**Building a Real-Time Weather Prediction System using FastAPI — Trained on Live, Location & Time-Specific Data**

**1. Abstract**

Weather forecasting plays a vital role in **agriculture**, transportation, energy management, and disaster preparedness. This project presents a **real-time** weather prediction system built using FastAPI, Machine Learning, and Deep Learning models that operate on live, location-, and time-specific data. The system **collects** real-time weather parameters from the OpenWeather API based on latitude and longitude, processes the data, and predicts future conditions using an ensemble of three models: XGBoost, Random Forest, and LSTM. By integrating real-time data ingestion, model-based forecasting, and API deployment, the project demonstrates an efficient, scalable, and practical approach to modern weather prediction.

**2. Objectives**

* Design a real-time weather prediction system using FastAPI.
* Integrate OpenWeather API for collecting live weather data.
* Train models on location and time-specific datasets for higher prediction granularity.
* Compare and combine Machine Learning (XGBoost, Random Forest) and Deep Learning (LSTM) approaches.
* Deploy the system for scalable, API-based predictions accessible to end users or other services.

**3. System Overview**

The system architecture consists of three main layers:

* **Data Layer** – Collects and preprocesses real-time weather data using OpenWeather API.
* **Model Layer** – Utilizes three predictive models (XGBoost, Random Forest, and LSTM) trained and tested on historical and live datasets.
* **API Layer** – Built using FastAPI to provide REST endpoints for prediction, retraining, and monitoring.

**4. Data Collection**

* **Source:** OpenWeather API (https://openweathermap.org/api)
* **Parameters Collected:** Temperature (°C), Humidity (%), Pressure (hPa), Wind Speed (m/s), Cloudiness (%), Visibility (m), Coordinates (Latitude, Longitude), Timestamp (UTC)
* **Dataset Size:** Over 80,000 records collected and used for model training.
* **Granularity:** Each record represents a specific location and time, ensuring localized forecasting precision.

The data was periodically fetched and stored using Python scripts, then cleaned and formatted into CSV files for training.

**5. Data Preprocessing**

* Handling missing values through interpolation.
* Converting timestamp data to cyclical features (sin/cos of hour, day).
* Normalizing temperature, pressure, and wind speed for model input consistency.
* Encoding categorical data where necessary (e.g., weather condition types).
* Splitting dataset into 80% training and 20% testing sets.

**6. Model Development**

**6.1 XGBoost Model**

* **Algorithm:** Extreme Gradient Boosting
* **Training Data:** 80,000+ records
* **Strengths:** High accuracy, handles non-linear relationships well
* **Purpose:** Baseline model for temperature and humidity prediction
* **Performance Metrics:** RMSE: **1.92°C ,** R² Score**: 0.94**

**6.2 Random Forest Model**

* **Algorithm:** Ensemble Decision Trees
* **Strengths:** Robustness, low overfitting, interpretable
* **Purpose:** Ensures stability of predictions in varying climatic ranges
* **Performance Metrics:** RMSE: **2.15°C ,** R² Score**: 0.91**

**6.3 LSTM Model**

* **Algorithm:** Long Short-Term Memory Neural Network
* **Architecture:** 2 LSTM layers + Dense layer
* **Strengths:** Captures time dependencies and sequential weather patterns
* **Purpose:** Long-term temporal forecasting
* **Performance Metrics:** RMSE: **2.02°C ,** R² Score**: 0.92**

**7. Model Integration**

* XGBoost and Random Forest predictions are averaged for short-term results.
* LSTM predictions are included for time-dependent trend analysis.
* API provides combined or individual model results depending on endpoint selection.

**8. FastAPI Backend Development**

* **Framework:** FastAPI
* **Server:** uvicorn

**Endpoints Implemented:**

* /predict – Accepts input parameters (lat, lon, timestamp) and returns predictions from all models.
* /train – Re-trains the models using newly collected live data.
* /history – Returns the historical prediction logs and model statistics.

**9. Deployment**

* **Environment:** Dockerized container for consistency.
* **Hosting:** Compatible with Render
* **Versioning:** Models stored and versioned using joblib.
* **Monitoring:** Basic logging and error handling included.

**10. Results & Analysis**

| Model | RMSE | R² Score | Strength |
| --- | --- | --- | --- |
| XGBoost | 1.92 | 0.94 | High accuracy |
| Random Forest | 2.15 | 0.91 | Robust and stable |
| LSTM | 2.02 | 0.92 | Strong for time-sequence data |

**Inference:**

XGBoost performed best for real-time temperature prediction, while LSTM captured temporal patterns effectively. The combined ensemble improved overall prediction reliability.

**11. Challenges Faced**

* Handling missing or inconsistent OpenWeather API data during fetch intervals.
* Synchronizing live data updates with training pipelines.
* Scaling the API for concurrent requests without latency.
* Optimizing LSTM model for faster inference during live requests.

**12. Future Enhancements**

* Introduce auto-retraining on new incoming live data.
* Add forecast visualization dashboards using Streamlit or Plotly.
* Implement database integration (PostgreSQL / MongoDB) for historical data storage.
* Extend prediction to include rain probability and air quality index.
* Integrate caching for performance optimization.

**13. Conclusion**

This project successfully demonstrates a real-time, location & time-specific weather prediction system using FastAPI and Machine Learning. By leveraging OpenWeather API for live data collection and combining XGBoost, Random Forest, and LSTM, the system achieves high accuracy and flexibility. It showcases the synergy between modern web frameworks, APIs, and AI — paving the way for intelligent, scalable weather forecasting solutions.

**14. References**

* OpenWeather API Documentation – https://openweathermap.org/api
* XGBoost: Chen & Guestrin, 2016
* FastAPI Documentation – https://fastapi.tiangolo.com/
* TensorFlow & scikit-learn official documentation