CropCast

DATA-DRIVEN FARM INSIGHTS

Data Engineering Group Project

Abstract

To design an automated system that predicts whether forecast for farmers and to predict their crop yield from their sensor's data using core principle of data engineering

LUMS

Contributions of Individuals

Faizan Ullah (23030015)

Led the **system architecture design** and implemented **Airflow DAGs,** airflow installation for orchestration. Added real time **logging** support for the Airflow dags via live teams alerts. Developed **Kafka producers/consumers**, configured **Apache Spark** for real-time processing, and integrated various components within the **AWS ecosystem** i.e. Ec2, s3 and RDS.

Salman Ahmed (24280002)

Designed and implemented the **machine learning pipeline**, including model training and evaluation. Also responsible for **scripting orchestration** to ensure seamless integration between data processing and prediction tasks.

Ahsan khan (24280044)

Contributed to the **system architecture** and assisted in developing the **ML pipeline and model**. Worked in pipeline scripts integration within the ecosystem and Took charge of **deploying the interface on Hugging Face Spaces**, enabling public access to the final product. Further took charge in coordinating with in the team remotely for better execution.

M Zafir (24280031)

Worked on developing the pipeline scripts, and their integration with sources and destination. Handled **Amazon S3** for object storage and set up the **PostgreSQL database** for structured storage and querying. Ensured proper integration of storage solutions within the overall architecture.

Project Overview

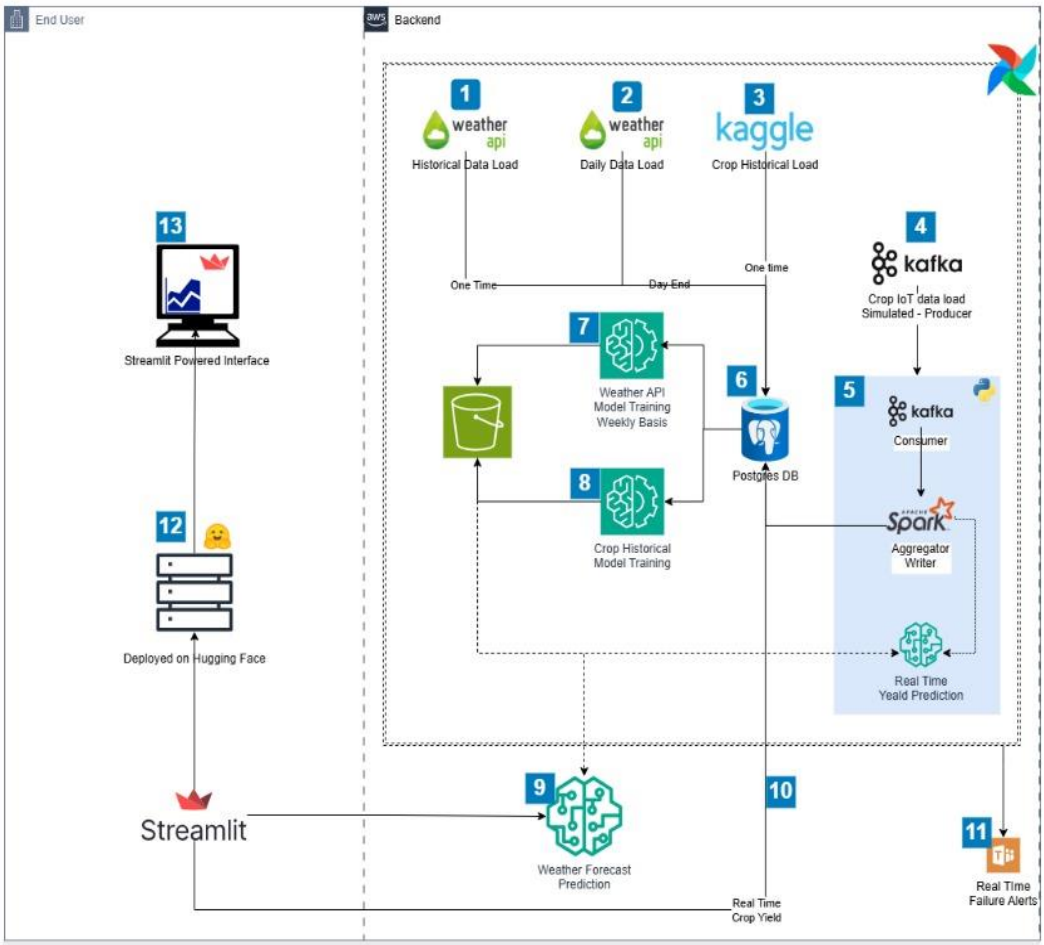
CropCast is an end-to-end data engineering project designed to assist farmers in making informed, data-driven decisions by providing short-term weather forecasts and crop yield predictions. The primary goal is to build a robust pipeline that integrates both real-time and historical data to offer actionable insights in the agriculture domain.

