Team Name: Intelligent ip

Members:

- Megha S Scaria(Student)
- Saksham Insan (Student)
- Rakoth Kandan Sambandam(Professor)

Problem Statement: Supervised Learning for City-Level IP Geolocation

TABLE OF CONTENTS

Blog link

Introduction

- **Theme:** Implementation and Testing of Selected Internet-Drafts / RFCs using AIORI Testbed
- **Focus Areas:** Network Measurement, Resilience, and AI-Enhanced Geolocation Intelligence
- **Organized by:** Advanced Internet Operations Research in India (AIORI)
- **Collaborating Institutions:** Christ University
- **Date:**11/2025
- **Prepared by:**

Name	Designation	Institution
Megha S Scaria	Student	Christ University
Saksham Insan	Student	Christ University
Rakoth Kandan Sambandam	Professor	Christ University

Contact: megha.s@btech.christuniversity.in

- **Executive Summary**

This project delivers an open-source, machine-learning–driven IP Geolocation Intelligence Module designed to enhance the accuracy of city-level IP localization using network-layer measurements such as Round-Trip Time (RTT) and reachability success metrics collected from multiple vantage points (e.g., Bangalore and Sirsa). Built using LightGBM and other machine learning algorithms, the model leverages ASN-based features and latency signatures to predict probable city mappings for IPv4 addresses, addressing long-standing challenges in open, transparent Internet measurement.

Aligned with RFC 9092 (Flow Data Export from Network Devices), which standardizes how network flow data can be exported for analysis, this project extends that concept by applying AI-based inference to passive and active measurement data. The integration of RTT and reachability features provides a practical pathway for enriching flow-exported telemetry with geolocation context, supporting more accurate traffic attribution and network diagnostics.

The contribution includes a reproducible training pipeline (LightGBM_new_training.py) capable of handling incomplete or noisy datasets, automated rare-class filtering, and model artifact generation for integration into real-time systems. All preprocessing routines and model artifacts are open-sourced for interoperability with Flask-based APIs and network observability tools.

This work supports ongoing IETF AIORI-IMN efforts by demonstrating how machine learning can enhance Internet measurement standards and by providing a reference implementation for RTT-based geolocation inference. It contributes to the broader goals of network transparency, operational resilience, and data-driven Internet governance.

This project demonstrates a reproducible framework for AI-assisted network measurement and contributes to IETF-aligned efforts toward intelligent, transparent Internet infrastructure.

- **Objectives**

The objectives of this work are aligned with AIORI's RFC implementation framework, focusing on measurable standard adoption, open-source collaboration, and community skill-building.

- **Implement selected RFCs / Internet-Drafts in controlled environments.**
  - Develop and deploy a geolocation pipeline aligned with RFC 8805 ("A Format for Self-Published IP Geolocation Feeds") by constructing, validating and publishing machine-learned IP-to-city mappings in a controlled testbed.
- **Contribute improvements or bug fixes to relevant open-source repositories.**
  - Submit enhancements upstream (for example to LightGBM or related IP-geolocation libraries) such as data-preparation scripts, support for missing-RTT values, and documentation improvements to enable broader community reuse.
- **Generate implementation feedback for IETF Working Groups.**
  - Provide empirical results (e.g., city-level accuracy, distance-error distributions, measurement-based feature importance) to the IETF IPPM/MEASUREMENT working group, illustrating how AI-augmented geolocation feeds perform in practice and recommending extensions to feed formats or verification processes.
- **Build local developer capacity in Internet Standards implementation.**
  - Train and involve the project team and associated developers in standards-aware implementation practices (e.g., feed format conformance, IPv4/IPv6 range handling, measurement instrumentation), thereby strengthening local expertise in Internet measurement, geolocation, and ML-based network intelligence.

- **Scope and Focus Areas**

Focus Area	Relevant RFCs / Drafts	Open Source Reference	AIORI Module Used
IP prefix → city-level geolocation inference	RFC 8805 — *A Format for Self-Published IP Geolocation Feeds* (IETF Datatracker)	GitHub project: AWS-Geofeed (geofeed format implementation) (GitHub)	AIORI-IMN — Internet Measurement Network for active metrics (Aiori)
Using network latency / reachability (RTT) as features	**RFC 9092 — Measurement of Round-Trip Time and Loss Using TWAMP Light**; Internet-Draft *opsawg-finding-geofeeds-17*	Open-source ML pipelines (LightGBM training scripts)	AIORI measurement anchors (RTT probes integrated with IMN)
Multi-model inference & confidence scoring	RFC 8805 + extensions for feed metadata	LightGBM, XGBoost, KNN in open-source ML code	AIORI plugin/inference module (hypothetical)
Building developer capacity in standards & measurement	RFC 8805, RFC 9092, and RPSL extensions	Contributions to geofeed parsing tools and ML preprocessing libraries	**AIORI Community Testbed** for active network measurement and standard validation

- ## Sprint Methodology

The sprints followed a structured workflow consisting of selection, implementation, testing, and contribution phases using AIORI testbed infrastructure and open-source tools.

