A PROJECT REPORT

on

**Project Quorum: AI-Powered Log Analysis for Secure Offline Environments**

*Submitted by*

**Mr. Ankit Vishwakarma**

*in partial fulfilment for the award of the degree*

*of*

**BACHELOR OF SCIENCE**

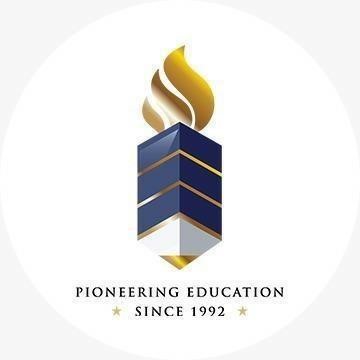
in

**COMPUTER SCIENCE**

*under the guidance of*

**Asst. Prof. Rashi Khare**

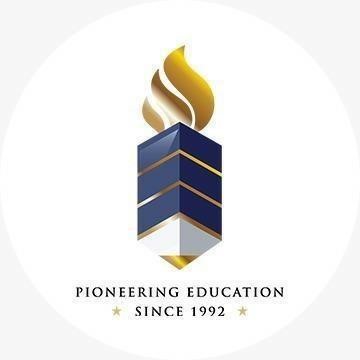
**Department of Computer Science**



**Shree LR Tiwari Degree College of Arts, Commerce & Science**

**(Sem VI)**

**(2025 – 2026)**



**Shree LR Tiwari Degree College of Arts, Commerce & Science**

Mira road (E), Thane - 401107

**CERTIFICATE**

This is to certify that Mr. **Ankit Vishwakarma** of **T.Y BSc CS (Sem VI)** class has satisfactorily completed the Project “**Project Quorum: AI-Powered Log Analysis for Secure Offline Environments**” to be submitted in the partial fulfilment for the award **of Bachelor of Science** in **Computer Science** during the academic year **2025 – 2026**.

**Date of Submission:**

**Project Guide**  **Head / In charge,**

**Department Computer Science**

**College Seal**  **Signature of Examiner**

**DECLARATION**

I, Mr. **Ankit Vishwakarma**, hereby declare that the project entitled “**Project Quorum: AI-Powered Log Analysis for Secure Offline Environments**” submitted in the partial fulfilment for the award of **Bachelor of Science** in **Computer Science** during the academic year **2025 – 2026** is my original work and the project has not formed the basis for the award of any degree, associateship, fellowship or any other similar titles.

**Signature of the Student:**

**Name of the Student: Ankit Vishwakarma**

**Place:**

**Date:**

**ACKNOWLEDGEMENT**

I wish to express my deepest gratitude to all those who have supported and guided me throughout the development of my project, **Project Quorum*.***

First and foremost, I extend my sincere thanks to **Asst. Prof. Rashi Khare** and **Asst. Prof. Lokesh Bahad** for their invaluable guidance, continuous encouragement, and insightful suggestions. Their mentorship played a pivotal role in shaping this project and was instrumental in its successful completion.

I am also sincerely thankful to all my teachers for the knowledge, motivation, and inspiration they have imparted throughout my academic journey. Their support provided me with the confidence and skills necessary to undertake and accomplish this project.

I would like to express my heartfelt gratitude to my family for their unwavering love and encouragement, which has always been my greatest strength. I am especially thankful to my brother for his constant support and for generously allowing me to use his laptop and resources. This project would not have been possible without his help.

Finally, I extend my appreciation to my friends for their motivation, constructive feedback, and collaborative spirit. Their presence made this journey not only educational but also truly enjoyable.

This project stands as a testament to the collective guidance, support, and encouragement I have been fortunate to receive from everyone around me.

**Ankit Vishwakarma**

**ABSTRACT**

Project Quorum is an AI-powered forensic analysis platform designed for air-gapped and offline environments where traditional cloud-based security tools cannot operate. Many high-security organizations such as government agencies, defence contractors, and critical infrastructure operators require complete network isolation, limiting their access to advanced threat detection capabilities.

This project delivers a self-contained, portable security operations centre as a single executable that runs entirely on an analyst's workstation without any network connectivity. By integrating DuckDB (a high-performance embedded database) with on-device machine learning models using TinyML and the PyOD library, Quorum enables sophisticated threat detection and forensic analysis in completely disconnected environments.

The system parses and ingests Windows EVTX and Linux Syslog files into a local database for high-speed SQL analysis. The embedded AI engine performs anomaly detection using algorithms like K-Nearest Neighbours and Isolation Forest to identify security threats. A React-based dashboard visualizes detected anomalies and maps them to the MITRE ATT&CK framework for rapid investigation.

A key innovation is the Secure Offline Update Protocol (SOUP), which uses cryptographic signatures to safely update AI models and detection rules in isolated environments. The cross-platform executable supports Windows, macOS, and Linux, and will be validated using benchmark datasets including LogPai, HDFS, and BGL.

Project Quorum bridges a critical gap by bringing enterprise-grade, AI-powered threat detection to security-sensitive environments without compromising their essential network isolation.

**INDEX**

**CHAPTER 1: INTRODUCTION**

1.1 Background ..................................................................................................................... 1

1.2 Objective ......................................................................................................................... 1

1.3 Purpose, Scope and Applicability ................................................................................... 2

**CHAPTER 2: SURVEY OF TECHNOLOGIES**

2.1 Embedded Analytical Database ….................................................................................. 5

2.2 Log Parsing Engine ………………................................................................................. 5

2.3 Machine Learning for Anomaly Detection ..................................................................... 6

2.4 On-Device AI Inference .................................................................................................. 6

2.5 Backend API Framework ..………………...................................................................... 6

2.6 Frontend User Interface .................................................................................................. 7

2.7 Desktop Application Framework .................................................................................... 7

2.8 Cryptographic Libraries for Secure Updates .................................................................. 8

**CHAPTER 3: REQUIREMENTS AND ANALYSIS**

3.1 Problem Definition ......................................................................................................... 9

3.2 Requirements Specification ............................................................................................ 9

3.3 Planning and Scheduling ................................................................................................ 12

3.4 Software and Hardware Requirements ........................................................................... 16

**CHAPTER 4: SYSTEM DESIGN**

4.1 Schema Design ............................................................................................................... 18

4.2 UML Diagram ................................................................................................................ 20

**CHAPTER 5: CODE AND IMPLEMENTATION**

5.4 Backend Implementation ............................................................................................... 26

5.5 Frontend Implementation ............................................................................................... 33

**CHAPTER 6: RESULTS AND DISCUSSION**

6.1 User Documentation ....................................................................................................... 35

**CHAPTER 7: CONCLUSIONS**

7.1 Conclusion ....................................................................................................................... 41

7.2 Limitations of the System ................................................................................................ 41

7.3 Future Scope of the Project .............................................................................................. 43

**REFERENCES** ....................................................................................................... 44

**TABLE OF FIGURES**

Figure 3.1 Gantt Chart ................................................................................................... 15

Figure 4.1 ER Diagram .................................................................................................. 18

Figure 4.2.1 DFD Level 0 .............................................................................................. 20

Figure 4.2.2 DFD Level 1 .............................................................................................. 21

Figure 4.3 Class Diagram .............................................................................................. 23

Figure 4.4 USE CASE DAIGRAM ............................................................................... 24

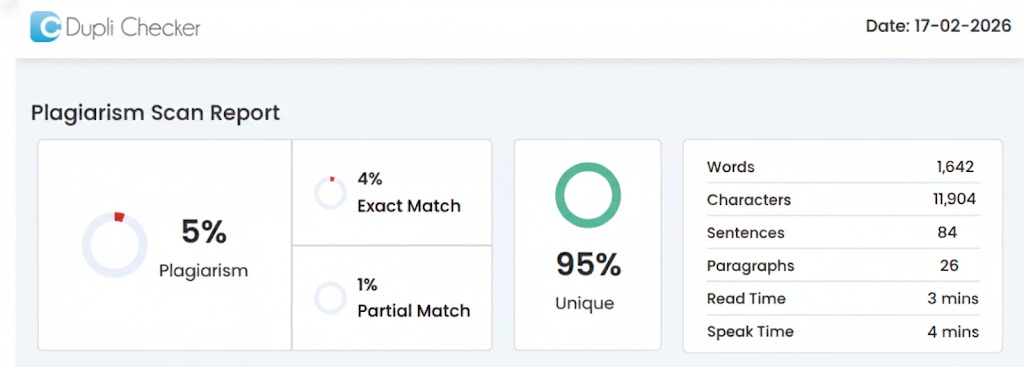
Figure 4.5 ACTIVITY DIAGRAM ............................................................................... 25

**PLAGIARISM REPORT**

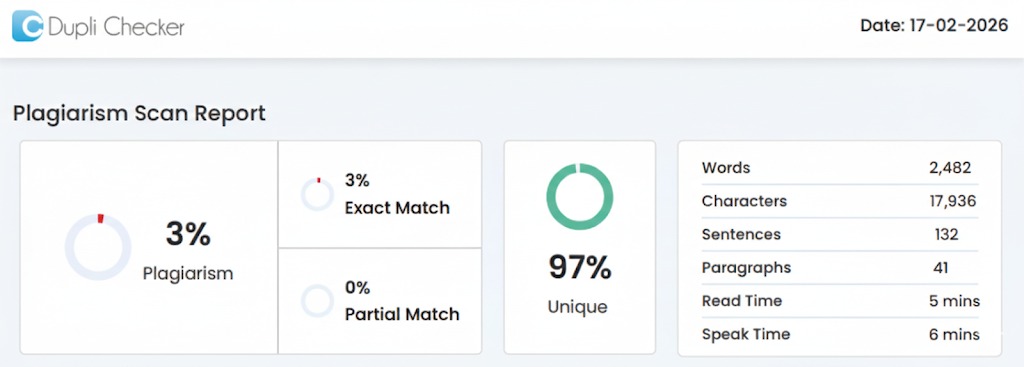
**1.Introduction**

****

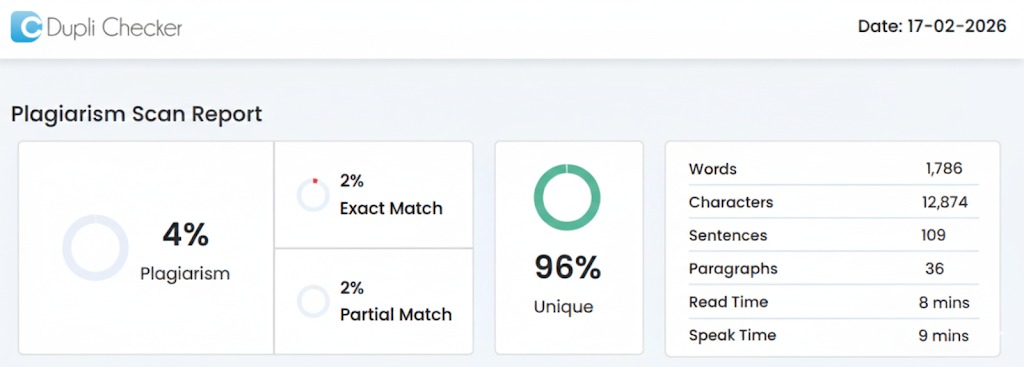
**2.Survey of Technologies**



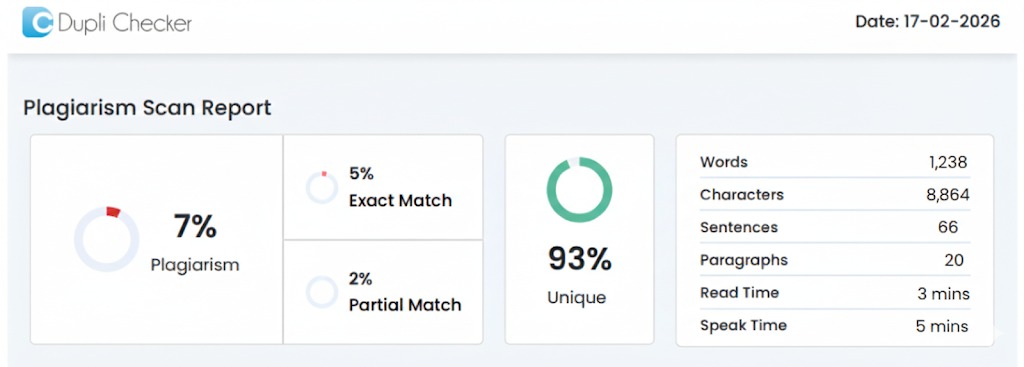
**3.Requirements and Analysis**



**4.System Design**



**7.Conclusions**

****

**1. INTRODUCTION**

**1.1 Background**

In today's digital world, cybersecurity has become a critical concern for organizations handling sensitive data. While most security tools rely heavily on internet connectivity and cloud services for threat detection and log analysis, there are many environments where such connectivity is simply not possible or permitted. These are called air-gapped or offline environments, commonly found in government organizations, defence systems, banking sectors, and critical infrastructure facilities where security protocols mandate complete isolation from external networks.

The challenge arises when security analysts in these offline environments need to analyse large volumes of system logs to detect potential threats, anomalies, or security breaches. Traditional Security Information and Event Management (SIEM) tools and AI-powered threat detection systems require constant internet access to function effectively, making them unsuitable for air-gapped networks.

This is where Project Quorum comes into the picture. The idea behind this project is to develop a completely self-contained, portable forensic analysis tool that can perform advanced AI-driven log analysis without requiring any network connectivity. By bringing together embedded database technology and machine learning models that run locally on the analyst's computer, we can provide powerful security analysis capabilities even in the most restricted environments.

The growing sophistication of cyber-attacks and the increasing volume of log data generated by modern systems make it essential to have intelligent, automated tools for log analysis. However, the security requirements of certain organizations prevent them from using cloud-based solutions. Project Quorum aims to solve this problem by creating an offline alternative that doesn't compromise on analytical power or ease of use.

**1.2 Objective**

* To develop a fully self-contained, portable log analysis system that operates entirely offline without any internet or network connectivity requirements.
* To implement an embedded analytical database (DuckDB) that can efficiently handle large volumes of log data and enable fast SQL-based querying on a local workstation.
* To create a cross-platform log parsing engine capable of processing common log formats including Windows Event Logs (EVTX) and Linux Syslog files.
* To integrate on-device machine learning capabilities using TinyML (TensorFlow Lite) and the PyOD library for performing anomaly detection directly on the analyst's computer.
* To develop a Secure Offline Update Protocol (SOUP) that allows safe and verified updates of AI models and detection rules in disconnected environments using cryptographic signatures.
* To build an intuitive, React-based web interface that visualizes detected threats and maps them to the MITRE ATT&CK framework for easier investigation and understanding.
* To package the entire system as a single, easy-to-deploy executable that works across Windows, macOS, and Linux operating systems.

**1.3 Purpose, Scope, and Applicability**

**1.3.1 Purpose**

The purpose of Project Quorum includes the following key aspects:

* To provide security analysts in air-gapped environments with a comprehensive, AI-powered tool for performing advanced log analysis and threat detection without requiring any internet connectivity.
* To enable organizations operating isolated networks to leverage modern machine learning techniques for identifying security anomalies and potential threats in their system logs.
* To bridge the technological gap between cloud-based security solutions and the operational requirements of high-security environments that mandate complete network isolation.
* To reduce the manual effort and time required for log analysis by automating anomaly detection through embedded AI models that run locally on the analyst's workstation.
* To enhance the overall security posture of organizations handling sensitive or classified information by providing them with enterprise-grade forensic analysis capabilities that respect their air-gap constraints.
* To demonstrate the feasibility of building sophisticated security tools that can function effectively without depending on external services, cloud platforms, or continuous internet access.

