
Project Scoping Submission - Electricity Demand Forecasting

Team Members:

Jahnavi Mishra

Nikhil Sirisala

Rajat Keshri

Samanvya Tripathi

Aakash Mahesha

Amogha Gadde

-

1. Introduction

In this project, we aim to build an electricity demand forecasting model leveraging the full MLOps lifecycle. The model will predict electricity demand for a given location based on real-time weather data (wind speed, temperature, humidity), population density, and time-related features. The primary goal is to develop an efficient forecasting system capable of predicting short-term electricity consumption, which is crucial for grid stability and energy management.

To achieve this, we will gather data from multiple cities across the United States, each with varying weather patterns and power consumption habits. We can dynamically feed this information into our model for timely and location-specific predictions by using APIs to collect real-time weather and population data. Our pipeline will ensure data versioning, cleanliness, and consistency, while Airflow will orchestrate the data collection and model training workflows. MLflow will manage model versioning, validation, and performance tracking, ensuring the model remains accurate and up-to-date. Lastly, a user interface will allow users to interact with the model by inputting a location and receiving predictions for electricity demand over the next few hours.

This end-to-end system integrates data collection, processing, model development, and user interaction into a seamless MLOps solution for forecasting electricity demand in various U.S. cities.

2. Dataset Information -

1. **Dataset Introduction**: The dataset comprises hourly weather data, population density, and electricity demand from multiple U.S. regions over the past five years. The primary goal is to predict short-term electricity consumption based on real-time and historical weather conditions, population, and other external factors. By building a comprehensive dataset, this project aims to model the correlation between these factors and electricity demand, helping to optimize grid management.

2. **Data Card**: **Size:** Estimated ~40,000 data points per region, with hourly data for 5 years across multiple cities.

Format: CSV (processed from API responses)

Features:

- **Weather:** Hourly data including temperature, humidity, wind speed, pressure, etc.
- **Population Density:** High-resolution data (~1 km grid) for population density.
- **Location:** City name, latitude, longitude.
- **Time:** Hourly timestamps over a 5-year period.

Label: Electricity demand (measured in kWh or MWh).

Data Types:

- **Numerical:** Temperature, wind speed, population density, electricity demand.
- **Categorical:** City names.

3. **Data Sources**:

Weather Data: World Weather Online API offers detailed weather data for different cities, including temperature, wind speed, humidity, etc., on an hourly basis(World Weather Online).

Electricity Demand Data: EIA API provides access to real-time and historical electricity consumption data for different regions across the U.S. We would be focussing on Texas - ERCO (ERCOT - Electric Reliability Council of Texas) - which has 8 regions within it covering the entire Texas state. We would expand this to other regions and states as well.

Population Density Data: WorldPop API delivers population density data at high resolution (1 km grid), adjusted to match UN estimates, allowing us to account for population factors affecting electricity consumption(WorldPop).

4. **Data Rights and Privacy**: All data from the above sources is publicly available and can be accessed through APIs under open usage terms. WorldPop data is licensed under Creative Commons Attribution 4.0 International, which allows for redistribution and adaptation, provided proper attribution is givens. There are no

privacy concerns as the dataset does not include personally identifiable information (PII), and it aligns with data protection standards such as GDPR.

3. Data Planning and Splits

Our goal is to build a reliable electricity demand forecasting model. For this, we need an effective data pipeline that ensures clean, well-structured, and adequately split data for training, validation, and testing. Below, I outline the steps for **loading**, **preprocessing**, and **managing** the data, along with the strategies we'll use to split it appropriately.

Loading

1. **Data Collection**:
    - The data will be sourced using three APIs:
        - **Weather Data**: World Weather Online API will provide hourly weather data (e.g., temperature, humidity, wind speed) for different cities over the last five years.
        - **Electricity Demand Data**: The EIA API will supply electricity demand data for corresponding regions and time periods.
        - **Population Density Data**: WorldPop API will give population density information, updated annually at high resolution (~1 km).
    - **Storage**: After fetching the data, it will be stored in a CSV format, with time as the key index.
2. **Data Integration**:
    - Data from different sources (weather, demand, population) will be merged using city names and timestamps. The integrated dataset will align these different data streams to create a comprehensive dataset for modeling.