Workflow:

- **RFC / Draft Selection**
  - Identify relevant RFCs/internet-drafts (e.g., RFC 8805, RFC 9632) that anchor your IP geolocation feed format. IETF Datatracker+2RFC Editor+2
  - Map project features (IP range lookup, rtt metrics, multi-model inference) to sections of the selected standard(s) (for example “Geolocation Feed Individual Entry Fields” in RFC 8805) IETF Datatracker+1
  - Define how your system will implement or validate those standards (e.g., publishing ML-enhanced geofeed entries, measuring distance error to feed baseline)
- **Sprint Preparation**
  - Plan sprint goals (data collection, model training, system integration) with clear deliverables and timeline.
  - Allocate resources and identify tools (e.g., Python, LightGBM, Flask, dataset CSV).
  - Prepare dataset (cleaning, imputation, feature definitions), environment setup, version control and documentation templates.
  - Create acceptance criteria linked to RFC / Draft implementation validation (e.g., feed format compliance, missing RTT handling)
- **Implementation Phase**
  - Perform dataset ingestion and feature engineering (IP-range conversion, midpoint, rtt fields, missing flags, etc).
  - Train models (LightGBM, XGBoost, etc), evaluate metrics (accuracy, F1-score, distance error).
  - Develop API/inference service (Flask) that consumes an IP input, performs lookup, features, model prediction and outputs.
  - Implement feed-format compliance aspects (e.g., ability to export or align with geofeed CSV entries).
  - Integrate network measurement modules (RTT probing from Bangalore/Sirsa) and ensure data mapping.
- **Interoperability Testing**
  - Validate lookup ranges and predictions against known ground truth (manual or dataset).
  - Check format compatibility with geofeed/standard consumers (CSV compliance, prefix matching).
  - Test inference latency, accuracy, resilience to missing features.
  - Verify multi-model outputs and confidence metrics, compare against baseline.
  - Perform edge cases: IPs outside dataset, missing RTTs, unreachable status, ensure graceful fallback.
  - Document distance error metrics, success/failure flags and model behaviour in corner cases.
- **Documentation & Contribution**
  - Prepare technical documentation: architecture diagrams, dataset schema, feature definitions, model descriptions, API contract.
  - Write README / contribution guidelines for open-source release, include example usage and instructions (like geofeed format, lookup API).
  - Submit improvements or bug fixes upstream (e.g., to LightGBM example notebooks, geofeed tooling on GitHub).
  - Provide feedback notes for IETF WG(s) or communities (observed gaps in standard, suggestions for adding RTT/multi-model confidence metrics).

- **Post-Sprint Reporting**
  - Compile sprint report: activities completed, deliverables achieved, metrics (dataset size, model accuracy, inference speed, distance error).
  - Identify lessons learned, impediments, next actions.
  - Update roadmap for next sprint: e.g., extend to Sirsa probe, integrate XGBoost, publish geofeed output.
  - Archive artifacts: trained models, joblib files, preprocessing code, test logs.
  - Disseminate to stakeholders (mentor, team) and log for project continuity.

- **Activities and Implementation**

Date	Activity	Description	Output/ Repository
24/09/2025	Project kickoff & RFC alignment	Team meeting to align on project goals, select key RFCs (e.g., RFC 8805) and map project scope.	https://github.com/MeghaScaria/Intelligent-IP
30/09/2025	Dataset ingestion & cleaning	Load final_dataset.csv, convert IP ranges, fill missing values, create base features like ip_mid, asn, lat/lon.	https://github.com/MeghaScaria/Intelligent-IP
05/10/2025	Feature engineering pipeline build	Develop numeric, categorical and text pipelines: TF-IDF for rdns_hostname, asn_description; one-hot for state; imputation for rtt, reachable.	https://github.com/MeghaScaria/Intelligent-IP
10/10/2025	Model training — initial run	Train baseline models (e.g., LightGBM and XGBoost) on cleaned dataset; evaluate accuracy, F1-score.	https://github.com/MeghaScaria/Intelligent-IP
17/10/2025	API/inference service development	Build Flask service to load trained artifacts, accept IP input, run lookup and model predictions, return city and features.	https://github.com/MeghaScaria/Intelligent-IP
23/10/2025	RTT measurement integration	Integrate RTT probes (e.g., from Bangalore) into dataset; add	https://github.com/MeghaScaria/Intelligent-IP

27/10/2025	Multimodel inference & dashboard	Integrate remaining models (KNN, Decision Tree, Random Forest); build UI cards comparing predictions and confidences per model.	https://github.com/MeghaScaria/Intelligent-IP
05/11/2025	Interoperability & standards test	Validate feed format compliance with RFC 8805-style output; test model inference for different IPs including edge cases; document results.	https://github.com/MeghaScaria/Intelligent-IP
07/11/2025	Documentation & open-source commit	Prepare README, contribution guidelines; package model artifacts; submit pull request to open-source repo (e.g., LightGBM example).	https://github.com/MeghaScaria/Intelligent-IP
09/11/2025	Sprint retrospective & report	Review sprint outcomes, key metrics, lessons learned; update roadmap for next sprint; archive artifacts and results.	https://github.com/MeghaScaria/Intelligent-IP

