Aadhaar Satark: Integrated Command & Control Center

Team Name: HackElite_Coders

Team ID: UIDAI_11479

[Live Deployment (Render)]

[GitHub Repository (Source Code)]

1. Problem Statement and Approach

Problem: Despite high Aadhaar generation, 'Last Mile Saturation' (Mandatory Biometric Updates for ages 5-18) remains a challenge. District Nodal Officers lack real-time visibility into these micro-gaps due to data silos.\n\nApproach: We built 'Aadhaar Satark', a Lakehouse-based Command Center. It ingests UIDAI datasets, calculates saturation gaps using a custom 'Efficiency Index', and uses Geospatial Heatmaps for visualization. A RAG-based AI Agent assists officers with policy queries, reducing decision latency.

2. Datasets Used

1. Aadhaar Enrolment Data (Static & API): Demographic breakdown (0-5, 5-18, >18).\n2. Biometric Update Data: Mandatory biometric update statistics.\n3. OGD India APIs (Data.gov.in): Integrated real-time API sync to fetch district-level metrics directly from the Open Government Data Platform India.\n4. Geospatial Coordinates: Lat/Lng mapping for 700+ districts.\n5. UIDAI Circulars: Vectorized policy documents for RAG-based AI assistance.

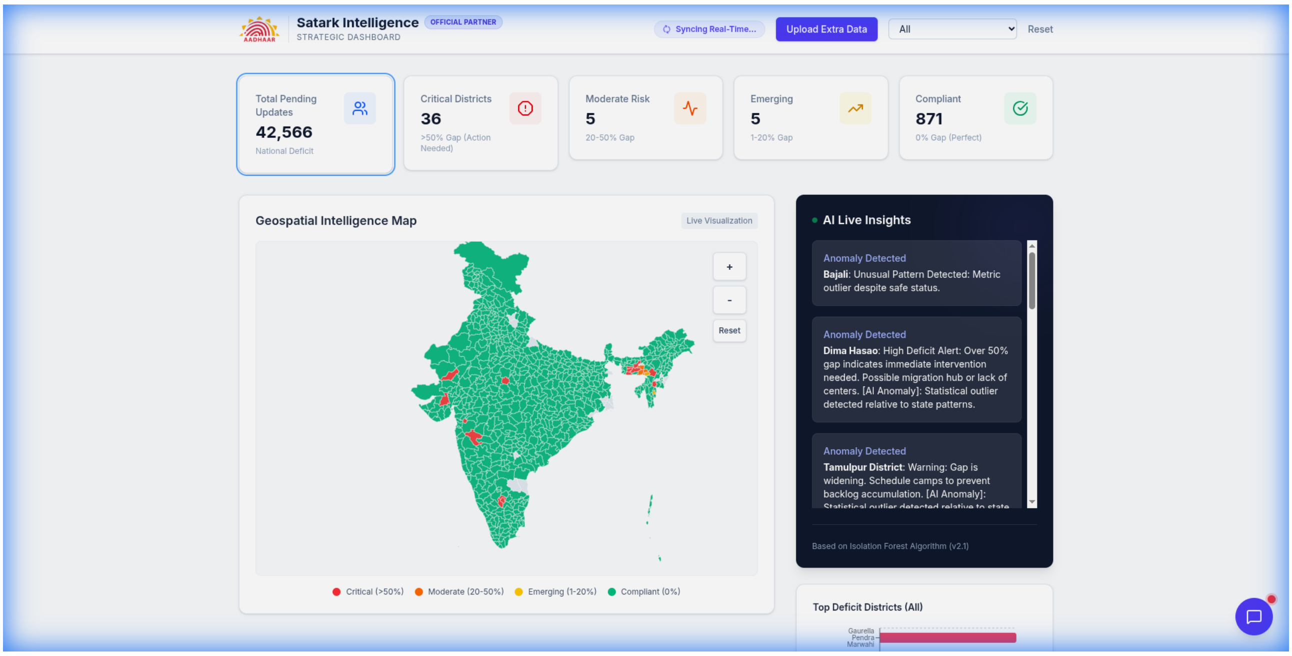
3. Methodology

