Project Scoping Submission IE7374

ClimaSmart: AI-driven Weather Prediction

Team Members:

- Darshan Jayeshbhai Shah
- Devansh Doshi
- Haritha Anand
- Keshika Arunkumar
- Muskan Naresh Jain
- Pramoth Guhan

1. Introduction

ClimaSmart is a next-generation AI-powered weather prediction system designed to provide highly accurate, real-time weather forecasts alongside personalized recommendations. The system offers features such as Weather-Based Commute Suggestions, Outdoor Adventure Recommendations, and an interactive chatbot for real-time queries about weather conditions.

The project leverages a combination of cutting-edge machine learning models like CNN, LSTM, RNN, and XGBoost, processing both real-time and historical weather data sourced from OpenWeather, Tomorrow.io, NOAA, and weather.gov. The platform is designed with CI/CD (Continuous Integration/Continuous Deployment) in mind, ensuring smooth and reliable updates to the data pipeline, model training, and the application.

By implementing a robust CI/CD pipeline using GitHub Actions, integrated with Airflow, the system automates testing, building, and deployment. Docker is used for containerization, allowing the system to run efficiently across different environments, while Kubernetes handles the deployment, scaling, and orchestration of these containers, ensuring the application can handle high traffic without degradation. For data storage and management, Snowflake serves as our primary data warehouse, and we plan to use a cloud platform such as AWS, GCP, or Azure, depending on the scalability and cost-efficiency requirements.

The backend of the system is built with FastAPI to handle real-time weather data and recommendations, while the frontend is developed using Streamlit, offering users a clean, interactive interface to receive forecasts and insights. Integrating advanced tools like DVC, MLflow, and TFDV, the platform ensures that data versioning, model tracking, and validation processes are seamless and efficient.

2. Dataset Information

Dataset Introduction:

ClimaSmart utilizes both real-time and historical data collected from multiple weather sources such as OpenWeather, Tomorrow.io, weather.gov, and NOAA. The dataset includes parameters like temperature, humidity, wind speed, precipitation, and pressure. These weather parameters will be used to train machine learning models that predict future weather conditions while generating actionable recommendations, such as what to wear or when to plan outdoor activities.

Historical data from OpenWeather (45 years of data) and NOAA will help train models to capture seasonal and temporal patterns. Real-time data from Tomorrow.io and weather.gov will ensure accurate and up-to-date forecasts. These datasets will be ingested and processed using Apache Airflow DAGs to automate the data pipeline.

Data Card:

Size: The dataset will grow continuously, with real-time updates occurring every hour from multiple APIs.

Format:

- JSON for real-time weather data from OpenWeather, Tomorrow.io, and weather.gov.
- CSV for historical data from NOAA and OpenWeather.

Data Types:

- Numerical: Temperature, humidity, wind speed, atmospheric pressure.
- Categorical: Weather conditions like sunny, rainy, cloudy, etc.

Data Sources:

- OpenWeather API: Provides minute-by-minute, hourly, and daily weather forecasts, alongside 45 years of historical data.

- Tomorrow.io API: Hyper-local weather data, essential for short-term forecasting.
- weather.gov API: U.S. weather data, including forecasts and weather alerts.
- NOAA: Historical data, crucial for training long-term predictive models.

Data Rights and Privacy:

The APIs comply with privacy regulations such as GDPR, and the project will adhere to the usage guidelines set by OpenWeather, NOAA, Tomorrow.io, and weather.gov. All data is used under appropriate licenses, adhering to pay-as-you-go models for higher API usage.

3. Data Planning and Splits

Loading:

Snowflake will serve as the primary data warehouse, offering scalability and speed. Data will be ingested via automated pipelines built with Airflow and will be regularly updated with real-time data pulled from APIs. Cloud storage options such as AWS S3, Google Cloud Storage, or Azure Blob Storage will be used to store intermediate datasets or raw data that may not need immediate processing but is required for long-term storage or archival purposes.

Preprocessing:

- TFDV (TensorFlow Data Validation) will validate the data pipeline by generating descriptive statistics and identifying data anomalies. It will help us monitor data consistency, ensuring real-time data is comparable with the training data.
- Preprocessing tasks will involve handling missing values, detecting outliers, and synchronizing timestamps for time-series forecasting. Features such as rolling averages for temperature and lag features will be created to capture the temporal dependencies.
- Airflow DAGs will automate the preprocessing tasks, executing scripts that prepare data for training or deployment.

