3 Related Research Work 2 3](#_Toc113957291)

[2.4 Literature Review Summary Table 3](#_Toc113957292)

[2.5 Research Gap 3](#_Toc113957293)

[2.6 Problem Statement 3](#_Toc113957294)

[Chapter 3: Requirements and Design 4](#_Toc113957295)

[3.1 Requirements 4](#_Toc113957296)

[3.1.1 Functional Requirements 4](#_Toc113957297)

[3.1.2 Non-Functional Requirements 4](#_Toc113957298)

[3.1.3 Hardware and Software Requirements 4](#_Toc113957299)

[3.2 Proposed Methodology 4](#_Toc113957300)

[3.3 System Architecture 4](#_Toc113957301)

[3.4 Use Cases 4](#_Toc113957302)

[3.4.1 Sample Use Case Name Here 4](#_Toc113957303)

[3.5 Database Design *(Optional)* 6](#_Toc113957304)

[3.6 Class diagram *(Optional)* 6](#_Toc113957305)

[3.7 Sequence diagram *(Optional)* 6](#_Toc113957306)

[3.8 Any Other Artifact… 6](#_Toc113957307)

[3.9 Graphical User Interfaces(GUI) (*Optional)* 6](#_Toc113957308)

[Chapter 4: Implementation and Test Cases 7](#_Toc113957309)

[4.1 Implementation 7](#_Toc113957310)

[4.1.1 Implementation of First Component/Algorithm 7](#_Toc113957311)

[4.2 Test case Design and description 7](#_Toc113957312)

[4.2.1 Sample Test case No.1 7](#_Toc113957313)

[4.2.2 Sample Test case No.2 7](#_Toc113957314)

[4.3 Test Metrics 8](#_Toc113957315)

[4.3.1 Sample Test case Matric.No.1 8](#_Toc113957316)

[4.3.2 Sample Test case Metric.No.2 8](#_Toc113957317)

[4.3.3 Sample Test case Metric.No.3 8](#_Toc113957318)

[Chapter 5: Experimental Results and Analysis 9](#_Toc113957319)

[Chapter 6: Conclusion and Future Directions 10](#_Toc113957320)

[References 11](#_Toc113957321)

[Appendix 12](#_Toc113957322)

[Appendix A: Guidelines 12](#_Toc113957323)

[Appendix B: Heading of Sample Appendix B 12](#_Toc113957324)

[Formatting Guidelines 13](#_Toc113957325)

[Chapter 1: Heading 1 13](#_Toc113957326)

[1.1 Heading 2 13](#_Toc113957327)

[1.1.1 Heading 3 13](#_Toc113957328)

[Tables and Figures 14](#_Toc113957329)

[Equations 15](#_Toc113957330)

[Header/Footer 15](#_Toc113957331)

[Other Formatting Guidelines 15](#_Toc113957332)

[References 15](#_Toc113957333)

List of Tables

[Table 1: This is Sample table caption 1](#_Toc55559466)

[Table 2: This is Sample table caption 1](#_Toc55559467)

List of Figures

Figure 1: List of Styles 1

Figure 2: IEEE Reference style 1

# Abstract

Threat Track AI is a clever, locally deployable project designed to help IT teams identify threats and analyze logs more quickly. It automatically scans and arranges system logs, identifies critical alerts, and makes real-time recommendations for workable solutions using sophisticated AI models like LLMs and Agentic AI. This solution simplifies the process in contrast to traditional systems, which frequently overwhelm with too many alerts and call for manual responses. Log collection, anomaly detection and classification, a solution, and a real-time dashboard are all combined into a single, adaptable, and scalable platform. The objective is to lessen operational stress, assist organizations in promptly identifying and addressing threats, and maintain the security and functionality of their systems.

# Introduction

Threat Track AI is an AI-powered system for automated log analysis and real-time threat detection. Uses LLMs and Agentic AI to classify logs, filter critical alerts, and provide instant solutions. Locally deployable, ensuring security, faster processing, and operational efficiency.

**The last paragraph of introduction chapter should contain an outline of the entire report. Summarize each chapter in one line to make the last paragraph.**

## Goals and Objectives

The main and foremost objective of developing this system is to help IT teams manage system logs more effectively by automatically detecting threats and providing actionable insights saving time, reducing manual effort, and improving response to potential attacks.

### **Goals:**

* To build a locally deployable AI system that automates the classification and analysis of log data.
* To enable real-time detection of security threats using LLMs and Agentic AI technologies.
* To develop a customizable, scalable solution suitable for various IT environments.