Preprocessing

1. **Handling Missing Data**:
    - **Imputation**: If any hourly weather data is missing, we will use linear interpolation to fill gaps, especially for time-series data where continuous hourly records are critical.
    - **Demand Data Gaps**: Any missing electricity demand will be treated carefully, possibly excluding those time periods if imputation may lead to inaccuracies.
2. **Feature Engineering**:
    - **Time-Based Features**: We'll generate features for hour of the day and holidays to account for the natural cyclicality of electricity demand.

- **Lagged Features**: For weather variables (e.g., temperature, humidity), lagged features will be created, e.g., weather from 1-3 hours prior, to capture temporal dependencies that impact electricity consumption.
- **Population Interaction**: Interaction features between population density and weather variables will help us better understand their combined effect on demand.

3. **Data Imbalance**:
    - If there's an imbalance between high and low demand periods (e.g., summer vs. winter), we may apply techniques like oversampling the underrepresented time periods or use SMOTE (Synthetic Minority Over-sampling Technique) to balance the data, especially during the training phase.

Managing Data

1. **Data Versioning**:
    - **DVC (Data Version Control)** will be employed to version both raw and processed datasets. This will allow us to track changes in the data, making it easier to reproduce results and manage different dataset versions used for model training.
2. **Efficient Data Storage**:
    - Once processed, the dataset will be stored in **Parquet** or **Feather** format, which allows for faster read and write operations compared to traditional CSV formats, particularly useful for large time-series datasets.

Splitting Strategies

1. **Train-Validation-Test Split (70-15-15 or 80-10-10)**:
    - For the initial split, we will divide the dataset into training (80%), validation (10%), and test (10%) sets. **Time-aware splits** will be used where the test set contains the most recent data points, preventing future data from leaking into the training set.
2. **Blocked Time-Series Splits**:
    - For handling potential seasonality in the data (e.g., summer vs. winter electricity consumption), we will also explore blocked splits, where each block corresponds to a specific season or month. This will help in assessing whether the model can generalize across different time periods with different consumption patterns.

4. GitHub Repository -

➔ Share the link to your GitHub repository and describe the folder structure.

Link to repo: https://github.com/sacredvoid/mlops-project/tree/dev

README.md has been added to every folder to describe the folder. There is a README.md file in the root directory which will describe the project entirely.

Folder structure:

- **.github/workflows**: Stores GitHub Action workflows for CI/CD pipelines.
- **backend**: Contains the backend logic and services, like APIs and server-side functionalities.
- **dags**: Houses Airflow DAGs (Directed Acyclic Graphs) for orchestrating and managing workflows.
- **dataset/data**: Includes raw or processed data (csv files for local storage)
- **dataset**: Stores dataset processing files - scraping, generating dataset, preprocessing, uploading to DVC, downloading from DVC
- **docs**: Holds project documentation.
- **experiments**: Contains Jupyter notebooks and scripts for exploratory analysis and experiments.
- **frontend**: Manages the frontend or UI part of the application.
- **model**: Contains model-related code for training, evaluation, and prediction, along with MLFlow
- **rag**: Stores code related to Retrieval-Augmented Generation (RAG) models or pipelines.

5. Project Scope

We aim to build a machine-learning application to forecast short-term electricity load around multiple locations. Our project is inspired by the study '**Analysis of Weather and Time Features in Machine Learning-aided ERCOT Load Forecasting** ' by Jonathan Yang, Mingjian Tuo, Jin Lu, and Xingpeng Li.

The study focuses on performing short-term electricity load forecasting for multiple regions of Texas state using the ERCOT data. The study revealed weather plays an important role in the increase or decrease of electricity usage and is an important feature in forecasting electricity usage and load.