Managing Data:

- Data Version Control (DVC) will be implemented to track changes in the datasets. DVC ensures that each version of the dataset can be reverted, compared, or reused in future experiments, which is critical for reproducibility and transparency in the machine learning development process.
- Snowflake will manage the processed and versioned data for efficient querying and analysis. It will centralize data storage for easy access by machine learning models during both training and prediction phases.

Splitting:

The dataset will be split into training, validation, and test sets using a time-series-based approach. Historical data from NOAA and OpenWeather will train the models, while recent data from Tomorrow.io and weather.gov will be used for validation and testing.

4. GitHub Repository

The GitHub repository will serve as the central hub for project code, documentation, and CI/CD pipelines. The repository will be organized into logical modules:

- README: Project setup instructions, usage guides, and environment configurations.
- /data: Contains datasets managed with DVC.
- /src: Scripts for data preprocessing, model training, and API handling.
- /models: Trained models, versioned with MLflow.
- /pipeline: CI/CD configuration for automated testing, model deployment, and containerization.

5. Project Scope

Problems:

Need for Real-Time, Hyper-Local Weather Insights: Users require highly accurate, real-time weather insights to make informed decisions about commuting, outdoor activities, and travel. Hyper-local predictions (e.g., city-block level, specific to smaller areas, like neighbourhoods) are becoming increasingly important for users who rely on minute-to-minute updates for planning their day.

Static Forecasts in Current Weather Apps: Current weather apps provide generalized, static forecasts and lack dynamic, personalized recommendations that account for a user's specific needs, location, or planned activities. Users also lack a real-time chatbot interface to query specific weather-related questions and receive tailored advice.

Lack of Robust Data Versioning and Model Management: Existing solutions often ignore the critical need for data versioning, making it difficult to track the datasets used in model training. This affects reproducibility and model performance monitoring over time, especially when training on rapidly changing data.

Manual Updates are Inefficient: Manually updating datasets and models is prone to error and delays. A fully automated system is required for continuous operation, ensuring that models stay up-to-date with the latest data and consistently deliver accurate predictions.

Current Solutions:

General Forecasts Without Context: Existing weather applications focus primarily on providing generalized forecasts, without taking user context into account, such as commute optimization, outdoor event planning, or personalized recommendations.

No Integration of Historical Data: While real-time forecasts are important, many apps fail to leverage historical data to predict long-term trends or seasonality, limiting their predictive accuracy over time.

Neglect of Data Versioning: Current apps and systems often fail to properly track and version datasets, which hampers the ability to reproduce machine learning experiments and track model evolution. This makes it difficult to trace model performance issues back to changes in data or model configurations.

Proposed Solutions:

Integration of Real-Time and Historical Data: ClimaSmart integrates both real-time weather data and historical data from sources like OpenWeather, Tomorrow.io, and NOAA, allowing for more accurate and granular predictions. Real-time data helps users with immediate decision-making, while historical data helps train models to understand long-term patterns (e.g., seasonal changes).

Personalized, Actionable Recommendations: The system will provide personalized insights based on user-specific needs, such as recommending the best times for outdoor activities or suggesting alternate routes based on weather-related road conditions. Additionally, the chatbot will allow users to interactively ask about weather forecasts and receive real-time, tailored responses.

Automation via Airflow, DVC, and CI/CD: The Airflow DAGs automate the data ingestion, preprocessing, and model retraining process. DVC (Data Version Control) is used to track changes in the datasets and ensure that all datasets are versioned, which enables reproducibility of machine learning experiments. The CI/CD pipeline, powered by GitHub Actions, Docker, and Kubernetes, ensures that model updates and data pipelines are continuously integrated and deployed with minimal human intervention.

TFDV for Data Validation: TFDV (TensorFlow Data Validation) will be used to ensure that real-time data remains consistent with the training dataset. It will detect anomalies and discrepancies, ensuring that the models continue to perform optimally, and that incoming data doesn't corrupt the predictions.

6. Current Approach Flow Chart and Bottleneck Detection

Flowchart:

The entire ClimaSmart project follows a well-structured, multi-phase workflow, from data ingestion to model deployment and real-time prediction delivery. The flowchart captures the steps of this end-to-end process, divided into key stages as outlined below:

1. Data Collection and Ingestion:

- Real-time and historical weather data are pulled from OpenWeather, Tomorrow.io, NOAA, and weather.gov APIs.
- Airflow DAGs automate the process of fetching data at regular intervals.
- Data is stored in Snowflake for centralized storage and scalable access.

