Design Document

Hybrid IRCTC Bot-Detection And Train Search Web App

Prepared by: Subradeep Das

11th August 2025

Centre for Development of Advanced Computing (CDAC) — CINE
IIT Guwahati Research Park, Guwahati, Assam

Contents

List of Figures

Chapter 1

Executive Summary

This document describes the system architecture, components, data flows, API contracts, database design, model training and retraining policy, deployment plan, security considerations, and testing strategy for the Hybrid IRCTC Bot-Detection and Train Search Web App.

Chapter 2

Scope and Goals

2.1 Purpose

Provide a detailed, implementable blueprint for the system so developers and reviewers can understand how each module is built and integrated.

2.2 Primary goals

- Reliable bot detection combining CAPTCHA and behavioral analysis.
- Fast and informative train search with probability estimates for confirmation and punctuality.
- Clear retraining and model governance process.
- Practical deployment and operational guidance with emphasis on security and privacy.

Chapter 3

High-level Architecture

3.1 Overview

The system is split into:

- Frontend: Login (behavior capture + CAPTCHA), Train Search UI, Admin Dashboard.
- Backend: API gateway (authentication, feature aggregation, inference orchestration).
- Inference services: Behavioral classifier (TensorFlow) and Ticket probability predictor (Random Forest).
- Persistence: PostgreSQL (recommended), model artifacts and logs on disk/object storage.
- Background workers: Celery + Redis for retraining, batch jobs, and reporting.
- Edge: Nginx for TLS termination, static asset serving, and basic rate-limiting.

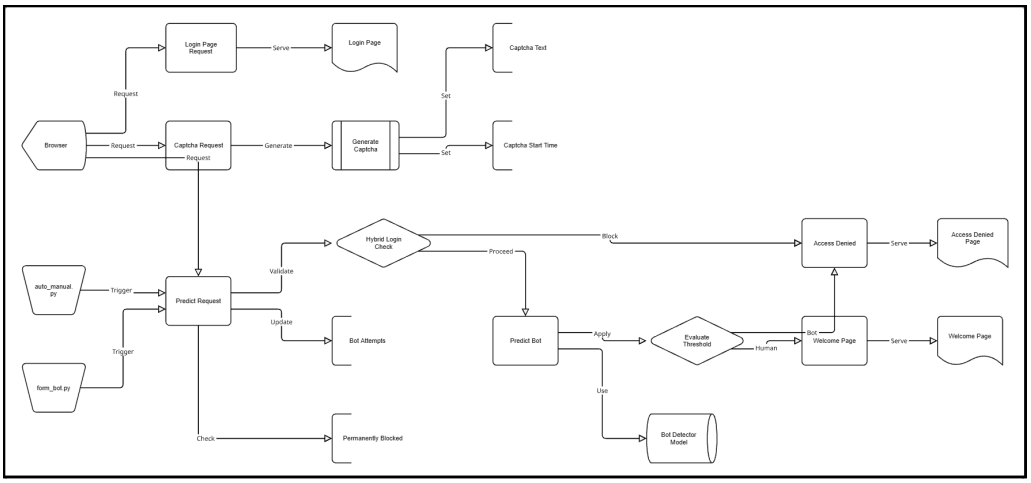


Figure 3.1: System workflow overview

Chapter 4

Component Design

4.1 Frontend (Browser)

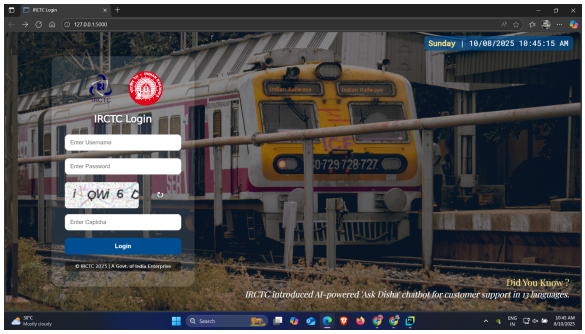
4.1.1 Responsibilities

- Collect behavioral telemetry (mouse, keyboard, focus/blur, timing) on login page.
- Display image CAPTCHA and validate token from server.
- Provide train search UI with station suggestions, calendar, and result list showing probabilities and metrics.
- Expose admin pages for authorized users.