### **Objectives:**

* To reduce alert fatigue by prioritizing and filtering only the most critical log alerts.
* To integrate intelligent agents that not only detect anomalies but also recommend suitable responses.
* To provide a user-friendly dashboard for visualizing threats and managing logs efficiently.
* To ensure the system functions entirely offline for better security and data privacy.
* To address the lack of integrated, AI-powered log monitoring systems available for local deployment.

## Scope of the Project

 Our system will be developed using FastAPI for the backend and React with Tailwind CSS for the frontend.

 We will develop an AI-powered platform that will allow IT teams to analyze log data, detect threats in real time, and get instant suggestions for resolution.

 Logs from multiple sources will be collected and analyzed using LLMs without the need for manual inspection.

 IT teams will receive filtered alerts based on severity, avoiding unnecessary alerts and helping them focus on what truly matters.

 The system will enhance the speed and accuracy of threat detection through continuous monitoring.

 Critical issues will be identified and responded to using Agentic AI, which will provide intelligent suggestions.

 The entire solution will be locally deployed to ensure data privacy and security.

# Literature Review

## Introduction

With the rising complexity of IT infrastructure and the increasing frequency of cyber threats, efficient and automated log analysis systems have become essential. Logs generated by servers, applications, and networks hold critical insights into the operational and security posture of a system. Manual monitoring of these logs is not only labor-intensive but also prone to human error, especially at scale. As a result, research and development have shifted towards AI-based systems capable of detecting anomalies and automating the response cycle.

## Background and Problem Elaboration

Traditional log monitoring systems rely heavily on predefined rules and signatures to detect unusual activity. These methods fail to adapt to new types of threats and generate a large number of false positives, overwhelming IT teams. As organizations adopt microservices and cloud-native technologies, the volume of generated logs has increased exponentially, making real-time monitoring even more challenging. There is a clear need for intelligent systems that can understand the context of log entries, detect anomalies, and suggest corrective actions automatically.

## Detailed Literature Review

### Definitions

 **Log Analysis**: The process of reviewing and interpreting log data generated by various systems to identify patterns, diagnose problems, and detect security breaches.

 **Anomaly Detection**: A technique in data analysis that identifies unusual patterns or events that do not conform to expected behavior.

 **LLM (Large Language Models)**: Advanced AI models trained on vast text corpora to understand, generate, and classify language, useful in interpreting complex logs.

 **Agentic AI**: AI systems capable of making decisions autonomously based on learned patterns or rules.

### Related Research Work 1

**Anomaly Detection in Web Logs**

Siwach and Mann (2022) discussed the increasing importance of anomaly detection in web log data as a key method to ensure the security of modern web applications. Their study explored statistical and early machine learning approaches that detect deviations from normal behavior in server logs. These systems, however, lacked the ability to understand context and often generated high false positive rates. This work was foundational in establishing the limitations of traditional log analysis and underscored the need for smarter tools.

### Related Research Work 2

**Automated Log Analysis with Machine Learning**

Shah et al. (2022) proposed a machine learning framework for automated log analysis, which improved accuracy by learning from historical log data. The system used classification models to detect known failure patterns. Although this approach marked progress over rule-based systems, it was limited in its ability to detect novel threats or offer remediation steps. The lack of natural language understanding also restricted the system’s interpretability and scalability across domains.

## Literature Review Summary Table

**Table 1: Summary of Related Work in Log Analysis and Anomaly Detection**

| **No.** | **Title & Reference** | **Authors** | **Year** | **Approach** | **Limitation** | **Relevance to This Project** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Anomaly Detection for Web Logs [1] | Siwach & Mann | 2022 | Statistical/Machine Learning | High false positives, lacks context | Highlights the need for contextual analysis |
| 2 | Automated Log Analysis Using ML [2] | Shah et al. | 2022 | ML-based classification | No handling of novel anomalies, no response suggestions | Shows potential for automation but lacks completeness |
| 3 | LLM-Based Log Parsing and Anomaly Detection [3] | Fariha et al. | 2024 | LLM with attention mechanism | No autonomous agents | Strong basis for using LLMs with improvement potential |

## Research Gap

Despite advancements in machine learning for log analysis, existing systems either focus solely on anomaly detection or lack real-time capabilities. Very few integrate LLMs for contextual understanding, and even fewer use autonomous agents to suggest real-time solutions. There is a significant gap in end-to-end solutions that combine intelligent log classification, critical alert filtering, and automated response mechanisms—particularly for localized deployments where privacy and latency are critical.