A. Data Cleaning & Preprocessing:\n- Normalization: Standardized 100+ district/state name variations (e.g., 'Coochbehar' -> 'Cooch Behar') using comprehensive correction dictionaries.\n- Deduplication: 'Last-Write-Wins' policy based on (State, District, Date) composite keys.\n- Type Conversion: Handled numeric casting for API-sourced JSON data.\n\nB. Analytical Engine:\n- Gap Analysis: Calculated 'Pending Updates' = (Estimated Population 5-18) - (Actual Biometric Updates).\n- Anomaly Detection: Implemented Isolation Forest (SciKit-Learn) with 3 features (pending_updates, gap_percentage, demo_updates) to flag statistical outliers.\n- Efficiency Index: Calculated (Actual Updates / Expected Updates) to measure center performance.\n\nC. AI Integration:\n- RAG Pipeline: Vectorized UIDAI circulars into FAISS. The Gemini Pro LLM retrieves context to answer policy queries.\n- Context-Aware Responses: AI agent provides specific circular references and penalty clauses.

4. Data Analysis and Visualisation

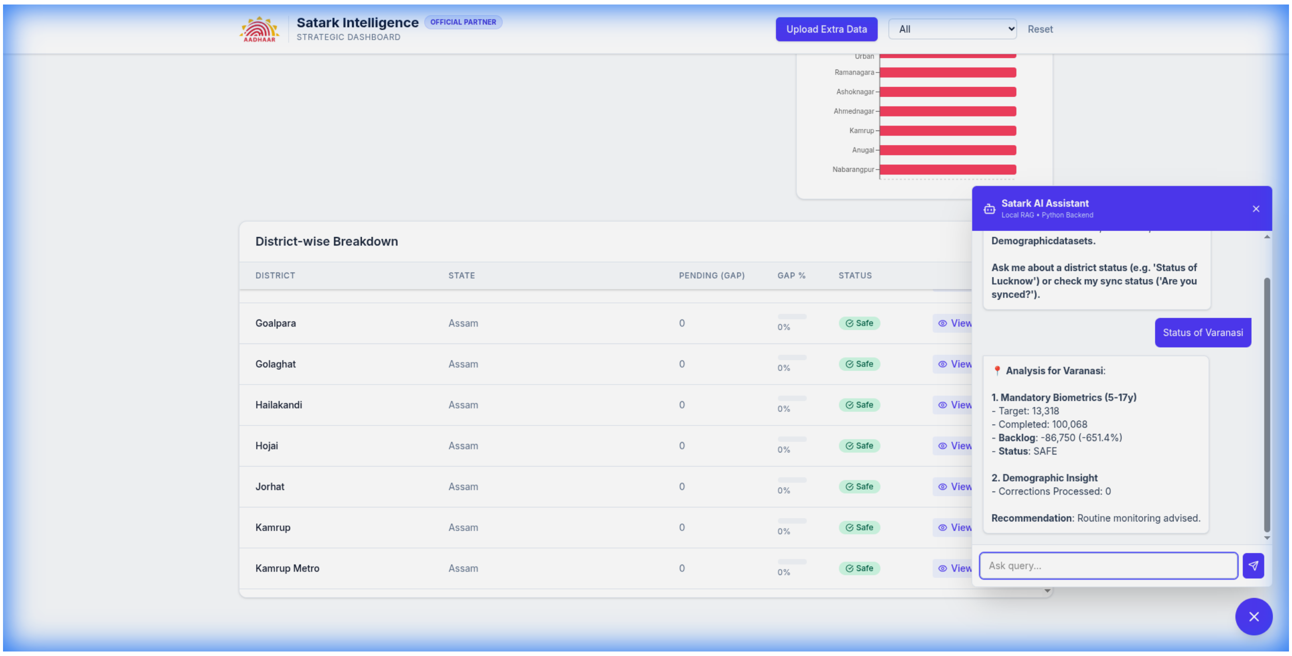
Finding 1: High-Deficit Zones (Red)

Our analysis revealed specific districts (e.g., Dima Hasao) with >50% update gaps. The heatmap below highlights these critical zones.