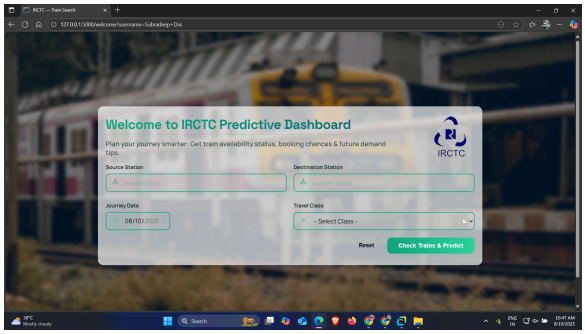
4.1.2 Behavioral payload (example)

```
{
  "session_id": "uuid-v4",
  "events": [
    {"type":"mousemove","t":120,"x":234,"y":105},
    {"type":"keydown","t":350,"key":"a"},
    {"type":"keyup","t":370,"key":"a"}
  ],
  "summary": {
    "mouse_total_distance": 523.2,
    "avg_mouse_velocity": 120.3,
    "typing_cpm": 210,
    "focus_changes": 2,
    "captcha_time_ms": 2350
  }
}
```

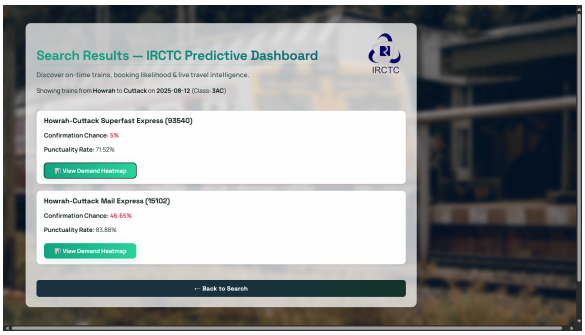
4.1.3 UI mockups (selected)



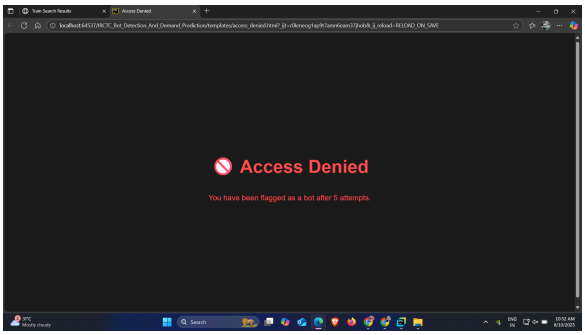
(a) Welcome / Dashboard



(b) Train Search UI



(c) Train Results with prediction badges



(d) Access Denied / Bot Detected

Figure 4.1: Frontend screens and flows

4.2 Backend: API Gateway (Flask)

4.2.1 Primary endpoints

Endpoint	Description / Payload
POST /api/login	{email, password, captcha_token, behavioral} → returns {status, label, confidence, session}
GET /api/search	Query params: source, dest, date, class → returns list of trains + prediction objects
POST /api/retrain	Admin-only trigger; optional force.
GET /api/admin/logs	Admin-only paginated logs.

4.2.2 Login sequence

1. Verify CAPTCHA server-side.
2. Aggregate behavioral events into features.
3. Call behavioral classifier for label and confidence.
4. Log attempt (JSONB) and decide flow: allow or deny (with human-review fallback).

4.3 Behavioral Classifier (TensorFlow)

4.3.1 Input / Output

Input: aggregated session-level features (and optional short sequences). Output: class probabilities for {human, bot, human-like-bot}.

4.3.2 Suggested lightweight architecture

Dense network for low-latency inference:

$$\text{Input (N)} \rightarrow \text{Dense(128, ReLU)} \rightarrow \text{Dropout(0.3)} \rightarrow \text{Dense(64, ReLU)} \rightarrow \text{Dense(3, Softmax)}$$

4.3.3 Preprocessing

- Winsorize extreme outliers.
- Z-score normalization using stored mean/std.
- Persist scaler and feature list alongside model.

4.4 Ticket Confirmation Predictor (Random Forest)

4.4.1 Features

days_before_journey, day_of_week, month, past_avg_waitlist, past_punctuality, class, quota, encoded station features, train_no.

4.4.2 Prediction contract

Return calibrated probability and model metadata:

```
{ "train_no": "12345", "confirmation_prob": 0.71, "model_version": "
   rf_v1.2" }
```

Chapter 5

Data Model and Database Design

5.1 Key entities

- **users** — registered users and roles.
- **login_attempts** — behavioral payloads, IP, UA, label, model_version.
- **trains** — train metadata and historical stats.
- **predictions** — runtime predictions with features and model metadata.

