RV COLLEGE OF ENGINEERING®, BENGALURU-560059

(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

TITLE OF THE PROJECT
AnomalyX: An Adaptive Threat Detection System

Mini - Project Report

Submitted by

AKSHAT GUPTA **USN 1RV23CS027**
AMOL VYAS **USN 1RV23CS032**

in partial fulfillment for the requirement of 5th Semester
DATABASE MANAGEMENT SYSTEMS (CD252IA)

Under the Guidance of
Shweta Babu P., Professor,
CSE, RVCE

Academic Year 2025 - 2026

RV COLLEGE OF ENGINEERING®, BENGALURU 560059
(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the project work titled **'AnomalyX: An Adaptive Threat Detection System'** is carried out by Akshat Gupta (1RV23CS027) and Amol Vyas (1RV23CS032), who are bonafide students of R. V. College of Engineering, Bengaluru, in partial fulfillment of the curriculum requirement of 5th Semester **Database Management Systems(CD252IA)** Laboratory Mini Project during the academic year **2024-2025**. It is certified that all corrections/suggestions indicated for the internal Assessment have been incorporated in the report. The report has been approved as it satisfies the academic requirements in all respect laboratory mini-project work prescribed by the institution.

Signature of Faculty In-charge

Head of the Department
Dept. of CSE, RVCE

External Examination

Name of Examiners	Signature with date
1	
2	

Acknowledgement

We are indebted to our guide and faculty, **Dr. Shweta Babu P., Professor, Dept of CSE, RV College of Engineering**, for her wholehearted support, suggestions, valuable comments and invaluable advice throughout our project work and helped in the preparation of this project.

Our sincere thanks to **Dr. Shanta Rangaswamy,** Professor and Head, Department of Computer Science and Engineering, RVCE for her support and encouragement.

We express sincere gratitude to our beloved Principal, **Dr. K. N. Subramanya** for his appreciation towards this project work.

We thank all the **teaching staff and technical staff** of the Computer Science and Engineering department, RVCE for their help.

Lastly, we take this opportunity to thank our **family members and friends** who provided all the backup support throughout the project work

Abstract

With the rapid growth of digital infrastructure and networked systems, monitoring large volumes of network traffic and identifying abnormal behavior has become increasingly challenging. Traditional rule-based and signature-driven intrusion detection systems are limited in their ability to detect previously unseen attacks, evolving traffic patterns, and subtle anomalies in real time. This project presents **AnomalyX**, a scalable and adaptive network anomaly detection system designed to identify suspicious network behavior using modern time-series forecasting and statistical analysis techniques.

AnomalyX employs a microservice-based architecture where network metrics are collected from a dedicated metrics service and processed asynchronously by a FastAPI inference engine. The system integrates the TimeGPT forecasting model to predict expected network behavior and dynamically compares observed values against probabilistic confidence intervals to detect anomalies. Instead of relying on heavy supervised model training, the platform uses lightweight anomaly scoring and adaptive thresholding, enabling faster deployment, lower computational overhead, and easier maintenance.

The system is evaluated using the UNSW-NB15 dataset to simulate realistic network traffic conditions. Experimental observations demonstrate that the framework is capable of detecting sudden traffic spikes, abnormal flow patterns, and persistent deviations while maintaining low latency and stable performance. The modular design allows seamless scalability and integration with existing monitoring pipelines. Overall, AnomalyX provides a practical and efficient solution for real-time network anomaly detection suitable for academic experimentation and real-world deployment.

Table of Contents

List of Figures

CHAPTER 1:

Introduction

Modern network infrastructures generate massive volumes of traffic data that must be continuously monitored to ensure security, reliability, and optimal performance. Traditional intrusion detection systems and monitoring tools rely heavily on predefined rules and static thresholds, which are often ineffective against emerging threats, dynamic traffic patterns, and zero-day attacks. Additionally, handling large-scale network data requires efficient storage, structured querying, and scalable database systems capable of managing high-velocity time-series records.

The **AnomalyX** project addresses these challenges by designing an intelligent network anomaly detection system supported by a robust database architecture. The system captures network flow information, protocol details, traffic metrics, and attack labels using a relational database implemented with PostgreSQL and Prisma ORM. This structured data model enables efficient storage, retrieval, and analysis of network behavior over time. In parallel, a NoSQL database is used to manage user account information, ensuring flexibility, scalability, and separation of authentication concerns from analytical data. The integration of relational and NoSQL databases provides both consistency and scalability, making the system suitable for real-world deployment scenarios.

AnomalyX leverages time-series forecasting to predict expected network behavior and dynamically identify abnormal deviations. The system processes network metrics asynchronously using a microservice-based architecture and provides near real-time anomaly detection while maintaining low computational overhead and operational simplicity.

The key innovation in AnomalyX lies in the integration of **transformer-based time-series forecasting with lightweight statistical anomaly detection and a hybrid database architecture**. Instead of training heavy machine learning models, the system utilizes a pre-trained forecasting engine (TimeGPT) to generate probabilistic predictions of network traffic. Anomalies are detected by evaluating deviations against confidence intervals and adaptive thresholds, enabling rapid deployment without extensive training or labeling requirements. Furthermore, the use of Prisma ORM ensures schema consistency and type safety, while the inclusion of a NoSQL database for user management improves scalability and flexibility. The combination of asynchronous microservices, modern forecasting intelligence, and hybrid data storage makes AnomalyX efficient, modular, and easy to maintain.

A custom MERN/TypeScript dashboard with React and TypeScript provides real-time visualization, featuring interactive Recharts-powered charts, color-coded threat indicators, and dynamic filtering. The responsive interface enables security analysts to explore incidents and support continuous machine learning model refinement. AnomalyX project addresses significant cybersecurity concerns by enhancing network security against increasingly sophisticated cyber threats that endanger personal data and critical infrastructure

1.1. Objectives:

- To design and implement a scalable database system for storing network flow records, protocol information, traffic metrics, and anomaly labels using a relational database.
- To integrate a NoSQL database for flexible and scalable management of user account information and authentication data.
- To develop a real-time data ingestion pipeline capable of handling continuous network traffic metrics with low latency.
- To apply time-series forecasting techniques to predict normal network behavior and identify abnormal deviations dynamically.
- To minimize computational complexity by avoiding heavy supervised model training and relying on lightweight statistical anomaly scoring.
- To ensure data consistency, integrity, and efficient querying through normalized schema design and ORM-based data management.

1.2. Scope:

The scope of the AnomalyX project includes the design and implementation of a hybrid database system combining relational and NoSQL storage, development of data ingestion and persistence pipelines, integration of a forecasting-based anomaly detection engine, and evaluation using realistic network datasets such as UNSW-NB15. The system focuses on detecting abnormal traffic patterns in near real time and maintaining historical auditability of network flows and anomalies. The project does not aim to replace enterprise-grade intrusion prevention systems or perform deep packet inspection but instead serves as an intelligent monitoring and analytics platform suitable for academic research, prototype deployments, and scalable extensions in production environments.