Leveraging historical weather data and time series forecasting models like Facebook Prophet, the system predicts 7-day forecasts for rainfall, temperature, and wind speed. Concurrently, a Kafka producer simulates the streaming of real-time farm attributes, such as sunlight exposure, rainfall levels, soil type, farm area, and fertilizer usage, which are consumed and processed using Apache Spark.

A yield prediction model, trained on a publicly available Kaggle dataset, ingests these features to estimate expected crop output. The entire data flow is orchestrated via Apache Airflow, enabling automation, scheduling, and real-time failure alerts (integrated with Microsoft Teams). Results are visualized using a user-friendly Streamlit dashboard deployed on Hugging Face Spaces. This project lies at the intersection of agritech and climate-tech, offering a scalable and modular solution tailored for smallholder and entry-level farmers in data-scarce regions.

Architecture

The following diagram illustrates the end-to-end architecture of CropCast, detailing how data flows through various components across ingestion, processing, model inference, storage, and deployment. Each step in the data engineering pipeline has been annotated for clarity.



1 Historical Data Load (Weather API)

Initial model training utilizes historical weather data from **WeatherAPI**. This dataset provides essential insights into past weather patterns (rainfall, sunlight, storms, etc.) crucial for training a robust forecasting model

2 Daily Weather Data Update (Weather API)

Daily updated weather data is fetched from **WeatherAPI**, enabling periodic retraining or fine-tuning of the forecasting model. Regular updates ensure predictions stay accurate and relevant to current conditions.

3 Crop Historical Data Load (Kaggle)

Historical crop data from **Kaggle datasets** is collected for initial model training. This data helps in building models capable of accurately predicting crop yields based on historical conditions and outcomes.

4 IoT-Based Crop Data Simulation (Kafka Producer)

Simulates real-time IoT-based crop sensor data streams using **Kafka Producer**. This simulated data mimics real-world sensor inputs (soil moisture, temperature, nutrient levels) enabling real-time yield predictions for specific farms.

5 Kafka Event Consumer with Spark Data Aggregator - Predicting Yeild

Kafka events are consumed, processed, and transformed in real-time. After transformation, the data is given to a model fetched form s3, the refined data is stored into the **PostgreSQL database** for streamlit to show real time prediction

6 Postgres Database

Centralized **Postgres DB** serves as a unified storage for structured data, hosting historical weather and crop data, and real-time processed data. This database acts as the single source of truth for data used in model training and predictions.

7 Weather Forecast Model Training (Weekly)

The weather forecasting model (using Prophet/ARIMA) is retrained weekly using newly collected and updated weather data. This ensures ongoing accuracy and adaptability to changing climatic conditions.

8 Crop Yield Model Training (Kaggle Data)

The crop yield prediction model is trained on the historical crop data sourced from **Kaggle**. This trained model becomes the baseline for yield prediction, which is further enhanced with real-time data.

9 Weather Forecast Model Deployment (Streamlit)

The trained weather forecasting model is deployed and exposed through a **FastAPI** service hosted on Hugging Face. This API delivers weather predictions (rainfall, sunlight, etc.) to the front-end interface in real-time.

10 Real-Time Yield Prediction Fetching

Fetching yield prediction from the Database which spark has written in the postgres database, after using crop prediction model

11 Real-Time Pipeline Failure Alerts

The data pipelines orchestrated using **Airflow** are continuously monitored. In the event of any pipeline failure, real-time alerts are automatically sent to the development team's communication channel (e.g., Teams), facilitating prompt issue resolution.

13 User Interface (Streamlit)

End-users interact with the system through an intuitive and interactive dashboard built with **Streamlit**. This user-friendly interface provides clear visualizations and detailed analyses of weather forecasts, real-time yield predictions, and insights tailored specifically for farmers.