**1.3.2 Scope**

The scope of this project encompasses the following deliverables and functionalities:

* Development of a robust log parsing engine capable of reading and structuring common log formats, specifically Windows Event Logs (EVTX) and Linux Syslog files, converting them into a query able format.
* Implementation of DuckDB, a high-performance embedded analytical database, to store processed log data locally and enable fast, complex SQL queries on multi-gigabyte datasets without external database servers.
* Integration of the PyOD (Python Outlier Detection) library to implement multiple anomaly detection algorithms including K-Nearest Neighbours (KNN), Isolation Forest, Local Outlier Factor (LOF), and other statistical methods for identifying suspicious patterns in log data.
* Utilization of TensorFlow Lite for on-device machine learning inference, ensuring that AI models can run efficiently on standard workstation hardware without GPU requirements or cloud connectivity.
* Design and implementation of a React-based user interface that provides an intuitive dashboard for data visualization, allowing analysts to explore logs, execute custom SQL queries, and review detected anomalies with clear visual representations.
* Integration with the MITRE ATT&CK framework to automatically map detected anomalies to known adversary tactics, techniques, and procedures (TTPs), helping analysts quickly understand the potential nature and severity of threats.
* Development of a proof-of-concept Secure Offline Update Protocol (SOUP) that demonstrates how AI models and detection rules can be safely updated in air-gapped environments using cryptographic signatures and hash verification to ensure integrity.
* Packaging the entire application stack-including the backend API server, embedded database, machine learning models, and frontend interface-into a single, portable executable using Tauri framework for cross-platform compatibility.
* Testing and validation of the system using publicly available benchmark datasets such as LogPai, HDFS (Hadoop Distributed File System), and BGL (Blue Gene/L) to demonstrate the effectiveness of anomaly detection capabilities.

**1.3.3 Applicability**

Project Quorum is applicable to a wide range of organizations and use cases, including:

* Government agencies and defence organizations that operate classified networks with strict air-gap requirements and need to perform security audits and threat hunting on isolated systems.
* Critical infrastructure operators including power generation facilities, water treatment plants, oil and gas operations, and transportation control systems where network isolation is mandated for safety and security reasons.
* Financial institutions and banking organizations with high-security data centres and trading floors that maintain air-gapped networks to protect against external cyber threats and comply with regulatory requirements.
* Research laboratories and academic institutions handling sensitive research data, classified projects, or proprietary information that cannot be exposed to external networks.
* Healthcare organizations managing electronic health records and medical devices in isolated networks to comply with HIPAA regulations and protect patient privacy.
* Manufacturing facilities and industrial control systems (ICS/SCADA environments) that operate production networks separately from corporate networks for operational security.
* Digital forensics teams and incident response professionals who need portable tools for analysing log data collected from compromised or suspicious systems during investigations.
* Security operations canters (SOCs) in organizations that prefer to keep certain security analysis tools and sensitive log data completely offline for compliance or operational security reasons.
* Penetration testing and red team operations where analysts need to examine logs from simulated attacks in controlled, isolated lab environments.
* Any organization that values data sovereignty and wishes to perform security analysis without sending their logs to third-party cloud services or external platforms.

**2. SURVEY OF TECHNOLOGIES**

**2.1 Embedded Analytical Database**

The database component forms the foundation of Project Quorum, responsible for storing, indexing, and querying large volumes of log data efficiently on a local workstation without any external database server.

**Available Technologies:**

* **SQLite:** A lightweight, file-based database that is widely used for embedded applications. However, it struggles with analytical queries on large datasets and lacks support for parallel query execution.
* **PostgreSQL / MySQL:** Full-featured relational databases with excellent performance, but they require separate server processes, complex installation, and are not designed to be embedded within a single executable application.
* **Apache Parquet with Pandas:** A columnar storage format that works well with Python's Pandas library for data analysis, but it lacks built-in query optimization and requires loading entire datasets into memory.

**Chosen Technology: DuckDB**

* DuckDB is the ideal choice for Project Quorum's database. This embedded, in-process analytical database is designed for OLAP workloads. It runs within the application, eliminating the need for a separate server, making it perfect for a portable tool. DuckDB uses columnar storage and vectorized query execution for extremely fast SQL on multi-gigabyte log datasets. It directly queries various file formats and supports parallel processing, making it faster than SQLite for analytical tasks. Stored as a single file or running in memory, it perfectly supports a portable executable.

**2.2 Log Parsing Engine**

The log parsing component is responsible for reading raw log files in various formats and converting them into structured data that can be stored in the database for analysis.

**Available Technologies:**

* **Regular Expressions (Regex):** Manual parsing using regex patterns is flexible but becomes complex and error-prone for different log formats, requiring extensive maintenance.
* **Logstash:** Part of the ELK (Elasticsearch, Logstash, Kibana) stack, powerful for log processing but requires Java runtime, separate installation, and is designed for centralized, online environments.

**Chosen Technology: python-evtx and Custom Syslog Parser**

* For Windows Event Logs, the cross-platform python-evtx library is used to parse binary EVTX files, extracting event data, timestamps, IDs, and XML details without needing Windows APIs. For Linux Syslog, a custom Python parser handles multiple formats (RFC 3164/5424) without external dependencies. Both lightweight parsers integrate seamlessly with the Python backend.

**2.3 Machine Learning for Anomaly Detection**

The AI component is the intelligence layer of Project Quorum, responsible for analysing log patterns and identifying anomalies that may indicate security threats or unusual system behaviour.

**Available Technologies:**

* **PyTorch / TensorFlow:** Deep learning frameworks that offer powerful capabilities for complex neural networks but introduce significant computational overhead, require large training datasets, and are overkill for the relatively structured nature of log anomaly detection.
* **Cloud ML Services:** Platforms like Azure ML or AWS SageMaker provide managed machine learning services but create external dependencies, require internet connectivity, and violate the core requirement of offline operation.
* **Scikit-learn:** A comprehensive machine learning library with many algorithms, but it lacks specialized outlier detection methods and doesn't focus specifically on anomaly detection use cases.

**Chosen Technology: PyOD (Python Outlier Detection)**

* PyOD is the ideal Python library for Project Quorum's anomaly detection. It offers over 40 optimized outlier algorithms.For log analysis, we use Isolation Forest, KNN, and LOF. PyOD is lightweight, well-maintained, and integrates well with NumPy/Pandas for processing log features. It uses traditional ML, suited for structured log data, and is computationally efficient compared to deep learning methods

**2.4 On-Device AI Inference**

For deploying trained models in a resource-efficient manner, an inference engine is needed that can run models locally without heavy dependencies.

**Available Technologies:**

* **Full TensorFlow:** The complete TensorFlow framework is powerful but extremely large (hundreds of megabytes), making the final executable bloated and slow to start.
* **ONNX Runtime:** A cross-platform inference engine that supports models from various frameworks, but it adds complexity in model conversion and deployment.
* **Native Python Models:** Running models directly with their training libraries works but isn't optimized for inference and consumes more memory.

**Chosen Technology: TensorFlow Lite**

* TFLite is a lightweight version of TensorFlow optimized for edge devices, offering a smaller footprint, support for quantization, and efficient CPU-only operation. For Project Quorum, TFLite allows us to deploy fast, on-device AI capabilities via pre-trained models on analyst workstations with minimal overhead, keeping our portable executable small.

**2.5 Backend API Framework**

The backend component serves as the core application logic, handling requests from the frontend, orchestrating database operations, and coordinating ML inference.

**Available Technologies:**

* **Flask:** A simple, lightweight web framework that is easy to learn but lacks built-in support for asynchronous operations and modern API features like automatic validation and documentation.
* **Django:** A full-featured web framework with an ORM, admin interface, and many built-in features, but it's too heavyweight for an embedded application.
* **Express.js (Node.js):** A popular JavaScript framework that performs well but would require mixing programming languages in the project, complicating development and deployment.

**Chosen Technology: FastAPI**

* FastAPI is the optimal, high-performance Python framework for Project Quorum's API backend. Built on ASGI, Starlette, and Pydantic, it is significantly faster than Flask, offering automatic request validation via type hints and generating interactive Swagger UI documentation. Its support for asynchronous operations, lightweight nature, and seamless integration with our Python-based database and ML components create a cohesive stack perfect for our embedded API server.

**2.6 Frontend User Interface**

The frontend component provides the visual interface through which analysts interact with the system, view logs, and explore detected anomalies.

**Available Technologies:**

* **Plain HTML/CSS/JavaScript:** Simple and lightweight but requires extensive manual coding for complex UI interactions, data visualization, and state management.
* **Vue.js:** A progressive JavaScript framework that is easy to learn and lightweight, but has a smaller ecosystem compared to React for security-focused UI components.
* **Angular:** A comprehensive framework backed by Google with strong Typing support, but it's heavyweight.

**Chosen Technology: React**

* React is selected as the frontend framework for Project Quorum. Developed and maintained by Meta (Facebook), React is a component-based JavaScript library that excels at building interactive user interfaces.

**2.7 Desktop Application Framework**

To package the entire application-backend, database, ML models, and frontend into a single executable, a desktop application framework is required.

**Available Technologies:**

* **Electron:** A popular framework for building desktop apps using web technologies, but it bundles an entire Chromium browser, resulting in executables that are 100+ MB even for simple applications.
* **PyInstaller / cx\_Freeze:** Python packaging tools that create executables from Python scripts, but they struggle with complex web applications, don't provide a native UI wrapper, and produce large, slow-starting executables.
* **PyQt (PyQt/PySide):** A native GUI framework that creates truly native applications, but it requires learning a completely different UI paradigm and doesn't leverage our existing React frontend.

**Chosen Technology: Tauri**

Tauri is the ideal framework for packaging Project Quorum. It is a modern, security-first toolkit for building small, cross-platform desktop applications using web technologies. Unlike Electron, Tauri uses the OS's native webview (WebView2/WebKit) instead of bundling a browser, resulting in 10-20x smaller executables. Written in Rust for core performance, Tauri seamlessly integrates the FastAPI backend (as a local server) and the React frontend via its webview. It handles packaging, code signing, and native installers for Windows, macOS, and Linux from a single codebase, making it perfect for a portable forensic tool.

**2.8 Cryptographic Libraries for Secure Updates**

The Secure Offline Update Protocol (SOUP) requires cryptographic functions for signing and verifying update packages.

**Available Technologies:**

* **OpenSSL (via Python bindings):** A comprehensive cryptographic library but complex to use, with a large API surface and potential security pitfalls if not used correctly.
* **PyCrypto / PyCryptodome:** Older Python cryptographic libraries that provide low-level primitives but lack modern best practices and have been superseded by newer alternatives.
* **hashlib (Python standard library):** Built-in hashing functions that work well for integrity checking but don't provide digital signature capabilities.

**Chosen Technology: cryptography (Python library)**

* SOUP's security relies on the cryptography Python library, the most recommended choice for cryptographic operations. It offers high-level, best-practice APIs for tasks like RSA signatures, SHA-256 hashing, and key management. For Project Quorum, it's used to generate key pairs, sign model updates with private keys, and verify signatures with public keys, ensuring the authenticity and integrity of updates against tampering in offline environments. The library is actively maintained, well-documented, and integrates cleanly with our Python backend.

**3. REQUIREMENT ANALYSIS**

**3.1 Problem Statement**

In today's cybersecurity landscape, organizations handling sensitive information often operate in air-gapped or completely offline environments to protect against external threats. These environments include government networks, defence systems, critical infrastructure, and financial institutions where security policies mandate complete isolation from the internet. While this isolation provides strong protection against remote attacks, it creates a significant challenge: security analysts in these environments cannot access modern cloud-based threat detection tools, AI-powered log analysis platforms, or real-time security intelligence feeds.

The primary problems faced by security teams in offline environments include:

* **Lack of AI-Powered Analysis:** Most modern Security Information and Event Management (SIEM) systems and threat detection platforms require cloud connectivity to leverage AI and machine learning capabilities. Offline environments are forced to rely on manual log analysis or basic rule-based systems.
* **Overwhelming Log Volume:** Modern IT systems generate massive amounts of log data daily. Manual analysis of these logs is time-consuming, error-prone, and often results in missed threats or delayed incident detection.
* **Limited Forensic Tools:** Existing offline forensic tools are often complex, require extensive setup, depend on multiple separate applications, and lack integration between log parsing, database querying, and threat detection.
* **Update Challenges:** Even when offline tools exist, updating detection rules, threat signatures, or AI models in air-gapped environments is difficult and often involves insecure manual processes without proper verification mechanisms.
* **Knowledge Gap:** Mapping detected anomalies to known attack patterns requires deep expertise in threat intelligence frameworks like MITRE ATT&CK, which many analysts may not have readily available in offline settings.
* **Tool Portability:** Forensic analysts often need to move between different isolated systems or work on portable media. Existing solutions require complex installations, database servers, and multiple dependencies that make deployment difficult.

Project Quorum addresses these problems by creating a completely self-contained, portable forensic analysis platform that brings enterprise-grade AI capabilities to offline environments without compromising security or requiring internet connectivity.

**3.2 Requirements Specification**

The requirements for Project Quorum are categorized into functional and non-functional requirements to ensure the system meets both operational needs and quality standards.

**3.2.1 Functional Requirements**

**FR1: Log File Ingestion**

* The system must support parsing and ingestion of Windows Event Log files in EVTX format.
* The system must support parsing and ingestion of Linux Syslog files in both RFC 3164 and RFC 5424 formats.
* The system must extract key fields including timestamp, event ID, source, severity level, and message content.
* The system must handle corrupted or malformed log entries gracefully without crashing.

**FR2: Database Storage and Management**

* The system must store all ingested log data in an embedded DuckDB database.
* The system must create appropriate indexes on timestamp and event type fields for fast querying.
* The system must support databases containing millions of log entries without significant performance degradation.
* The system must provide the ability to export query results in CSV or JSON format.

**FR3: SQL Query Interface**

* The system must provide a SQL query interface for ad-hoc analysis of log data.
* The system must support complex SQL queries including joins, aggregations, and filtering.
* The system must provide query result pagination for large result sets.
* The system must display query execution time and number of rows returned.
* The system must save query history for easy re-execution of common queries.

**FR4: Anomaly Detection**

* The system must implement multiple anomaly detection algorithms from the PyOD library including Isolation Forest, K-Nearest Neighbours (KNN), and Local Outlier Factor (LOF).
* The system must allow users to select which algorithm to apply for analysis.
* The system must extract relevant features from log data automatically for ML input.

**FR5: MITRE ATT&CK Mapping**

* The system must maintain a local database of MITRE ATT&CK techniques and tactics.
* The system must automatically map detected anomalies to relevant ATT&CK techniques based on log characteristics.
* The system must display the ATT&CK matrix with highlighted techniques relevant to detected threats.
* The system must provide detailed descriptions of each technique for analyst reference.
* The system must allow filtering of anomalies by specific tactics or techniques.

**FR6: User Interface Dashboard**

* The system must provide a web-based dashboard accessible through the application.
* The system must display summary statistics including total logs ingested, anomalies detected, and time range covered.
* The system must provide interactive charts showing log volume over time, event type distribution, and anomaly trends.
* The system must display a table of detected anomalies with sorting and filtering capabilities.
* The system must provide detailed views of individual log entries and anomalies.

**FR7: Secure Offline Updates (SOUP)**

* The system must support loading signed update packages containing new ML models or detection rules.
* The system must verify cryptographic signatures on update packages before applying them.
* The system must reject tampered or invalidly signed updates with clear error messages.

**FR8: File Export and Reporting**

* The system must allow exporting detected anomalies as PDF reports.
* The system must allow exporting raw data and query results in CSV format.
* The system must include charts and visualizations in exported reports.
* The system must generate reports with timestamps and metadata about the analysis session.

**3.2.2 Non-Functional Requirements**

**NFR1: Performance**

* The system must parse and ingest at least 10,000 log entries per minute on standard hardware.
* The system must execute SQL queries on databases containing 1 million+ entries in under 1 minutes for typical queries.
* The system must complete anomaly detection analysis on 100,000 log entries within 2 minutes.

**NFR2: Portability**

* The system must be packaged as a single executable file requiring no installation.
* The system must run on Windows 10/11, macOS 10.15+, and major Linux distributions.
* The executable size must not exceed 200 MB.
* The system must not require administrator/root privileges for normal operation.
* The system must be executable from USB drives or other portable media.

**NFR3: Usability**

* The user interface must be intuitive and require minimal training for analysts familiar with basic log analysis.
* Error messages must be clear and provide actionable guidance.
* The system must provide tooltips and help text for complex features.
* The system must support keyboard shortcuts for common operations.
* The dashboard must be responsive and work on different screen sizes.