CHAPTER 2:

Software Requirement Specification

The Software Requirement Specification (SRS) defines the functional and non-functional requirements of the AnomalyX system. It outlines the software and hardware dependencies, system capabilities, operational constraints, and quality attributes required to ensure reliable deployment and performance. The goal of this specification is to provide a clear blueprint for development, testing, deployment, and maintenance of the anomaly detection platform.

2.1 Software Requirements:

The AnomalyX system is developed using modern open-source technologies to ensure scalability, maintainability, and cross-platform compatibility. The core backend services are implemented using Python and Node.js. The inference service utilizes the FastAPI framework for building asynchronous REST APIs, while the metrics service uses Express.js to expose network data endpoints. These frameworks enable high-performance, non-blocking communication between microservices.

The forecasting engine integrates the TimeGPT model via the Nixtla client library. This allows the system to perform advanced time-series forecasting without requiring local model training or specialized hardware. Supporting libraries such as NumPy and Pandas are used for data preprocessing, normalization, and numerical computations. HTTP communication between services is handled using asynchronous clients such as HTTPX to ensure low latency and efficient resource utilization.

The relational database is implemented using PostgreSQL with Prisma ORM to provide schema enforcement, migration support, type safety, and reliable transaction management. Prisma enables seamless interaction between the application and the database while maintaining data integrity. A NoSQL database (such as MongoDB) is used to store user account information, enabling flexible schema evolution and horizontal scalability for authentication-related data.

2.2 Hardware Requirements

The hardware requirements for AnomalyX are modest due to the system's lightweight architecture and avoidance of heavy machine learning training workloads. A standard server or virtual machine is sufficient for development and deployment. The recommended minimum configuration includes a multi-core processor (Intel i5 or equivalent), 8 GB RAM, and at least 50 GB of available storage for logs, datasets, and database files. Solid-state storage is preferred to improve database performance and reduce I/O latency.

The system can run on cloud platforms, on-premise servers, or local development machines. For higher throughput scenarios, additional CPU cores and memory can be provisioned to support concurrent API requests and larger data volumes. Network connectivity should be stable with sufficient bandwidth to support continuous data ingestion and external API communication with the forecasting service.

No specialized hardware such as GPUs or TPUs is required since the forecasting engine operates externally through API calls. This significantly reduces infrastructure cost and simplifies maintenance. Backup storage and periodic snapshots are recommended for data reliability and recovery. The system supports horizontal scaling by deploying multiple instances behind a load balancer if required.

2.3 Functional Requirements

The AnomalyX system is organized into modular components, each responsible for specific functional responsibilities.

2.3.1 Metrics Collection Module

- Collects network flow metrics from upstream sources or simulated datasets.
- Exposes REST endpoints for querying historical and real-time metrics.
- Supports filtering by time range, host identifiers, protocol type, and service.
- Ensures data validation and schema consistency before forwarding to downstream services.
- Handles concurrent client requests efficiently.

2.3.2 Data Ingestion and Windowing Module

- Retrieves metrics asynchronously from the metrics service.
- Constructs sliding windows of historical observations for time-series analysis.
- Normalizes and preprocesses raw data to remove noise and missing values.
- Maintains rolling buffers to preserve temporal continuity.

2.3.3 Forecasting Module

- Integrates the TimeGPT forecasting engine through Nixtla client APIs.
- Generates probabilistic forecasts including confidence intervals.
- Supports configurable forecast horizons and confidence levels.
- Handles API failures, retries, and timeout management gracefully.

2.3.4 Anomaly Detection Module

- Computes deviation scores by comparing observed values against forecast bounds.
- Maintains adaptive thresholds using rolling statistical measures.
- Classifies anomalies into severity levels based on deviation magnitude and persistence.
- Reduces false positives through dynamic sensitivity tuning.

2.3.5 Persistence and Database Module

- Stores network flows, metrics, forecasts, and anomaly records in PostgreSQL.
- Maintains referential integrity using Prisma ORM.
- Supports indexed queries for fast retrieval and audit trails.
- Stores user accounts and authentication data in a NoSQL database.

2.3.6 API and Integration Module

- Exposes REST APIs for retrieving anomaly results and historical analytics.
- Supports authentication and authorization mechanisms.
- Enables integration with dashboards, monitoring tools, and external services.

2.4 Non-Functional Requirements

- **Performance:** The system shall support near real-time anomaly detection with low latency and high throughput.
- **Scalability:** The architecture shall support horizontal scaling of services.
- **Reliability:** The system shall tolerate transient failures and recover gracefully.
- **Security:** Data access shall be authenticated and sensitive information encrypted.
- **Maintainability:** Modular design shall enable independent updates and easy debugging.
- **Portability:** The system shall run on Linux and cloud environments without modification.
- **Usability:** APIs shall be well-documented and easy to integrate.
- **Availability:** The system shall maintain high uptime and operational stability.
- **Data Integrity:** All persisted records shall maintain consistency and referential integrity.
- **Extensibility:** New features and analytics modules shall be easy to integrate.

CHAPTER 3

Entity Relationship Diagram (ERD)

The following figure 3.1 illustrates the design of a database system for network traffic management. This system is intended to efficiently handle information regarding servers, clients, connections, firewalls, routes, and attacks, along with the relationships that bind these entities. By implementing a structured database, network administrators can optimize operations, enhance security measures, and gain insights into network behavior. This figure details the entities within the system, their attributes, and the significant relationships that connect them. These relationships define the interactions and dependencies among various components, ensuring a cohesive and effective information management system for network traffic.