2. Data Preprocessing and Validation:

Data Preprocessing:

- Airflow DAGs automate data cleaning tasks: handling missing values, removing duplicates, and normalizing timestamps.
- TFDV (TensorFlow Data Validation) is used to generate data statistics and detect anomalies.

Feature Engineering:

- Create rolling averages for temperature and wind speed.
- Generate lag features for time-series analysis.
- Encode categorical weather conditions (e.g., sunny, rainy) for model input.

3. Model Training and Versioning:

Model Training:

- Models like CNN LSTM, RNN, and XGBoost are trained using the preprocessed data stored in Snowflake.

Model Performance Check:

- Evaluate models using performance metrics like accuracy, precision, recall, and F1-score.
- Models are tuned or retrained if performance falls below a threshold.

Model Versioning:

- The best-performing models are versioned and stored in Snowflake using MLflow for tracking.

4. CI/CD and Model Deployment:

CI/CD Pipeline: The entire pipeline is automated using GitHub Actions to continuously test, build, and deploy code and models.

Containerization: The application is containerized using Docker, ensuring consistent environments from development to production.

Kubernetes Deployment: Kubernetes manages scaling, orchestration, and deployment of Docker containers. It also handles high-availability needs.

5. Frontend Integration:

Frontend (Streamlit): Streamlit is used to create an interactive user interface that displays real-time weather predictions and recommendations.

Model Serving: The backend models are served through Kubernetes, providing low-latency predictions to users.

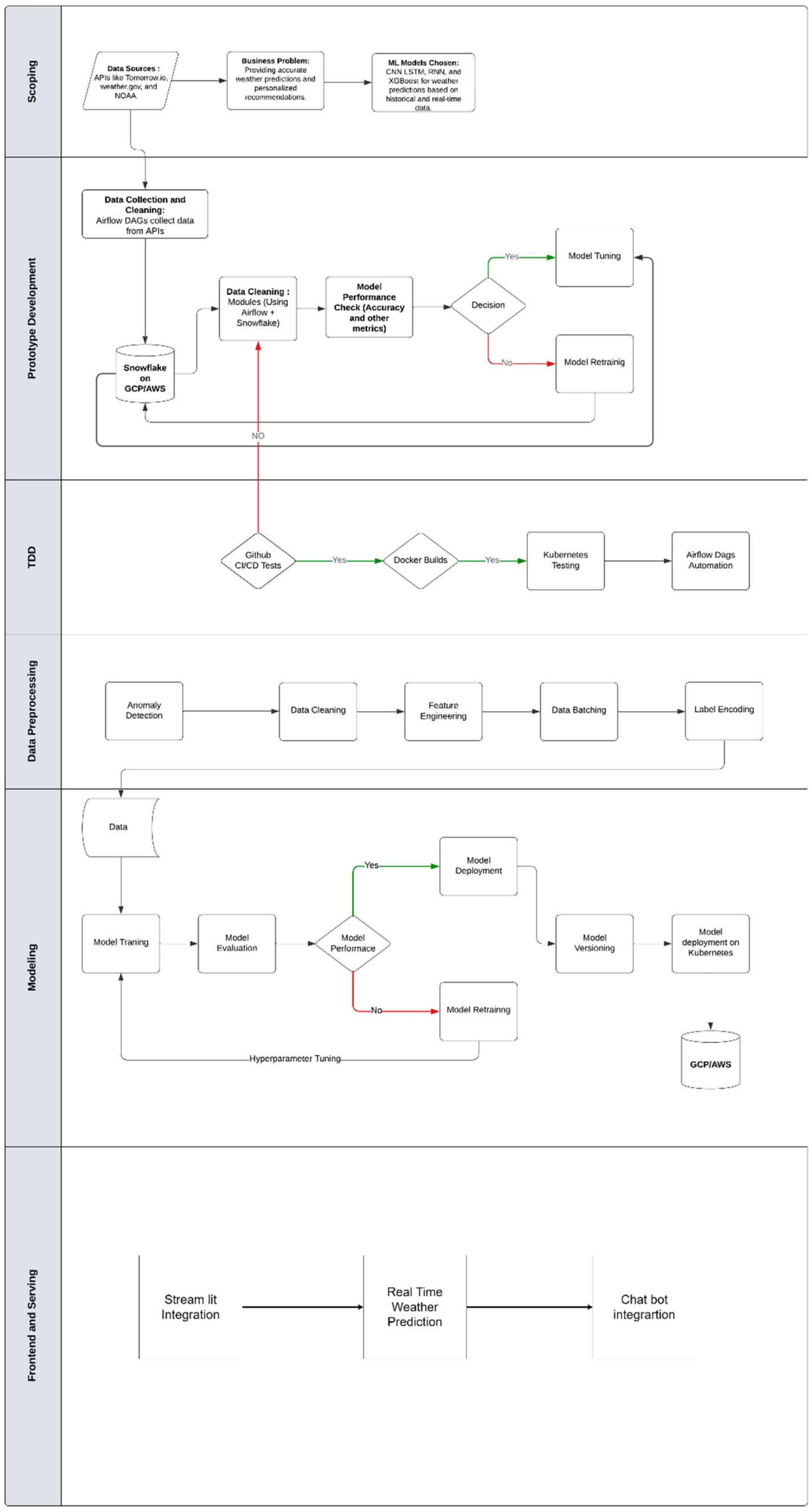
Chatbot Integration: A chatbot (using Dialogflow or Rasa) is integrated to provide real-time, conversational interactions with the user regarding weather predictions.

