Project Description: IoT Data Streaming, Local & Global Model Training, and Analytics Pipeline

Project Overview

This project aims to build a complete IoT data processing and analytics pipeline that simulates **live streaming from IoT devices**, performs **per-device local model training**, aggregates these into a **global model**, and evaluates data with **batch and streaming analytics**. The system is designed to be **modular, scalable, and real-time**, allowing seamless addition of devices and expansion of analytics capabilities.

Project Objectives

1. **Simulate live IoT data ingestion** using an open-source dataset.
2. **Train local models per IoT device** to capture device-specific patterns.
3. **Aggregate local models into a global model** using federated learning principles.
4. **Evaluate batch and streaming data** using Apache Spark with the global model.
5. **Store and manage all data, models, and results** efficiently in MongoDB.
6. **Visualize per-device and global analytics** using Superset.

Data Source

- The project will use **Intel Lab Data**, a public IoT sensor dataset containing readings from multiple devices over time (temperature, humidity, light, voltage).

- The dataset is downloaded and preprocessed to simulate real-time streaming of sensor readings.

System Architecture

The system is composed of the following components:

1. Kafka – Live Data Streaming

- **Purpose:** Serve as the backbone for real-time data flow.
- **Functionality:**
  - Read rows from the dataset and send each row as a Kafka message to the topic `iot_stream`.
  - Each message contains:
    - `device_id`
    - Timestamp
    - Sensor readings (temperature, humidity, light, voltage)
  - **Loop over the dataset repeatedly**: once the dataset ends, start again from the beginning to maintain a continuous stream.
- **Output:** Kafka topic `iot_stream` for consumption by Flink and Spark.

2. Apache Flink – Per-Device Processing and Local Modeling

- **Purpose:** Process live streaming data and maintain device-specific models.
- **Functionality:**
  - Consume messages from Kafka topic `iot_stream`.
  - Group data by `device_id`.

- For each device:
  - Maintain a **local neural network**.
  - Perform **feedforward, backpropagation, and weight updates** continuously.
  - Send periodic model updates to the global model server.
- Dynamically handle **new devices**, creating models as needed.

- **Output:** Local model updates sent to the global server.

3. Local Neural Network Models

- **Purpose:** Capture patterns and anomalies specific to each device.
- **Structure:** Lightweight feedforward neural networks.
- **Training:** Incremental training on live streaming data.
- **Storage:** Local models are stored in `/models/local` and updates are sent to the global model server.

4. Global Model Server

- **Purpose:** Aggregate local models into a single global model.
- **Functionality:**
  - Collect model updates from local nodes periodically.
  - Perform federated aggregation to update the global neural network weights every half hour.
  - Maintain a **historical record of updates** with timestamps.
  - Provide the global model to Spark for batch and streaming evaluations.
- **Storage:** Global model stored in `/models/global` and MongoDB.

5. Apache Spark – Batch and Streaming Analytics

- **Batch Analysis:**
  - Read the full dataset to perform heavy analytics.
  - Evaluate predictions using the **latest global model**.
- **Streaming Analysis:**
  - Consume Kafka topic `iot_stream`.
  - Evaluate predictions in near real-time using the global model.
- **Output:** Store both batch and streaming predictions in MongoDB for visualization.

6. MongoDB – Data Storage

- **Collections:**
  - `local_models`: Updates from per-device models.
  - `global_model`: Aggregated global model weights.
  - `predictions`: Batch and streaming evaluation results.
- **Purpose:** Centralized storage to support analytics and visualization.

7. Visualization

- **Tool:** Superset
- **Visualizations:**
  - Per-device sensor readings and predictions.
  - Global model evaluation metrics.
  - Comparison between streaming and batch results.
  - Any other helpful insights

Data Flow

1. Dataset is **downloaded and preprocessed**.

2. Kafka streams each row of the dataset as a message to the topic `iot_stream`, and automatically loops back to the beginning once the dataset ends — continuously simulating a real IoT data stream since no live data source is available.