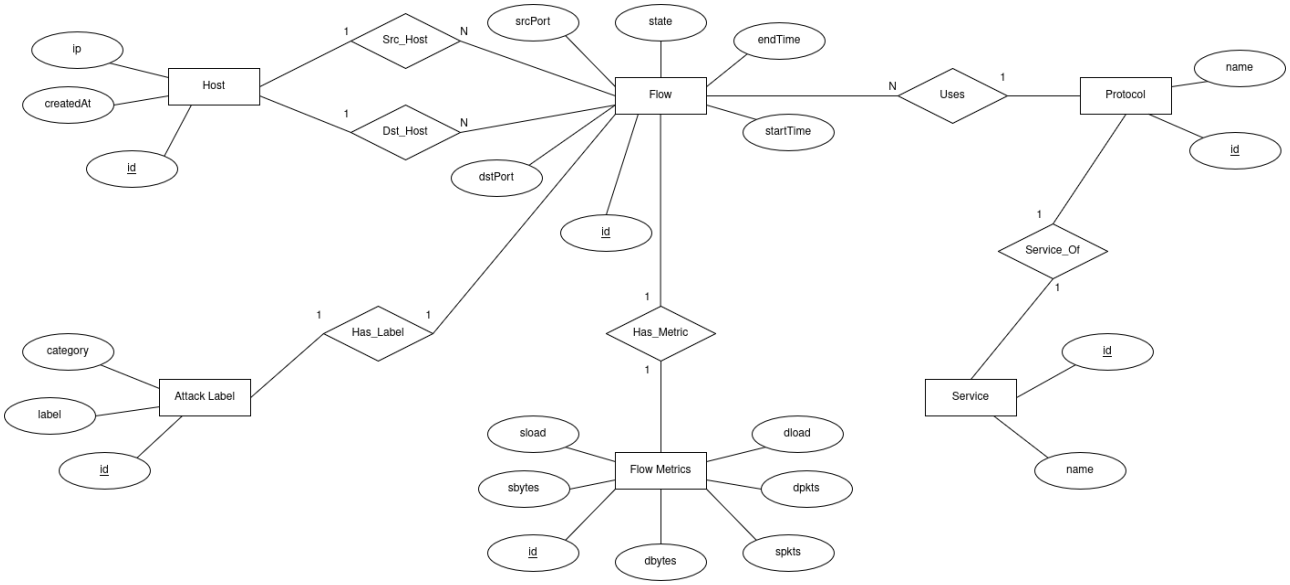


Fig 3.1. Entity Relationship Diagram for AnomalyX

3.1 Entity And their Attributes:

1. Host

The **Host** entity represents a network endpoint participating in communication flows. A host can act as a source or destination in a network connection.

- **id:** Unique identifier assigned to each host.
- **ip:** IP address of the host, used to uniquely identify the device on the network.
- **createdAt:** Timestamp indicating when the host record was created in the system.

1. Flow

The **Flow** entity represents a single network communication session between two hosts. It stores connection-level metadata and timing information.

- **id:** Unique identifier for each network flow.
- **srcPort:** Source port number from which the connection originated.
- **dstPort:** Destination port number to which the connection was established.
- **state:** Current state of the connection (e.g., active, closed, reset).
- **startTime:** Timestamp representing when the flow started.
- **endTime:** Timestamp representing when the flow ended.

2. **Protocol**

The **Protocol** entity stores information about the network protocol used in a flow.

- **id:** Unique identifier for each protocol.
- **name:** Name of the protocol (e.g., TCP, UDP, ICMP).

3. **Service**

The **Service** entity represents the application-level service associated with a network flow.

- **id:** Unique identifier for each service.
- **name:** Name of the service (e.g., HTTP, DNS, FTP, SSH).

4. **Flow Metrics**

The **Flow Metrics** entity stores quantitative measurements related to a network flow. These metrics are used for traffic analysis and anomaly detection.

- **id:** Unique identifier for each metrics record.
- **sbytes:** Total number of bytes sent from the source host.
- **dbytes:** Total number of bytes received by the destination host.
- **spkts:** Total number of packets sent from the source host.
- **dpkts:** Total number of packets received by the destination host.
- **sload:** Data transmission load from the source host.
- **dload:** Data transmission load to the destination host.

5. **Attack Label**

The **Attack Label** entity stores classification information related to a network flow. It is used for labeling traffic during dataset evaluation and model validation.

- **id:** Unique identifier for each label record.
- **category:** Category of the attack (e.g., DoS, Exploit, Reconnaissance).
- **label:** Binary or numeric value indicating whether the flow is normal or malicious.

3.2 Relationships and Their Descriptions

1. SRC_HOST

The **SRC_HOST** relationship connects the **Host** entity with the **Flow** entity to represent the source of a network connection. Each flow originates from exactly one source host, while a single host can initiate multiple flows over time. This relationship allows the system to track outgoing traffic patterns, identify source-based anomalies, and analyze communication behavior at the host level.

Cardinality: One Host → Many Flows (1 : N)

2. DST_HOST

The **DST_HOST** relationship links the **Host** entity with the **Flow** entity to represent the destination of a network connection. Each flow terminates at exactly one destination host, while a host can receive traffic from multiple flows. This relationship enables the system to analyze incoming traffic behavior and identify abnormal access patterns.

Cardinality: One Host → Many Flows (1 : N)

3. USES

The **USES** relationship associates the **Flow** entity with the **Protocol** entity. Each network flow uses exactly one protocol, whereas a single protocol can be used by many flows. This relationship supports protocol-wise traffic analysis, filtering, and performance evaluation.

Cardinality: One Protocol → Many Flows (1 : N)

4. SERVICE_OF

The **SERVICE_OF** relationship links the **Flow** entity with the **Service** entity to indicate the application-level service associated with a flow. A flow may optionally belong to a service, while a service can be associated with many flows. This relationship allows application-level monitoring and service-specific anomaly detection.

Cardinality: One Service → Many Flows (0..1 : N)

5. HAS_METRIC

The **HAS_METRIC** relationship connects the **Flow** entity with the **Flow Metrics** entity. Each flow has exactly one corresponding metrics record containing quantitative traffic measurements. This relationship enables detailed performance analysis and anomaly computation based on traffic statistics.

Cardinality: One Flow → One Flow Metrics (1 : 1)

6. HAS_LABEL

The **HAS_LABEL** relationship associates the **Flow** entity with the **Attack Label** entity. A flow may optionally have a label indicating whether it represents normal or malicious traffic. This relationship supports supervised evaluation and dataset validation.

Cardinality: One Flow → Zero or One Attack Label (1 : 0..1)

CHAPTER 4

Data Flow Diagram

This section delineates the logical flow of information within the **AnomalyX** ecosystem, abstracting away implementation details to focus purely on data transformation and movement. The Data Flow Diagrams (DFD) presented herein serve as the architectural blueprint for the system's information pipelines, utilizing standard notation to represent external entities, processes, data stores, and data vectors.

The architecture adopts a **pipe-and-filter** design pattern, where network telemetry acts as the primary input stream, undergoing successive transformations—normalization, windowing, and statistical scoring—before persistence and visualization.

4.1 Level 0 DFD (Context Diagram)

The Level 0 DFD, or Context Diagram, defines the **system boundary** and identifying the external interfaces that interact with the AnomalyX environment. At this abstraction level, the entire anomaly detection platform is encapsulated as a single "Black Box" process, labeled **0.0 AnomalyX System**.