6. Monitoring and Alerting:

System Monitoring: Tools like Prometheus and Grafana monitor system performance (CPU, memory, network usage) across the infrastructure.

Airflow Monitoring: Airflow provides task monitoring, failure alerts, and retries for its DAGs to ensure smooth data flow.

Model Performance Monitoring: MLflow tracks model performance over time, ensuring models continue to meet accuracy requirements.


Scoping
Data Sources : APIs like Tomorrow.io, weather.gov, and NOAA.
Business Problem: Providing accurate weather predictions and personalized recommendations.
ML Models Chosen: CNN LSTM, RNN, and XGBoost for weather predictions based on historical and real-time data.
Prototype Development
Data Collection and Cleaning: Airflow DAGs collect data from APIs
Snowflake on GCP/AWS
Data Cleaning : Modules (Using Airflow + Snowflake)
Model Performance Check (Accuracy and other metrics)
Decision
Yes
Model Tuning
No
Model Retrainig
NO
TDD
Github CI/CD Tests
Yes
Docker Builds
Yes
Kubernetes Testing
Airflow Dags Automation
Data Preprocessing
Anomaly Detection
Data Cleaning
Feature Engineering
Data Batching
Label Encoding
Modeling
Data
Model Traning
Model Evaluation
Model Performace
Yes
Model Deployment
No
Model Retrainng
Hyperparameter Tuning
Model Versioning
Model deployment on Kubernetes
GCP/AWS
Frontend and Serving
Stream lit Integration
Real Time Weather Prediction
Chat bot integrartion

Bottlenecks:

1. Data Latency: Real-time weather data collection from multiple APIs may introduce delays in fetching and processing the data.

Solution:

- Airflow DAGs are optimized to run parallel tasks, ensuring that data collection is efficient.
- Snowflake's scalable architecture handles large datasets quickly, allowing real-time data ingestion without bottlenecks.

2. Data Quality and Consistency: Inconsistent or noisy real-time data can reduce model accuracy and increase errors.

Solution: TFDV is used to validate real-time data against the training data schema. If anomalies or inconsistencies are detected, Airflow triggers alert and retries to prevent poor-quality data from entering the models.

3. Model Accuracy and Overfitting: Models may underperform due to overfitting, poor data quality, or suboptimal hyperparameters, leading to low accuracy.

Solution: MLflow tracks all experiments, ensuring model performance is continually monitored. If accuracy drops below a set threshold, Airflow DAGs automatically trigger retraining with hyperparameter tuning. Experiments can be compared, and the best-performing models are selected for deployment.

4. CI/CD Pipeline Scalability: As the system scales, ensuring continuous integration, automated testing, and deployment of new models with low latency may become challenging.

Solution: Kubernetes handles auto-scaling of Docker containers, ensuring the infrastructure scales dynamically with user demand. The CI/CD pipeline (using GitHub Actions) automates the entire process from model versioning, testing, and deployment, ensuring a continuous workflow without manual intervention.