12 Model Serving via Hugging Face

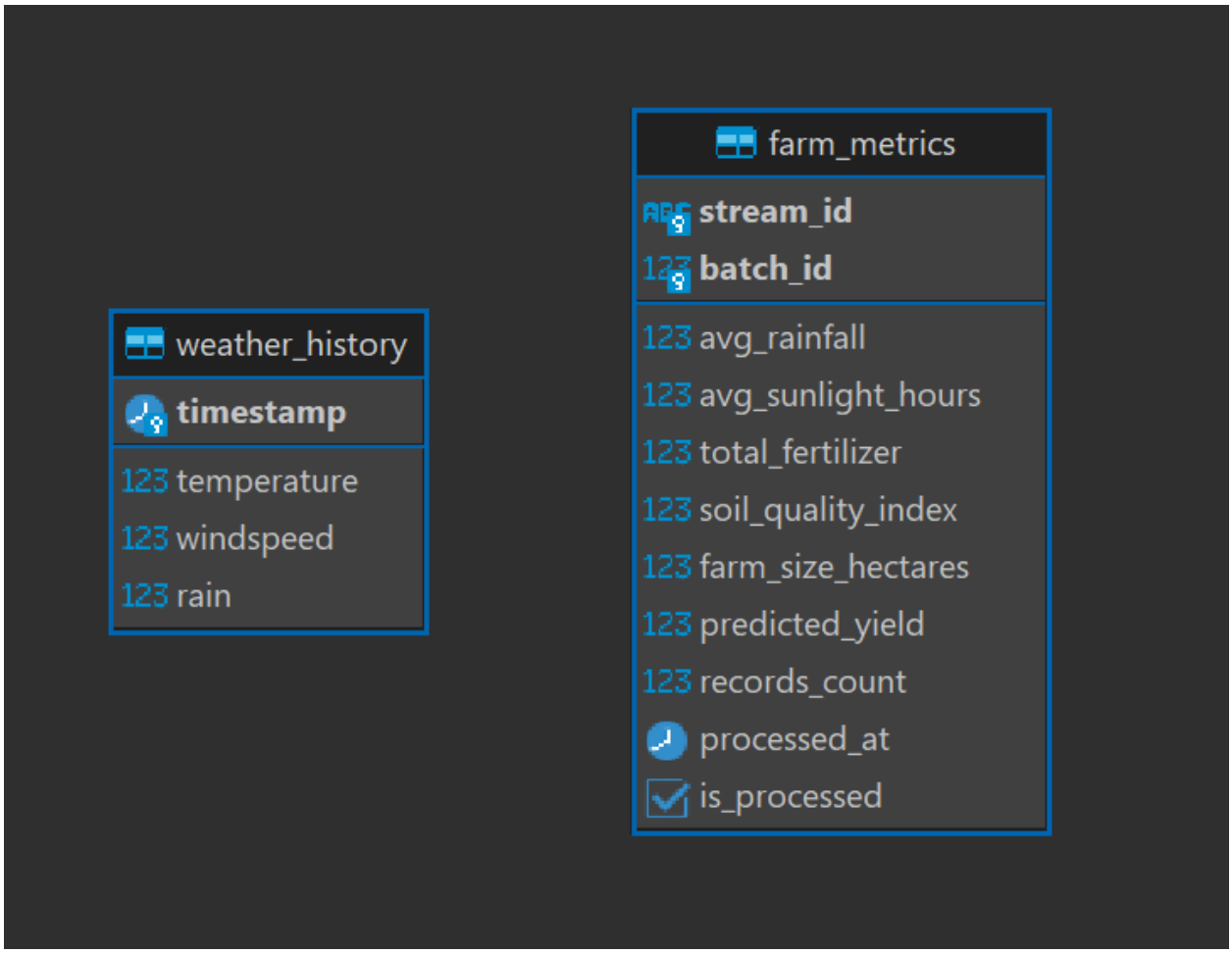
Both yield prediction and weather forecast data are exposed as plotly graphs using **Streamlit** and hosted on **Hugging Face Spaces**. This facilitates scalable deployment and easy integration with various applications.

Refer to the numbered components for a step-by-step breakdown of how real-time and historical data are handled to produce actionable weather forecasts and yield predictions.

Together, these modular components form a scalable, fault-tolerant system tailored for precision agriculture and climate-aware farming

Schema Diagram

The database schema for *CropCast* consists of two primary tables: weather_history and farm_metrics, each designed to capture different but interrelated aspects of the system's functionality. Furthermore, we have used object storage like S3 to save other additional data of our project, like CSV, pkl models etc.



---

1. weather_history

This table stores historical weather observations used for training and forecasting via time series models.

Column	Description
timestamp (PK)	The date and time of the recorded weather observation. Serves as the primary key.
temperature	Measured air temperature at the given timestamp.
windspeed	Wind speed recorded at the timestamp.
rain	Amount of rainfall in millimeters.

These features are fed into forecasting models like Prophet to generate 7-day predictions used in yield estimation and planning.

2. farm_metrics

This table stores aggregated and processed farm-level metrics which are used as input features for the crop yield prediction model.