Key Components & Interaction Logic:

- **External Entity: Metrics Source**
    - **Role:** The originator of raw network telemetry. In the current implementation, this represents the simulation engine or live network tap feeding flow metrics (e.g., source/dest IP, bytes, packet counts) into the system.
    - **Data Vector:** Network Flow Metrics (Ingress).
- **External Entity: Forecasting Service (TimeGPT)**
    - **Role:** An external SaaS dependency providing "Forecasting-as-a-Service." The system offloads the complex time-series regression tasks to this entity to minimize local computational overhead.
    - **Data Vector (Egress):** Time-Series Window & Forecast Request.
    - **Data Vector (Ingress):** Forecast & Confidence Interval.
- **External Entity: User**
    - **Role:** The consumer of the actionable intelligence generated by the system. This entity interacts with the dashboard for monitoring and historical analysis.
        - **Data Vector:** Anomaly Reports & Analytics (Egress).

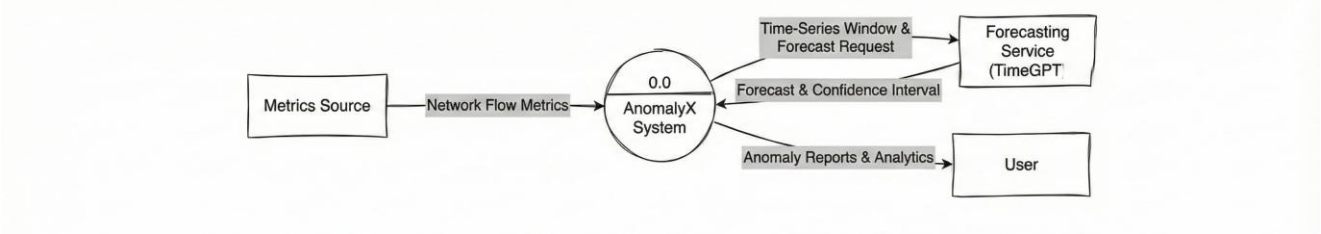


Fig. 4.1 DFD - Level 0

4.2 Level 1 DFD (Functional Decomposition)

The Level 1 DFD explodes the single "0.0" process into its constituent functional subprocesses, revealing the internal data processing logic and storage interactions. This level exposes the microservice architecture and the asynchronous handshakes required for the anomaly detection pipeline.

- **Process Decomposition:**
- **1.0 Collect Metrics:**
    - Acts as the ingress gateway (Metrics Service). It accepts raw telemetry, validating the payload structure before passing it downstream. This process effectively decouples data ingestion from analysis.
- **2.0 Preprocess & Windowing:**
    - Responsible for feature engineering. It normalizes raw metric values and aggregates them into fixed-size sliding windows (time-series formatting) required by the forecasting engine.
- **3.0 Forecast Generation:**
    - The interface layer for the external inference engine. It serializes the windowed data, handles the API handshake with the Forecasting Service, and deserializes the returned predicted values and confidence bounds.
- **4.0 Anomaly Detection:**
    - The core decision engine. It compares the *Actual* values (from 2.0) against the *Predicted* values (from 3.0).
    - **Logic:** Computes the deviation score and applies the adaptive thresholding algorithm ($\tau=\mu+\lambda\times\sigma$). If the deviation exceeds τ, the flow is flagged as anomalous.
- **5.0 Data Persistence:**
    - Handles the transactional integrity of the system. It routes structured data to the appropriate storage solutions.
    - **D1 Flow Database (PostgreSQL):** Stores high-fidelity flow records, metrics, and anomaly flags for audit trails.

- **D2 User Store (NoSQL):** Stores user authentication profiles and session data.

- **6.0 Report Generation:**
  - The query layer serving the frontend client. It aggregates data from **D1** and **D2** to produce visualizable analytics, historical trend reports, and real-time alerts for the **User**.

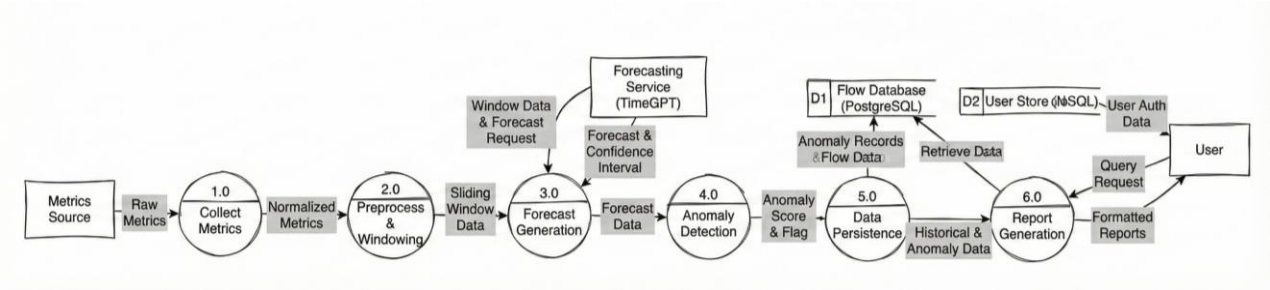


Fig 4.2 DFD – Level 1

CHAPTER 5

Relational Schema and Normalization

This relational schema (Fig 5.1) outlines the structure of the database designed for the **AnomalyX** system. It utilizes a modular, microservices-oriented architecture to represent network entities, traffic flows, and statistical metrics. The design prioritizes write-heavy performance for time-series ingestion while maintaining strict referential integrity for historical analysis.