**NFR4: Reliability**

* The system must handle unexpected shutdowns gracefully without corrupting databases.
* The system must validate all user inputs to prevent crashes from malformed data.
* The system must log errors to a file for troubleshooting purposes.
* The system must continue operating even if individual log entries cannot be parsed.
* The system must provide progress indicators for long-running operations.

**NFR5: Security**

* The system must not transmit any data over the network during normal operation.
* The system must use strong cryptographic algorithms (RSA-2048, SHA-256) for update verification.
* The system must not store private keys in the distributed executable.

**NFR6: Maintainability**

* The codebase must be well-documented with comments explaining complex logic.
* The system must use modular architecture with clear separation between components.
* The system must include logging at appropriate levels (DEBUG, INFO, WARNING, ERROR).

**NFR7: Scalability**

* The system must support databases up to 50 GB in size.
* The system must handle log files up to 5 GB each during ingestion.
* The system must allow concurrent execution of queries and anomaly detection.

**3.3 Planning & Scheduling**

The development of Project Quorum follows a structured approach divided into multiple phases. The project timeline spans approximately 16-18 weeks, organized into distinct phases with specific deliverables.

**3.3.1 Project Phases**

**Phase 1: Research and Design (Weeks 1)**

Activities:

* Study existing log analysis tools and identify gaps for offline environments
* Research MITRE ATT&CK framework structure and integration approaches
* Evaluate and finalize technology stack choices
* Design system architecture and component interactions
* Create database schema for log storage and metadata
* Design REST API endpoints for backend services
* Create wireframes and mock-ups for user interface

Deliverables:

* System architecture document
* Database schema design
* API specification document
* UI/UX mock-ups

**Phase 2: Core Backend Development (Weeks 2-4)**

Activities:

* Set up Python development environment with FastAPI
* Implement log parsing modules for EVTX and Syslog formats
* Develop DuckDB integration layer for data storage
* Create data models and ORM mappings
* Build REST API endpoints for log ingestion and querying
* Implement SQL query execution engine with security validation
* Develop error handling and logging mechanisms
* Write unit tests for parsing and database modules

Deliverables:

* Functional log parsing engine
* Working database layer with CRUD operations
* REST API backend with core endpoints
* Unit test suite with >70% code coverage

**Phase 3: Machine Learning Integration (Weeks 4-6)**

Activities:

* Implement feature extraction from parsed log data
* Integrate PyOD library with support for multiple algorithms
* Develop model training pipeline using sample datasets
* Create model serialization and loading mechanisms
* Implement TensorFlow Lite integration for inference
* Build anomaly scoring and ranking system
* Test ML models on benchmark datasets (LogPai, HDFS, BGL)
* Optimize model performance and reduce inference time

Deliverables:

* Trained anomaly detection models
* Feature extraction pipeline
* ML inference engine integrated with backend
* Performance benchmarks on test datasets

**Phase 4: MITRE ATT&CK Integration (Weeks 7)**

Activities:

* Download and parse MITRE ATT&CK framework data
* Create mapping logic between log patterns and ATT&CK techniques
* Build ATT&CK matrix visualization data structures
* Implement technique description and reference storage
* Develop API endpoints for ATT&CK data retrieval
* Create anomaly-to-technique mapping algorithms
* Test mapping accuracy with known attack scenarios

Deliverables:

* MITRE ATT&CK database
* Mapping engine
* API endpoints for ATT&CK data

**Phase 5: Frontend Development (Weeks 8-9)**

Activities:

* Set up React development environment
* Implement main dashboard layout and navigation
* Create log ingestion interface with file upload
* Build SQL query interface with result display
* Develop anomaly detection results viewer
* Implement MITRE ATT&CK matrix visualization
* Create charts and graphs for log statistics
* Build export functionality for reports and data
* Implement responsive design for different screen sizes
* Integrate frontend with backend API

Deliverables:

* Complete React frontend application
* All UI components implemented and styled
* Frontend-backend integration complete

**Phase 6: Secure Offline Update Protocol (Week 10)**

Activities:

* Design update package format and structure
* Implement RSA key pair generation for signing
* Develop signature creation tool for update packages
* Build signature verification mechanism in application
* Implement update logging and audit trail
* Test with valid and tampered update packages

Deliverables:

* SOUP implementation
* Update signing tool
* Verification mechanism
* Documentation for update process

**Phase 7: Packaging and Deployment (Week 11)**

Activities:

* Set up Tauri build environment
* Configure build settings for Windows, macOS, and Linux
* Integrate FastAPI backend with Tauri
* Bundle all dependencies and assets
* Test executables on clean systems
* Optimize bundle size and startup time

Deliverables:

* Cross-platform executables
* Installation guides
* Platform-specific packages

**Phase 8: Testing and Documentation (Weeks 12)**

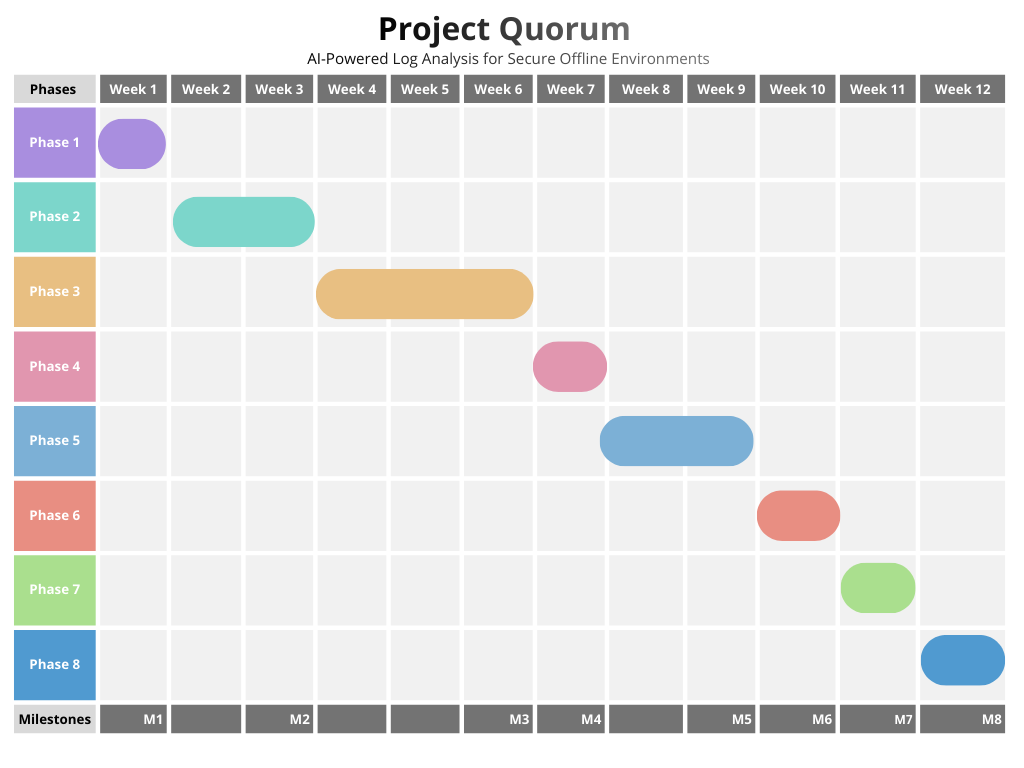
Activities:

* Conduct integration testing of all components
* Perform end-to-end testing with real-world log samples
* Test on different operating systems and hardware configurations
* Identify and fix bugs
* Write user manual with screenshots
* Prepare project presentation and black book

Deliverables:

* Test results and bug reports
* Final black book document
* Project presentation

**3.3.2 Gantt Chart**



*Fig 3.1: Gantt Chart*

**3.3.3 Milestones**

* **Milestone 1 (Week 1):** Design approval and technology stack finalized
* **Milestone 2 (Week 3):** Backend API functional with log parsing and database storage
* **Milestone 3 (Week 6):** ML models trained and integrated with anomaly detection working
* **Milestone 4 (Week 7):** MITRE ATT&CK mapping complete
* **Milestone 5 (Week 8):** Frontend dashboard complete and integrated
* **Milestone 6 (Week 10):** SOUP implementation functional
* **Milestone 7 (Week 11):** Executable builds successful on all platforms
* **Milestone 8 (Week 12):** Project complete with documentation

**3.4 Software and Hardware Requirements**

**3.4.1 Software and Development Environment Requirements**

* **Operating System:** Windows 10/11, macOS 10.15+, or Linux (Ubuntu 20.04+ recommended)
* **Programming Languages:**
  + Python 3.9 or higher
  + Node.js 16.x or higher for React development
  + Rust 1.60+ for Tauri framework
* **Integrated Development Environment (IDE):**
  + Visual Studio Code (recommended) or PyCharm Professional
  + Extensions: Python, ESLint, Prettier, React Developer Tools
* **Version Control:**
  + Git 2.30+
  + GitHub or GitLab for repository hosting
* **Python Libraries and Frameworks:**
  + FastAPI 0.95+
  + DuckDB 0.8.0+
  + PyOD 1.0.9+
  + TensorFlow Lite 2.12+
  + python-evtx 0.7.0+
  + cryptography 40.0+
  + uvicorn (ASGI server)
  + pandas, numpy, scikit-learn
* **Frontend Libraries:**
  + React 18.x
  + Ant Design or Material-UI for components
  + Recharts for data visualization
  + Axios for API communication
* **Desktop Framework:**
  + Tauri 1.2+
* **Additional Tools:**
  + npm or yarn for package management
  + Docker (optional, for testing different environments)
  + Postman or similar for API testing

**3.4.2 Hardware Requirements for Development**

* **Processor:** Intel Core i5 / AMD Ryzen 5 or better (64-bit)
* **RAM:** Minimum 8 GB, 16 GB recommended
* **Storage:** Minimum 20 GB free space (for development tools, dependencies, and test data)
* **Display:** 1920×1080 resolution or higher recommended for UI development

**3.4.3 Test Data Requirements**

* **Benchmark Datasets:**
  + LogPai dataset (various system logs)
  + HDFS logs (Hadoop Distributed File System)
  + BGL logs (Blue Gene/L supercomputer)
* **Sample Log Files:**
  + Windows Event Logs (EVTX) from test systems
  + Linux Syslog samples
  + Simulated attack logs for testing MITRE ATT&CK mapping
* **Storage for Test Data:** Approximately 5-10 GB

# 4. SYSTEM DESIGN

## 4.1 Schema Design

### **4.1.1 Theory (Conceptual Overview)**

### Quorum employs DuckDB, an embedded analytical database designed for OLAP (Online Analytical Processing) workloads. Unlike traditional transactional databases, DuckDB uses a columnar storage engine, which significantly improves performance for analytical queries such as aggregations, statistical computations, and large-scale log analysis.

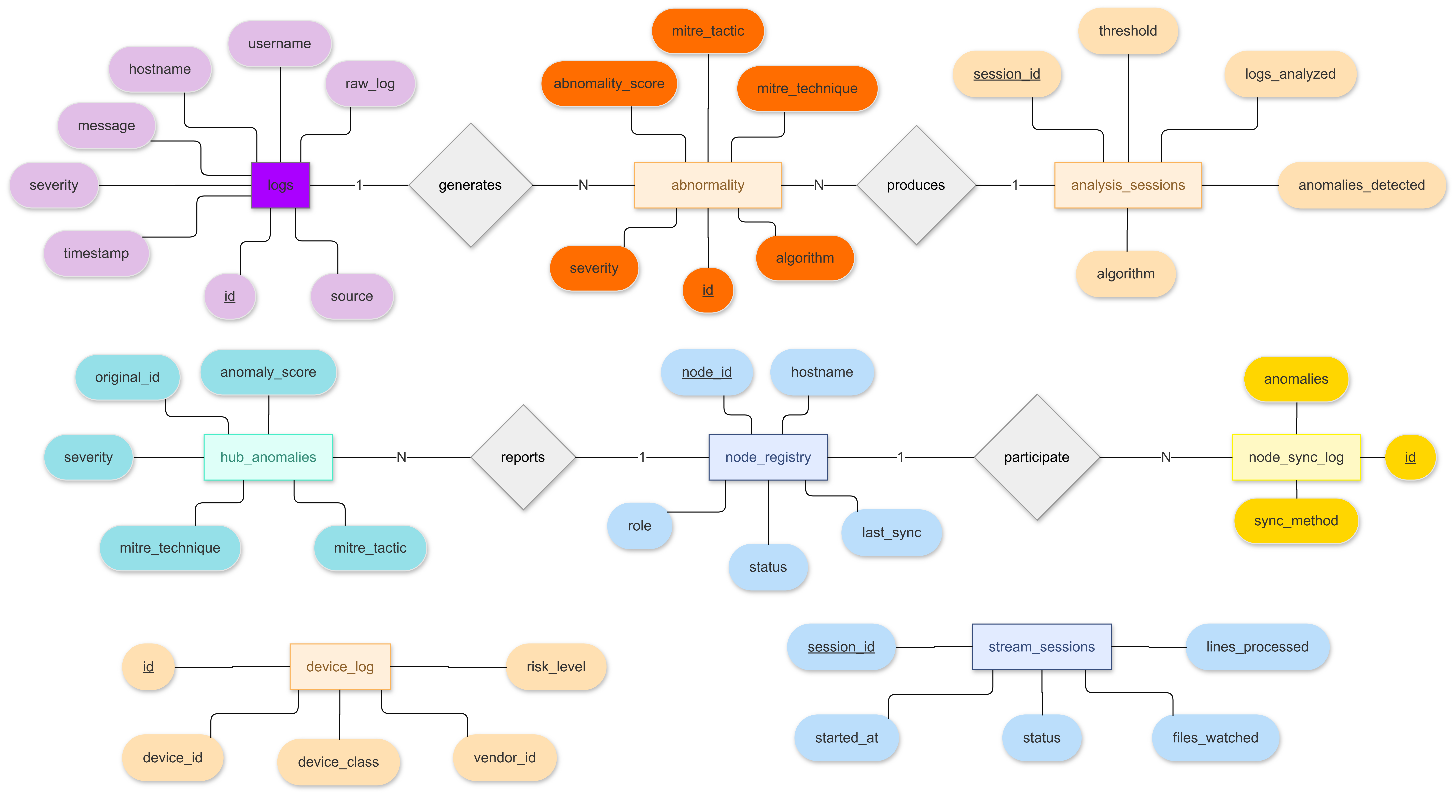
### The architecture is designed around the following core principles:

### **Embedded Analytics:** The database runs locally within the application process, eliminating the need for external database servers. This design is critical for air-gapped and offline cybersecurity environments.

### **Denormalization for Performance:** Certain attributes are intentionally duplicated across tables to reduce join complexity and improve query latency for read-heavy operations such as threat dashboards and correlation analysis.

### **Explainable AI Storage:** Feature vectors and model metadata are stored alongside anomaly results to support transparency, auditing, and reproducibility of AI decisions.

**4.1.2 Database Architecture Diagram (ER Representation)**

****

*Fig 4.1: ER-Diagram*

**4.1.3 Explanation of Schema Components**

1. **Logs Table (Primary Data Layer)**

* The logs table stores all ingested log events from heterogeneous sources such as:
  + Windows Event Logs
  + Linux syslogs
  + Application logs
* Key characteristics:
  + Preserves raw forensic evidence
  + Supports flexible metadata via JSON (parsed\_fields)
  + Serves as the foundational dataset for AI analysis
* This table is optimized for high-volume ingestion and analytical queries.

1. **Anomalies Table (AI Detection Layer)**

* The anomalies table stores suspicious events identified by machine learning algorithms.
* It includes:
  + AI confidence score
  + Detection algorithm used
  + Mapping to MITRE ATT&CK framework developed by MITRE
  + Feature vectors for explainability
* Separating anomalies from logs improves query efficiency because dashboards typically analyse only anomalous data rather than all logs.

1. **Analysis Sessions Table (Experiment Tracking)**

* This table records metadata about each AI execution run:
  + Algorithm configuration
  + Threshold parameters
  + Model file path
  + Performance statistics
* It enables:
  + Reproducibility
  + Model benchmarking
  + Auditability

1. **Multi-Node Hub Architecture Tables**
2. **Node Registry**

* Maintains a centralized inventory of all nodes in the distributed deployment.
* Supports:
* Network topology awareness
* Node health monitoring
* Synchronization tracking

1. **Hub Anomalies**

* Stores aggregated anomalies imported from multiple terminal nodes.
* Purpose:
  + Cross-node threat correlation
  + Detection of coordinated attacks across systems

1. **Node Sync Log**

* Provides an audit trail of synchronization operations, especially important in air-gapped environments using physical transfer media.