5. Frontend Latency and User Interaction: Ensuring that the frontend remains responsive with high user traffic while maintaining low latency for real-time predictions.

Solution: Kubernetes provides auto-scaling to ensure that backend services handle increasing user queries without bottlenecking. Streamlit optimizations and fast API responses from the model ensure that users receive predictions and recommendations in real-time, with minimal delays.

6. Pipeline Failures and Resource Utilization: Failures in Airflow tasks or Kubernetes resource exhaustion (e.g., CPU, memory) can lead to interruptions in the workflow.

Solution: Airflow has built-in retries and error handling for task failures, and Kubernetes auto-scales resources as needed. Monitoring tools like Prometheus and Grafana will alert the team if system resources are under strain, allowing for early intervention and resource adjustments.

7. Metrics, Objectives, and Business Goals

Key Metrics:

Prediction Accuracy: Achieve at least 90% accuracy in weather predictions.

Measurement: Track metrics like Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R2 score using MLflow to monitor model performance. Automatic retraining via Airflow DAGs will ensure models are updated to maintain accuracy.

Response Time: Ensure chatbot responses are within 2 seconds.

Measurement: System logs, Prometheus, and Grafana will monitor response times, ensuring low-latency interaction through the chatbot and API services, even under heavy load.

Data Consistency: TFDV will monitor the consistency between training and real-time datasets, detecting and reporting data anomalies.

Measurement: Regular reports from TFDV will highlight data drift and schema anomalies. Airflow DAGs will handle data validation, triggering retraining if necessary.

Automation Efficiency: Ensure that Airflow DAGs automate over 95% of pipeline processes, minimizing the need for manual intervention.

Measurement: Track task success rates and failures through Airflow's logs. Monitor retry counts and time-to-completion for each task, with a goal of reducing manual overhead.

Business Goals:

Differentiation with Personalized Weather Recommendations: ClimaSmart will stand out by offering real-time, personalized recommendations, such as weather-based commute suggestions and outdoor activity recommendations, tailored to user preferences and local conditions. This personalization will drive user satisfaction and retention.

Seamless Deployment and Scaling Using Docker and Kubernetes: Use Docker to containerize the system for consistent deployments, and Kubernetes to handle scaling and fault tolerance. Kubernetes' auto-scaling will manage fluctuations in user traffic, ensuring high availability and efficient resource use.

Operational Efficiency Through Automation: Automate the majority of processes, from data ingestion to model deployment, through Airflow DAGs, GitHub Actions, and CI/CD pipelines, ensuring that the system can scale with minimal manual oversight. This reduces both costs and operational complexity.

8. Failure Analysis

Risks:

Inconsistent Real-Time Data: Data discrepancies between real-time data and the training datasets may lead to inaccurate predictions, caused by missing data points, formatting changes in the APIs, or latency issues.

Model Drift: Over time, the model's performance may deteriorate due to changing weather patterns or seasonal shifts, causing the model to lose accuracy as the data distributions drift from those seen in the training phase.

Pipeline Failures: Airflow tasks may fail due to network issues, API rate limits, or infrastructure problems. This could lead to delays in data ingestion or preprocessing, ultimately causing disruptions in the predictions.

Infrastructure Scaling Challenges: During peak usage, the system may struggle to scale quickly enough to meet the demand, causing slow predictions or potential downtime if Kubernetes fails to scale in time.

Data Versioning Issues: Version mismatches between datasets and models could cause experiment reproducibility issues, leading to confusion or errors when comparing model performance across versions.

Mitigation:

Inconsistent Real-Time Data: Use TFDV to monitor and validate real-time data against the schema. Alerts will be triggered for anomalies or missing data, and fallback mechanisms with backup APIs will be implemented to ensure data continuity.

Model Drift: Track model performance using MLflow and schedule periodic retraining via Airflow DAGs. If model accuracy drops below a defined threshold, retraining will be automatically triggered to incorporate the latest data.

Pipeline Failures: Airflow's built-in retries will handle task failures. Kubernetes will auto-heal and restart any failed pods. Additionally, API rate monitoring will prevent rate-limit errors, ensuring smooth data flow.

Infrastructure Scaling Challenges: Kubernetes' Horizontal Pod Autoscaler (HPA) will manage auto-scaling based on traffic and system load. Monitoring tools like Prometheus and Grafana

9. Deployment Infrastructure

Cloud-Based Infrastructure:

Primary Data Warehouse: Snowflake will be the primary data warehouse, offering both scalability and performance for storing real-time and historical weather data. Snowflake's architecture will allow efficient data retrieval for training machine learning models and serving real-time predictions.