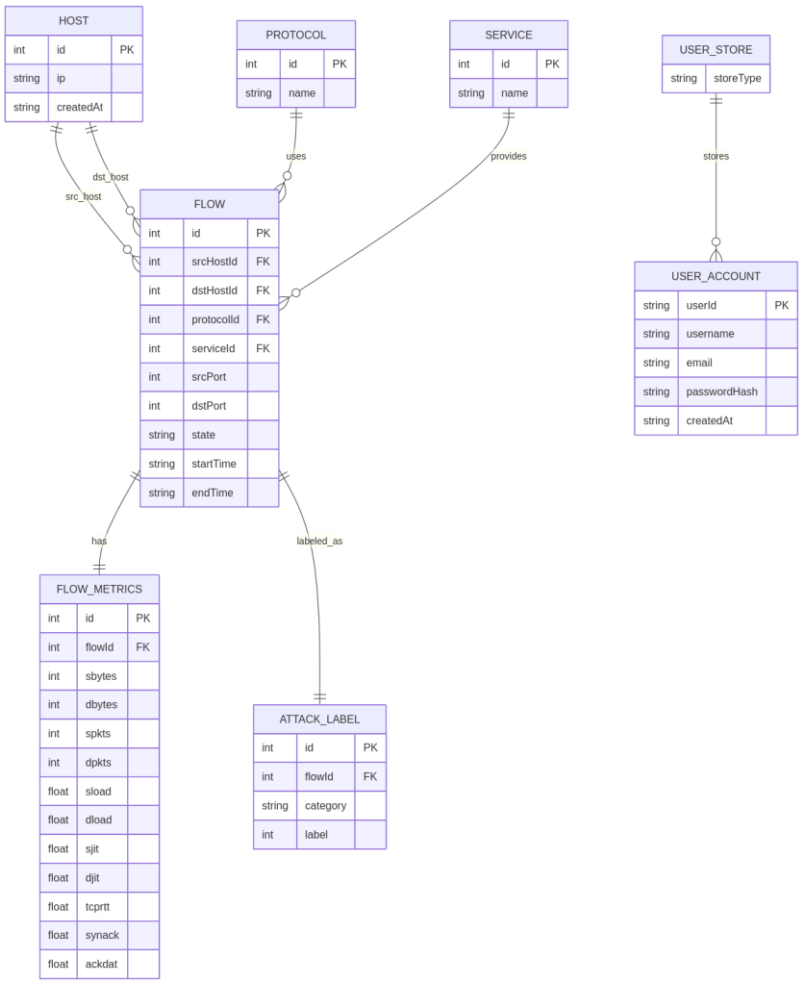


Fig. 5.1 Relational Schema

5.1 Tables

1. Host (id, ip, createdAt): Stores information about unique network endpoints (nodes) identified within the monitored ecosystem.

- This table serves as the central registry for all source and destination entities. By normalizing IP addresses into a dedicated table, the system avoids the redundancy of storing string-based IP representations in millions of flow records.

- **Primary Key (PK):** id (Surrogate key ensuring unique identification).
- **Attributes:** ip (Unique IPv4/IPv6 address), createdAt (Timestamp of first discovery).

2. Protocol (id, name): Acts as a lookup dictionary for standard network protocols (e.g., TCP, UDP, ICMP).

- This table eliminates the need to store protocol names as repeated strings in the main flow table, significantly reducing storage overhead.
- **Primary Key (PK):** id.
- **Attributes:** name (Standard protocol designation).

3. Service (id, name): Classifies the specific application layer service associated with a protocol (e.g., HTTP, SSH, FTP).

- It aids in distinguishing traffic types for the anomaly detection engine, as different services exhibit different statistical baselines.
- **Primary Key (PK):** id.
- **Foreign Key (FK):** Linked hierarchically to Protocol in specific normalization views to enforce valid protocol-service mappings.

4. Flow (id, srcHostId, dstHostId, protocolId, srcPort, dstPort, state, startTime, endTime): The central fact table representing a specific network communication session between two hosts.

- This table captures the "5-tuple" identity of a connection along with its lifecycle state. It acts as the anchor for 1:1 extensions like metrics and labels.
- **Primary Key (PK):** id (Unique flow identifier).
- **Foreign Keys (FK):**
    - srcHostId references id in the **Host** table.
    - dstHostId references id in the **Host** table.
    - protocolId references id in the **Protocol** table.

5. FlowMetrics (id, flowId, sbytes, dbytes, spkts, dpkts, sload, dload, sjit, djit): Stores the quantitative statistical features extracted from a specific flow.

- This table is vertically partitioned from the Flow table. It contains the heavy numerical data required for the TimeGPT forecasting engine (e.g., Source Bytes sbytes, Destination Load dload). Segregating this data allows for lighter metadata queries on the main Flow table.
- **Primary Key (PK):** id.
- **Foreign Key (FK):** flowId references id in the **Flow** table (Unique constraint enforces 1:1 cardinality).

6. AttackLabel (id, flowId, label, category): Stores the ground-truth or predicted classification of a flow.

- This table helps in categorizing flows as 'Normal' or specific attack vectors (e.g., 'Exploits', 'Generic'). It is essential for supervised evaluation and historical reporting.
- **Primary Key (PK):** id.
- **Foreign Key (FK):** flowId references id in the **Flow** table.

5.2 Normalization

The provided schema is designed to adhere to high standards of database normalization to ensure data integrity and optimize storage. We analyze the schema through the First, Second, and Third Normal Forms.

1. First Normal Form (1NF)

The schema satisfies 1NF by ensuring:

- **Atomic Value Constraint (AVC):** All attributes within Flow, Host, and FlowMetrics contain atomic, indivisible values. For example, srcPort stores a single integer, not a list of ports.
- **Elimination of Repeating Groups (ERG):** There are no nested arrays or duplicate columns (e.g., metric1, metric2) within a single table.
- **Primary Key Requirement (PKR):** Every table utilizes a unique surrogate primary key (id).

Key Technical Implications:

- → Prevents parsing overhead during high-velocity write operations.
- → Establishes a foundational structure for efficient indexing in PostgreSQL.

Tables in 1NF:

- **Host_1NF:** Captures discrete endpoint data (ip, createdAt) without multi-valued attributes.
- **Flow_1NF:** Represents connection metadata (srcPort, dstPort) as singular data points.
    - **Metrics_1NF:** Stores statistical counters (sbytes, dbytes) as individual numeric fields.

2. Second Normal Form (2NF)

To achieve 2NF, the schema must eliminate **Partial Dependencies**, ensuring that all non-key attributes depend on the *entire* primary key.

- **Full Functional Dependency (FFD):** Since the AnomalyX schema utilizes **Surrogate Keys** (single-column UUIDs or Integers) for all tables rather than composite natural keys, Partial Dependency is structurally impossible.
- **Optimization Strategy:** If a composite key (e.g., srcIP + dstIP + timestamp) were used, attributes like srcLocation would depend only on srcIP (Partial Dependency). By extracting Host into its own table and using hostId, we satisfy 2NF.

Tables in 2NF:

- **Host_Registry:** Isolates host identity from flow transactions.
- **Flow_Transaction:** Contains only data dependent on the specific unique flow instance (startTime, state), not the inherent properties of the Host or Protocol.
- **Metric_Extension:** The FlowMetrics table relies entirely on its own ID (or the flowId), ensuring statistical data is tied to the specific flow instance.

3. Third Normal Form (3NF)

In 3NF, the schema must address **Transitive Dependencies**, ensuring that non-key attributes do not depend on other non-key attributes.

- **Transitive Dependency Removal (TDR):**
  - *Scenario:* If the Flow table contained a column ProtocolName, it would be dependent on ProtocolID (if it existed) or the concept of the protocol, creating a transitivity.
  - *Resolution:* The schema creates a separate Protocol table. The Flow table only stores protocolId. The name "TCP" exists only once in the Protocol table, not in millions of flow records.
- **Functional Dependency Minimization (FDM):**
  - Similarly, the AttackLabel is separated. Detailed descriptions of attack categories are stored via the label reference, ensuring the Flow table remains lean.