## Problem Statement

IT teams are often overwhelmed by a flood of alerts from traditional log monitoring systems, leading to alert fatigue and delayed responses. Current AI-based tools may detect anomalies but do not provide contextual solutions or integrate seamlessly with IT workflows. There is a clear need for a locally deployable, AI-powered system that can intelligently classify logs, detect threats, and suggest immediate, actionable steps to improve system security and operational efficiency.

# Requirements and Design

Describe all modules of requirements and design in clear English text along with the necessary diagram and figures. Anyone reading your report should be able to reproduce your system/results after reading it.

**For each chapter provide a paragraph of introduction and in the end a paragraph of conclusions.** Make sure no heading/subheading is blank. Write text to introduce each section as well.

Introduce sub-heading as:

## Requirements

### Functional Requirements

### **FR.1 – Log Ingestion System**

| **FR No.** | **Title** | **Description** |
| --- | --- | --- |
| FR-1.1 | Log Format Support | Accept logs in various formats (.txt, .log, .json) from multiple sources (servers, endpoints, apps). |
| FR-1.2 | Upload Modes | Provide real-time or scheduled log uploads. |

### **FR.2 – LLM-Based Threat Detection**

| **FR No.** | **Title** | **Description** |
| --- | --- | --- |
| FR-2.1 | Semantic Analysis | Use a Large Language Model (LLM) to analyze logs semantically. |
| FR-2.2 | Threat Detection | Identify malicious behavior, anomalies, and suspicious sequences based on learned patterns. |

### **FR.3 – Agentic AI Integration**

| **FR No.** | **Title** | **Description** |
| --- | --- | --- |
| FR-3.1 | Continuous Monitoring | Agentic AI will continuously monitor incoming logs. |
| FR-3.2 | Reactive Threat Handling | React to detected threats (e.g., generate alerts, suggest actions). |
| FR-3.3 | Adaptive Learning | Adaptively learn from feedback or historical threat data. |

### **FR.4 – Threat Classification**

| **FR No.** | **Title** | **Description** |
| --- | --- | --- |
| FR-4.1 | Severity Levels | Categorize threats based on severity: Low, Medium, High, and Critical. |
| FR-4.2 | Analyst Insights | Display summarized insights about threats for analysts. |

### **FR.5 – User Management**

| **FR No.** | **Title** | **Description** |
| --- | --- | --- |
| FR-5.1 | Role-Based Access | Role-based access: Admin, Analyst, and Viewer. |
| FR-5.2 | Secure Authentication | Secure login and session control for all users. |

### **FR.6 – Visualization Dashboard**

| **FR No.** | **Title** | **Description** |
| --- | --- | --- |
| FR-6.1 | Real-Time Monitoring | Real-time display of threat detection, flagged logs, and system metrics. |
| FR-6.2 | Filtering and Search | Filtering and search options based on threat type, time, or log source. |

### **FR.7 – Notification and Alert System**

| **FR No.** | **Title** | **Description** |
| --- | --- | --- |
| FR-7.1 | Alert Mechanisms | Alert users via in-app notifications, email, or external integrations. |
| FR-7.2 | Alert Logging | Maintain alert logs for auditing and traceability. |

### Non-Functional Requirements

Non-functional requirements define the system's operational characteristics and constraints, ensuring it performs reliably, securely, and efficiently under defined conditions. These requirements are essential to maintain the system's usability, maintainability, and performance.

| **NFR No.** | **Title** | **Description** |
| --- | --- | --- |
| NFR-1 | Performance | The system must process incoming logs and detect threats within 2 seconds. |
| NFR-2 | Scalability | Must support horizontal scaling to handle increased data from multiple sources. |
| NFR-3 | Security | All data must be encrypted at rest and in transit; secure authentication required. |
| NFR-4 | Availability | System should ensure 99.9% uptime in a production environment. |
| NFR-5 | Usability | User interface should be intuitive and accessible for non-technical users. |
| NFR-6 | Offline Operability | Full functionality must be available in a fully offline, local deployment. |
| NFR-7 | Maintainability | The codebase must be modular and well-documented to ease updates. |

### Hardware and Software Requirements

This section outlines the essential hardware and software needed to build, run, and maintain the system. It ensures all stakeholders are aware of what is needed for deployment.