1. **Problems**: Many studies and implementations exist for long-term and mid-term electricity load forecasts but not short-term forecasts. This is because many external factors such as human behaviour, appliance usage, etc affect electricity consumption making it very hard to predict the demand. Though it is evident that weather conditions play an important role in load forecasting, there are many challenges in incorporating

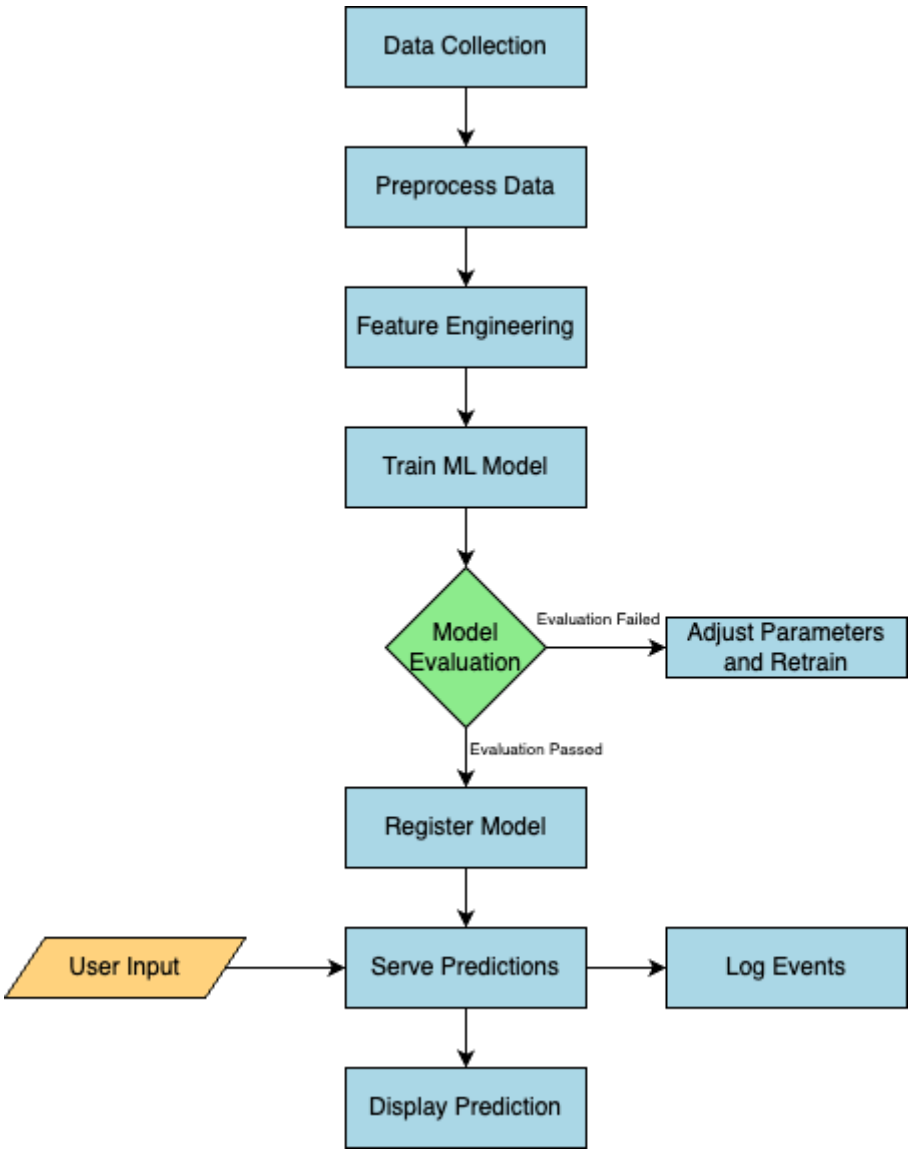
them efficiently into machine learning models making sure not to introduce redundant or irrelevant data that would degrade the model performance.

2. **Current Solutions**: Currently, many applications are available in the market that focus on forecasting electricity load and usage taking in many factors including weather. The drawback of these applications is they focus on long-term and mid-term forecasts. Very few applications focus on short-term forecasting. Moreover, the applications are designed to be used on end-users' proprietary data, making the applications work specifically for the user's data but not in the general case scenario. Also, these applications are intended to be used by experts having in-depth knowledge of the domain. This causes the applications to have a limited user pool.