Tables in 3NF:

- **Protocol_Ref (PROT):** Standardizes protocol definitions, removing string redundancy from the Flow table.
- **Service_Ref (SERV):** Centralizes service names, ensuring that if a service is renamed, it is updated in only one row.
- **Flow_Core (FLOW):** Contains strictly the Foreign Keys (srcHostId, protocolId) and transaction-specific timestamps. It contains no descriptive text data that belongs in a lookup table.
- **Metrics_Isolated (MET):** Contains only the numerical deviations derived from the flow, ensuring clear separation of *identity* (Flow table) and *behavior* (Metrics table).

CHAPTER 6

NoSQL Implementation

While the core anomaly detection engine of **AnomalyX** relies on a relational database (PostgreSQL) for structured time-series and transaction integrity, the system adopts a **Polyglot Persistence** architecture by utilizing **MongoDB** for user management and authentication.

The decision to implement a NoSQL solution for the administrative layer is driven by the following architectural advantages:

1. **Schema Flexibility:** User profiles often evolve. MongoDB's document-oriented structure allows for the seamless addition of user preferences, UI customization settings (e.g., dashboard themes), or third-party OAuth tokens without requiring rigid schema migrations or downtime.
2. **High-Performance Read Operations:** Authentication checks and session validation are high-frequency read operations. MongoDB's storage engine (WiredTiger) provides low-latency retrieval for JSON documents, which aligns natively with the Node.js/Express backend used for the Metrics Service.
3. **Horizontal Scalability:** As the user base grows, MongoDB's sharding capabilities offer a clear path for scaling the user data store independently from the massive read/write load of the network flow database.

6.1 Data Description: User Collection

The user management module utilizes a single collection implicitly named users (default Mongoose pluralization). This collection stores authentication credentials and profile metadata. The data is modeled using a strict Mongoose schema to ensure data consistency before it reaches the MongoDB store.

The schema definition imposes specific validation rules and formatting constraints (e.g., trimming whitespace, forcing lowercase emails) to sanitize input at the application level.

Field	Data Type	Properties	Description
_id	ObjectId	**PK**, Auto-gen	A unique 12-byte BSON identifier automatically assigned by MongoDB upon document creation.
email	String	**Unique**, Required, Lowercase, Trimmed	The user's email address. The schema enforces lowercase to prevent case-sensitive duplicates (e.g., "User@Test.com" vs "user@test.com") and trim to remove accidental whitespace.
password	String	Required, Min Length: 6	Stores the **bcrypt** hash of the user's password. The schema enforces a minimum input length of 6 characters before hashing to ensure basic security complexity.
name	String	Required, Trimmed	The user's display name. The trim property ensures clean formatting by removing leading/trailing whitespace.
createdAt	Date	Auto-managed	Timestamp generated automatically by the Mongoose timestamps: true option upon creation.
updatedAt	Date	Auto-managed	Timestamp generated automatically by the Mongoose timestamps: true option, updated whenever the document is modified.

Below is a representation of a standard user document stored within the database:


```
{
  "_id": { "$oid": "64f1b2c8e1a9c3d4f5a6b7c8" },
  "email": "admin@anomalyx.net",
  "password": "$2b$10$EixZaYVK1fsbw1ZfbX3OXePaWrn96pzvPJQucVcn0Jn...",
  "name": "System Administrator",
  "createdAt": { "$date": "2025-10-15T08:30:00.000Z" },
  "updatedAt": { "$date": "2025-10-20T14:15:00.000Z" },
  "__v": 0
}
```


Fig. 6.1. Sample Data in MongoDB User Collection

The schema implements document middleware and instance methods to encapsulate security logic directly within the data model:

- **Pre-Save Hook (Password Hashing):** A pre('save') hook is triggered before any document is persisted to the database. It checks if the password field has been modified. If so, it generates a salt (10 rounds) and replaces the plain-text password with a secure **bcrypt** hash. This ensures that raw passwords are never stored in the database.
- **Instance Method (comparePassword):** The schema defines a custom method comparePassword(candidatePassword) which utilizes bcrypt.compare(). This securely verifies login attempts by comparing the provided plain-text password against the stored hash without ever exposing the hash itself.

CHAPTER 7

Conclusion

AnomalyX successfully demonstrates a paradigm shift in network anomaly detection by prioritizing statistical forecasting over traditional heavy supervised learning. By decoupling data ingestion from inference through a microservices architecture, the system achieves near real-time latency without the operational burden of continuous model retraining.

The integration of **Nixtla's TimeGPT** as a forecasting engine validated the hypothesis that foundation models can effectively generalize to network traffic patterns (Zero-Shot Inference), eliminating the "cold start" problem typical of deep learning-based NIDS. Furthermore, the adoption of a **Polyglot Persistence** strategy—leveraging PostgreSQL for rigid transactional integrity of flow records and MongoDB for flexible user management—proved essential for building a scalable, production-ready backend.

The project highlights that modern network security tools need not be monolithic. Instead, a hybrid approach combining lightweight statistical profiling with external AI inference services offers a viable, low-overhead path for detecting protocol anomalies in dynamic network environments.

CHAPTER 8

References

[1] N. Moustafa and J. Slay, ‘‘UNSW-NB15: A comprehensive data set for network intrusion detection systems,’’ in Proc. Military Communications and Information Systems Conference (MilCIS), Canberra, Australia, 2015, pp. 1–6.

[2] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, ‘‘A detailed analysis of the KDD CUP 99 data set,’’ in Proc. IEEE Symposium on Computational Intelligence for Security and Defense Applications, Ottawa, Canada, 2009, pp. 1–6.

[3] A. L. Buczak and E. Guven, ‘‘A survey of data mining and machine learning methods for cyber security intrusion detection,’’ IEEE Communications Surveys & Tutorials, vol. 18, no. 2, pp. 1153–1176, 2016.

[4] T. Kim and H. Kim, ‘‘Anomaly detection for network intrusion using recurrent neural network,’’ in Proc. IEEE International Conference on Big Data and Smart Computing, Jeju, South Korea, 2016, pp. 1–4.

[5] S. Hochreiter and J. Schmidhuber, ‘‘Long short-term memory,’’ Neural Computation, vol. 9, no. 8, pp. 1735–1780, 1997.

[6] A. Vaswani et al., ‘‘Attention is all you need,’’ in Proc. Advances in Neural Information Processing Systems (NeurIPS), Long Beach, CA, USA, 2017, pp. 5998–6008.

[7] A. Garza, C. Challu, and M. Mergenthaler-Canseco, ‘‘TimeGPT-1,’’ arXiv preprint arXiv:2310.03589, 2023.

[8] P. Boniol, J. Paparrizos, and T. Palpanas, ‘‘An interactive dive into time- series anomaly detection,’’ in Proc. IEEE International Conference on Data Engineering (ICDE), Utrecht, Netherlands, 2024, pp. 5382–5386.