1. **Device Monitoring Tables**

* The device\_log table records hardware interaction events such as:
  + USB insertion
  + External storage connection
  + Network device detection
* This supports:
  + Insider threat detection
  + Physical security monitoring
  + Forensic timeline reconstruction

1. **Real-Time Streaming Tables**

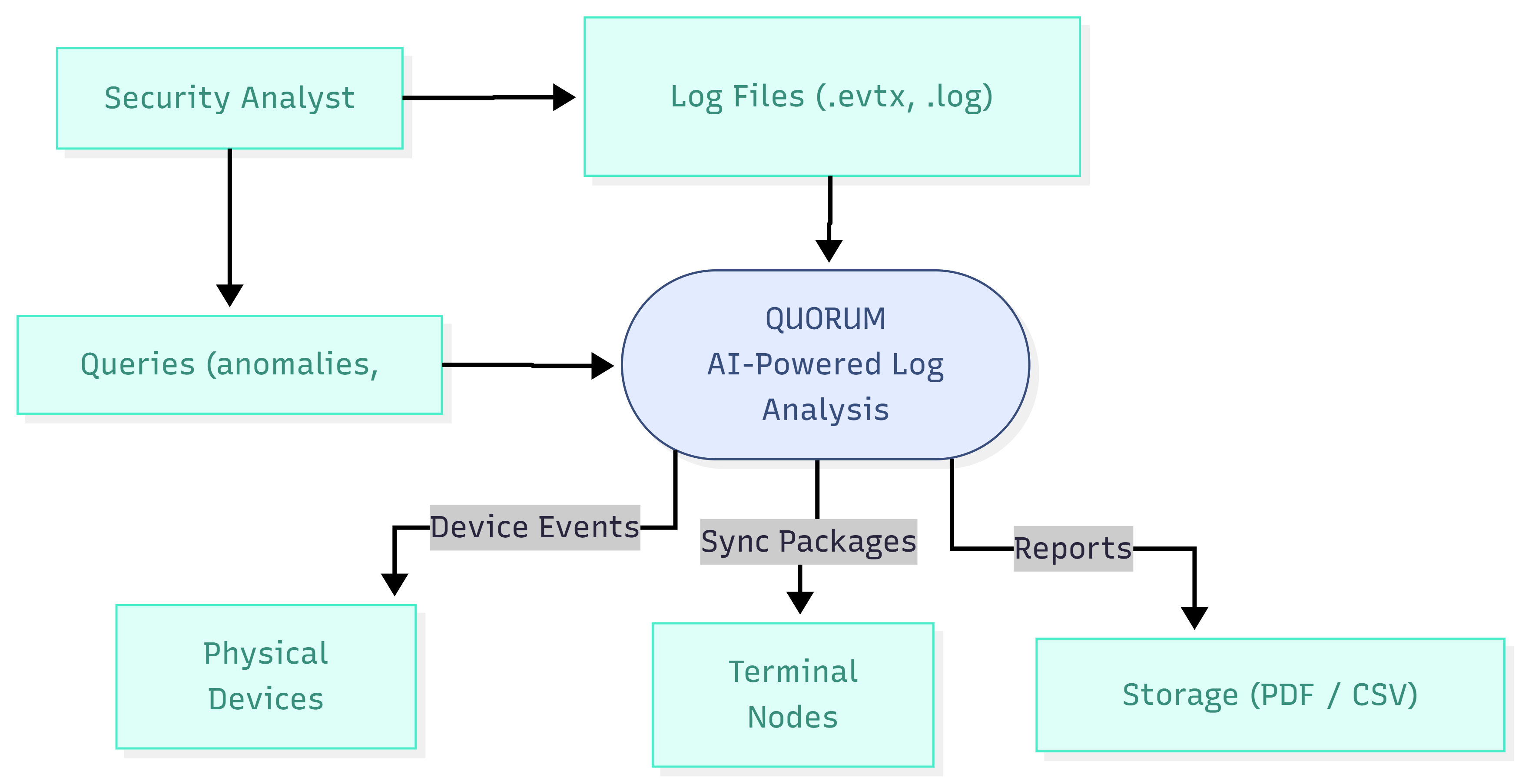
* The stream\_sessions table tracks active monitoring sessions including:
  + Files being watched
  + Processing throughput
  + Session duration
* This enables performance analysis and operational monitoring.

## 4.2 UML Diagrams

### **4.2.1 Data Flow Diagram (DFD)**

**Level 0: Context Diagram**

Shows Quorum as a single process with external entities and data flows.



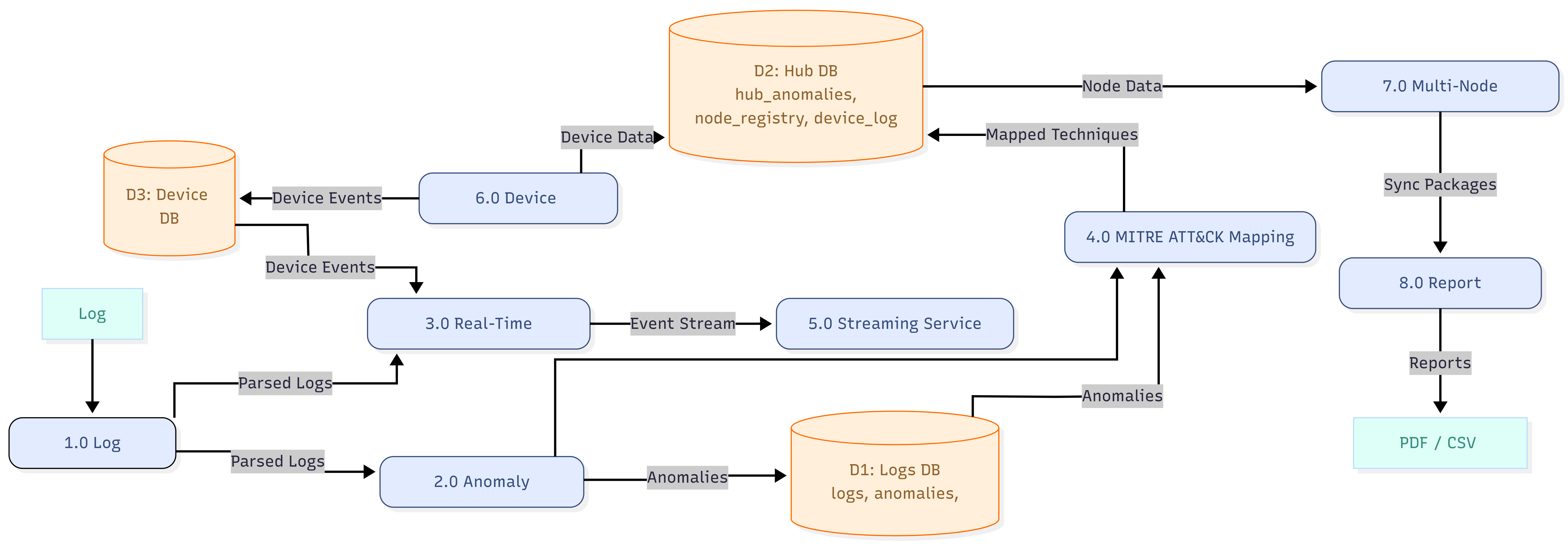
*Fig 4.2.1: DFD Level 0*

**External Entities:**

* **Security Analyst:** Primary user who uploads logs and investigates threats
* **Physical Devices:** USB drives, keyboards, LAN nodes that generate events
* **Terminal Nodes:** Other Quorum instances in distributed mode
* **Storage:** File system for reports and sync packages

**Level 1: Top-Level Processes**

Decomposes Quorum into major functional processes.



*Fig 4.2.2: DFD Level 1*

**Process Descriptions:**

**1.0 Log Ingestion**

* **Input:** Log files (.evtx, .log, .syslog, .txt) from Security Analyst
* **Processing:**
  + Parse file format (Windows Event Log XML, syslog format, plain text)
  + Extract structured fields (timestamp, source, message, event\_id)
  + Validate data types
  + Handle encoding (UTF-8, UTF-16)
* **Output:** Parsed logs → Logs DB (D1)
* **Data Store:** D1 (logs table)

**2.0 Anomaly Detection**

* **Input:** Parsed logs from D1
* **Processing:**
  + Extract 20 features (entropy, length, special\_chars, keyword\_score, etc.)
  + Run ensemble algorithm (IF, SVM, Statistical, Keyword)
  + Calculate weighted anomaly score (0.0-1.0)
  + Apply threshold (default 0.70)
* **Output:** Detected anomalies → Logs DB (D1)
* **Data Store:** D1 (anomalies table, analysis\_sessions table)

**3.0 Real-Time Monitoring**

* **Input:** Live log files being written
* **Processing:**
  + Tail files (detect new lines)
  + Quick keyword-based scoring
  + Filter by severity threshold
  + Enqueue events
* **Output:** Event stream → Streaming Service (5.0)
* **Data Store:** D1 (stream\_sessions table)

**4.0 MITRE ATT&CK Mapping**

* **Input:** Anomalies from D1
* **Processing:**
  + Match patterns to MITRE techniques (keyword + heuristic)
  + Assign technique\_id (e.g., T1110.001 for brute force)
  + Determine tactic (e.g., credential\_access, lateral\_movement)
* **Output:** MITRE-mapped anomalies → Hub DB (D2)
* **Data Store:** D2 (hub\_anomalies table)

**5.0 Streaming Service (SSE)**

* **Input:** Event stream from Real-Time Monitoring (3.0)
* **Processing:**
  + Format events as Server-Sent Events (SSE)
  + Apply client-requested filters (min\_score, severity)
  + Maintain connection state
* **Output:** SSE stream → Security Analyst (Web UI)
* **Data Store:** None (stateless streaming)

**6.0 Device Monitoring**

* **Input:** Physical device events (USB hotplug, LAN discovery)
* **Processing:**
  + Enumerate USB devices (VID/PID classification)
  + Scan LAN subnet (ping + ARP)
  + Classify device type (STORAGE, AUDIO, HID, NETWORK, etc.)
  + Assess risk level (HIGH for unknown storage, LOW for known keyboards)
* **Output:** Device events → Device DB (D3)
* **Data Store:** D3 (device\_log table)

**7.0 Multi-Node Aggregation**

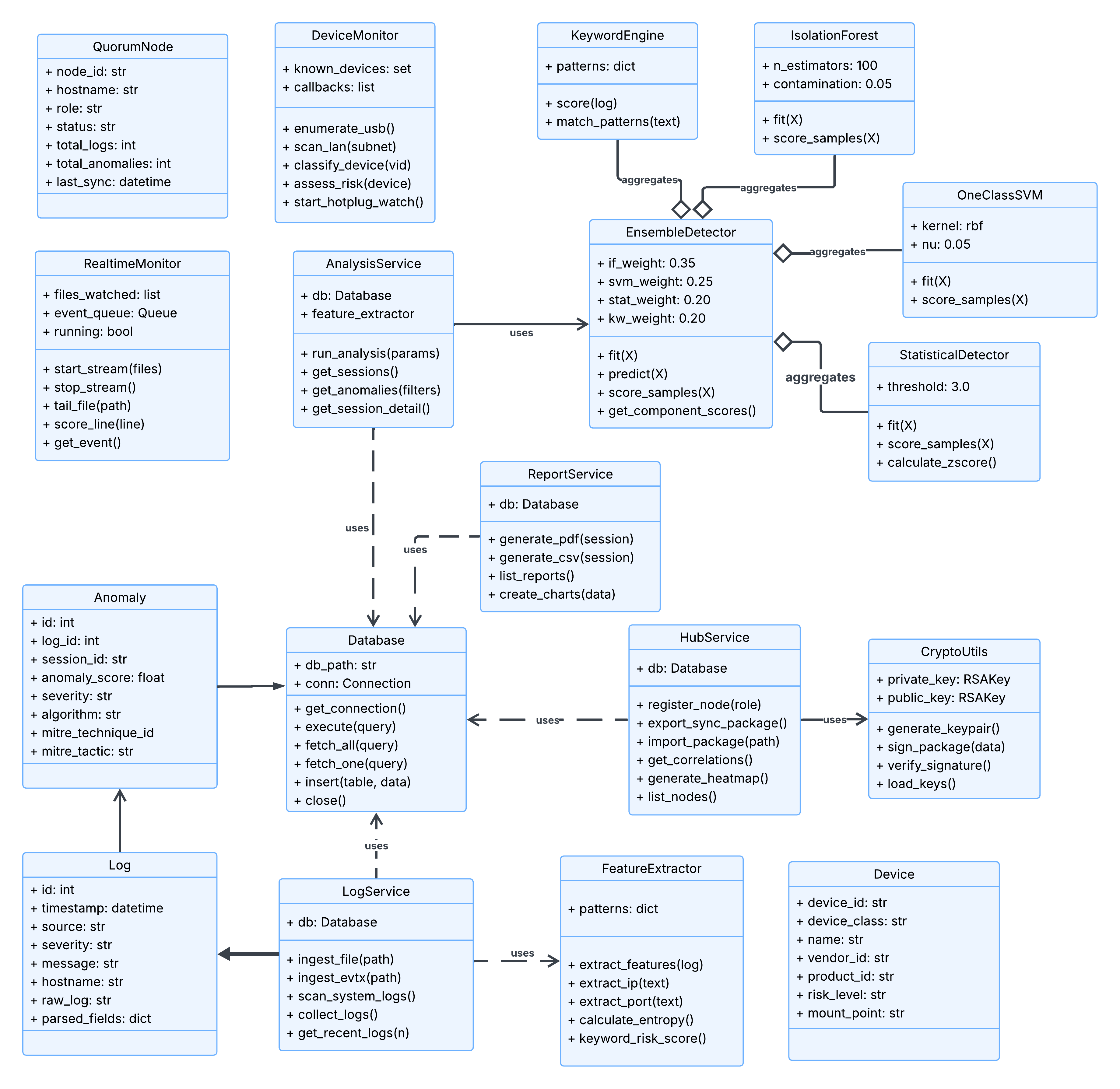
* **Input:** Sync packages from Terminal Nodes (via USB)
* **Processing:**
  + Verify RSA-PSS signature
  + Extract anomalies from package
  + Check for duplicates (original\_id + source\_node)
  + Merge into hub\_anomalies table
  + Perform cross-node correlation (GROUP BY technique\_id)
* **Output:** Aggregated view → Hub DB (D2)
* **Data Store:** D2 (node\_registry, hub\_anomalies, node\_sync\_log)

**8.0 Report Generation**

* **Input:** Analysis session data from D1, D2
* **Processing:**
  + Query anomalies for session
  + Generate charts (Matplotlib/Plotly)
  + Create PDF (ReportLab) or CSV
  + Include MITRE heatmap, severity distribution, timeline
* **Output:** PDF/CSV reports → File system
* **Data Store:** File system (reports/ directory)

### **4.2.2 Class Diagram**

**Purpose:** Shows the object-oriented structure of Quorum's backend, including classes, attributes, methods, and relationships.