3. **Proposed Solutions**: Our motive for the project is to extend the study by conducting our ablation study on different models trained on electricity load and weather data. Additionally, develop a comprehensive application that can automate short-term load forecasting, not just limited to Texas State but for multiple geographical locations, by retrieving the relevant electricity load and weather data through multiple API sources. On top of that, the application is designed to pull information from the web and inform the user of potential events and activities happening in those locations that might affect the electricity load forecast. This allows the users to be prepared for extreme situations.

We provide users with an integrated chatbot interface to interact with the application seamlessly. This allows a smooth and hassle-free usage experience for the end-user to focus on obtaining accurate forecasts without exploring underlying data inputs and complexities by themselves.

6. Current Approach Flow Chart and Bottleneck Detection



This architecture shows the flow of an electricity demand forecasting system, using tools like Airflow, MLflow, and a Google Cloud setup. Each step of the process, from data collection to model training and serving predictions, is automated and designed to handle real-time data. Here's a breakdown of the system in simpler terms:

1. **Data Collection:** The system gathers electricity demand and weather data through an API. This happens at the end of each day or when a new location is added (triggered by a pull request). GitHub Actions automates the process

2. **Data Processing:** The collected data is collated and processed to clean it up and format it into CSV files, which are then stored in DVC in Google Cloud Storage for later use in model training.

3. **Model Training and Evaluation:** It pulls the latest processed data from Google Cloud and trains a machine learning model to predict electricity demand. Once training is done, the model is evaluated to see how well it performs.

4. **Model Registry and Serving:** The best-performing model is saved and registered in MLflow, which tracks and versions models.

A FastAPI server then loads this model to serve predictions. Users interact with the system through a simple web interface (built with Streamlit), where they can input a location and get a demand forecast.

5. **Logging and Monitoring:** Throughout the process, logging tracks key events, errors, and the overall health of the system to ensure everything runs smoothly.

Bottlenecks

1. **Data Scraping and API Calls:** Frequent API calls for each data update can be inefficient, especially if network issues or data availability problems arise. To mitigate this, batch processing can be used to reduce the frequency of real-time requests. Additionally, parallelizing API requests or implementing asynchronous programming can help handle multiple calls concurrently, speeding up data collection and improving system performance.
2. **Real-Time Data Delays:** Real-time weather and electricity demand data may be delayed or arrive with inconsistencies, especially when relying on external APIs. To address this, buffering techniques will be implemented to manage delays.
3. **Data Merge conflicts:** Discrepancies in time zones, data collection intervals, or inconsistent timestamps can create challenges when aligning data from multiple sources. To address this, robust timestamp synchronization across all data sources will be implemented Along with timestamp synchronization, Data validation checks will be implemented to ensure consistency and accuracy before merging data from different APIs.

7. Metrics, Objectives, and Business Goals

Metrics

Root Mean Squared Error (RMSE)

RMSE is a widely used metric for regression models and is especially useful for time series data. It provides an indication of how well the model is predicting actual demand by calculating the square root of the average squared differences between predicted and observed values. Lower RMSE indicates better predictive performance.

Mean Absolute Error (MAE)

Mean Absolute Error (MAE) is a metric that provides the average size of the errors in the model's predictions, without considering whether the predictions are over or

underestimating. This makes it a straightforward measure of forecast accuracy, as it presents the errors in the same units as the target variable, which in this case is energy demand. As a result, it is easy to interpret and directly relate to the practical significance of prediction errors in real-world scenarios like energy consumption forecasting.

Training Loss (Training and Validation):
Tracking the training and validation loss over each epoch helps monitor model performance. If validation loss is consistently higher than training loss, it suggests **overfitting**. If both losses remain high, the model may be **underfitting** and failing to learn the data's patterns.

Data Drift:
Data drift measures how much the input data's distribution changes over time. When the drift is significant, the model may encounter new patterns not seen during training, which can lead to performance degradation. High drift levels indicate that **retraining** the model might be necessary.

Data Processing Time:
This metric tracks how long data pipelines (such as Airflow or Kubeflow) take to process and clean data before feeding it to the model. **Efficient pipelines** reduce delays and ensure that training or inference happens on fresh and accurate data.

Prediction Latency:
Prediction latency measures the time it takes for the model to generate a prediction after receiving input data. This is critical in **real-time** or **near-real-time** applications, where low latency is required for immediate responses.