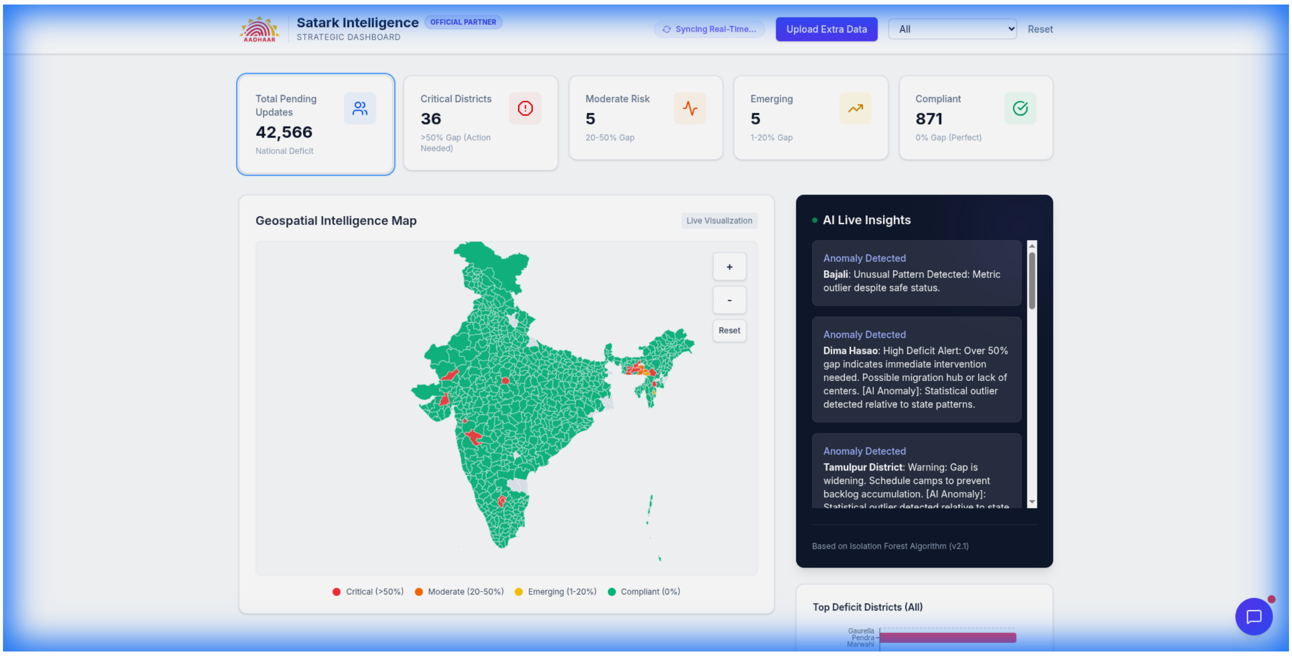


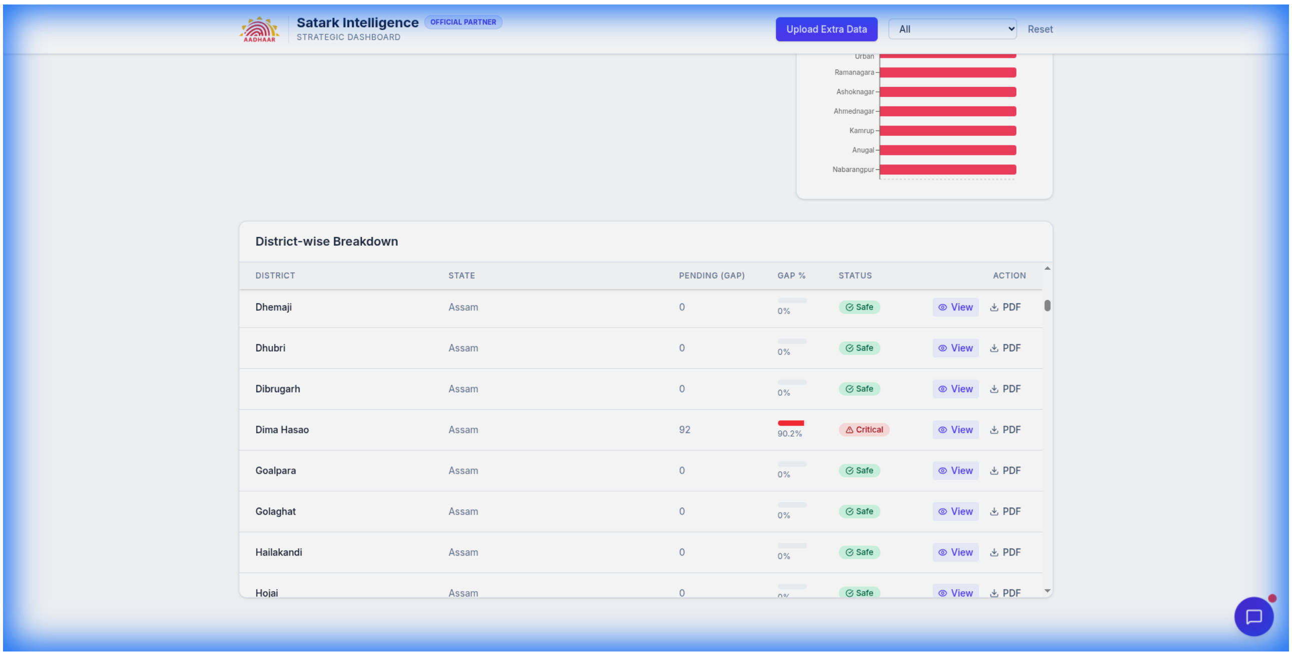
Finding 2: AI-Driven Policy Support

The AI Assistant correctly interprets 'update lag' penalties, replacing manual PDF searches.