[9] S. Alnegheimish, L. Nguyen, L. Berti-Equille, and K. Veeramachaneni, ‘‘Large language models can be zero-shot anomaly detectors for time series?,’’ arXiv preprint arXiv:2405.14755, 2024.

[10] Z. Latif, Q. Umer, C. Lee, K. Sharif, F. Li, and S. Biswas, ‘‘A machine learning-based anomaly prediction service for software-defined networks,’’ Sensors, vol. 22, no. 21, p. 8434, 2022.

[11] M. Verkerken et al., ‘‘A novel multi-stage approach for hierarchical intrusion detection,’’ IEEE Transactions on Network and Service Management, vol. 20, no. 4, pp. 4459–4472, 2023.

[12] O. Belarbi, M. Bouziane, and A. Bouridane, ‘‘An intrusion detection system based on deep belief networks,’’ in Science of Cyber Security. Cham, Switzerland: Springer, 2022, pp. 377–392.

[13] J. Su et al., ‘‘Large language models for forecasting and anomaly detection: A systematic literature review,’’ arXiv preprint arXiv:2402.10350, 2024.

[14] T. Jafarian, M. Masdari, A. Ghaffari, and K. Majidzadeh, ‘‘Security anomaly detection in software-defined networking based on a prediction technique,’’ International Journal of Communication Systems, vol. 33, no. 15, p. e4524, 2020.

[15] I. Ullah and Q. H. Mahmoud, ‘‘Design and development of RNN anomaly detection model for IoT networks,’’ IEEE Access, vol. 10, pp. 62722– 62750, 2022.