Serving Uptime:
Serving uptime tracks the availability of the model's API or service in production. High uptime is essential for **mission-critical systems**, ensuring the model remains accessible when predictions are needed.

Model Versioning:
Model versioning helps track which model is deployed and compare its performance to previous versions. This ensures that new models are **improvements** and allows for easy **rollback** to earlier versions if issues arise.

Model Drift:
Model drift measures how much the model's performance deteriorates over time in production. Regular monitoring helps decide when to **retrain or update** the model, keeping its performance aligned with real-world changes.

Training Duration:
Training duration refers to how long it takes to train the model. This metric is critical in

continuous integration and deployment (CI/CD) workflows, where shorter training times can speed up the release of updated models.

Resource Utilization (CPU, GPU, Memory):
Monitoring CPU, GPU, and memory usage during training and inference ensures that resources are used efficiently. Tracking resource utilization helps **optimize costs** and prevent system bottlenecks.

Project Objectives

1. Build a Highly Accurate Electricity Demand Forecasting Model
2. Real-Time Forecasting Integration
3. Automation and updation based on data changes, model performance, data drifts, and additional user feedback
4. MLOps **-**
    - Tracking model iteration, along with hyperparameters and data versions.
    - Model Versioning
    - Automated Model Retraining
    - Monitoring and Alerting
5. Scalability: The model will be designed to scale to other cities or regions.

Business Goals:

Our electricity demand forecasting model aims to optimize energy supply and grid management for utility companies by accurately predicting electricity demand, which helps maintain a balance between supply and demand, minimizes financial losses, and reduces the risk of blackouts.

This model supports cost efficiency by enabling participation in demand response programs, allowing customers to reduce usage during peak hours and leading to optimized pricing strategies. Additionally, as we integrate renewable energy sources like solar and wind, aligning electricity demand with their availability promotes sustainability and reduces reliance on fossil fuels. Enhanced forecasts also improve customer satisfaction by mitigating outages and providing insights into peak usage periods.

Incorporating a Retrieval-Augmented Generation (RAG) approach with an Agentic LLM, our solution not only predicts demand based on location but also retrieves relevant news articles and external information, such as upcoming events, that could influence demand, creating a dynamic and context-aware forecasting system that enhances decision-making for grid operators and energy providers.

8. Failure Analysis

1. Data Collection and Ingestion Risks

Risks:

- **API Downtime or Rate Limits**: The APIs used for weather, electricity demand, and population density data could face downtime, rate limiting, or provide incomplete data.
- **Inconsistent Data**: Variations in data collection frequencies between weather, population, and electricity demand data could lead to misaligned timestamps or gaps in data.
- **Missing or Outdated Data**: Some regions or time periods may have missing data, or the population data may not be frequently updated.

Mitigation Strategies:

- **Redundant Data Sources**: Use multiple weather data providers to ensure data availability. Set up failover mechanisms where another API is triggered if the primary one fails.
- **Error Handling in Data Pipelines**: Build error-handling logic in Airflow DAGs (Directed Acyclic Graphs) to gracefully handle API failures and retry data collection.
- **Caching Data**: Store cached versions of recent data locally or in a cloud database to handle API downtime without service interruptions.
- **Imputation Techniques**: Use interpolation or historical data to fill missing weather or electricity demand data where reasonable.
- **Monitor API Quotas**: Use API quota monitoring tools to avoid hitting rate limits, and optimize calls by reducing redundant or unnecessary requests.

2. Data Preprocessing and Cleaning Risks

Risks:

- **Data Inconsistency**: If data from different sources isn't cleaned or normalized consistently, it can lead to faulty features (e.g., unit mismatches in weather data).
- **Poor Feature Engineering**: Time-lagged or interaction features could be improperly generated, leading to models that miss important temporal or spatial dependencies.
- **Outliers**: Extreme weather conditions or unusual electricity demand spikes (due to events like blackouts) could distort model performance.