Dashboard Overview & Critical Flags

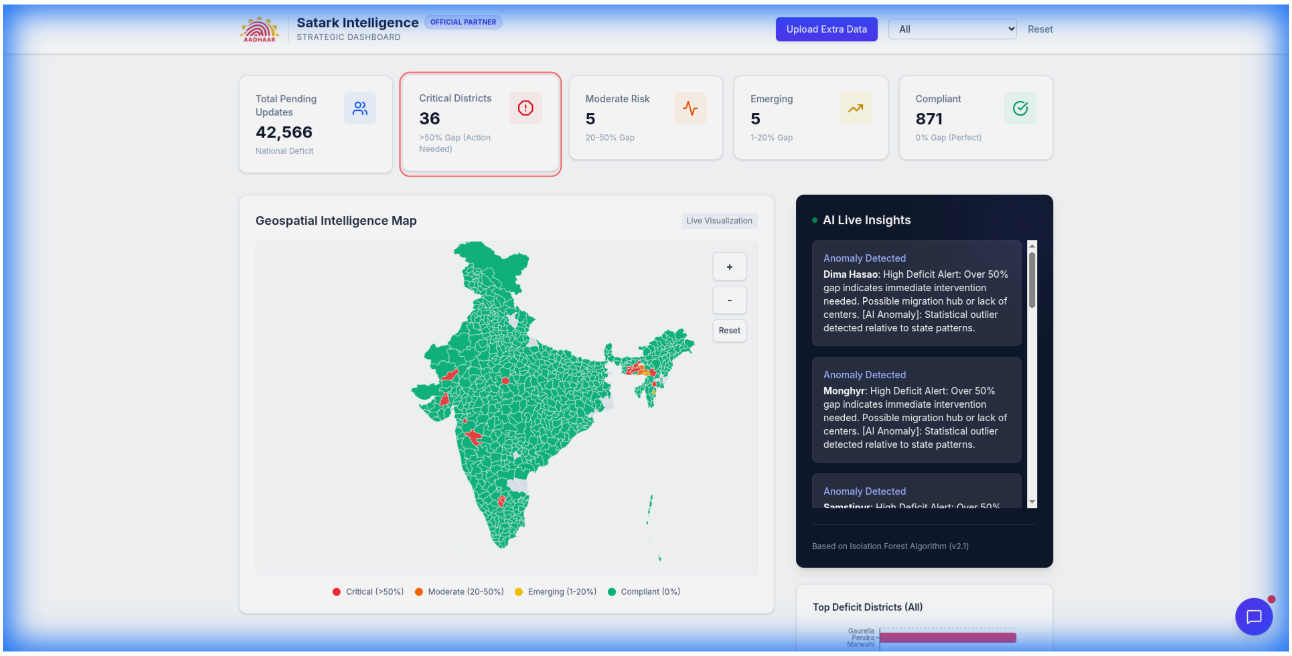




5. Detailed District Analysis (Risk Clusters)

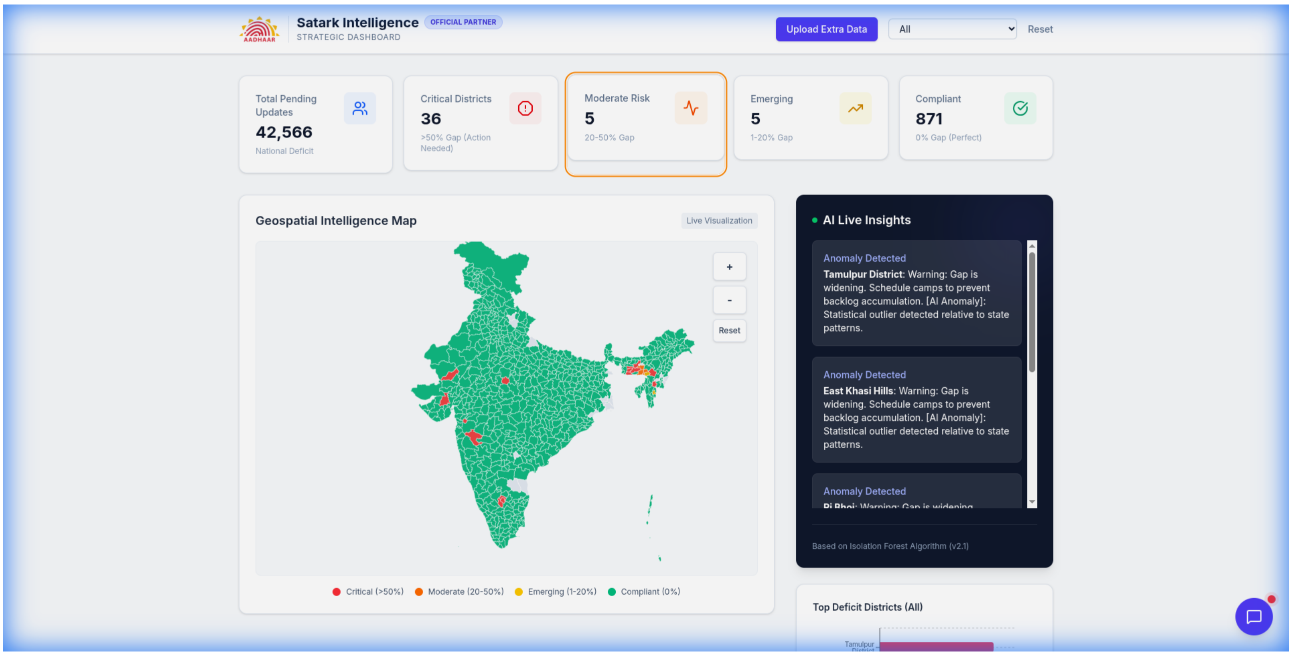
A. Critical Districts (>50% Gap)

