Weather Forecasting System - Concept & Execution Notes

1. Project Overview

- **Goal:** Build a weather forecasting system using historical and live weather data.
- **ML Focus:** Time-series temperature prediction using LSTM (Long Short-Term Memory) neural networks.
- **API Hosting:** FastAPI service serving predictions.
- **Data Management:** ETL pipeline feeding new data into PostgreSQL.
- **Visualization:** Power BI dashboard (planned post-ETL phase).

---

2. Core Concepts & Decisions

2.1 Time Series Forecasting

- **Why LSTM:** Handles sequential dependencies in weather patterns better than classical ML models.
- **Input Shape:** (batch_size, time_steps, features) → (1, 24, 4) in this project.
- **Features Used:**
- temperature_2m
- relative_humidity_2m
- precipitation
- wind_speed_10m

2.2 Data Normalization

- **Purpose:** Standardizes feature scale for stable LSTM training.
- **Method:** MinMaxScaler (0–1 range).
- **Important:**
- Fit scaler only on training data.
- Denormalize model outputs before showing results.

2.3 FastAPI Service

- **Reason:** Lightweight, Python-native REST API framework.
- **Endpoints:**
- `/predict`: Accepts feature input, returns temperature prediction.
- `/health`: Basic service check.
- **Simplification:**
- Single feature input repeated as dummy sequence.
- True sequence input planned as a future improvement.

2.4 Model Format

- **Why .keras Format:** Official TensorFlow recommendation.
- **Why .h5 Format Used Locally:** Ensures compatibility with TensorFlow 2.12 (local version).

2.5 ETL Pipeline (Planned)

- **Purpose:**
- Automate data collection.
- Feed new weather data into PostgreSQL.
- **API Source:** Open-Meteo Archive API.
- **Scheduling:** GitHub Actions or local cron jobs.

2.6 Database Decision

- **Why PostgreSQL:**
- Supports external dashboard tools like Power BI.
- Scales better than SQLite for larger datasets.

---

3. Development Phases Summary

Phase 1: Model Development

- Collected 1 year of historical hourly weather data.
- Preprocessed using MinMaxScaler.
- Built and trained LSTM model.
- Evaluated MAE and RMSE.
- Saved model in .keras and .h5 formats.

Phase 2: API Development

- Built FastAPI app locally.
- Integrated model loading and prediction.
- Added denormalization using scaler min-max values.
- Confirmed local testing using Swagger UI.

Phase 3: ETL (Pending)

- PostgreSQL setup.
- Build ETL script.
- Connect ETL output to FastAPI and Power BI.

Future Improvements (Noted in System Memory)

- Accept true 24-hour feature sequences instead of dummy repeated input.
- Load scaler values dynamically via configuration instead of hardcoding.

---

4. Tools & Libraries Summary

Purpose	Tools
Data Collection	Requests, Pandas
Model Training	TensorFlow, Keras
API Service	FastAPI, Uvicorn
Data Storage	PostgreSQL
Dashboard	Power BI
Scheduling	GitHub Actions (Planned)

---

5. Notes

- Keep track of scaler values from training.
- Test API with normalized inputs if denormalization is not yet active.
- Always use chronological train-test splits for time series forecasting.