5.2 Recommended schemas (Postgres)

```
CREATE TABLE users (
  user_id SERIAL PRIMARY KEY,
  name TEXT,
  email TEXT UNIQUE NOT NULL,
  password_hash TEXT NOT NULL,
  role TEXT DEFAULT 'user',
  created_at TIMESTAMP DEFAULT now()
);

CREATE TABLE login_attempts (
  attempt_id SERIAL PRIMARY KEY,
  user_id INTEGER REFERENCES users(user_id),
  timestamp TIMESTAMP DEFAULT now(),
  ip TEXT,
  user_agent TEXT,
  features_json JSONB,
  label TEXT,
  model_version TEXT
);

CREATE TABLE trains (
  train_no TEXT PRIMARY KEY,
  train_name TEXT,
  source TEXT,
  destination TEXT,
  departure_time TIME,
  arrival_time TIME,
  punctuality_rate REAL
);

```

```
CREATE TABLE predictions (
  prediction_id SERIAL PRIMARY KEY,
  train_no TEXT REFERENCES trains(train_no),
  timestamp TIMESTAMP DEFAULT now(),
  features_json JSONB,
  predicted_prob REAL,
  model_version TEXT
);
```

5.3 Indexes and performance

Index on `login_attempts(timestamp)`, partial index for denied attempts, and indexes on `predictions(train_no)` and `predictions(timestamp)`.