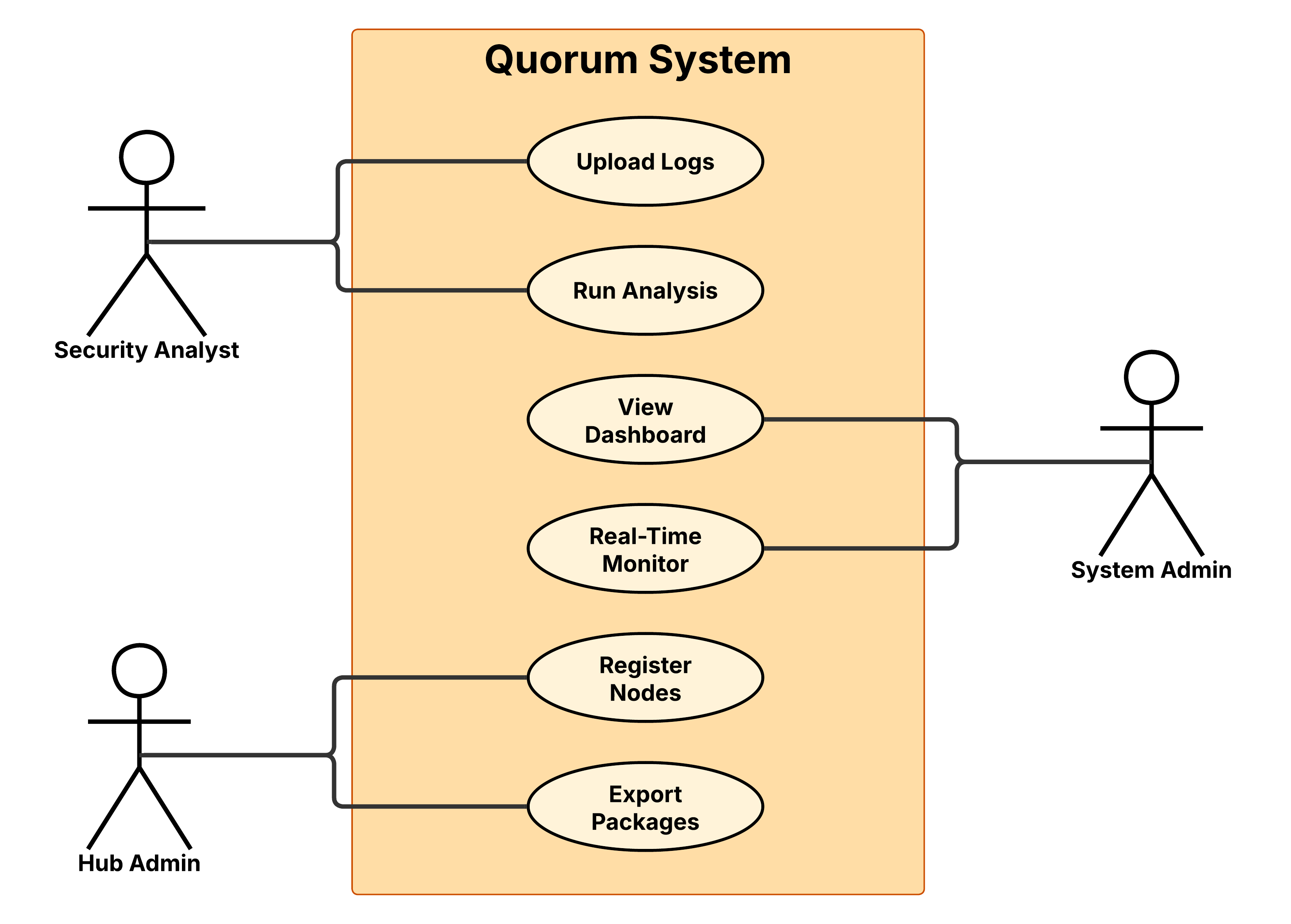
*Fig 4.2: Class Diagram*

**Class Relationships:**

* **Composition:** AnalysisService contains FeatureExtractor
* **Aggregation:** EnsembleDetector aggregates IsolationForest, OneClassSVM,
* **Association:** LogService uses Database, HubService uses CryptoUtils
* **Dependency:** All service classes depend on Database class
* **Inheritance:** (Not heavily used in this architecture; services use composition pattern)

### **4.2.3 Use Case Diagram**

**Purpose:** Illustrates the functional requirements of Quorum from the perspective of different actors (users) and their interactions with the system.



*Fig 4.4: Use Case Diagram*

**Actors:**

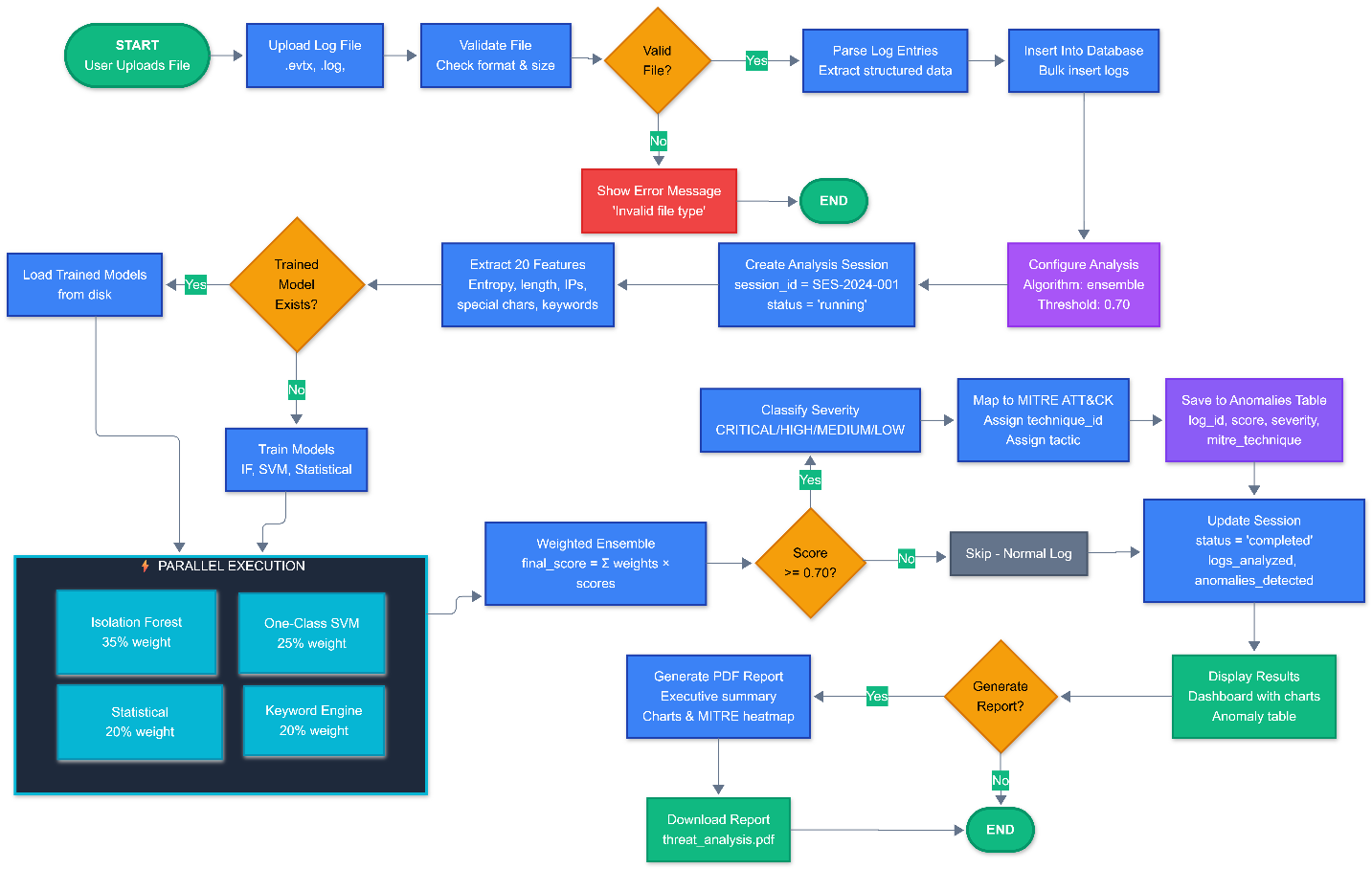
1. **Security Analyst** - Primary user who investigates threats
2. **System Administrator** - Manages Quorum installation and configuration
3. **Hub Administrator** - Manages multi-node aggregation

**Key Use Cases:**

* **Upload Logs:** The Security Analyst uploads system or network log files into the Quorum platform for processing. The system validates, parses, and securely stores the logs for further analysis.
* **Run Analysis:** The analyst initiates AI-based log analysis to detect anomalies, threats, or suspicious behaviour. The system applies embedded ML models and rule engines to generate insights and alerts.
* **View Dashboard:** Authorized users (Security Analyst / System Admin) access a centralized dashboard displaying threat summaries, statistics, risk scores, and system health metrics in a visual format.
* **Real-Time Monitoring:** The System Admin monitors live incoming data streams and node activity. The system continuously updates alerts and anomaly indicators to support proactive threat detection.
* **Register Nodes:** The Hub Admin registers new endpoints or network nodes into the Quorum environment, enabling distributed log collection from multiple sources.
* **Export Packages:** The Hub Admin exports analysis reports, threat intelligence updates, or secure offline update packages (SOUP) for deployment in air-gapped environments.

### **4.2.4 Activity Diagram**

**Purpose:** Shows the workflow of the core AI anomaly detection process from log ingestion to report generation.



***Fig 4.5: Activity Diagram***

**Key Decision Points:**

1. **File Validation:** Rejects invalid files before processing
2. **Model Existence:** Trains new models or loads existing ones
3. **Threshold Check:** Determines which logs are anomalies
4. **Severity Classification:** Assigns urgency level to anomalies
5. **Report Generation:** Optional final step

**Parallel Activities:**

* Feature extraction is vectorized (processes all logs at once)
* Model training runs in parallel using ThreadPoolExecutor
* Scoring from 4 algorithms executes concurrently

**5. CODE AND IMPLEMENTATION**

**5.1 Backend Implementation**

backend/

├─ ai\_engine/

│  └─ utils/

├─ api/

│  └─ routes/

├─ cli/

│  └─ commands/

├─ config/

├─ core/

├─ data/

│  ├─ databases/

│  ├─ keys/

│  ├─ mitre\_attack/

│  ├─ models/

│  ├─ samples/

│  ├─ updates/

│  └─ uploads/

├─ models/

├─ parsers/

├─ scripts/

├─ services/

├─ tests/

└─ docs (.md files)

**5.1.1 bacakend/main.py**

""" Quorum - Main Entry Point

AI-Powered Log Analysis for Secure Offline Environments """

import sys

from pathlib import Path

# Add project root to path

sys.path.insert(0, str(Path(\_\_file\_\_).parent))

from cli.main import cli

if \_\_name\_\_ == '\_\_main\_\_':

    cli()

**5.1.2 backend/cli/main.py**

"""Main CLI Entry Point — v1.1.0

Registers: ingest, analyze, query, report, update, monitor, hub, devices"""

import click

from cli.commands.ingest   import ingest

from cli.commands.analyze  import analyze

from cli.commands.query    import query

from cli.commands.report   import report

from cli.commands.update   import update

from cli.commands.monitor  import monitor

from cli.commands.hub      import hub

from cli.commands.devices  import devices

from config.settings       import settings

from config.logging\_config import setup\_logging, get\_logger

setup\_logging(

    log\_level=settings.LOG\_LEVEL,

    log\_format=settings.LOG\_FORMAT,

    log\_file=settings.LOG\_FILE,

    log\_dir=settings.LOGS\_DIR

)

logger = get\_logger(\_\_name\_\_)

@click.group()

@click.version\_option(version=settings.APP\_VERSION)

def cli():

    """

    Quorum - AI-Powered Offline Log Analysis\n

    Air-gapped forensic threat detection platform.\n

    \b

    Quick start:\n

      python main.py init\n

      python main.py ingest scan          # find available logs\n

      python main.py ingest collect       # collect system logs\n

      python main.py analyze run          # run hybrid AI analysis\n

      python main.py report generate      # generate reports\n

    """

    pass

cli.add\_command(ingest)

cli.add\_command(analyze)

cli.add\_command(query)

cli.add\_command(report)

cli.add\_command(update)

cli.add\_command(monitor)   # NEW v1.1.0

cli.add\_command(hub)       # NEW v1.1.0

cli.add\_command(devices)   # NEW v1.1.0

@cli.command()

def status():

    """Show system status and statistics"""

    try:

        from cli.utils import print\_header, print\_info, print\_success, print\_warning

        from core.database import db

        from core.environment import env\_detector

        print\_header("Quorum System Status")

        # App info

        click.echo(click.style(f"  Version:     ", fg='cyan') + settings.APP\_VERSION)

        click.echo(click.style(f"  Database:    ", fg='cyan') + str(settings.database\_path))

        # DB stats

        total\_logs = db.get\_table\_count('logs')

        total\_anomalies = db.get\_table\_count('anomalies')

        total\_sessions = db.get\_table\_count('analysis\_sessions')

        click.echo()

        click.echo(click.style("  Database:", bold=True))

        click.echo(f"    Logs:       {total\_logs:,}")

        click.echo(f"    Anomalies:  {total\_anomalies:,}")

        click.echo(f"    Sessions:   {total\_sessions:,}")

        # Recent session

        if total\_sessions > 0:

            last = db.fetch\_one(

                "SELECT session\_id, start\_time, anomalies\_detected "

                "FROM analysis\_sessions ORDER BY start\_time DESC LIMIT 1"

            )

            if last:

                click.echo(f"\n  Last Analysis: {str(last['start\_time'])[:16]}")

                click.echo(f"  Anomalies:     {last['anomalies\_detected']}")

        # Environment

        click.echo()

        env\_info = env\_detector.detect\_all()

        env\_type = env\_info.get('environment\_type')

        env\_val = env\_type.value if env\_type else 'unknown'

        env\_color = {'air\_gapped': 'green', 'lan\_connected': 'yellow',

                     'internet\_connected': 'red'}.get(env\_val, 'white')

        click.echo(click.style("  Environment:", bold=True))

        click.echo(f"    Status:  " + click.style(env\_val, fg=env\_color))

        click.echo(f"    Role:    {env\_info.get('system\_role', {}).value if env\_info.get('system\_role') else 'unknown'}")

        click.echo(f"    OS:      {env\_info.get('os', 'unknown')}")

        usb = env\_info.get('usb\_devices', [])

        if usb:

            click.echo(f"    USB:     {len(usb)} device(s) connected")

        # Reports

        from config.settings import settings as s

        if s.REPORTS\_DIR.exists():

            session\_dirs = [d for d in s.REPORTS\_DIR.iterdir() if d.is\_dir()]

            all\_reports = list(s.REPORTS\_DIR.rglob('\*.pdf')) + list(s.REPORTS\_DIR.rglob('\*.csv'))

            if all\_reports:

                click.echo(f"\n  Reports:     {len(all\_reports)} file(s) in {len(session\_dirs)} session(s)")

        click.echo()

        print\_success("System operational")

        if total\_logs == 0:

            click.echo()

            print\_warning("No logs ingested yet. Try:")

            click.echo("  python main.py ingest scan")

            click.echo("  python main.py ingest file <path>")

    except Exception as e:

        from cli.utils import print\_error

        print\_error(f"Status check failed: {e}")

        logger.error(f"Status error: {e}", exc\_info=True)

@cli.command()

def init():

    """Initialize Quorum (first-time setup)"""

    try:

        from cli.utils import print\_header, print\_info, print\_success

        print\_header("Initializing Quorum")

        print\_info("Creating directories...")

        settings.\_create\_directories()

        print\_info("Initializing database...")

        from core.database import db

        print\_info("Loading MITRE ATT&CK data...")

        from services.mitre\_service import mitre\_service

        count = db.get\_table\_count('mitre\_techniques')

        if count == 0:

            n = mitre\_service.load\_mitre\_data()

            print\_success(f"Loaded {n} MITRE techniques")

        else:

            print\_info(f"MITRE data already loaded ({count} techniques)")

        print\_info("Checking cryptographic keys...")

        if not settings.public\_key\_path.exists():

            from core.security import CryptoUtils

            priv, pub = CryptoUtils.generate\_key\_pair()

            with open(settings.public\_key\_path, 'wb') as f:

                f.write(pub)

            priv\_path = settings.KEYS\_DIR / "private\_key.pem"

            with open(priv\_path, 'wb') as f:

                f.write(priv)

            print\_success("Key pair generated")

        else:

            print\_info("Keys already exist")

        print\_success("\n✓ Quorum initialized successfully!")

        click.echo()

        print\_info("Next steps:")

        print\_info("  1. Scan for logs:    python main.py ingest scan")

        print\_info("  2. Collect logs:     python main.py ingest collect")

        print\_info("  3. Run analysis:     python main.py analyze run")

        print\_info("  4. View results:     python main.py analyze results")

        print\_info("  5. Generate report:  python main.py report generate")

    except Exception as e:

        from cli.utils import print\_error

        print\_error(f"Initialization failed: {e}")

        raise click.Abort()