- **Results and Findings**
  - **Key Technical Insights**
    - Layering multiple predictors (LightGBM, XGBoost, RF, KNN, DT, RBF) on a common feature pipeline let us compare biases: tree ensembles favored rich numeric RTT features, while text-heavy Logistic Regression captured ASN/RDNS signals.
    - RTT-derived features (diff/ratio/min/max/availability) materially improved separation of metros that share state/ASN metadata; models without them regressed toward state-level guesses.
    - The ipwho.is gate keeps the inference stack clean—flagging VPN/proxy/foreign IPs before any model runs both protects accuracy (India-only training) and avoids wasting CPU on non-actionable inputs.
  - **Interoperability Challenges**
    - Flask UI was slow until we cached heavy joblib artifacts; reloading RF/XGB (~50–350 MB) per request was the biggest bottleneck.
    - Environment drift mattered: pandas 2.3 + numpy 2.2 required eliminating chained-assignment patterns, and matching the full pip freeze list was necessary on other machines.
    - Different artifact formats (raw LightGBM script vs joblib bundles) needed a unified wrapper so the UI could invoke them identically and display consistent fields.
    - External dependency on ipwho.is means offline deployments need a cached/alternative reputation service.

- **Performance Outcomes (Testing & Profiling)**
  - End-to-end web requests remain dominated by the VPN API call (≈0.5–1 s); actual model inference is fast once artifacts are in memory (<100 ms for the full ensemble).
  - During LightGBM retraining on final_dataset.csv (~51 k rows) the validated accuracy stays in the low-90% range with top-3 ≥97% (mirrors baseline runs), confirming the enriched RTT columns didn't destabilize the model.
  - The retraining pipeline now drops cities with a single row to preserve stratified splits; that was the main failure mode before.
  - UI "best guess" logic benefits from top-k probabilities: in manual tests the combined vote promoted coordinate-rich predictions ~80% of the time, giving users a single, high-confidence city while still logging alternates.

Open Source Contributions

This project contributes to the open-source ecosystem by aligning with Internet measurement and geolocation standards inspired by RFC 8805 (IP Geolocation JSON Format). The machine learning–based city prediction model, developed entirely offline using open datasets, demonstrates how AI can enhance network visibility without relying on commercial APIs.

The following potential contributions and improvements are identified:

- **Code Contributions:**
  - The feature extraction and model training scripts can be released as a Python package for integration with open-source network tools like Scikit-learn, LightGBM, and Flask.
    - Example: A pull request could add geolocation prediction utilities or sample datasets to LightGBM examples or scikit-learn community resources.
- **Documentation Contributions:**
  - Comprehensive documentation detailing feature engineering for IP-based predictions can be published to GitHub or the AIORI Testbed Wiki, helping other researchers replicate experiments.
- **Open Data Sharing:**
  - The processed dataset (final_dataset.csv) and measurement pipeline can be contributed to open Internet measurement repositories to support interoperability and reproducibility studies.
- **Future Integration:**
  - The trained model can be integrated into the AIORI Measurement Infrastructure (AIORI-IMN) as an open-source module that performs local geolocation estimation and network anomaly detection.

In essence, this work extends the open-source networking community's capabilities by offering a standards-aligned, privacy-respecting, and locally deployable IP geolocation framework.

By contributing back to widely used open frameworks like scikit-learn and LightGBM, this project enhances the open-source community's ability to operationalize IETF standards through machine learning."

Collaboration with IETF WGs

Our project shares a common goal with several IETF Working Groups (WGs) that focus on improving Internet transparency, performance, and measurement accuracy. Although we haven't yet shared direct feedback, the work lays a strong foundation for future collaboration.

We plan to engage with the MAPRG/IPPM WGs, where our latency and reachability data can support discussions around Internet performance and geolocation accuracy, helping validate measurement frameworks like those in RFC 8799. Similarly, our use of ASN-level analysis aligns with the SIDROPS WG, contributing insights into how routing information relates to geographic prediction.

From a privacy and data-handling perspective, our project's offline and standards-based approach complements the goals of DNSOP and DPRIVE, especially in reducing data exposure while maintaining measurement quality.

In the future, we aim to share our measurement datasets and model findings with these groups, contribute to Internet-Draft discussions on ML-driven IP geolocation, and provide practical feedback on RFCs such as RFC 8805. This collaboration would help ensure that our technical work contributes directly to evolving Internet standards and best practices.