**Hardware Requirements**

| **Component** | **Specification** |
| --- | --- |
| Processor | Quad-core (Intel i5/Ryzen 5 or better) |
| RAM | Minimum 8 GB |
| Storage | Minimum 100 GB SSD |
| GPU (optional) | NVIDIA GPU with 4 GB VRAM for LLM |

**Software Requirements**

| **Component** | **Version/Details** |
| --- | --- |
| Operating System | Ubuntu 20.04 LTS or Windows 10+ |
| Backend Framework | FastAPI |
| Frontend Framework | React with Tailwind CSS |
| Database | MongoDB or PostgreSQL |
| AI Models | TinyLLaMA, Agentic AI (custom agents) |
| Python Version | Python 3.9+ |
| Node.js Version | Node.js 16+ |

## Proposed Methodology

This section presents the methodology used to develop Threat Track AI. It outlines the data flow, model integration, and logic used in real-time threat detection and resolution.

We adopt a modular AI-driven methodology consisting of the following steps:

1. **Log Collection**: Collect logs from various systems in real-time or batch mode.
2. **Preprocessing**: Normalize and parse log formats.
3. **LLM-Based Classification:** Use LLM to interpret logs and detect anomalies.
4. **Agentic AI Response**: Trigger appropriate agents for alerting or suggesting solutions.
5. **Dashboard Visualization**: Display findings and insights on a secure UI.

This methodology ensures accuracy, scalability, and maintainability across various IT environments

## System Architecture

This section provides a high-level architecture of the system, showing the interaction among different modules.

**System Architecture Components:**

* **Log Sources** – Application, system, network, cloud logs.
* **Ingestion Layer** – Collects and formats logs.
* **LLM Engine** – Parses logs for anomalies and threats.
* **Agentic AI Engine** – Recommends or triggers actions.
* **Storage** – Database for logs, alerts, user roles.
* **Dashboard** – React-based UI for monitoring and control.

## Use Cases

### Real-Time Log Upload and Analysis

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Real-Time Log Upload and Analysis | | |
| Actors | | Admin, Analyst | | |
| Summary | | The user uploads logs (file or stream) for real-time parsing and analysis. Detected threats are displayed on the dashboard. | | |
| Pre-Conditions | | - User must be authenticated with the appropriate role (Admin or Analyst). - System must be online and ready to accept uploads. - Log format must be supported. | | |
| Post-Conditions | | - Logs are successfully ingested. - Anomalies are detected and results are shown. | | |
| Special Requirements | | - Secure upload channel (HTTPS or authenticated API). - Upload size limit: 100MB. - Supported formats: .log, .txt, .json, .csv | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | User opens the log upload interface. | | 2 | Displays upload form (file picker or stream input). |
| 3 | User clicks "Analyze Now" or enables auto-analysis. | | 4 | Parses logs, detects anomalies, and displays results on the dashboard. |
| **Alternative Flow** | | | | |
| 3 | Unsupported file format is uploaded. | | 4-A | Shows error message: “Unsupported file format.” |
| 2 | No anomalies are found during analysis. | | 2-A | Displays message: "No threats detected in this log." |