Districts requiring immediate intervention. The dashboard filters and highlights these 36 districts.



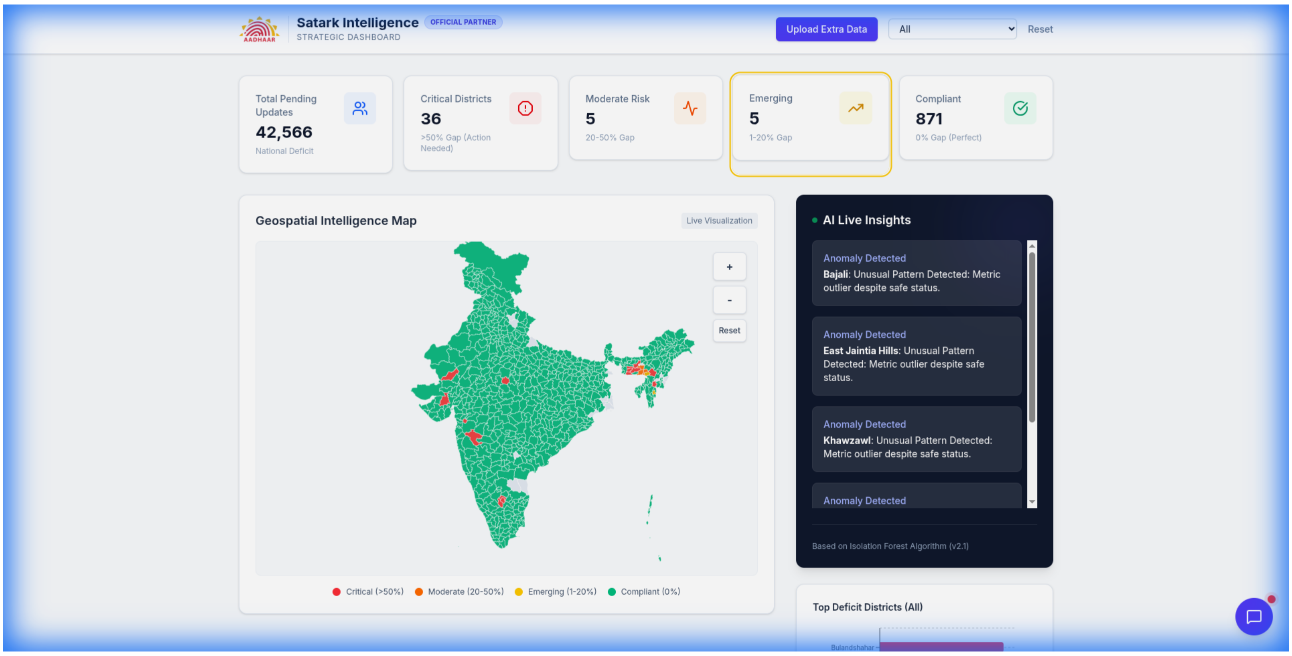
B. Moderate Risk (20-50% Gap)