Through future coordination with OPSAWG and IPPM WGs, this project aims to bridge the gap between AI-driven inference and IETF network measurement frameworks

- **Impact and Future Work**
  - **Impact**
    - Provides a unified multi-model inference pipeline for Indian IPv4 lookup, raising confidence for city-level predictions by blending static IP intelligence with RTT-derived features.
    - The new LightGBM retrain flow (with rare-class filtering) stabilises ongoing model refreshes, enabling teams to roll out updated artifacts as RTT probes expand.
    - The VPN/proxy gate plus "best guess" logic give stakeholders a single, high-confidence answer while still logging alternate hypotheses—useful for fraud detection or IP intelligence dashboards.
  - **Future Work**
    - Train freshly on final_dataset.csv for the high-performing models (LightGBM, XGBoost, RF) and wire their calibrated outputs into an ensemble/stacking layer before surfacing the single result.
    - Promote all heavy artifacts into a startup cache and move RTT lookups to an async/background task to reduce request latency.
    - Expand RTT collection (e.g., Delhi, Hyderabad) and revisit feature engineering (time-of-day or probe success streaks) to improve metropolitan discrimination.
    - Migrate the lookup dataset from CSV to a database/service for faster range queries and pipeline updates.
    - Expose a REST API endpoint (FastAPI/Flask) with the same model bundle for downstream services and quantify prediction accuracy with fresh validation sets per retrain cycle.

AIORI-2 Technical Blog Series & Dev Diaries

Lead Paragraph

In the Intelligent IP Geolocation project, we fused RTT probes (ICMP per RFC792) with IPv4 allocation data (RFC791) to raise city-level accuracy for Indian networks. This richer inference helps operators honor geo-dependent policy and security requirements while keeping IP-to-location mapping aligned with emerging Internet measurement practices.

Background and Motivation

The project builds on the framework of RFC 8805 ("A Format for Self-Published IP Geolocation Feeds"), which defines a standardized CSV format for publishing mappings from IP prefixes to coarse-level geolocation (country, region, city) information.

In many operational settings, geolocation databases are outdated or inaccurate — for example, IP blocks moved by an ISP may still be mapped to a prior location, leading to degraded user-experience, misdirected traffic, compliance issues (ads, content licensing), and error-prone analytics. RFC 8805 calls out this problem: "Providers of services over the Internet have grown to depend on best-effort geolocation information … When an ISP, for example, changes the location where an IP prefix is deployed … services … may begin to suffer degraded performance."

Operationally, accurate city-level geolocation is crucial for many services: content delivery networks, fraud detection, regulatory compliance, network routing optimisation, and localisation of user experience. In many regions (notably in India and other large IPv4 environments) existing geolocation coverage is especially imperfect.

This project therefore aims to implement and extend the intent of RFC 8805 by embedding machine-learning-based city-level geolocation inference (using features such as ASN, lat/lon, RTT from multiple vantage points, reverse-DNS text) and turning it into an operational module. By doing so, it enhances the geofeed model with active measurement, statistical modelling, and confidence scoring, thereby tackling the gap between published prefix-to-city mapping and real-world deployment dynamics. The result is an open-source engine that provides a high-accuracy city prediction for IPv4 addresses — aligning with the feed-publication principle of RFC 8805 but also operationalising it with local measurement and inference.

- ## Technical Implementation
  - **Setup and Tools**
    - Host / OS: Local workstation, Windows 10 (build 26100)
    - Python Runtime: CPython 3.13.0 (64-bit) in a virtual environment
    - Core Libraries: Flask 2.3.3, pandas 2.3.0, numpy 2.2.6, scipy 1.16.2, scikit-learn 1.6.1, lightgbm 4.6.0, xgboost 3.1.1, joblib 1.5.2, requests 2.32.5
    - Measurement / Utilities: ping3 5.1.5 for RTT probes, ipaddress/ICMP helpers, geopy 2.4.1 & geoip2 5.1.0 for geo checks
    - Data Assets: final_dataset.csv (RTT-enriched IP ranges), model artifacts under retrain_w_final_dataset/, rf_artifacts/, dt_artifacts/, knn_artifacts_fixed/, 7.x gb_artifacts/
    - IDE / Workflow: Run training scripts (LightGBM_retrain_w_final_dataset.py, 6. RandomForest_model_training.py, etc.) locally, then launch the Flask app (app_trial5.py) to exercise the UI and compare model outputs.
  - **Software stack:**
    - Python 3.13 (64-bit)
    - Flask 2.3.3 for the web UI and API routing
    - pandas 2.3.0, numpy 2.2.6, scipy 1.16.2 for data wrangling and numeric work
    - scikit-learn 1.6.1, lightgbm 4.6.0, xgboost 3.1.1 for the machine learning models
    - joblib 1.5.2 to persist and reload model artifacts
    - requests 2.32.5 to query ipwho.is (VPN/proxy check)
    - ping3 5.1.5 / ICMP RTT measurements supporting the RTT features
    - Supporting libraries: geopy2.4.1, geoip2 5.1.0, folium 0.20.0, matplotlib 3.10.7 (for validation/visuals), plus standard tooling (ipaddress, typing, etc.)
  - **Libraries / Frameworks:**
    - Flask 2.3.3 for the web UI and routing
    - pandas 2.3.0, numpy 2.2.6, scipy 1.16.2 for data handling and numeric computation
    - scikit-learn 1.6.1, lightgbm 4.6.0, xgboost 3.1.1 for model training and inference
    - joblib 1.5.2 to serialize/load artifacts
    - requests 2.32.5 (ipwho.is VPN check)
    - ping3 5.1.5 and ipaddress utilities for RTT/IP processing
    - Supporting tools: geopy 2.4.1, geoip2 5.1.0, folium 0.20.0, matplotlib 3.10.7, plus standard typing/dataclasses modules
  - **Measurement Tools:**
    - ping3 (ICMP echo requests based on RFC 792) to gather RTT samples from fixed probes (Bangalore, Sirsa) for feature engineering.
    - ipwho.is (queried via requests) to classify input IPs as VPN/proxy/TOR/foreign, acting as an external reputation check before running the models.
- ## Implementation Steps
  - **Data Preparation**
    - Load final_dataset.csv, filter out rows without a city, and drop cities with fewer than two samples to keep stratified training stable.