### Threat Classification and Notification

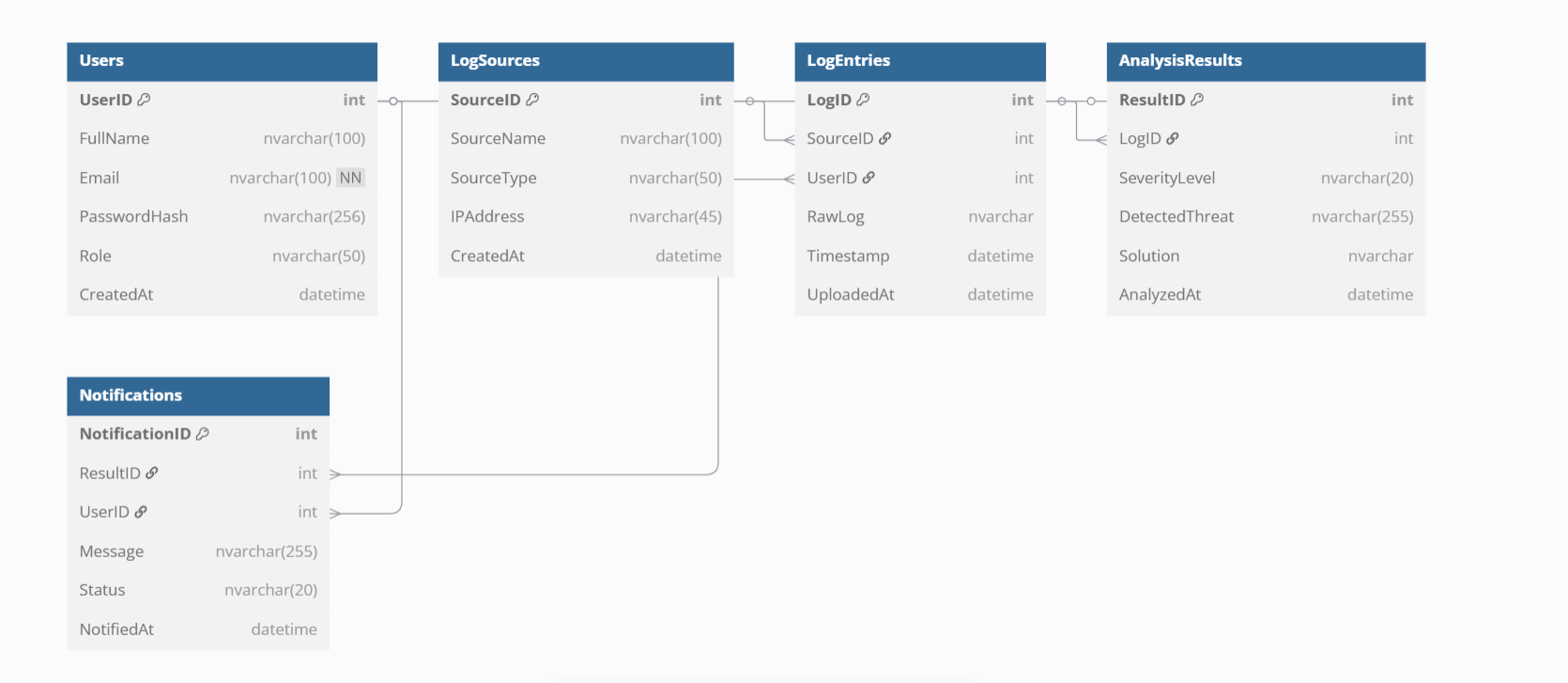
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Threat Classification and Notification | | |
| Actors | | Admin, Analyst | | |
| Summary | | The system classifies detected anomalies into severity levels (Normal, Anomalous, Critical) and notifies the appropriate user(s). | | |
| Pre-Conditions | | - Logs must be ingested and parsed. - Classification model must be loaded and ready. - Notification settings must be configured. | | |
| Post-Conditions | | - Threats are classified and stored. - Notifications are sent for Critical-level threats. | | |
| Special Requirements | | - Reliable classification model with high accuracy. - Notification system (email/SMS/alerts) must be active and integrated. | | |
| Basic Flow | | | | |
| Actor Action | | | **System Response** | |
| 1 | System completes anomaly detection. | | 2 | Proceeds to classify anomalies using the AI model. |
| **Alternative Flow** | | | | |
| 3 | Model fails to load or crashes. | | 4-A | Displays error: "Classification model unavailable. Please contact administrator." |
| 2 | Notification service is down. | | 2-A | Logs failure and queues the notification for retry. |
| 2 | No Critical threats found. | | 2 | No alert is sent; results are only stored in the database. |

## Database Design *(Optional)*

### **Entities and Relationships**

| **Table Name** | **Description** |
| --- | --- |
| Users | Stores registered users including Admins and Analysts. |
| LogEntries | Stores each uploaded or streamed log entry. |
| AnalysisResults | Contains classification and anomaly detection results. |
| Notifications | Stores notifications triggered by critical threats. |
| LogSources | Stores information about log origin (e.g., server name). |

### **ERD Diagram**



## Class Diagram (*Optional)*

The class diagram outlines the relationships between core system classes.

**Key Classes:**

* LogParser
* ThreatClassifier
* AgenticResponder
* UserManager
* DashboardController

## Sequence diagram *(Optional)*

The sequence diagram models the flow of operations during a real-time log analysis.

**Scenario: Real-Time Threat Detection**

1. User uploads a log.
2. System parses and classifies it.
3. Agentic AI recommends action.
4. Result shown on dashboard.

## Any Other Artifact…

Other potential artifacts include:

* **Alert Flow Chart**
* **Deployment Diagram**
* **Configuration Files** (e.g., config.yaml, .env)
* **Test Reports**
* **Log Format Mapping Guide**

## GUI Graphical User Interfaces (*Optional)*

This section includes GUI mockups/screens and explains their functionality and navigation flow for each user role.

**Login Page**

* **Functionality:** Secure login.
* **Navigation:** Redirects to dashboard based on role.