Chapter 6

Data Flow Diagrams

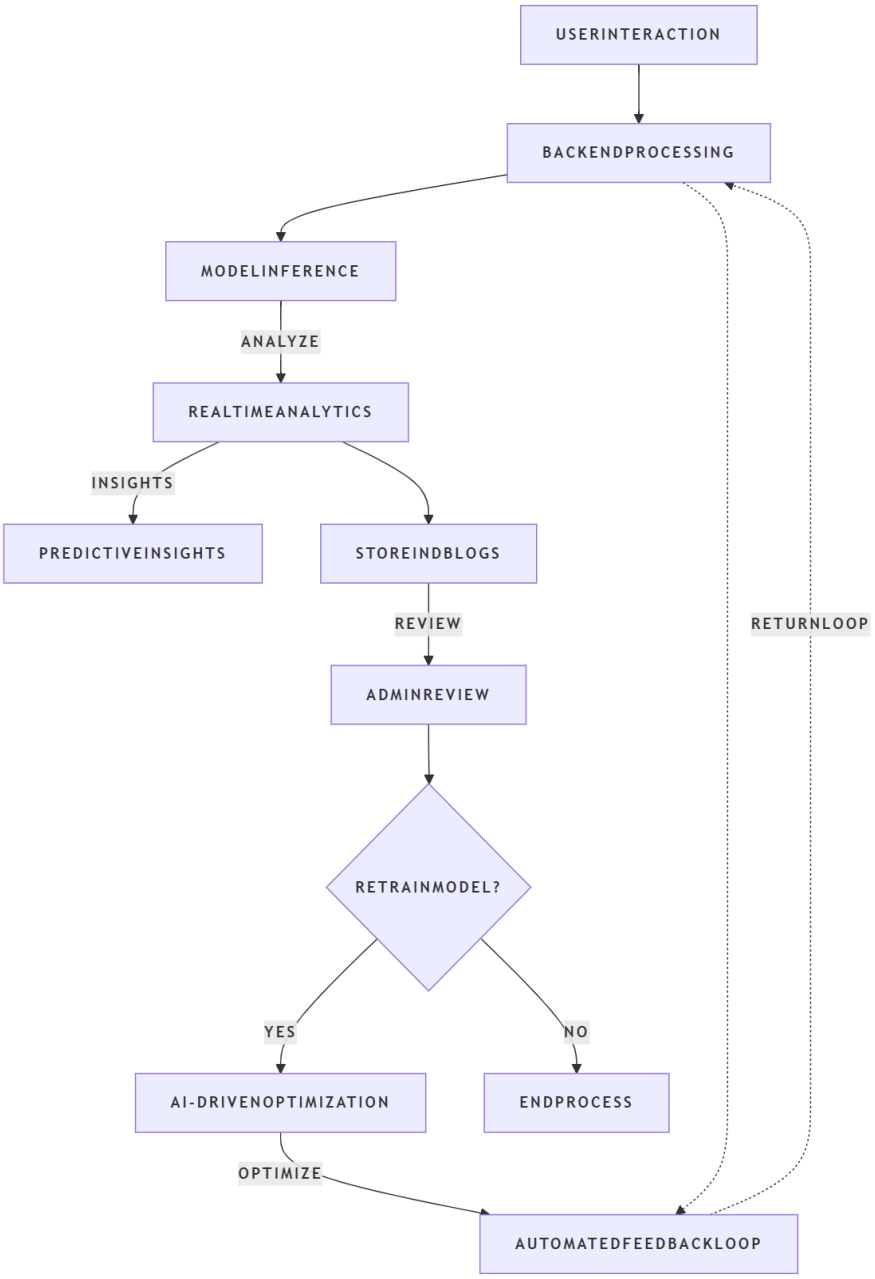


Figure 6.1: Detailed data flow showing feature aggregation and model inference

Chapter 7

API & Integration Contracts

7.1 POST /api/login

Request:

```
{
  "email": "user@example.com",
  "password": "strongpassword",
  "captcha_token": "...",
  "behavioral": { ... }
}
```

Response (allowed):

```
{ "status":"ok", "label":"human", "confidence":0.98, "session":"sid
   -..." }
```

Response (denied):

```
{ "status":"denied", "label":"bot", "confidence":0.92, "message":"
   Access denied" }
```

7.2 Admin endpoints

Require role-based authentication and IP allowlist: `POST /api/retrain`, `GET /api/admin/logs`, `GET /api/models`.

Chapter 8

Model Training, Retraining and MLOps

8.1 Datasets

- Behavioral dataset with aggregated features and labels.
- Historical ticket dataset with waitlist, booking, punctuality records.

8.2 Retraining policy

- Auto-trigger retrain when new login data reaches a threshold (e.g., +10 new entries since last train) and minimum totals satisfied.
- Promote a new model only if validation accuracy improves and validation loss decreases compared to the current production model.
- Archive prior models and log retraining metadata (version, samples, metrics).

8.3 Model artifacts and metadata

Store: model_file (HDF5 or SavedModel), scaler (joblib), metadata.json (version, date, features, metrics).

Chapter 9

Deployment and Operations

9.1 Containerization

Use Docker to package application and inference components; orchestrate with docker-compose for dev and a container platform for prod.

9.2 Suggested process

- Nginx as reverse proxy for TLS and gzip/static assets.
- Gunicorn (WSGI) for Flask app; persistent TF serving or in-process model loaded at startup for low latency.
- Redis for caching and Celery broker.
- Postgres for structured persistence.

Chapter 10

Security and Privacy

10.1 Authentication and secrets

Passwords stored with bcrypt/Argon2; sessions use secure cookies; admin endpoints restricted.

10.2 Privacy

Avoid storing raw typed input. Persist aggregated features instead of raw event dumps whenever possible. Hash any PII before logs.

Chapter 11

Performance and Scalability

11.1 Targets

- Behavioral model inference: target under 500 ms.
- Train search latency: under 2 seconds for baseline concurrency.

11.2 Optimization suggestions

Warm models at service start, reuse inference process, cache common queries, tune Gunicorn worker count and DB pools.

Chapter 12

Testing and Validation

12.1 Test types

- Unit tests for feature aggregator and model wrapper.
- Integration tests for full login $\rightarrow$ search flow.
- Load tests for login and search endpoints, fuzz tests for malformed telemetry.

Chapter 13

Admin Dashboard and Monitoring

13.1 Dashboard features

Model version and metrics, denied attempts summary, retrain controls, CSV export for audit.

13.2 Monitoring

Export metrics to Prometheus and set alerts for latency spikes, inference errors, and model regression events.

Chapter 14

Risks and Mitigations

Risk	Mitigation
Behavioral spoofing	Multi-signal approach (CAPTCHA + behavior), continual retraining, threshold tuning
False positives blocking humans	Conservative initial thresholds, human-review flow, clear logging and rollback
Model drift	Time-aware retraining and monitoring for metric degradation
Data privacy breaches	Minimize retention of raw events, hashing, and access controls

Chapter 15

Appendices

15.1 Operational files and snippets

15.1.1 Requirements (example)

```
Flask
gunicorn
tensorflow
scikit-learn
joblib
pandas
numpy
celery
redis
psycopg2-binary
bcrypt
```

15.1.2 Useful artifacts (included images)

- Workflow.png — system workflow overview (used in architecture).
- Dataflow.png / Dataflow1.png — data flow diagrams.
- train_search.png, train_results.png, welcome_page.png — frontend mockups.
- demand_graph1.png, demand_graph2.png — sample demand graphs used in analysis.
- access_denied.png — access-denied page mockup.

