


CAMPUS SECURITY SOLUTIONS

AI-Powered Monitoring & Prediction Platform

Team Members:

1. Nilotpal Gupta
2. Sankarsaan Mandal
3. Sonmitra Seth
4. Arnav De

The background features several solid green circles of varying sizes. A faint, complex network diagram with many interconnected nodes and lines is centered behind the text.

This report details an AI-powered security platform designed to unify multi-modal campus data. The system utilizes a microservices architecture with a Next.js/Drizzle ORM application serving as a frontend and Bridge API, and a Python/Flask service for machine learning inference. Core algorithms include a Logistic Regression model for probabilistic entity resolution and a First-Order Markov Chain for predictive location analysis during timeline gaps. The system emphasizes Explainable AI (XAI) by providing human-readable evidence for all predictions. Performance analysis confirms low-latency API responses (<700ms P95 for ML tasks) and high model accuracy (~85%). The architecture incorporates robust security safeguards, including Next.js Middleware for route protection and data hashing for privacy.

System Architecture

- Next.js Full-Stack Application:

- Frontend & API: A Next.js 14 application serves the React-based dashboard and exposes API routes under ``app/api/``.
- Authentication & Authorization: NextAuth.js manages user authentication (credentials, 2FA, QR) using JWTs. Next.js Middleware (``middleware.ts``) protects sensitive routes by validating session tokens at the server edge.
- Bridge API Pattern: The API routes (``/api/predict/*``) act as a "Bridge," performing efficient, bulk data aggregation from the database to gather all necessary context for an ML task. This single, structured payload is then sent to the Python service, preventing inefficient, numerous queries from the ML environment.

- Python Machine Learning Service:

- Inference Engine: A stateless Flask API hosts the trained Scikit-learn models (``owner_prediction_model.pkl``, ``location_prediction_model.pkl``).
- Function: It receives data payloads from the Next.js bridge, performs real-time feature engineering, and returns JSON predictions that include a confidence score and a human-readable evidence array for explainability.

- PostgreSQL Database:

- Data Store: The central repository for all user, device, and activity log data.
- Query Layer: The Next.js application uses Drizzle ORM (``database/schema.ts``) for type-safe, efficient, and complex relational data fetching.

Data Fusion & Timeline Generation

The system's core capability is the programmatic fusion of disparate data logs into a single, coherent narrative. This process is executed entirely on the server within the `/api/timeline/[userId]` route.

1. Unified Relational Query: For a given `userId` and optional `from/to` date parameters, a single database query is executed using Drizzle ORM. The query fetches the user record along with all related records from multiple tables (`swipeLogs`, `wifiLogs`, `roomBookings`, etc.) using nested `with` clauses that incorporate the date-range filters directly.

2. Data Normalization: The heterogeneous results from the query are processed into a standardized `TimelineEvent` object array. Each event type is mapped to a common schema, ensuring consistency for rendering:

```
type TimelineEvent = {  
  type: string; // e.g., 'Card Swipe', 'Wi-Fi Connection'  
  timestamp: Date; // The primary sorting key  
  details: string; // Human-readable description  
  location: object; // Standardized location data  
};
```

3. Chronological Sorting: The final `TimelineEvent` array is sorted in descending order based on the `timestamp` property. This produces the chronological sequence of events that is paginated and sent to the frontend client for visualization.

Machine Learning Methodologies

1. Entity Resolution Algorithm (Owner Prediction)

To resolve the ownership of unassigned campus cards, a Logistic Regression model is employed.

Process Flow:

1. Anchor Identification

All swipe logs for the unknown cardId are identified.

2. Evidence Gathering

For each swipe, the Bridge API fetches all co-located, temporally-proximate (± 5 min) events (Wi-Fi, bookings, CCTV).

3. Feature Engineering

For each candidate user, the predict_owner.py service computes a feature vector from the evidence.

Key features include:

- time_diff_wifi (seconds)
- same_location_wifi (binary)
- is_in_booking (binary)
- has_alibi (binary)

4. Inference

The trained model (owner_prediction_model.pkl) predicts the probability of ownership for each candidate based on their feature vector.

5. Explainability (XAI)

The API response includes an evidence array detailing the features that led to the prediction.

Example:

"A known device connected to a nearby Wi-Fi AP within 45 seconds."

2. Predictive Monitoring Algorithm (Location Prediction)

Timeline gaps are filled using a First-Order Markov Chain to predict the user's most probable location.

Process Flow:

1. Training: An offline script analyzes historical location data to compute a transition_matrix.pkl. This matrix stores the conditional probability $P(L_{next} | L_{current})$ for all location pairs.

2. State Identification: The predict_location.py service receives the user's last known location (locationBefore) as the current state.

3. Inference: The model performs a lookup in the transition matrix to identify the L_{next} with the highest probability given the current state.

4. Explainability (XAI): The prediction is explained by referencing the model's logic (e.g., "Based on historical data, users at [Location Before] are most likely to next visit [Predicted Location].").

Performance Analysis

The Logistic Regression model was evaluated on a held-out test set with an 80/20 class distribution.

Class	Precision	Recall	F1-Score
0 (Not Owner)	0.80	1.00	0.89
1 (Is Owner)	0.82	0.75	0.78

Runtime & Scalability

- **API Latency:** End-to-end performance measurements show low latency for key operations.
 - **Timeline Generation** (/api/timeline): P95 latency is ~250ms.
 - **Owner Prediction** (/api/predict/owner): P95 latency is ~700ms, including the network hop to the Python service and model inference.
 - The Bridge API pattern is critical to this performance, preventing timeouts by minimizing database round-trips.
- **Scalability:**
 - **Horizontal Scaling:** The Next.js and Python services are stateless, allowing them to be replicated behind a load balancer to handle increased request volume.
 - **Bottleneck:** The primary scaling bottleneck is the single PostgreSQL database instance. Production deployments would require a vertical scaling strategy or a read-replica configuration to distribute query load.

Safeguards & Failure Mode Analysis

Highlighting key advantages of the AI-powered platform

Privacy & Security Safeguards

- Authentication: NextAuth.js manages user sessions with secure, HTTP-only JWTs, implementing credentials login with 2FA and a one-time-use QR code system.
- Authorization: Next.js Middleware protects all sensitive pages and API routes by validating session tokens at the server edge before any data is processed.
- Data Privacy: Passwords are salted and hashed with bcrypt. Personal device identifiers are stored as non-reversible SHA-256 hashes to prevent direct identification.

Failure Mode Analysis

- ML Service Failure: In the event of a Python service outage, the Next.js Bridge API will time out and return a graceful error. The frontend UI will disable the specific predictive feature and notify the user, while core functionality remains operational.
- Database Failure: The database is a single point of failure. An outage will cause the entire application to fail. High-availability database configurations are the recommended mitigation for production.
- Incorrect Prediction: This risk is mitigated by system design. All ML outputs are presented as a probabilistic score accompanied by explainable evidence, ensuring the system serves as a decision-support tool where the human operator makes the final judgment.

System Architecture Overview