Cloud Storage Options: We will evaluate cloud platforms such as AWS (Amazon Web Services), Google Cloud Platform (GCP), or Azure to host the application and store real-time and historical data, depending on the project's requirements for scalability, cost, and available features.

- AWS S3 for storage: A widely used, cost-effective storage solution integrated with Snowflake for seamless data management.

- Google Cloud Storage or Azure Blob Storage as alternatives, depending on cost-efficiency and integration with other services.

Containerization Using Docker: Docker will be used to containerize the services, including data ingestion, model training, API serving, and the Streamlit frontend. By using Docker, the system ensures portability and consistency across different environments (development, testing, production). Docker images will be created for the individual components to allow seamless deployment across platforms

Orchestration and Scaling with Kubernetes: Kubernetes will handle the orchestration and scaling of Docker containers. Kubernetes ensures that as application usage grows, the infrastructure scales automatically to meet demand. It provides fault tolerance and high availability, ensuring that even under heavy loads, services remain responsive, and any failed pods are automatically restarted. Also manages load balancing across different microservices, ensuring efficient resource usage during peak traffic periods.

Airflow in Kubernetes: Airflow will be deployed as a service in the Kubernetes cluster. This ensures that DAGs (Directed Acyclic Graphs) scale with the workload, especially for heavy data processing or complex model retraining pipelines. Running Airflow in Kubernetes will also ensure resilience and scalability, allowing the system to handle large volumes of tasks (data ingestion, preprocessing, model training, serving, etc.) without interruption.

CI/CD Pipeline and Deployment Flow:

CI/CD Pipeline:

- The CI/CD pipeline will be automated using GitHub Actions. This will manage the processes of testing, building, and deploying Docker containers across the cloud infrastructure.
- The CI/CD pipeline will be configured to test every code commit automatically, ensuring that changes to the application, machine learning models, or data pipelines are verified and don't introduce issues in production.

Airflow DAGs for End-to-End Automation:

Airflow DAGs will automate the entire end-to-end pipeline, ensuring smooth data ingestion, preprocessing, model training, and model serving. Key tasks such as:

- Fetching new data from APIs.
- Validating data using TFDV.
- Retraining models when performance thresholds are met.
- Deploying new versions of models to Kubernetes.

The Airflow DAGs will ensure the entire machine learning pipeline is reproducible, scalable, and efficient without manual intervention.

Docker and Kubernetes for Deployment:

Docker images will be built and pushed to a container registry (e.g., Docker Hub, AWS ECR, Google Container Registry). Kubernetes will then pull these images and deploy them to the selected cloud infrastructure (AWS, GCP, or Azure).

This process ensures consistency in application behavior across different environments and allows for quick scaling of services in response to user demand.

Deployment Flowchart:

A deployment flowchart will be used to visualize the entire CI/CD pipeline. It will illustrate how data moves from Snowflake to the machine learning models and how predictions are served to the Streamlit frontend via FastAPI.

The flowchart will include the following steps:

1. Data is ingested and stored in Snowflake.
2. Preprocessing is automated via Airflow DAGs.
3. Models are trained and versioned (using DVC and MLflow).
4. Docker containers are built and pushed to a registry.
5. Kubernetes handles the orchestration and scaling of services.
6. Streamlit frontend receives real-time predictions via FastAPI.

10. Monitoring Plan

Metrics to Monitor:

Prediction Accuracy: Continuously track and report on model prediction accuracy using MLflow for experiment tracking and performance monitoring.

Data Validation: TFDV will ensure that real-time data remains consistent with training data, tracking metrics on data distribution and schema validation.

System Load: Monitoring tools will track CPU, memory, and network usage for Kubernetes clusters. This will ensure efficient scaling and resource management.

Model Drift: MLflow will track the performance of models over time, ensuring that any signs of model drift (reduced accuracy over time) are detected early.

Airflow Task Monitoring: Airflow has built-in monitoring to ensure DAG tasks are executed correctly. It will report task failures, retries, and completion times. This will be integrated with email alerts and cloud monitoring systems to notify the team of any failures.