Column	Description
stream_id (PK)	Unique identifier for each real-time data stream ingestion.
batch_id	Identifier for the batch in which this record was processed.
avg_rainfall	Average rainfall over the period for the given stream/batch.
avg_sunlight_hours	Aggregated sunlight hours during the monitored window.
total_fertilizer	Amount of fertilizer used (in kg or units).
soil_quality_index	A numeric index representing the quality of the soil (scaled or scored).
farm_size_hectares	Size of the farm in hectares.
predicted_yield	Crop yield prediction output from the ML model (e.g., kg/hectare).
records_count	Number of raw records from kafka aggregated into this entry.
processed_at	Timestamp of when the data was processed.

Column	Description
is_processed	Boolean flag to indicate whether this entry has been processed and predicted.

This table represents the output of the data pipeline after Kafka ingestion, Spark processing, and ML inference. It supports both traceability and further analytical queries.

Data Engineering Lifecycle Overview

1. Data Collection & Ingestion

Data for weather forecasting was collected using the weatherapi, chosen for its free access, open usage policy, and detailed historical and forecast data. It provided reliable hourly data on variables like temperature, windspeed, and rainfall. For yield prediction, simulated real-time sensor data, including sunlight hours, soil type, fertilizer usage, and farm size, was generated and ingested using Apache Kafka. Kafka was selected for its distributed, fault-tolerant architecture and seamless handling of high-throughput, real-time data streams, making it ideal for scaling to multi-farm environments.

To ensure data completeness and quality during ingestion, basic validation checks were implemented. Missing numeric values were imputed using the mean imputation method, balancing simplicity with effective retention of critical records for model accuracy.

2. Data Cleaning, Transformation & Quality

Once ingested, the data underwent transformation using vanilla python and using Apache Spark for streaming data, enabling processing of both historical and real-time data. Sensor values were aggregated in micro-batches, and temporal features were engineered for time-series modeling.

3. Data Storage & Modeling

We utilized PostgreSQL as our centralized storage solution due to its cloud-hosted flexibility, structured schema support, and integration with analytical tools like Spark. It allowed all team members to collaboratively access, inspect, and query ingested datasets with ACID-compliant integrity.

Model artifacts and filing data outputs were stored on AWS S3, providing scalable, durable, and cost-efficient object storage. This separation of hot (PostgreSQL) and cold (S3) storage

optimized access patterns across different parts of the pipeline.

4. Machine Learning

For forecasting weather, we trained a Facebook Prophet model on historical API data. Prophet was selected due to its robustness in capturing seasonality and trends in time-series data with minimal tuning. For crop yield prediction, we developed a two-layer dense neural network, trained on a curated Kaggle dataset. This model consumed features from simulated sensor streams and provided quick, interpretable yield predictions in near real-time.

Both models were kept lightweight to reduce computational overhead and latency, enabling fast inference within a streaming context.

5. Streaming & Real-Time Processing

Simulated Real-time data was continuously ingested using Kafka, with each record tagged by batch and stream identifiers. These were processed using Apache Spark Structured Streaming, which performed rolling aggregations and feature extraction. The architecture supports scalability and fault tolerance, allowing it to ingest sensor data from multiple farm units with minimal latency.

6. Pipeline Orchestration

Apache Airflow was used to schedule, manage, and monitor each stage of the pipeline. DAGs were configured for daily weather fetches, periodic streaming aggregation, batch processing, model inference, and storage. Airflow ensured pipeline reliability and traceability, while custom alerting (via Microsoft Teams integration) enabled timely failure detection and incident response. The modular DAG structure also simplified debugging and future scalability.

7. Deployment & Frontend Interface

The system was deployed using Streamlit, chosen for its simplicity in building interactive dashboards with minimal frontend overhead. The application was hosted on Hugging Face Spaces, making it publicly accessible without provisioning extra infrastructure. This lightweight deployment approach caters well to end-users like farmers who might use mobile devices or low-end systems.

The interface features two core views:

1. *Weather Forecast Tab* – Displays 7-day projections of rainfall, temperature, and windspeed.
2. *Yield Prediction Tab* – Visualizes yield estimates using Kafka-fed, real-time features such as sunlight, soil index, rain, farm size and fertilizer.