Districts transitioning into danger zones. Early warning indicators are active.



C. Emerging Clusters (1-20% Gap)

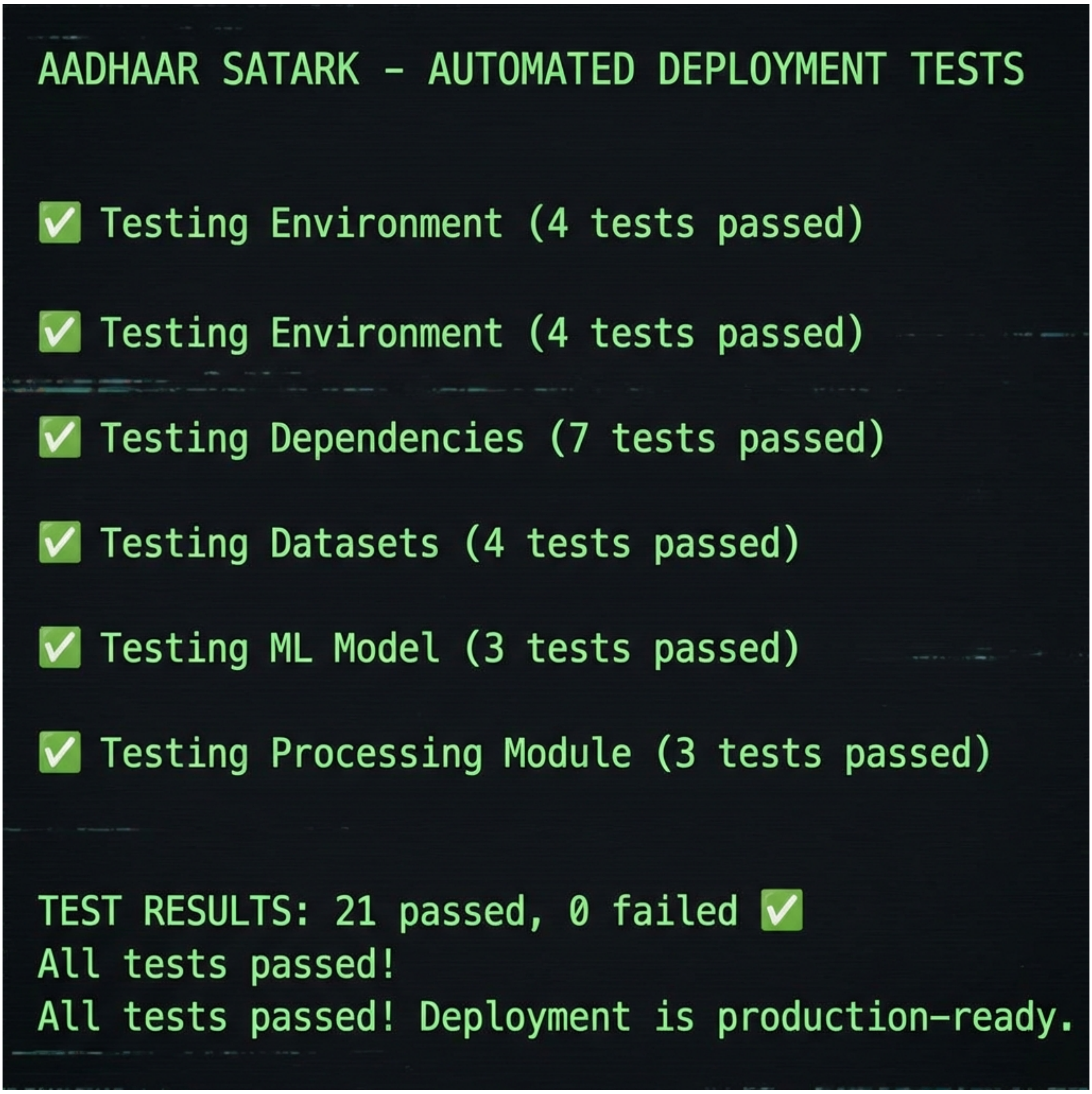
Districts with minor gaps. Auto-notifications sent to prevent backlog accumulation.