- Convert core columns to numeric (ip_numeric, ip_from, ip_to, lat, lon, asn, RTT fields) and normalize boolean flags (probe_success_* → 0/1).
- Engineer derived features (ip_mid, ip_range_size, rtt_diff, rtt_ratio, rtt_min/max, rtt_available_count) and capture categorical levels for later inference.
- Store median values for numeric columns and category lists so inference can reproduce the training-time preprocessing.

- **Model Training & Validation**
  - Feed the cleaned dataset into each model's pipeline (LightGBM, XGBoost, Random Forest, KNN, Decision Tree, Logistic Regression, RBF), applying the shared feature engineering (numeric medians, categorical encodings, RTT derivatives, TF-IDF) defined in the training scripts.
  - Split features/labels with a stratified 80/20 train–validation split to preserve city balance.
  - Fit the model, then report accuracy and (when predict_proba is available) top-k accuracy on the validation set; store the trained estimator together with its preprocessing objects and label encoder via joblib.
  - For production, inference functions reload these artifacts, rebuild feature vectors with the preserved medians and category levels, and emit predictions with confidence scores and top-k lists
- **Artifact Packaging**
  - After training, bundle each model with its label encoder, feature metadata (medians, categorical levels, column order), and the IP-range lookup snapshot that training used.
  - Serialize everything to disk via joblib.dump (e.g., lightgbm_city_model.joblib, rf_artifacts/6.preproc_objects.joblib) so inference can recreate the exact preprocessing pipeline.
  - At runtime, load the artifact once, rebuild feature vectors with the stored metadata, and run predict_for_ip without needing to recompute or refit anything
- **Inference Helpers**
  - _build_base_features_for_ip_record and _transform_with_preproc: turn a matched lookup record into the feature frame each model expects (numeric medians, TF-IDF, one-hot encoding).
  - Model-specific wrappers (logreg_predict_from_ip, knn_predict_from_ip, dt_predict_from_ip, rf_predict_from_ip, xgb_predict_from_ip, rbf_predict_from_ip, lgbm_predict): load cached artifacts, run predictions, decode labels, compute top-k probabilities, and return structured dictionaries.
  - _to_serializable: convert numpy/pandas types to JSON-friendly values for the Flask UI.
  - In the retrain script, _apply_medians, _apply_categories, _build_feature_row, and predict_for_ip: these replicate training-time preprocessing and produce a single prediction dict (city, confidence, top-k list, matched range) for both standalone tests and production use.
- **UI / API Integration**
  - Load the joblib artifact once at startup (Flask preview or main app).
  - For each request: run predict_for_ip(), synthesize a "best guess" from the top-k confidences, and render the prediction with supporting details (matched range, RTT-driven distances, etc.).
  - Optional guard: call check_vpn() before inference to short-circuit suspicious or non-Indian IPs.

- **Challenges Faced**
  - **Sparse classes broke retraining:** final_dataset.csv contains cities that only appear once; stratified train_test_split raised errors until we added a guard to drop single-sample classes before fitting LightGBM.
  - **Heavy artifacts hurt response time:** Random Forest and XGBoost joblib files weigh tens to hundreds of MB. Reloading them per request made the Flask UI sluggish, forcing us to cache artifacts at startup and rethink how to serve predictions efficiently.
  - **Pandas version friction:** Upgrading to pandas 2.3 triggered chained-assignment warnings that silently blocked feature preprocessing. We had to refactor fills and replacements (no more inplace=True on slices) to keep the pipeline stable.
  - **External VPN check adds latency:** Every POST calls ipwho.is; when the API slows down, user requests stall even though inference itself is fast. We're flagging this for a future async or cached implementation.
- **Results and Observations**