APPENDIX

Snapshots

Fig A.1 Home Page

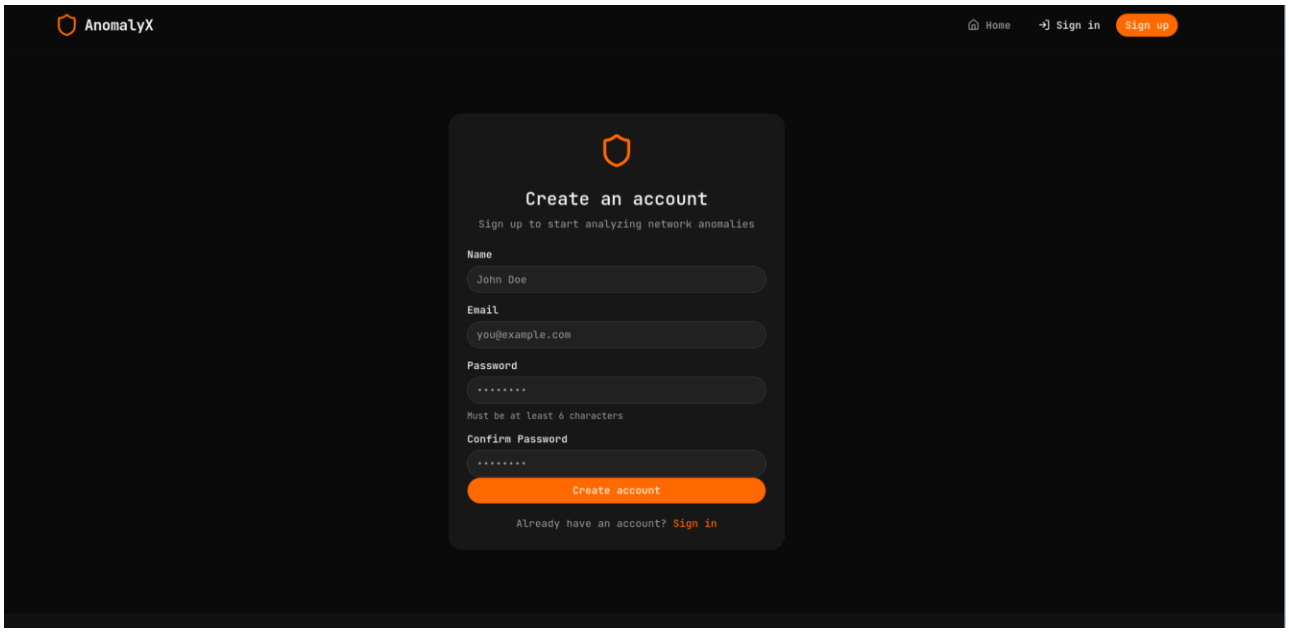


Fig A.2 Sign Up Page

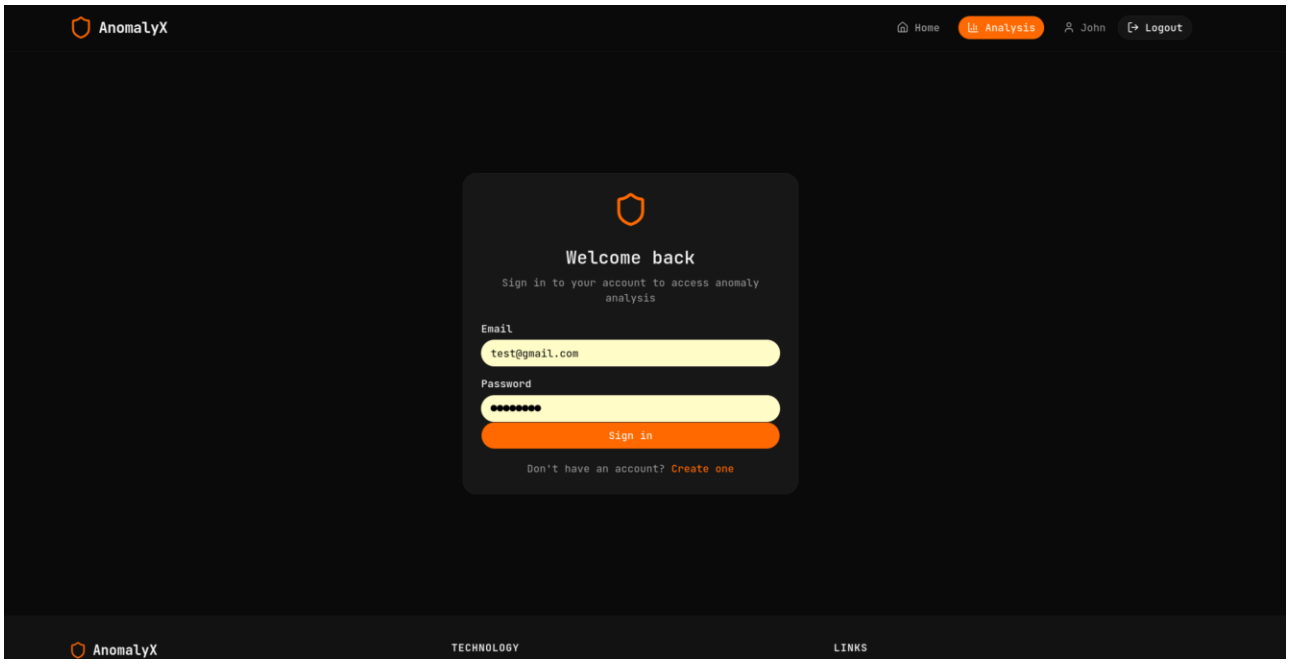
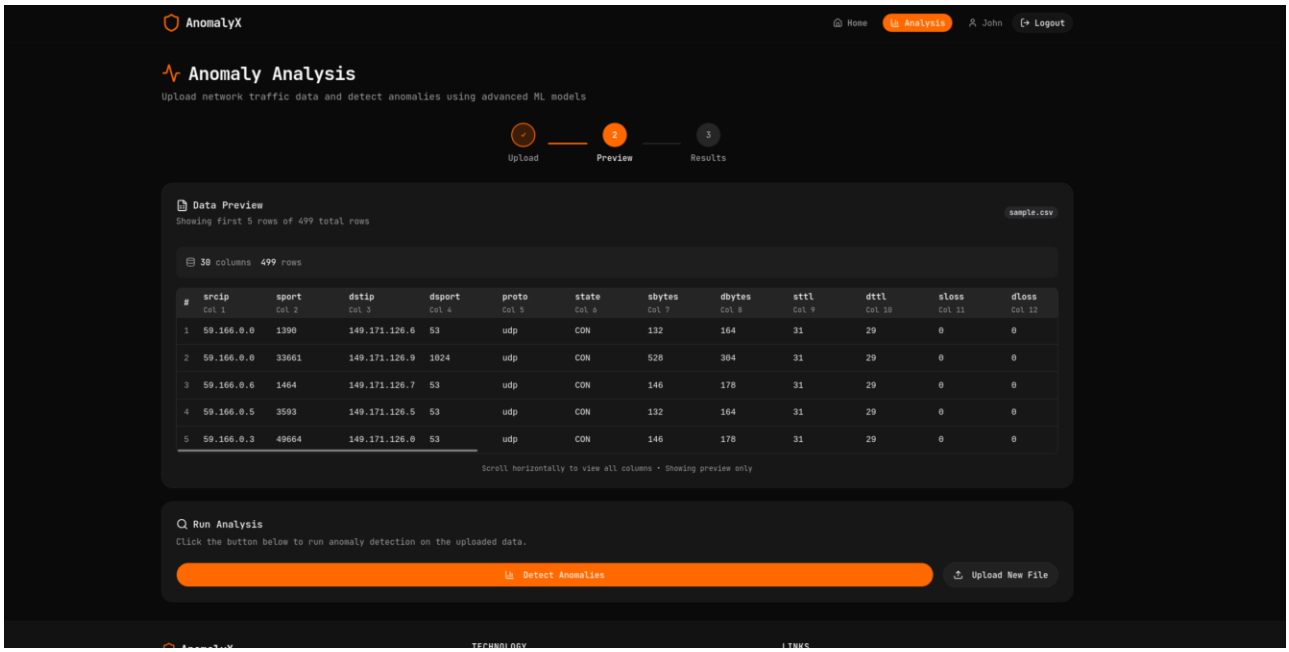


Fig A.3. Sign In Page



#	srcip	sport	dstip	dsport	proto	state	sbytes	dbytes	sttl	dttl	sloss	dloss
	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8	Col 9	Col 10	Col 11	Col 12
1	59.166.0.0	1390	149.171.126.6	53	udp	CON	132	164	31	29	0	0
2	59.166.0.0	33661	149.171.126.9	1024	udp	CON	528	304	31	29	0	0
3	59.166.0.6	1464	149.171.126.7	53	udp	CON	146	178	31	29	0	0
4	59.166.0.5	3593	149.171.126.5	53	udp	CON	132	164	31	29	0	0
5	59.166.0.3	49664	149.171.126.0	53	udp	CON	146	178	31	29	0	0

Fig A.4 Data Upload Page

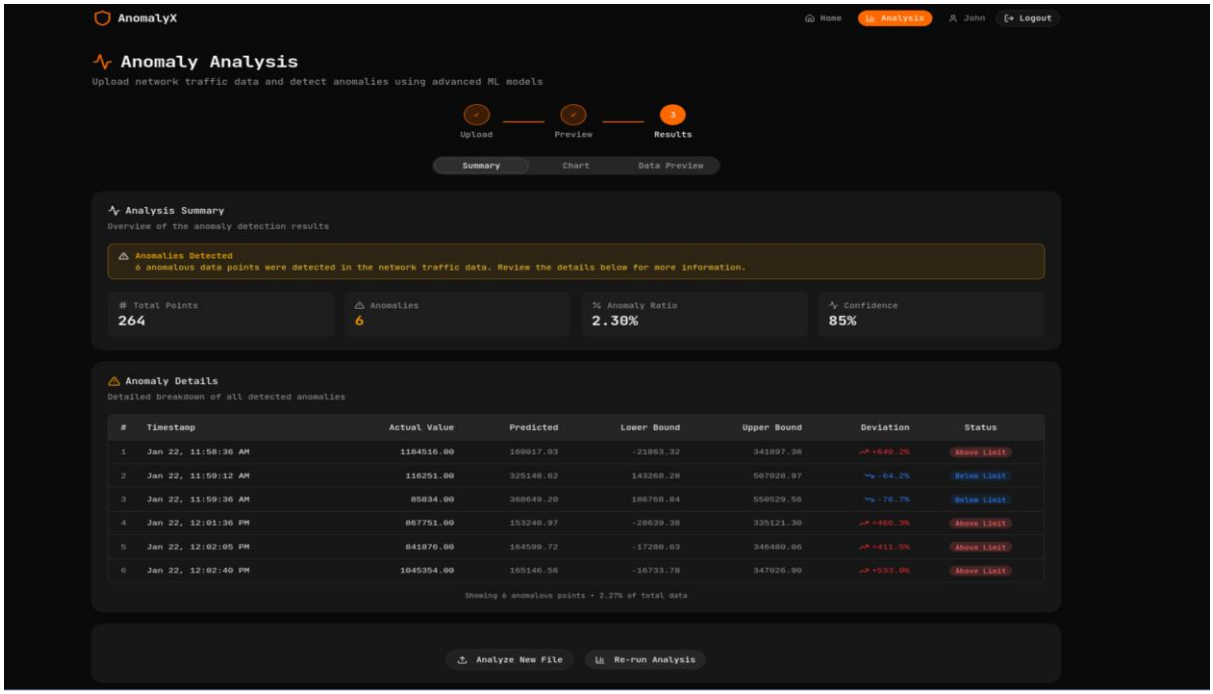


Fig A.5 Analysis Summary

Fig A.6. Anomaly Chart