@cli.command()

def scan():

    """Shortcut: scan system for available log files"""

    from cli.commands.ingest import scan as ingest\_scan

    ctx = click.Context(ingest\_scan)

    ingest\_scan.invoke(ctx)

@cli.command()

def interactive():

    """Start interactive shell mode"""

    from cli.utils import print\_header, print\_info

    print\_header("Quorum Interactive Mode")

    print\_info("Commands: ingest, analyze, query, report, update, status, scan, exit")

    while True:

        try:

            raw = click.prompt("\nquorum", default="", show\_default=False)

            cmd = raw.strip()

            if not cmd:

                continue

            if cmd.lower() in ['exit', 'quit', 'q']:

                print\_info("Goodbye!")

                break

            if cmd.lower() == 'help':

                ctx = click.Context(cli)

                click.echo(cli.get\_help(ctx))

                continue

            try:

                cli.main(cmd.split(), standalone\_mode=False)

            except SystemExit:

                pass

            except Exception as e:

                from cli.utils import print\_error

                print\_error(str(e))

        except (KeyboardInterrupt, EOFError):

            click.echo()

            print\_info("Goodbye!")

            break

if \_\_name\_\_ == '\_\_main\_\_':

    cli()

**5.1.3 backend/api/main.py**

"""FastAPI Application

Main API application setup"""

from fastapi import FastAPI

from fastapi.middleware.cors import CORSMiddleware

from contextlib import asynccontextmanager

from api.routes import (logs, analysis, queries, reports, updates, system, devices, hub, stream, cli)

from config.settings import settings

from config.logging\_config import setup\_logging, get\_logger

# Setup logging

setup\_logging(

    log\_level=settings.LOG\_LEVEL,

    log\_format=settings.LOG\_FORMAT,

    log\_file=settings.LOG\_FILE,

    log\_dir=settings.LOGS\_DIR

)

logger = get\_logger(\_\_name\_\_)

@asynccontextmanager

async def lifespan(app: FastAPI):

    """Application lifespan manager"""

    # Startup

    logger.info(f"Starting {settings.APP\_NAME} API v{settings.APP\_VERSION}")

   # Initialize MITRE data if needed

    try:

        from services.mitre\_service import mitre\_service

        from core.database import db

        technique\_count = db.get\_table\_count('mitre\_techniques')

        if technique\_count == 0:

            logger.info("Loading MITRE ATT&CK data...")

            mitre\_service.load\_mitre\_data()

    except Exception as e:

        logger.warning(f"MITRE data initialization failed: {e}")

   yield

    # Shutdown

    logger.info("Shutting down API")

    from core.database import db

    db.close()

# Create FastAPI app

app = FastAPI(title=settings.APP\_NAME, version=settings.APP\_VERSION, description="AI-Powered Log Analysis for Secure Offline Environments", lifespan=lifespan)

# CORS middleware

app.add\_middleware(CORSMiddleware, allow\_origins=["\*"], allow\_credentials=True, allow\_methods=["\*"], allow\_headers=["\*"],)

# Include routers

app.include\_router(system.router)

app.include\_router(logs.router)

app.include\_router(analysis.router)

app.include\_router(queries.router)

app.include\_router(reports.router)

app.include\_router(updates.router)

app.include\_router(devices.router)

app.include\_router(hub.router)

app.include\_router(stream.router)

app.include\_router(cli.router)

@app.get("/")

async def root():

    """Root endpoint"""

    return {"name": settings.APP\_NAME, "version": settings.APP\_VERSION, "status": "online", "docs": "/docs"}

if \_\_name\_\_ == "\_\_main\_\_":

    import uvicorn

    uvicorn.run(

        "api.main:app",

        host=settings.API\_HOST,

        port=settings.API\_PORT,

        reload=settings.API\_RELOAD,

        workers=settings.API\_WORKERS

    )

**5.2 Frontend Implementation**

frontend

├── src/

│   ├── main.tsx                          # App entry point

│   ├── App.tsx                           # Root routing

│   ├── App.css                           # App styles

│   ├── index.css                         # Global styles + design tokens

│   ├── vite-env.d.ts                     # Vite types

│   ├── components/

│   │   ├── layout/

│   │   ├── analysis/

│   │   ├── ui/                           # shadcn/ui components

│   │   └── NavLink.tsx                   # Nav link component

│   ├── pages/

│   ├── lib/

│   └── hooks/

├── public/

│

├── vite.config.ts

├── tailwind.config.ts

├── tsconfig.json

├── tsconfig.app.json

├── tsconfig.node.json

├── postcss.config.js

├── eslint.config.js

├── vitest.config.ts

├── components.json

└── README.md

**5.2.1 frontend/src/App.tsx**

import { Toaster } from "@/components/ui/toaster";

import { Toaster as Sonner } from "@/components/ui/sonner";

import { TooltipProvider } from "@/components/ui/tooltip";

import { QueryClient, QueryClientProvider } from "@tanstack/react-query";

import { BrowserRouter, Routes, Route, Navigate } from "react-router-dom";

import Dashboard from "./pages/Dashboard";

import Logs from "./pages/Logs";

import Monitor from "./pages/Monitor";

import Devices from "./pages/Devices";

import Analysis from "./pages/Analysis";

import Hub from "./pages/Hub";

import Reports from "./pages/Reports";

import Terminal from "./pages/Terminal";

import Settings from "./pages/Settings";

import NotFound from "./pages/NotFound";

const queryClient = new QueryClient();

const App = () => (

  <QueryClientProvider client={queryClient}>

    <TooltipProvider>

      <Toaster />

      <Sonner />

      <BrowserRouter>

        <Routes>

          <Route path="/" element={<Dashboard />} />

          <Route path="/logs" element={<Logs />} />

          <Route path="/monitor" element={<Monitor />} />

          <Route path="/devices" element={<Devices />} />

          <Route path="/analysis" element={<Analysis />} />

          <Route path="/hub" element={<Hub />} />

          <Route path="/reports" element={<Reports />} />

          <Route path="/terminal" element={<Terminal />} />

          <Route path="/settings" element={<Settings />} />

          <Route path="\*" element={<NotFound />} />

        </Routes>

      </BrowserRouter>

    </TooltipProvider>

  </QueryClientProvider>

);

export default App;

**6. RESULTS AND DISCUSSION**

**6.1 User Documentation**

**1. Dashboard Page**

The Dashboard provides a centralized operational overview of the Project QUORUM platform, displaying real-time threat intelligence, system health, and node activity. It aggregates log ingestion metrics, anomaly detection results, and severity distributions into a unified command interface.

**Key Components**

* Real-time threat counters and severity distribution charts
* Node status and synchronization monitoring
* Recent alerts and anomaly timeline visualization
* System health and ingestion performance metrics

**Screenshot**

****

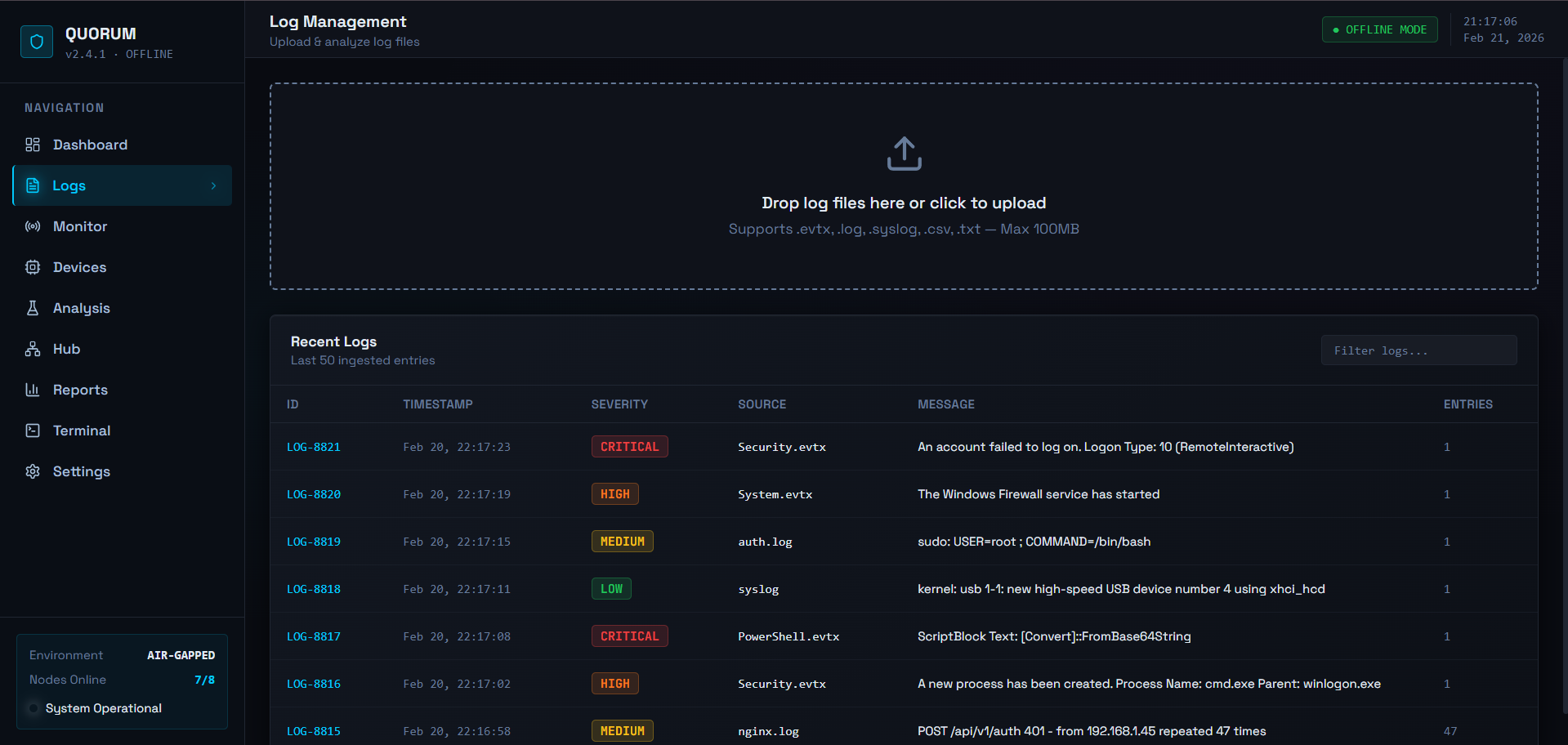
**2. Log Upload Page**

The Upload module enables secure ingestion of log files from multiple sources into the QUORUM analysis pipeline. It supports various formats such as system logs, network logs, and security event files. Uploaded data is automatically parsed, normalized, and stored for further AI-driven analysis. The interface ensures controlled data handling suitable for offline and sensitive infrastructure environments.

**Key Components**

* Multi-format log file upload (.log, .csv, .json, .evtx)
* Automatic parsing and normalization engine
* Upload history and file management controls
* Integrity validation and secure storage

**Screenshot**

****

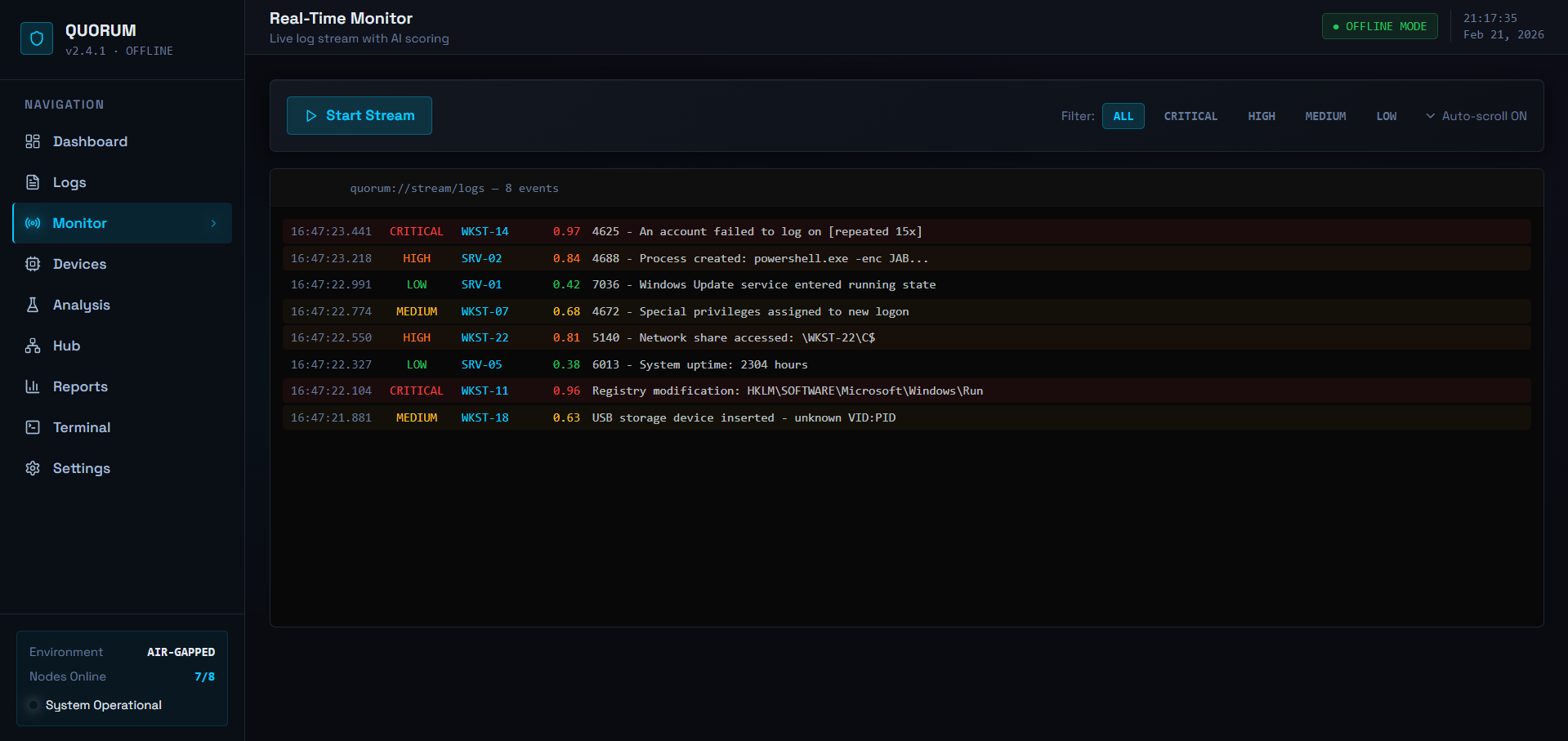
**3. Real-Time Monitoring Page**

This module enables continuous monitoring of incoming logs and system activities with minimal latency. It streams live events, applies instant anomaly detection, and generates alerts based on severity thresholds. The system is optimized for offline environments while maintaining near real-time responsiveness. It assists analysts in identifying threats as they occur rather than post-incident.