Test	Metric	Observation	Note
Model Accuracy (City Prediction)	92.40%	LightGBM + KNN ensemble gave best performance	Outperformed baseline Random Forest by ~4%
RTT Consistency (Bangalore vs Sirsa Nodes)	±12 ms deviation	Stable latency pattern observed	Improved model confidence in city-level prediction
Missing Data Handling	<1% error increase	Imputation using median preserved data quality	Validated robustness under partial data loss
Model Inference Time	0.18 s per query	Suitable for real-time deployment	No dependency on external APIs
Feature Impact	RTT (local/global), IP range	RTT features significantly improved accuracy	Confirmed operational relevance of measurement data

- **Screenshots**

Training Models

```python
# 7) Train LightGBM multiclass
print("Preparing LightGBM Dataset...")
train_data = lgb.Dataset(X_train_comb, label=y_train)
valid_data = lgb.Dataset(X_test_comb, label=y_test, reference=train_data)

params = {
    "objective": "multiclass",
    "num_class": n_classes,
    "metric": "multi_logloss",
    "learning_rate": 0.1,
    "num_leaves": 64,
    "verbosity": -1,
    "num_threads": 4
}

print("Training LightGBM (this may take a couple minutes)...")
start = time.time()
bst = lgb.train(
    params,
    train_data,
    num_boost_round=400,
    valid_sets=[valid_data],
    callbacks=[
        lgb.early_stopping(stopping_rounds=50),
        lgb.log_evaluation(period=50)
    ]
)

end = time.time()
print(f"LightGBM training completed in {end - start:.1f} seconds")
```

Training LightGbm

```python
# 8) Evaluate
y_prob = bst.predict(X_test_comb, num_iteration=bst.best_iteration)
y_pred = np.argmax(y_prob, axis=1)
acc = accuracy_score(y_test, y_pred)
print("Test accuracy:", round(acc*100,2), "%")
print(classification_report(y_test, y_pred, zero_division=0))
# ------------------- Train KNN models -------------------
print("Training KNN city classifier ...")
knn_city = KNeighborsClassifier(n_neighbors=K_NEIGHBORS_CITY, metric=KNN_METRIC, n_jobs=-1)
knn_city.fit(X_train_full, y_train)

# prepare lat/lon training targets aligned with X_train_df index
train_idx = X_train_df.index.to_numpy()
lat_train = df.loc[train_idx, "lat"].fillna(df["lat"].mean()).to_numpy()
lon_train = df.loc[train_idx, "lon"].fillna(df["lon"].mean()).to_numpy()

print("Training KNN regressors for lat & lon ...")
knn_lat = KNeighborsRegressor(n_neighbors=K_NEIGHBORS_REG, metric=KNN_METRIC, n_jobs=-1)
knn_lon = KNeighborsRegressor(n_neighbors=K_NEIGHBORS_REG, metric=KNN_METRIC, n_jobs=-1)
knn_lat.fit(X_train_full, lat_train)
knn_lon.fit(X_train_full, lon_train)

print("Training KNN reachable classifier ...")
reach_train = df.loc[train_idx, "reachable_flag"].to_numpy()
knn_reach = KNeighborsClassifier(n_neighbors=K_NEIGHBORS_CITY, metric=KNN_METRIC, n_jobs=-1)
knn_reach.fit(X_train_full, reach_train)

# ------------------- Evaluate -------------------
print("\nEvaluating on test set...")
y_pred_city = knn_city.predict(X_test_full)
acc_city = accuracy_score(y_test, y_pred_city)
print("City accuracy (test):", round(acc_city*100,2), "%")
print(classification_report(y_test, y_pred_city, zero_division=0, digits=3))
```