15.2 Sample graphs

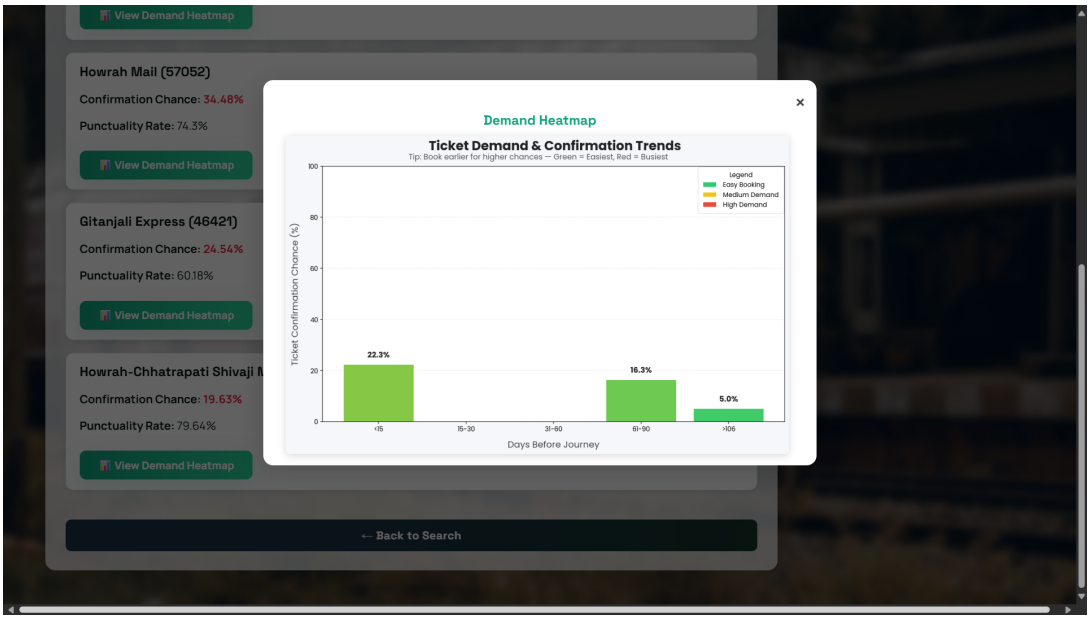


Figure 15.1: Results with predictions

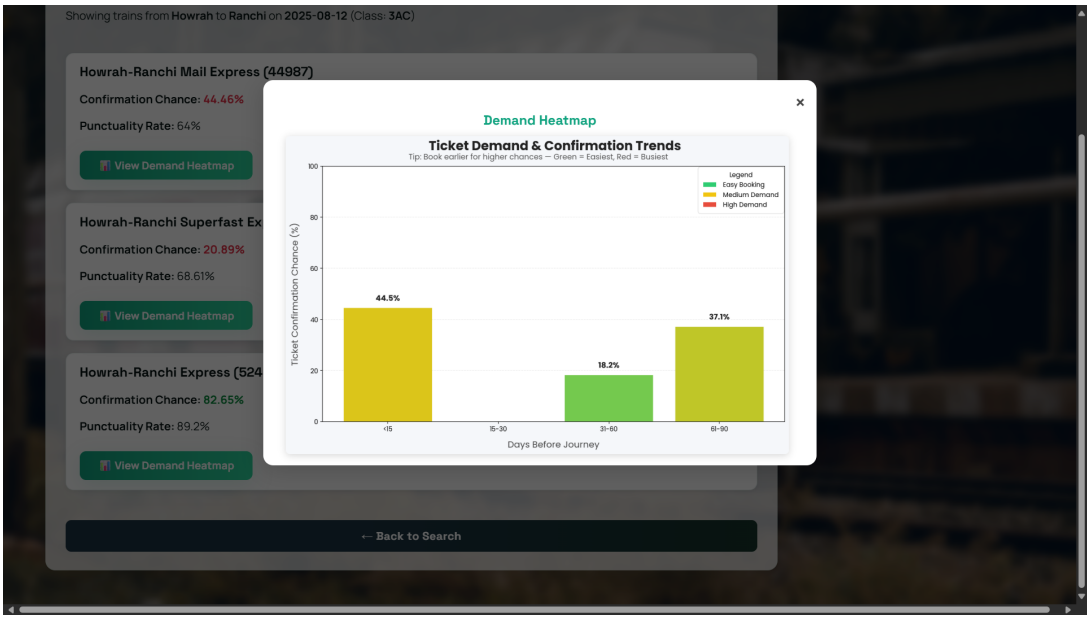


Figure 15.2: Demand heatmap — example 1

Notes

This document captures the architecture and component design required to implement the Hybrid IRCTC Bot-Detection and Train Search Web App.