Links

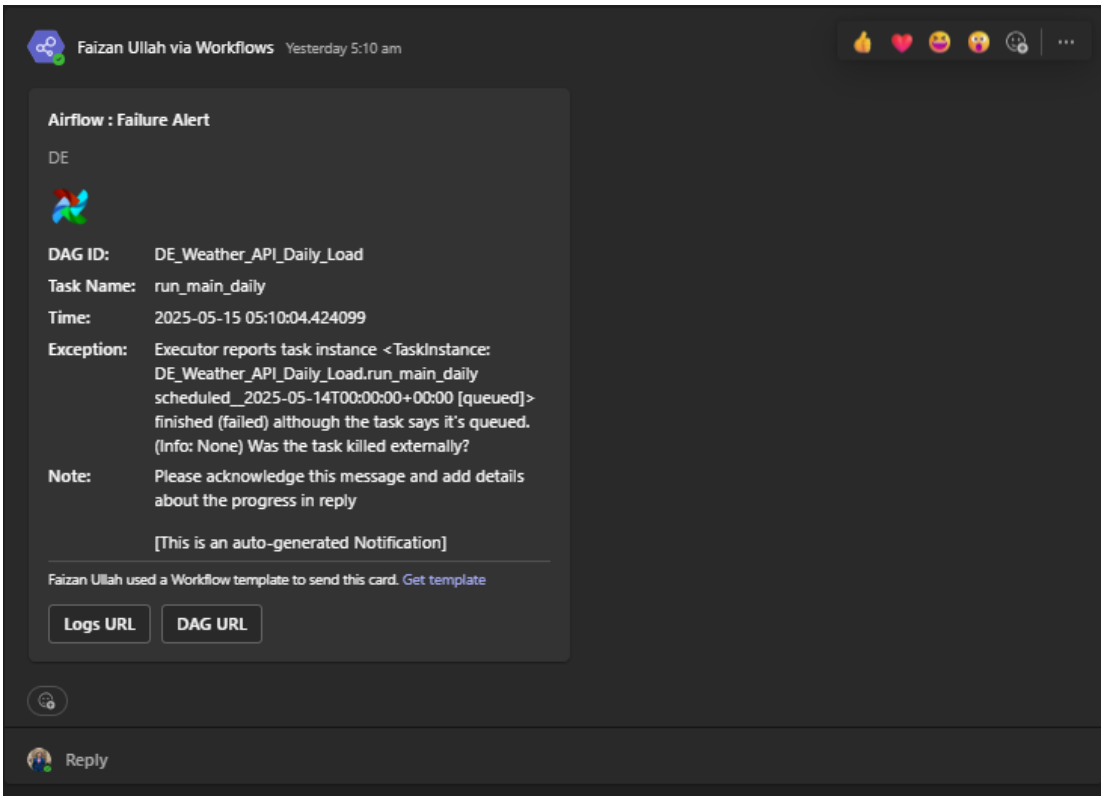
- Deployed Demo: **https://huggingface.co/spaces/AI71/DE2**
- GitHub Repository: **https://github.com/FaizanSh/cropcast-de-gp**

AI Use Across the DE lifecyle

AI, specifically Large Language Models (LLMs), were actively utilized throughout the development of this project. During coding, we encountered challenges across multiple platforms—including Airflow, Kafka, PostgreSQL, and Hugging Face Spaces—for which we consulted LLMs to receive guidance, debug errors, and understand best practices. This support significantly streamlined our development process, especially on unfamiliar tools.

- **Data Ingestion**: GPT & Copilot helped us structure our API calls, handle missing values, and simulate farm sensor streams through Kafka, even suggesting realistic feature values.
- **Data Cleaning & Spark**: Understanding what kind of cleaning is required and how we can use spark to transform the simulated data. Exploring option to save data so downstream system can process it. Presenting the Architecture and conversing with it. We previously planned on predicting later, GPT suggested to use it while parsing the simulated crop data.
- **Schema & Storage**: We drafted our PostgreSQL schema, then asked GPT to sanity-check our decisions, and even got help generating clean DLL and exploring options when we got issues in inserting same batch name after a failure.
- **ML Models**: From model choice (Prophet vs. ARIMA vs. LSTM) to understanding how to keep our neural net light, we debated our options with GPT before writing any code.
- **Airflow & Orchestration**: GPT helped us with installation guide. However, it wasn't too sure about the logging mechanism and suggested emails. However, we explored the teams linkage from the web and then it helped us formulate the structure of

teams events.



- **Deployment**: It compared platforms for us, helped structure our Streamlit UI, and clarified Hugging Face deployment issues.
- **Creative Work**: From generating our **logo prompts**, to **writing this report**, to **naming the project "CropCast"**, AI was everywhere, but always with us directing into which direction and how it should focus.

Furthermore, the documentation, including what you are currently reading, was initially drafted by the team and then paraphrased and refined using an LLM to enhance clarity, coherence, and readability. All AI-generated content was reviewed and validated by the team to ensure technical accuracy and project alignment. This approach allowed us to maintain high-quality documentation while accelerating our workflow.