Mitigation Strategies:

- **Automated Data Validation**: Set up validation checks in Airflow to ensure data consistency across features (e.g., matching units, formats).
- **Outlier Detection**: Implement outlier detection techniques (like Z-score or IQR) and decide whether to cap, remove, or investigate extreme values based on domain knowledge.
- **Continuous Monitoring of Data**: Use MLflow for tracking data drifts or shifts during preprocessing to ensure the data fed to the model remains consistent.

3. Model Development Risks

Risks:

- **Model Overfitting**: The model may overfit to specific regions or seasonal trends, failing to generalize to new data.
- **Inadequate Model Evaluation**: Without proper evaluation metrics, the model may perform well in training but poorly in real-world scenarios (e.g., during extreme weather).
- **Feature Importance Misinterpretation**: Poor understanding of feature importance could lead to overemphasizing less relevant variables, degrading the model's performance.

Mitigation Strategies:

- **Regularization Techniques**: Implement L1/L2 regularization to prevent overfitting and ensure that the model generalizes well across different cities and time periods.
- **Cross-Validation**: Use time-series-specific cross-validation methods (like rolling-window or walk-forward validation) to ensure the model performs well across time.
- **Performance Metrics**: Track both traditional metrics (MAE, RMSE) and time-specific metrics (e.g., performance during peak vs. off-peak hours) to evaluate model reliability across different scenarios.
- **Model Interpretability**: Use SHAP or LIME for feature importance analysis and model interpretability to understand what factors drive the predictions.

4. Model Deployment Risks

Risks:

- **Model Drift and Data Drift**: The model may become less accurate over time due to changing demand patterns (e.g., due to population growth or changes in energy consumption behavior) or shifts in weather data trends.
- **Scalability Issues**: Handling increasing API requests or scaling predictions to multiple cities could lead to performance degradation.
- **Pipeline Downtime**: Airflow, MLflow, or the deployed user interface could experience downtimes, leading to disrupted model availability.
- **Inaccurate Predictions Leading to Grid Instability**: Bad predictions might cause imbalances in the grid if relied upon for real-time energy management decisions.

Mitigation Strategies:

- **Model Retraining**: Set up automated retraining schedules using Airflow to trigger model updates when new data becomes available (e.g., monthly or seasonally).
- **Data Drift Detection**: Use MLflow's drift detection tools or build custom logic to alert the team when data distribution shifts are detected, prompting model updates.
- **Horizontal Scaling**: Ensure the pipeline can scale horizontally using cloud-based solutions like Kubernetes to handle increased API requests or multiple city predictions.
- **High Availability Setup**: Design redundant Airflow/MLflow setups (active-passive) to minimize downtime, and use load balancers to distribute traffic across multiple instances.
- **Alerting for Anomalous Predictions**: Implement alerts for unusual or anomalous electricity demand predictions, allowing grid managers to manually override or verify predictions in critical situations.