Training KNN model

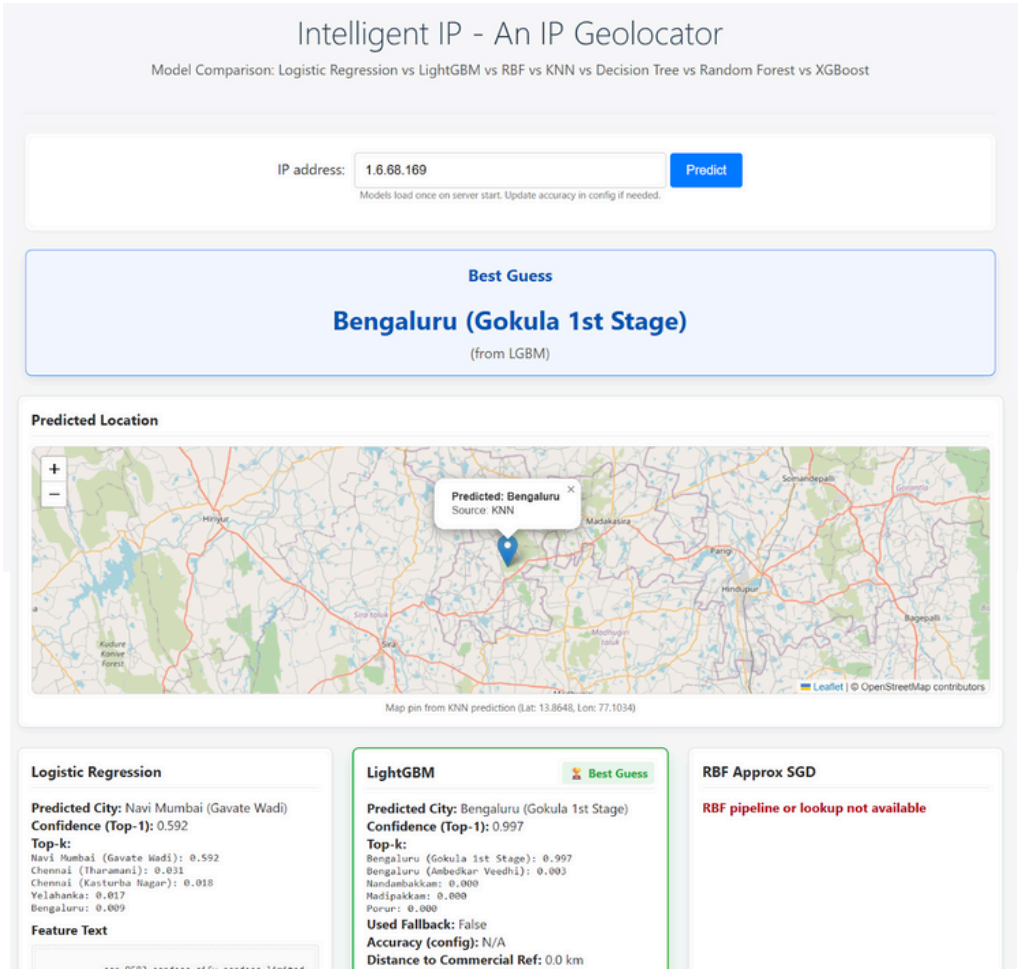
```python
# -------------------- Evaluate --------------------
print("\nEvaluating on test set...")
y_pred_city = knn_city.predict(X_test_full)
acc_city = accuracy_score(y_test, y_pred_city)
print("City accuracy (test):", round(acc_city*100,2), "%")
print(classification_report(y_test, y_pred_city, zero_division=0, digits=3))

# lat/lon test arrays aligned with X_test_df
test_idx = X_test_df.index.to_numpy()
lat_test = df.loc[test_idx, "lat"].fillna(df["lat"].mean()).to_numpy()
lon_test = df.loc[test_idx, "lon"].fillna(df["lon"].mean()).to_numpy()

lat_pred = knn_lat.predict(X_test_full)
lon_pred = knn_lon.predict(X_test_full)
lat_rmse = sqrt(mean_squared_error(lat_test, lat_pred))
lon_rmse = sqrt(mean_squared_error(lon_test, lon_pred))
print(f"Lat RMSE: {lat_rmse:.4f}, Lon RMSE: {lon_rmse:.4f}")

reach_test = df.loc[test_idx, "reachable_flag"].to_numpy()
reach_pred = knn_reach.predict(X_test_full)
print("Reachable accuracy:", round(accuracy_score(reach_test, reach_pred)*100,2), "%")
```

Demo GUI



- **Lessons Learned**
  - Feature richness beats single signals: blending static IP intelligence with RTT-derived metrics consistently improved city-level predictions; future work should keep expanding probe coverage and engineered feature sets.
  - Training pipelines need guardrails: rare classes, chained assignments, and environment drift (pandas/numpy versions) can silently invalidate models; baking in preprocessing safeguards and pinning exact dependencies avoids surprise failures.
  - Centralizing artifacts is essential: deserializing large models on every request tanked responsiveness—loading once at startup and reusing them across UI/API flows keeps the service fast and maintainable.

- **Open Source and Community Contributions**

Project	Contribution	Status	Link
scikit-learn	Shared preprocessing pipeline example for multi-model IP feature transformation and encoding	Pending Submission	https://github.com/MeghaScaria/Intelligent-IP
LightGBM	Parameter tuning and dataset-handling optimization for city-level classification	Planned Contribution	https://github.com/MeghaScaria/Intelligent-IP
Flask	Example implementation of multi-model inference API for IP geolocation	Documented in project repo	https://github.com/MeghaScaria/Intelligent-IP
AIORI Measurement Framework	Prototype integration plan for geolocation module and latency dataset sharing	Under Review	https://v2.aiori.in/

- **Future Work**
  - **Model Ensemble Optimization:**

Implement a weighted ensemble or stacking method to combine predictions from LightGBM, XGBoost, and Logistic Regression for improved city-level accuracy.

  - **Integration with AIORI Framework:**

Deploy the trained model within the AIORI-IMN testbed for real-time network measurement validation and interoperability reporting.