Alerting:

Set up automated alerts using Kubernetes dashboards and cloud-native monitoring tools (AWS CloudWatch, GCP Monitoring, Azure Monitor) for:

Data Anomalies: TFDV will trigger alerts when data quality or consistency deviates significantly from the expected distribution, preventing the use of corrupted or skewed data in predictions.

Model Degradation: Alerts when model accuracy drops below a defined threshold, triggering retraining or rollback to a previous version.

Infrastructure Load: Alerts for high CPU, memory, or disk usage that may indicate scaling is needed.

Airflow Task Failures: Alerts will be configured to notify the team of any task failures in the Airflow DAGs, allowing quick troubleshooting and pipeline recovery.

Infrastructure Monitoring: Kubernetes dashboards, along with cloud-native monitoring tools (AWS CloudWatch, GCP Monitoring, Azure Monitor), will alert the team in case of infrastructure issues, such as high resource consumption or system downtime.

11. Success and Acceptance Criteria

Success Metrics:

- Prediction Accuracy: Achieving at least 90% accuracy in weather forecasts.
- Chatbot Response Time: Maintain a response time of less than 2 seconds for the chatbot during user interactions.
- Airflow Automation: Achieving 95% automation in the data pipeline via Airflow DAGs with minimal manual intervention.
- Scalability: Ensure the system seamlessly handles increased user traffic using Kubernetes for scaling without performance degradation.
- Data Consistency: Ensure no significant data skew or anomalies are detected during operations using TFDV.
- MLflow for Model Tracking and Versioning: Ensure robust tracking and versioning of models using MLflow, allowing for easy rollback in case of failures or performance degradation.

Acceptance Criteria:

The project will be considered successful if:

- The system can handle real-time weather data from multiple sources (e.g., Tomorrow.io, OpenWeather, NOAA, etc.) and provide accurate, personalized insights to users.
- The system demonstrates smooth scalability using Docker and Kubernetes, handling increased user loads and large volumes of data.
- The models remain accurate and adaptable through regular retraining and monitoring using CI/CD pipelines and MLflow.
- Airflow DAGs effectively automate the end-to-end pipeline, from data ingestion to model retraining and deployment.

12. Timeline Planning

- <u>October 1st - October 14th:</u> Set up the CI/CD pipeline with GitHub Actions. Begin data collection with Airflow DAGs and implement DVC for data versioning.
- <u>October 15th - October 31st:</u> Train initial machine learning models using CNN LSTM, RNN, and XGBoost. Implement MLflow for tracking model experiments and performance.

- November 1st - November 14th: Integrate models with FastAPI for backend deployment and Streamlit for the frontend. Complete integration of TFDV for data validation, ensuring real-time data matches training data.
- November 15th - November 24th: Conduct thorough testing and optimization of the entire pipeline. Automate data ingestion, preprocessing, and model retraining via Airflow DAGs. Set up scaling via Kubernetes on the selected cloud platform.
- November 25th - December 4th: Perform final tests, deploy to the cloud infrastructure, and prepare for the project presentation.

13. Additional Information

APIs in Use:

OpenWeather API: Provides real-time and historical weather data, including detailed weather conditions for various locations globally.

Tomorrow.io API: Hyper-local real-time weather data, essential for providing precise short-term predictions and recommendations.

NOAA: Offers extensive historical weather data, used for long-term model training to capture seasonal patterns.

weather.gov API: U.S.-based weather data, including hourly and daily forecasts and severe weather alerts, providing short-term predictive insights.

Tech Stack:

Frontend: Developed using Streamlit for an interactive, user-friendly interface.

Backend: FastAPI for serving predictions and handling requests.

Data Warehouse: Snowflake for centralized storage and data management.

Cloud Infrastructure: AWS, GCP, or Azure will be used to host the application and manage real-time and historical data.

Data Versioning: DVC for tracking dataset changes.

Model Tracking & Versioning: MLflow for experiment tracking and version control of machine learning models.

Data Validation: TFDV to ensure real-time data matches the training dataset and schema.

Automation: Airflow for orchestrating automated workflows via DAGs, ensuring that data ingestion, model training, and deployment are handled seamlessly.

Containerization: Docker to containerize the system, ensuring consistency across environments.

Orchestration & Scaling: Kubernetes to manage the deployment, scaling, and monitoring of Docker containers.

CI/CD Pipeline: GitHub Actions for continuous integration, ensuring automated testing, building, and deployment.