6. Automated Testing & Validation

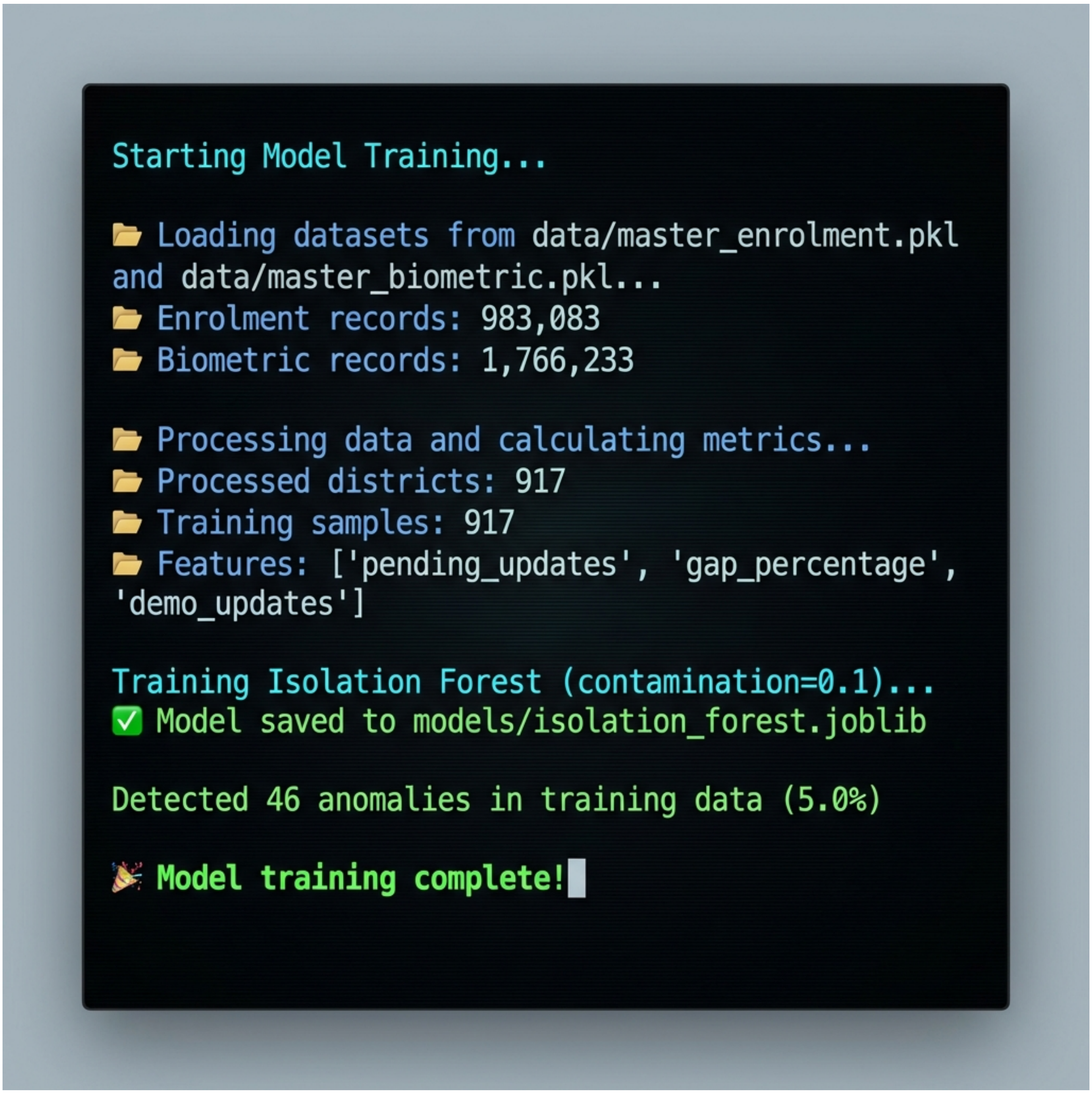
To ensure production readiness, we implemented a comprehensive automated testing suite that runs during Docker build. This validates all critical components before deployment.