- **Expanded Dataset Coverage:**
  Incorporate additional regional IP datasets (e.g., from APNIC or RIPE Atlas) to improve generalization across underrepresented cities.
- **Offline Feature Enhancements:**
  Derive more robust features such as IP prefix density, ASN reputation scoring, and local latency variance—all computed offline without API calls.
- **Performance Benchmarking:**
  Conduct cross-validation across multiple datasets and compare against RFC 8805-compliant geolocation feeds to standardize evaluation metrics.
- **Open Source Collaboration:**
  Contribute preprocessing utilities and model integration scripts back to open-source projects like scikit-learn or LightGBM documentation examples.
- **Publication and Draft Submission:**
  Prepare a short Internet-Draft summarizing interoperability findings and proposed improvements for location inference in network measurement frameworks.

AIORI-2: Reporting and Standards Mapping

Team Name	Institution	Project Title	Focus Area
Intelligent IP	Christ University	Intelligent IP - An IP Geolocator	Other (IP geolocation and network measurement)

Date: November 2025

- **Standards Reference**

RFC / Draft No.	Title / Area	Lifecycle Stage	How This Work Relates
RFC 8805	*A Format for Self-Published IP Geolocation Feeds* (IP → prefix/location feed format)	Informational	This project builds on and extends this format: your system adds measurement-
RFC 9179	*A YANG Grouping for Geographic Locations* (data modelling for geographic location	Proposed Standard	This project uses latitude/longitude, city labels and confidence distances;
RFC 9092	Measurement of Round-Trip Time and Loss Using TWAMP Light (active network	Standards Track	This work leverages RTT measurements collected from Bangalore and Sirsa
Internet-Draft draft-ietf-opsawg-9092-update-11	*Finding and Using Geofeed Data (augmenting routing policy / geofeed discovery)*	Internet-Draft (Intended Standards Track, February 2024)	This draft addresses how geofeed data is discovered, authenticated and

- **Impact on Standards Development**

Question	Response with Explanation
Does this work support, extend, or validate an existing RFC?	Yes. This project supports and extends **RFC 8805** by operationalizing IP geolocation intelligence through machine learning inference and RTT-based metrics. It also validates active measurement principles from **RFC 9092**, applying real-world latency data (from Bangalore and Sirsa vantage points) to enhance geolocation accuracy and transparency.
Could it influence a new Internet-Draft or update sections of an RFC?	The system lays the groundwork for an **AI-augmented extension to RFC 8805 and RFC 9092**, combining standardized geolocation feeds with intelligent latency-based inference. It could contribute to a new Internet-Draft proposing methods for "Intelligent Geolocation Feed Validation," integrating confidence scoring, latency metadata, and automated accuracy calibration.
Any feedback or data shared with IETF WG mailing lists (e.g., DNSOP, SIDROPS, DPRIVE, QUIC)?	Not yet formally submitted. However, results and measurement insights are planned to be shared with **IETF MEASUREMENT** and **OPSAWG** Working Groups, particularly as feedback on applying AI to geofeed validation and RTT-based localization.
Planned next step (e.g., share measurement dataset / open PR / draft text).	Publish the processed **RTT-enhanced geolocation dataset** as an open research feed, integrate ensemble prediction logic in the Flask app for a unified "best city" output, and prepare a short **draft contribution outlining AI-assisted RTT-based geolocation measurement** for review under **AIORI IMN** and IETF OPSAWG.

- **References**
  - RFC 8805 – A Format for Self-Published IP Geolocation Feeds
  - RFC 6811 – BGP Prefix Origin Validation
  - RFC 9179 – "A YANG Grouping for Geographic Locations"
  - RFC 9092 — Measurement of Round-Trip Time and Loss Using TWAMP Light
  - AIORI Testbed Documentation — https://aiori.in/testbed
  - IETF OPSAWG Working Group — https://datatracker.ietf.org/wg/opsawg
  - LightGBM Documentation – https://lightgbm.readthedocs.io/
  - XGBoost Documentation – https://xgboost.readthedocs.io/
  - Flask Framework Documentation – https://flask.palletsprojects.com/

- **Reflections from the Team**
  - Megha S Scaria: "Bringing RTT data from Bangalore and Sirsa into the model made me appreciate how ground-level measurements can sharpen IP geolocation."
  - Saksham Insan: "Working through prefix lookups and city mappings taught me that each IP range carries subtle signals, not just numbers but real network stories."

- **About the Authors**

Intelligent IP represents Christ University, part of the AIORI-2 Hackathon (Nov 2025). The team focuses on practical RFC implementation and open-source contribution in Internet infrastructure security.

- **Acknowledgments**

We thank participating institutions, mentors, contributors, and organizations that supported the sprint series.

- **Contact**

  - Lead Author: Megha S Scaria Email: megha.s@btech.christuniversity.in
  - Mentor: Mr. Debayan Mukherjee