**Admin Dashboard**

* **Functionality:** View all logs, users, roles, and threat statistics.
* **Navigation:** Sidebar with "Users", "Logs", "Settings".

**Analyst Dashboard**

* **Functionality:** Focused on flagged logs and AI recommendations.
* **Navigation:** Filtering by severity, time, source.

**Viewer Page**

* **Functionality:** Read-only view of logs and system health.

**GUI Navigation Flow:**

1. Login → Dashboard → Logs List → Threat Detail → Recommendations

# Implementation and Test Cases

**For each chapter provide a paragraph of introduction and in the end a paragraph of conclusions.**

## Implementation

This chapter describes the implementation progress made in FYP-1 for the project "Threat Track AI", focusing on the development of a prototype capable of handling multiple types of logs, detecting anomalies, and fine-tuning language models for threat classification.

### Implementation of First Component/Algorithm

#### **Platforms and Tools Used**

| **Component** | **Technology / Platform Used** |
| --- | --- |
| Programming Language | Python |
| Data Handling | Pandas, NumPy |
| Anomaly Detection | Scikit-learn (Isolation Forest) |
| Text Vectorization | Scikit-learn (TF-IDF Vectorizer) |
| LLM Fine-Tuning | Hugging Face Transformers, TinyLLaMA, LLaMA2 |
| Log Format Unification | Custom Python Scripts |
| File Format Handling | CSV, JSON |
| Preprocessing | Regex, NLTK, Custom Parsers |

#### **Log Preprocessing and Normalization**

We collected logs from **8 different sources** including:

* Server Logs
* Web Logs
* Windows Event Logs
* Access Logs
* Hadoop Logs
* HDFS Logs
* Apache Logs
* OpenStack Logs

Each log format was unique in structure. To standardize them:

* We developed **custom parsers** for each log type.
* Each log entry was transformed into a **unified structured format** with fields like timestamp, source\_type, log\_level, message, and ip\_address.
* The output was saved as a **CSV file**, making it suitable for text processing and ML-based analysis.

#### **Feature Extraction Using TF-IDF**

To convert textual log data into numerical features suitable for machine learning:

* We used the **TF-IDF (Term Frequency-Inverse Document Frequency)** vectorizer from scikit-learn.
* This method helped highlight important terms by reducing the weight of common but less informative words.
* Output: A sparse matrix of term weights representing each log line.

#### **Anomaly Detection Using Isolation Forest**

To identify suspicious or unusual log entries:

* We implemented **Isolation Forest**, a tree-based anomaly detection algorithm.
* This algorithm works well for high-dimensional, sparse TF-IDF data.
* Each log entry was assigned an **anomaly score**.
* Based on thresholding, each entry was **labeled as "normal" or "anomalous"** (or critical, in some cases).

#### **Preparing Dataset for LLM Fine-Tuning**

To prepare data for fine-tuning an LLM to understand and classify threats:

* We transformed the labeled dataset into **instruction format** compatible with language models:
  + **Prompt:** The original unified log text.
  + **Response:** Whether it is normal, anomalous, or critical along with a suggested solution if applicable.

**Example JSON Format:**

json

{

"prompt": "timestamp=2024-04-12 10:02:33 source=web message='Unauthorized access attempt from IP 192.168.1.10'",

"Response": "Critical: Potential intrusion attempt detected. Block IP and review access logs."

}

#### **Tokenization and Fine-Tuning**

* We used **tokenizers from Hugging Face** (AutoTokenizer) to tokenize the prompt-response pairs.
* The model selected for fine-tuning is **TinyLLaMA and LLaMA2**, due to its small size and compatibility with low-resource environments (e.g., 4GB GPU).
* Fine-tuning was done using Trainer API from Hugging Face's transformers library.
* Goal: Teach the model to classify logs and generate actionable security suggestions.

#### **Output & Evaluation (Planned for FYP-2)**

Although full evaluation will be done in FYP-2, preliminary results show:

* The model begins to learn patterns of normal vs. anomalous logs.
* Isolation Forest achieves good separation of obvious anomalies in TF-IDF space.

#### **Conclusion**

The current prototype demonstrates a **functional log processing pipeline**:

1. Converts heterogeneous log formats into a unified schema.
2. Extracts important text features.
3. Applies anomaly detection via machine learning.
4. Structures data for LLM understanding.
5. Begins fine-tuning for log understanding and threat classification.

In FYP-2, this will be extended to **real-time processing**, **streaming log ingestion**, and **deployment of the fine-tuned LLM** with a user interface.