5. Post-Deployment Risks

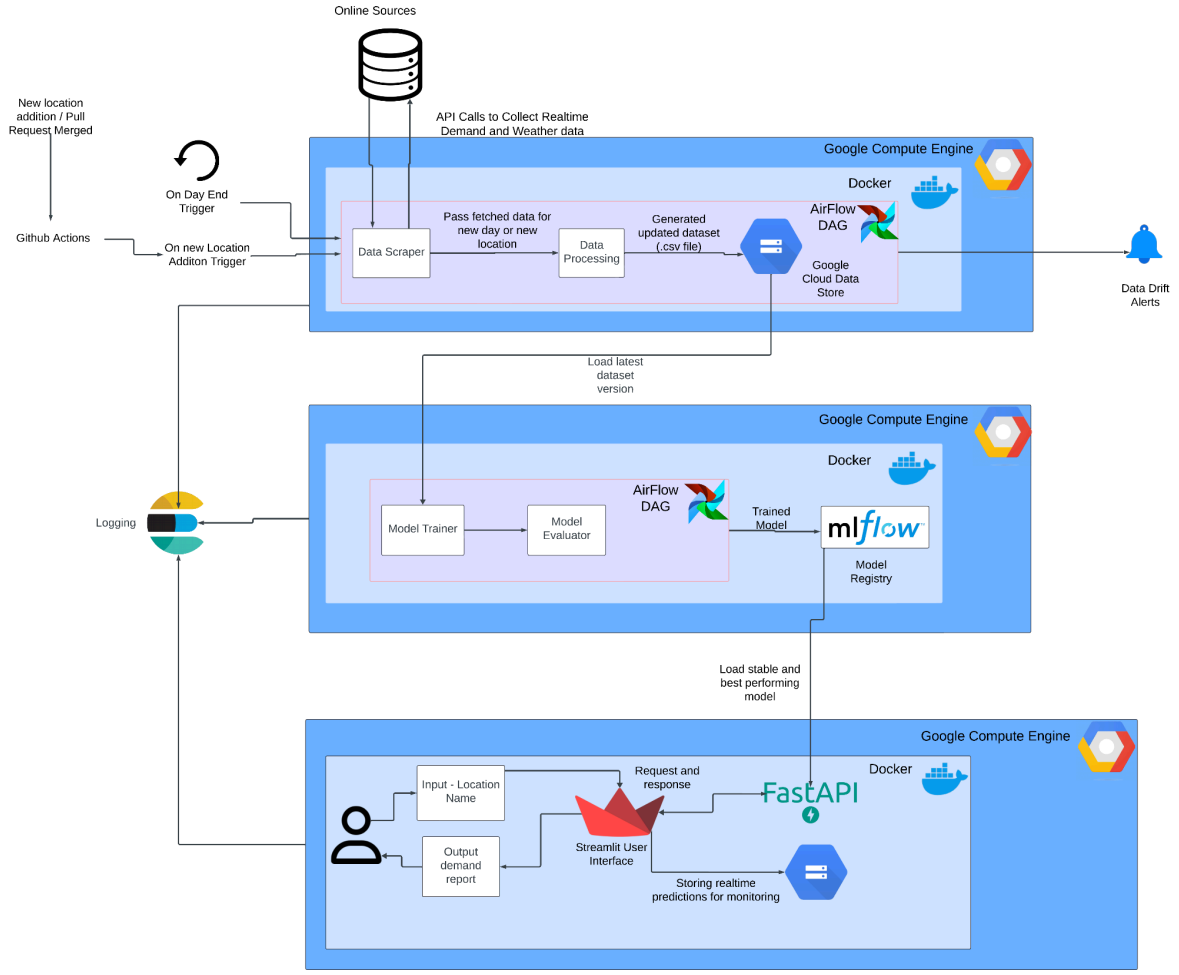
Risks:

- **Unexpected Environmental Changes**: Natural disasters, sudden changes in energy policies, or infrastructural changes could lead to demand shifts that the model isn't trained on.
- **Stakeholder Misuse**: Misinterpretation of model predictions by energy providers or grid operators could lead to poor decision-making.

Mitigation Strategies:

- **Regular Model Audits**: Perform periodic audits to assess how well the model is handling new data and whether it needs retraining or additional tuning.
- **Education for Users**: Provide stakeholders with clear documentation, training, and guidelines on how to interpret and use model predictions effectively.

9. Deployment Infrastructure



10. Monitoring Plan

A comprehensive monitoring plan is required to ensure optimal performance and reliability of the electricity demand forecasting system. This strategy encompasses critical aspects of the MLOps pipeline, including model accuracy, data quality, and system performance. Careful monitoring of key metrics will help maintain high forecast precision, ensure efficient resource utilization, and enable prompt identification and resolution of potential issues.

1. **Model Performance Metrics:**
    a. Metrics: RMSE, MAE, training loss, and validation loss

b. Why: RMSE and MAE directly quantify prediction accuracy, with RMSE being sensitive to large errors and MAE providing average error magnitude. Training and validation loss curves help detect overfitting or underfitting.