3. Flink **consumes messages**, trains local models, and sends updates to the global model server.

4. Global server **aggregates local updates** periodically, updates the global model, and stores it in MongoDB.

5. Spark **performs batch and streaming evaluation** using the global model and stores predictions in MongoDB.

6. Superset connects to MongoDB to **visualize results** for monitoring and analysis.

Project Structure

```
IoT-Streaming-ML-Pipeline/
│
├── data/
│   ├── raw/
│   │   └── intel_lab_data.csv              # Original downloaded
dataset
│   ├── processed/
│   │   └── processed_iot_data.csv          # Cleaned &
preprocessed version
│   ├── download_dataset.py                 # Script to
automatically download the dataset
│   └── preprocess_dataset.py               # Script to clean &
prepare data
```

```
|
├── kafka/
|   ├── kafka_producer.py                      # Reads dataset rows &
sends to Kafka topic
|   ├── kafka_consumer_test.py                 # Simple test consumer
(for debugging)
|   ├── docker-compose.yml                     # Kafka + Zookeeper
setup
|   ├── kafka_setup.sh                         # Helper script to
create topics & start broker
|   └── config/
|       └── kafka_config.json                  # Topic names, broker
addresses, configs
|
├── flink/
|   ├── flink_job.py                           # Main Flink streaming
job (consumes Kafka)
|   ├── flink_local_model_manager.py           # Handles per-device
local training logic
|   ├── flink_utils.py                         # Helper functions for
state mgmt & aggregation
|   └── requirements.txt                       # Flink-related
dependencies
|
├── models/
|   ├── local/
|   |   ├── model_template.py                  # Feedforward NN
architecture for local nodes
|   |   ├── device_001_model.pkl               # Example stored model
|   |   ├── device_002_model.pkl
|   |   └── ...
|   ├── global/
|   |   ├── global_model.py                    # Defines the global
model & aggregation logic
|   |   ├── aggregator.py                      # Aggregates weights
from local models
|   |   ├── global_model_weights.pkl           # Saved global model
weights
```

```
|   |   └── global_update_scheduler.py         # Runs every X minutes
to update the global model
|   └── utils/
|       └── model_utils.py                     # Shared functions for
model loading/saving
|
├── global_server/
|   ├── app.py                                 # FastAPI or Flask
server for handling updates
|   ├── endpoints/
|   |   ├── receive_update.py                  # Endpoint: receive
local model weights
|   |   ├── get_global_model.py # Endpoint: return the latest global
model
|   |   └── ...
|   ├── scheduler/
|   |   └── aggregate_job.py # Periodic job that triggers model
aggregation
|   ├── config.json                            # Server configurations
|   └── requirements.txt  # Python dependencies for global server
|
├── spark/
|   ├── spark_batch_analysis.py # Batch analysis using full dataset
|   ├── spark_streaming_analysis.py   # Stream evaluation using Kafka
data
|   ├── spark_global_evaluator.py # Evaluates global model performance
|   └── requirements.txt     # Spark-related dependencies
|
├── storage/
|   ├── mongodb_init.py    # Initialize MongoDB collections
|   ├── mongodb_connection.py     # Connection manager for MongoDB
|   ├── schemas/
|   |   ├── device_data_schema.json
|   |   ├── local_model_schema.json
|   |   ├── global_model_schema.json
|   |   └── predictions_schema.json
|   └── mongo_config.json   # DB configuration (host, port, db name)
|
```

```
├── visualization/
│   ├── superset_setup.sh                      # Setup Apache Superset
│   ├── dashboards/
│   │   ├── per_device_dashboard.json
│   │   ├── global_model_metrics.json
│   │   └── comparisons_dashboard.json
│   └── superset_config.py           # Connects Superset to MongoDB
│
├── orchestration/
│   ├── run_all.sh          # Starts the full pipeline end-to-end
│   ├── start_streaming.py           # Starts Kafka producer
│   ├── start_flink.py                # Starts Flink job
│   ├── start_global_server.py     # Starts global aggregation server
│   ├── start_spark_jobs.py      # Starts Spark jobs (batch +
streaming)
│   └── stop_all.sh             # Stops all running services
│
├── config/
│   ├── project_config.yaml       # General project-level settings
│   ├── paths.json           # Defines directories used in project
│   ├── scheduler_config.json  # Defines timing for global aggregation
│   └── environment.env         # Environment variables
│
├── utils/
│   ├── logger.py           # Centralized logging system
│   ├── metrics.py        # Functions for evaluating model accuracy
│   ├── helpers.py                # Generic utility functions
│   └── time_utils.py  # Functions for time formatting and scheduling
│
├── README.md              # Full documentation and setup instructions
├── requirements.txt         # Top-level dependencies
└── LICENSE                # Open-source license (MIT or Apache 2.0)
```

Technologies Used

- Python (scripts, ML models)
- Apache Kafka (streaming)
- Apache Flink (stream processing, local models)
- Neural networks (feedforward, backpropagation)
- Global model server (Python/Flask or FastAPI)
- Apache Spark (batch & streaming analytics)
- MongoDB (data storage)
- Superset (visualization)

Conclusion

This project establishes a **full end-to-end IoT data streaming and ML analytics system**, combining **real-time streaming, federated local model training, global model aggregation, and heavy analytics**, all integrated with **MongoDB storage and BI visualization**. The architecture is **scalable, modular, and designed for continuous improvement** as new IoT devices or datasets are added.