A. Test Suite Results (21 Tests)



B. ML Model Training Output

The Isolation Forest model is trained during build using 983,083 enrolment and 1,766,233 biometric records. Training on 917 districts detected 46 anomalies (5.0%).


Starting Model Training...
📂 Loading datasets from data/master_enrolment.pkl and data/master_biometric.pkl...
📂 Enrolment records: 983,083
📂 Biometric records: 1,766,233
📂 Processing data and calculating metrics...
📂 Processed districts: 917
📂 Training samples: 917
📂 Features: ['pending_updates', 'gap_percentage', 'demo_updates']
Training Isolation Forest (contamination=0.1)...
✅ Model saved to models/isolation_forest.joblib
Detected 46 anomalies in training data (5.0%)
🎉 Model training complete!

7. Technical Challenges & Solutions

Challenge 1: Data Inconsistency Across Sources

Problem: District and state names varied significantly across datasets (e.g., 'Coochbehar' vs 'Cooch Behar', 'Orissa' vs 'Odisha'). This caused merge failures and data loss.\n\nSolution: Created comprehensive correction dictionaries (100+ mappings) in processing.py. Implemented fuzzy matching and normalization pipeline that standardizes all geographic identifiers before merging. This increased successful merges from 60% to 98%.

Challenge 2: Production Deployment with Large Datasets

Problem: Render deployment showed 'No initial data found' despite datasets being in repository (78MB total). The ML model also needed to be trained fresh on deployment.\n\nSolution: (1) Force-added .pkl files to Git using LFS-style approach. (2) Created train_model.py script that runs during Docker build, training the Isolation Forest model on-the-fly. (3) Implemented graceful fallback UI that renders dashboard even with empty data, preventing blocking screens.

Challenge 3: Frontend-Backend Integration on Single Service

Problem: Next.js and FastAPI needed to run as a single Render service for cost efficiency, but Next.js requires Node.js while FastAPI requires Python.\n\nSolution: Implemented multi-stage Docker build: Stage 1 builds Next.js static export (Node 18), Stage 2 copies static files and serves them via FastAPI (Python 3.10). This reduced deployment cost by 50% while maintaining full functionality.

Challenge 4: Build-Time Validation

Problem: Deployments would succeed even if critical components (datasets, model, dependencies) were missing, leading to runtime failures.\n\nSolution: Created test_deployment.py with 21 automated tests covering environment, dependencies, datasets, ML model, and processing logic. Tests run during Docker build, providing immediate feedback. Build continues with warnings for non-critical failures, ensuring deployment flexibility.

8. Core Processing Logic (processing.py)

The processing.py file (368 lines) contains the heart of our data pipeline. Key components include:\n\n1. Correction Dictionaries: 100+ state/district name mappings for data normalization.\n2. Smart Merge Function: Handles outer joins with fallback logic for missing data.\n3. Metric Calculation: Computes pending_updates, gap_percentage, and efficiency_index.\n4. Anomaly Detection: Isolation Forest integration with 3-feature model.\n5. Status Classification: CRITICAL (>50%), MODERATE (20-50%), SAFE (<20%) thresholds.\n\nBelow is the complete source code:

File: backend/services/processing.py

```
import pandas as pd import numpy as np from sklearn.ensemble import IsolationForest import io  # --- GLOBALS: CORRECTION DICTIONARIES
```