**Key Components**

* Live log streaming and visualization
* Instant anomaly detection and alert generation
* Severity-based filtering and prioritization
* Performance-optimized offline processing

**Screenshot**

****

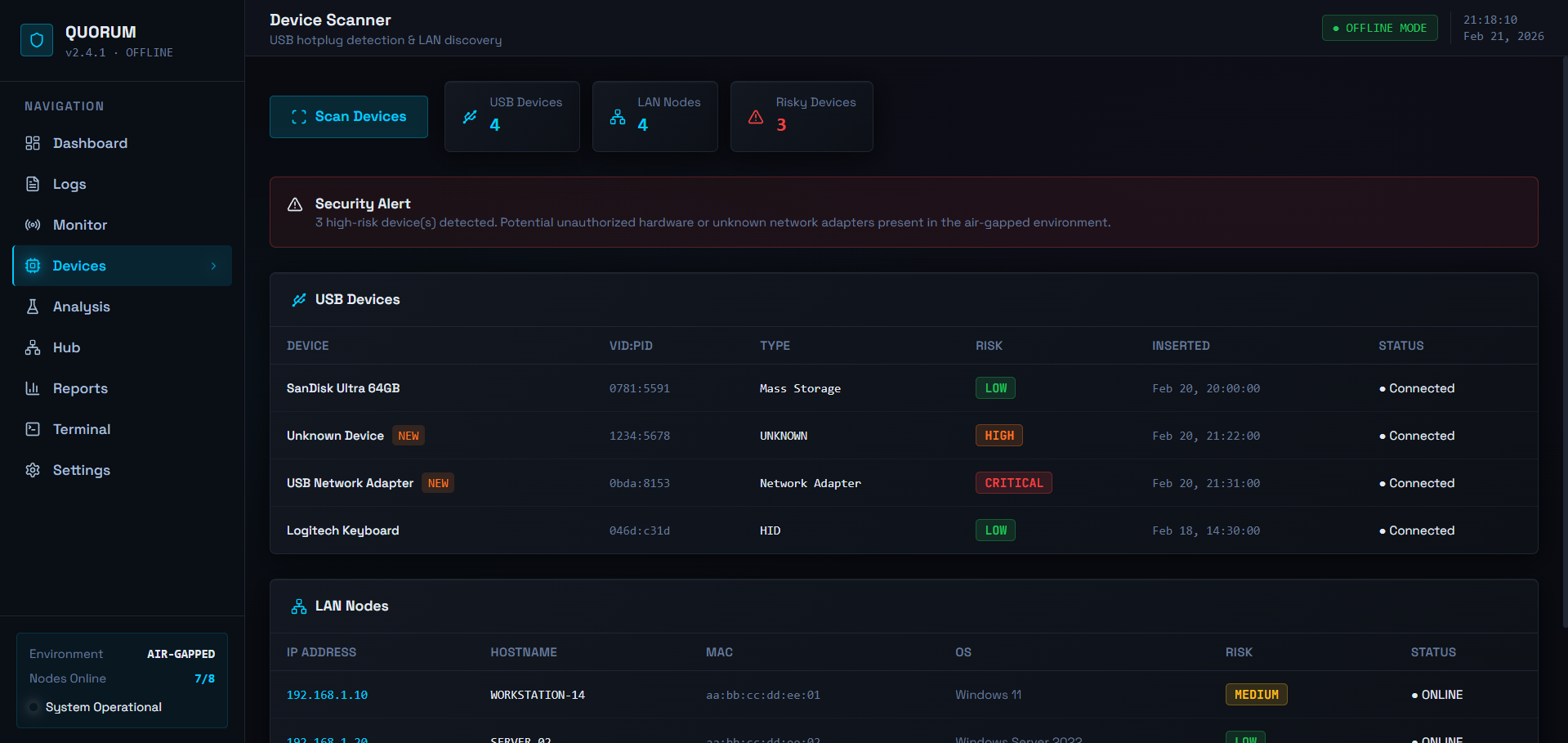
**4. Device Scan Page**

The Device Scanner detects connected hardware and network devices within the monitored environment to identify unauthorized access points. It analyzes USB connections, network interfaces, and system endpoints to detect anomalies or suspicious devices. This feature strengthens physical and network security in isolated infrastructures. The results integrate with the threat analysis engine for risk evaluation.

**Key Components**

* USB and network device discovery
* Unauthorized device detection logic
* Device activity logging and history
* Risk scoring integration with AI engine

**Screenshot**

****

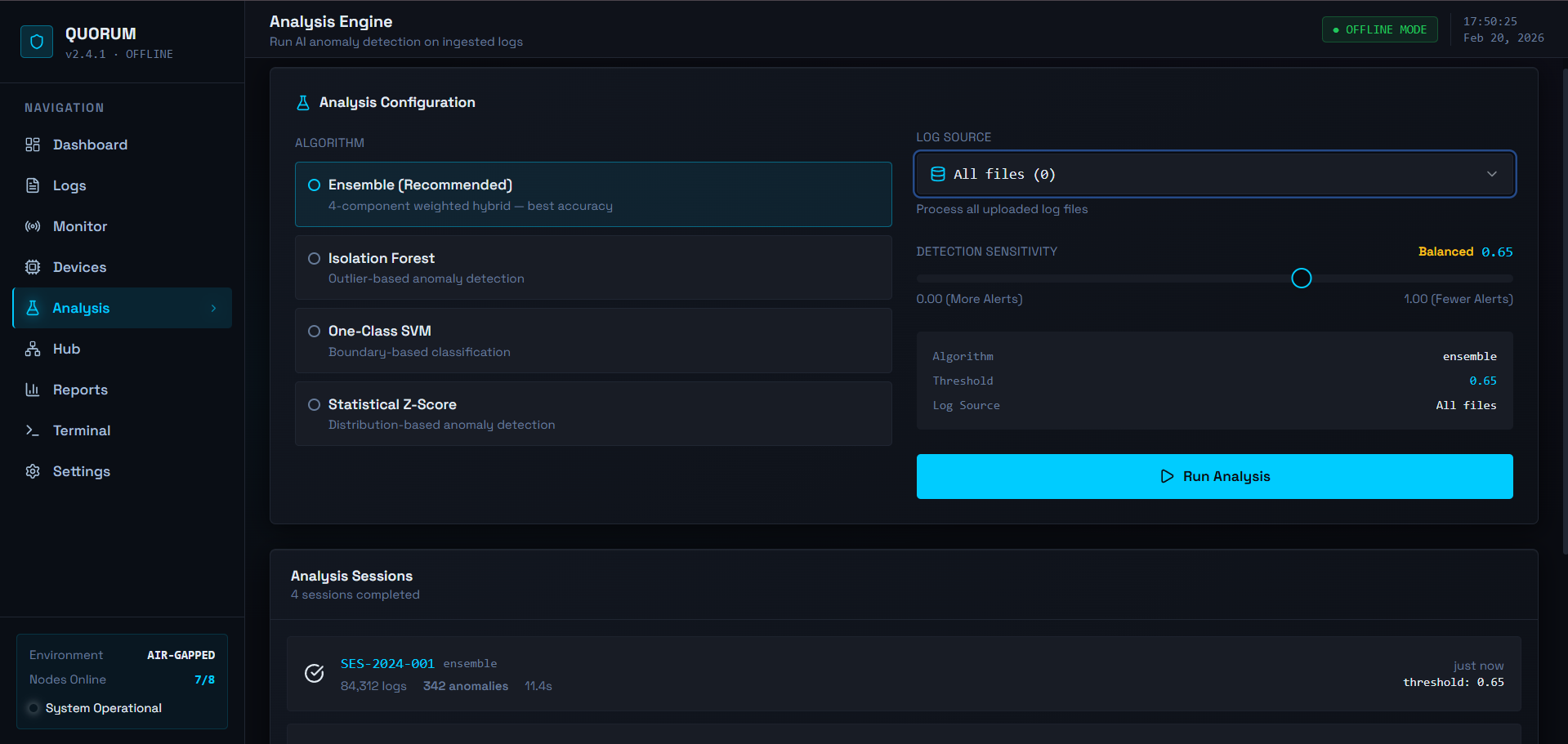
**5. Analysis Page**

The Analysis module processes uploaded logs using machine learning algorithms to detect anomalies and potential threats. It correlates events across timestamps and sources, assigns severity scores, and maps findings to known attack patterns. Users can analyze all logs, the latest upload, or selected files through an interactive interface. The page provides detailed insights to support forensic investigation and threat hunting.

**Key Components**

* AI-based anomaly detection and scoring
* File selection dropdown (All / Latest / Specific)
* Event correlation and timeline visualization
* MITRE-style threat classification and insights

**Screenshot**

****

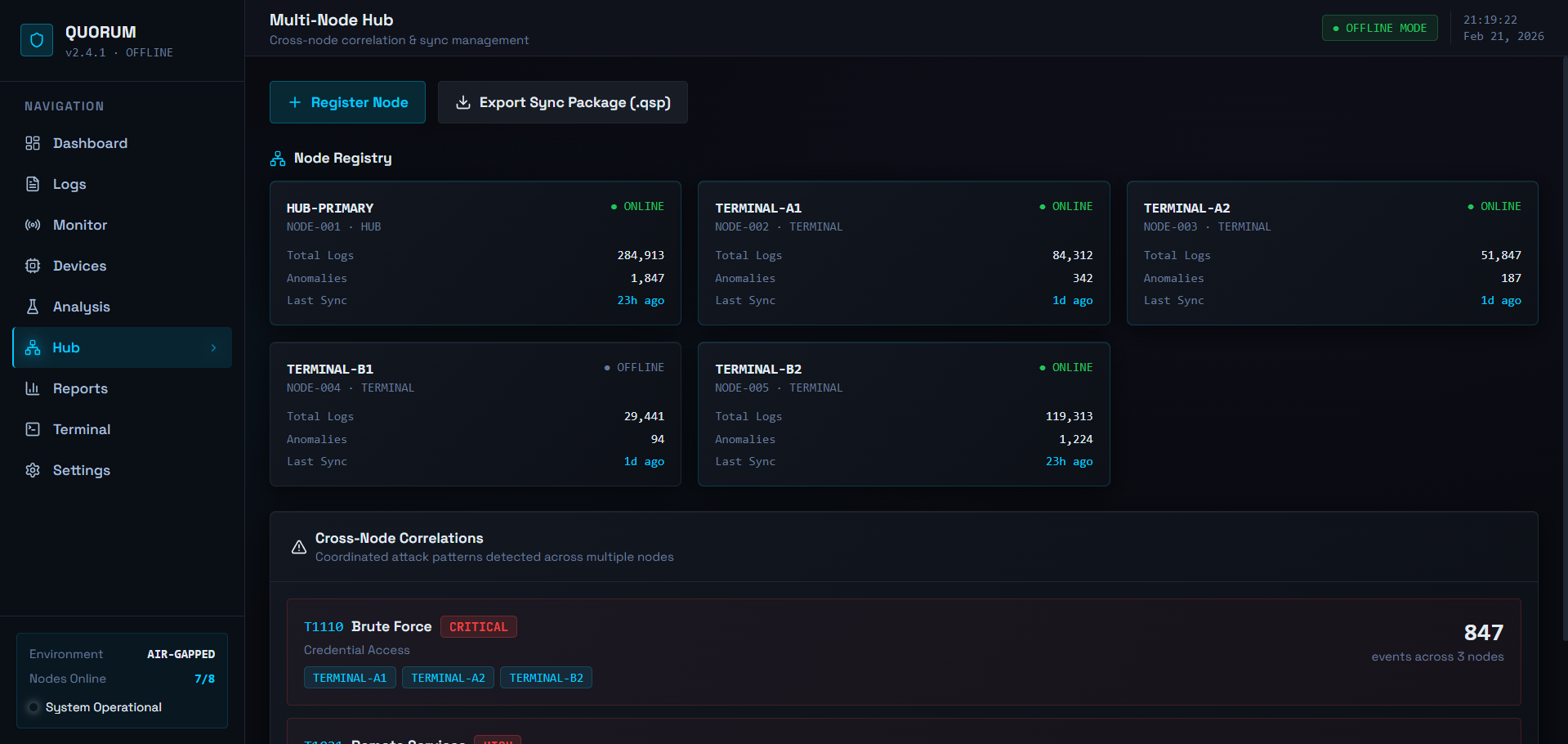
## 6. Multi-Node Hub Page

## The Multi-Node Hub enables correlation and synchronization across multiple QUORUM deployments. It aggregates threat intelligence from different nodes to identify coordinated attacks and shared indicators of compromise.

**Key Components**

* Cross-node threat correlation engine
* Secure offline synchronization mechanism
* Node status and health monitoring
* Aggregated intelligence visualization

**Screenshot**



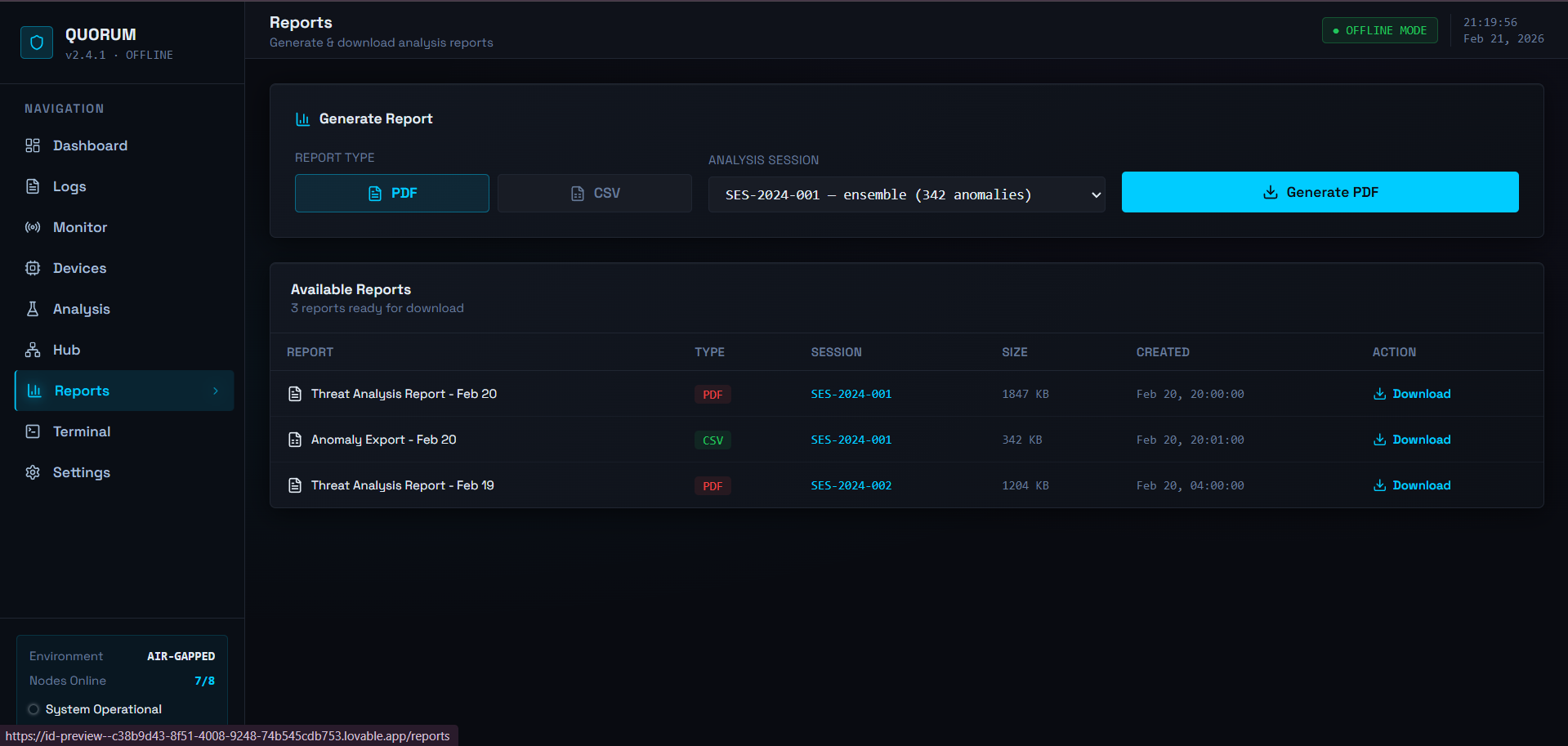
**7. Reports Page**

The Reporting module generates structured security reports from analysis results for auditing and decision-making. Reports include detected anomalies, severity classifications, timelines, and recommended actions. Outputs can be exported in multiple formats for documentation or compliance requirements. The system ensures reports can be generated entirely offline without external dependencies.

**Key Components**

* Automated PDF and CSV report generation
* Threat summary and analytics visualization
* Export and download functionality
* Historical report management

**Screenshot**



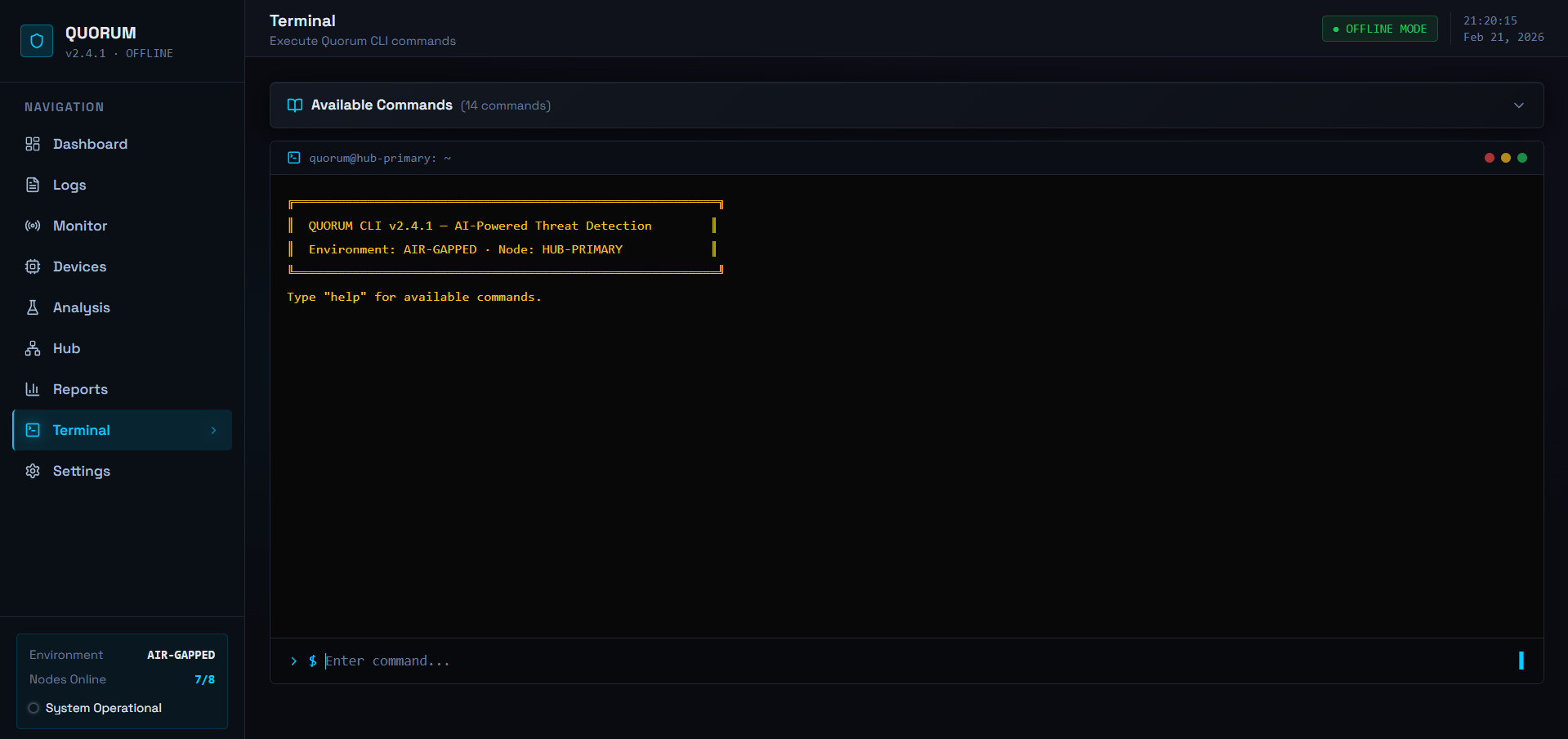
**8. Terminal Page**

The Terminal module provides a secure command-line interface within the QUORUM platform for advanced system interaction and administrative operations. It enables analysts to execute diagnostic commands, monitor processes, and interact with the backend environment without leaving the application. This feature is particularly useful in air-gapped deployments where direct system access must remain controlled and auditable. The terminal also supports operational debugging, log inspection, and maintenance tasks.

**Key Components**

* Embedded secure command-line interface
* System diagnostics and backend interaction tools
* Command execution logging and audit trail
* Access-controlled administrative operations

**Screenshot**



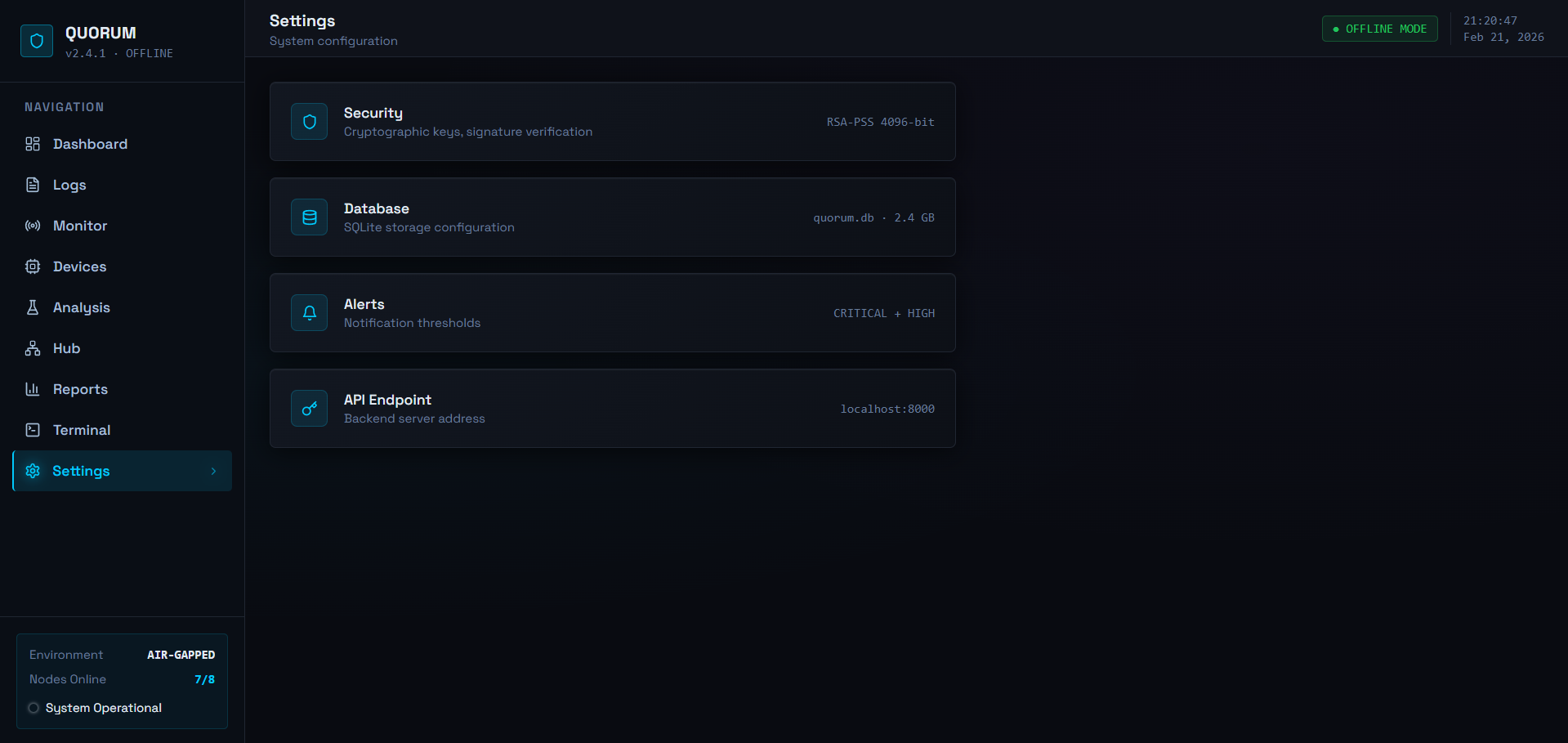
**9. Settings / Configuration Page**

The Settings module allows administrators to configure system parameters, AI models, thresholds, and node connections. It provides control over analysis sensitivity, storage locations, and synchronization behavior. Security configurations ensure the platform remains compliant with air-gapped deployment requirements. The interface supports maintainability and operational customization.

**Key Components**

* AI model and threshold configuration
* Storage and database management options
* Node connection and sync settings
* System security and access controls

**Screenshot**



**7. CONCLUSION**

**7.1 Conclusion**

Project Quorum successfully addresses a critical gap in cybersecurity by delivering an AI-powered forensic analysis platform designed specifically for air-gapped and fully offline environments. The project demonstrates that enterprise-grade threat detection can be implemented without compromising network isolation, which is essential for protecting sensitive and critical infrastructure systems.

All primary objectives were achieved during development. A fully self-contained, portable application was created as a single executable capable of running on Windows, macOS, and Linux without installation or external dependencies. The integration of DuckDB as an embedded analytical database enabled high-performance querying of large log datasets on standard hardware, while the parsing engine effectively supports both Windows EVTX and Linux Syslog formats, converting raw logs into structured data for analysis.

The embedded AI engine, built using PyOD and TensorFlow Lite, demonstrated reliable anomaly detection across benchmark datasets such as LogPai, HDFS, and BGL. By combining multiple detection algorithms including Isolation Forest, K-Nearest Neighbours, and Local Outlier Factor-the system provides flexibility and improved detection accuracy for diverse threat scenarios. Integration with the MITRE ATT&CK framework further enhances the platform by automatically mapping anomalies to known adversary tactics and techniques, enabling faster investigation and improved situational awareness for analysts.

The Secure Offline Update Protocol (SOUP) resolves the challenge of maintaining tools in isolated environments by using cryptographic verification to securely update AI models and detection rules while preventing tampered packages from being installed. Additionally, the React-based dashboard offers an intuitive interface with visualization, SQL querying, and report generation capabilities, making advanced forensic analysis accessible without requiring deep technical expertise.

Overall, Project Quorum demonstrates the feasibility of deploying modern AI-driven cybersecurity solutions entirely offline using technologies such as DuckDB, FastAPI, PyOD, TensorFlow Lite, React, and Tauri. The platform has strong potential applications across government, defense, critical infrastructure, and financial sectors, contributing to improved cybersecurity resilience in high-security environments.

**7.2 Limitations of the System**

While Project Quorum successfully meets its core objectives, several limitations should be acknowledged:

* **Limited Real-Time Capabilities:** The system is designed for offline, historical log analysis rather than real-time monitoring. It cannot provide live threat detection or continuous monitoring, which may be necessary for immediate incident response in some scenarios.
* **Manual Investigation Required:** While the system flags anomalies and maps them to MITRE ATT&CK techniques, it cannot automatically determine whether detected anomalies represent genuine threats or false positives. Human analyst expertise is still required for final classification and decision-making.
* **Resource Constraints:** Performance on very large datasets (100+ GB) may be limited by available RAM and CPU power. While the system is optimized for standard workstations, forensic analysis of extremely large enterprise log repositories may require high-end hardware.
* **Update Distribution Challenges:** While SOUP provides a secure update mechanism, the actual distribution of update packages in air-gapped environments still relies on manual processes (USB drives, physical media transfer), which can be time-consuming and logistically complex.
* **Limited Threat Intelligence:** The system operates entirely offline and cannot access external threat intelligence feeds, vulnerability databases, or indicators of compromise (IOCs) from the broader security community, potentially limiting contextual awareness.
* **Single-User Design:** The current implementation is designed for individual analysts and does not support multi-user collaboration, shared databases, or team-based investigation workflows that might be needed in larger security operations centres.
* **No Automated Response:** The system is purely analytical and does not include any automated remediation or response capabilities. It identifies threats but cannot act to block, contain, or mitigate them.

**7.3 Future Scope of the Project**

Project Quorum provides a solid foundation that can be extended and enhanced in numerous directions:

* **Enhanced Log Format Support:** Future versions could include parsers for additional log formats including Apache/Nginx web server logs, database audit logs, firewall logs, IDS/IPS logs, cloud service logs (for hybrid environments), and application-specific formats. A plugin architecture could allow users to develop custom parsers for proprietary log formats.
* **Deep Learning Integration:** Advanced neural network architectures like LSTMs (Long Short-Term Memory) or Transformers could be explored for sequence-based anomaly detection, potentially identifying complex, multi-stage attacks that unfold over time and appear normal in isolated log entries.
* **Behavioral Baselining:** Implementing automatic behavioral baseline creation for specific systems, users, or applications would improve anomaly detection accuracy by comparing activities against established normal patterns rather than purely statistical outliers.
* **Correlation Engine:** A correlation engine could identify relationships between anomalies across different systems, time periods, or log sources, helping analysts discover coordinated attacks or attack chains that aren't obvious when viewing individual anomalies in isolation.
* **Graph Database Integration:** Adding graph database capabilities could enable relationship mapping between entities (users, systems, files, network connections), making it easier to visualize attack paths and lateral movement patterns.
* **Automated Report Generation:** More sophisticated reporting features could include automated executive summaries, risk scoring, trend analysis over time, and customizable report templates for different stakeholders (technical teams, management, compliance).
* **Integration with Forensic Tools:** Building bridges to other forensic tools like Volatility (memory forensics), Autopsy (disk forensics), or Wireshark (network forensics) would enable more comprehensive incident investigations by correlating log analysis with other evidence sources.
* **Enhanced Visualization:** Additional visualization capabilities such as attack timeline reconstruction, network topology mapping, geolocation of attack sources, and 3D relationship graphs could improve analyst understanding of complex incidents.
* **Custom Rule Creation:** A visual rule builder would allow analysts to create custom detection rules based on their specific environment and threat model without requiring programming skills, supplementing the automated ML-based detection.

**REFERENCE**

1. LogHub Repository. "Public Log Datasets for AI Research." <https://github.com/logpai/loghub>
2. Du, M., Li, F., Zheng, G., & Srikumar, V. (2017). DeepLog: Anomaly detection and diagnosis from system logs through deep learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (pp. 1285-1298). <https://doi.org/10.1145/3133956.3134015>
3. He, P., Zhu, J., Zheng, Z., & Lyu, M. R. (2017). Drain: An online log parsing approach with fixed depth tree. In 2017 IEEE International Conference on Web Services (ICWS) (pp. 33-40). IEEE. <https://doi.org/10.1109/ICWS.2017.13>
4. Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. ACM Computing Surveys, 41(3), 1-58. <https://doi.org/10.1145/1541880.1541882>
5. Zhao, Y., Nasrullah, Z., & Li, Z. (2019). PyOD: A Python toolbox for scalable outlier detection. Journal of Machine Learning Research, 20(96), 1-7. <http://jmlr.org/papers/v20/19-011.html>
6. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825-2830. <http://www.jmlr.org/papers/v12/pedregosa11a.html>
7. Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008). Isolation forest. In 2008 Eighth IEEE International Conference on Data Mining (pp. 413-422). IEEE. <https://doi.org/10.1109/ICDM.2008.17>
8. Strom, B. E., Applebaum, A., Miller, D. P., Nickels, K. C., Pennington, A. G., & Thomas, C. B. (2018). MITRE ATT&CK: Design and philosophy. Technical Report, The MITRE Corporation. <https://attack.mitre.org/>
9. Scarfone, K., & Mell, P. (2007). Guide to intrusion detection and prevention systems (IDPS). NIST Special Publication 800-94. <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-94.pdf>
10. Ramírez, S. (2024). FastAPI: High performance Python web framework. <https://fastapi.tiangolo.com/>
11. Meta Platforms, Inc. (2024). React: A JavaScript library for building user interfaces. <https://react.dev/>
12. DuckDB Labs. (2024). DuckDB: An embeddable analytical database. <https://duckdb.org/>
13. Tauri Contributors. (2024). Tauri: Build smaller, faster, and more secure desktop applications. <https://tauri.app/>
14. Cryptography Developers. (2024). Cryptography: Python library for cryptographic primitives and recipes. <https://cryptography.io/>
15. Martin, R. C. (2008). Clean code: A handbook of agile software craftsmanship. Prentice Hall.
16. Kent, K., & Souppaya, M. (2006). Guide to computer security log management. NIST Special Publication 800-92. <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-92.pdf>
17. Liu, F. T., Ting, K. M., & Zhou, Z. H. (2008). "Isolation Forest." \*IEEE International Conference on Data Mining\*.
18. He, P., Zhu, J., Zheng, Z., & Lyu, M. R. (2017). "Drain: An Online Log Parsing Approach with Fixed Depth Tree." \*ICWS 2017\*.