2. **Data Quality and Conceptual Integrity:**
    a. Metrics: Data drift, concept drift, missing values, and outliers
    b. Why: Tracking these metrics ensure data reliability and model relevance. Data drift detects input distribution changes, while concept drift identifies shifts in feature-target relationships. Monitoring missing values and outliers maintains data integrity. This approach helps maintain forecast accuracy amid changing conditions, indicating when to adjust preprocessing, retrain, or redesign the model.
3. **Resource Utilization:**
    a. Metrics: CPU usage, GPU utilization, and memory consumption
    b. Why: Monitoring these metrics helps optimize resource allocation and manage costs. Tracking CPU and GPU usage ensures efficient utilization of compute resources, while memory consumption monitoring prevents out-of-memory errors. This approach facilitates more cost-effective model training and enables faster iteration cycles in the MLOps pipeline.
4. **MLflow Tracking:**
    a. Metrics: Experiment metrics, parameters, model versions, and deployment artifacts
    b. Why: MLflow tracking provides comprehensive version control and experiment management for the machine learning lifecycle. It enables systematic logging of model parameters, performance metrics, and artifacts across different experiments. This facilitates easy comparison between model versions, reproducibility of results, and efficient management of model deployments in the production environment.
5. **Log Management:**
    a. Metrics: Application logs, error logs (including deployment errors), and system logs
    b. Why: Log management is crucial for troubleshooting, performance optimization, and maintaining system reliability. Application logs and error logs, particularly deployment errors, help identify and diagnose issues in the MLOps pipeline and during model deployment. System logs provide insights into the overall health of the infrastructure. Implement ELK Stack for centralized log management, enabling system behavior tracking and successful model deployment monitoring in the MLOps pipeline.

So the plan is to utilize Kibana for log analysis and visualization, enabling efficient troubleshooting and performance monitoring of the MLOps pipeline. Implement Grafana for

real-time metrics dashboards, focusing on system resource utilization, model performance, and deployment success rates.

11. Success and Acceptance Criteria

- **Accurate Predictions**: The model should deliver high accuracy in forecasting short-term electricity demand, evaluated using performance metrics like Mean Absolute Error (MAE) or Root Mean Square Error (RMSE).
- **Real-Time Forecasting**: It should generate real-time, location-specific predictions for electricity demand, with continuous updates using real-time data from external sources.
- **Seamless Data Pipeline**: Airflow should orchestrate the data collection, preprocessing, and training workflows smoothly, with APIs integrated for real-time weather and population data, as well as historical electricity demand.
- **Data Consistency and Cleanliness**: The pipeline should handle missing or incomplete data through robust imputation and preprocessing techniques while maintaining data versioning.
- **Effective User Interface**: The UI should allow users to query the system for electricity demand predictions through a chatbot powered by LLMs, offering an intuitive experience for both experts and non-experts.
- **User Engagement via LLMs**: The LLM-powered chatbot should handle complex queries, offer predictions, and provide interactive engagement for users, making the system more accessible.

12. Timeline Planning

Weeks 1-2: Dataset Preparation

In the first two weeks, we will focus on dataset preparation. This includes gathering, cleaning, and exploring real-time weather, electricity demand, and population data. We'll ensure the dataset is comprehensive, filling any missing values and preparing it for further analysis.

Weeks 2-3: Automation, Model Experimentation, and GitHub Setup

In the next two weeks, we will automate the data collection process through APIs and begin experimenting with different model architectures. Simultaneously, we will set up the GitHub repository, establish GitHub Actions for continuous integration, and implement unit tests on pull requests to ensure code quality and functionality.

Weeks 3-4: Dataset Splitting, Airflow Pipeline, and Model Architecture

During this phase, we will split the dataset into training, validation, and test sets. We will build the Airflow pipeline for automated data collection and processing, define the model

architecture, and initiate model training and evaluation. Additionally, we will set up the cloud infrastructure and orchestration to handle scalable training and deployment processes.

Weeks 4-5: Model Training, Evaluation, and MLflow Integration
In these weeks, we will continue model training and evaluation while integrating the MLflow pipeline within Airflow to manage model versioning, hyperparameters, and performance tracking. We will focus on testing the model on the validation and test sets to ensure accuracy and reliability.

Weeks 6-7: UI Development, LLM Integration, and Pipeline Monitoring
We will shift our focus to building the user interface, where users can interact with the model. During this time, we will also integrate LLM components and agents to enhance predictions with context-aware information. Additionally, we will monitor the entire ML pipeline for performance and functionality.

Week 8: Model Tweaking and Expansion
In the final week, we will refine the model based on its performance and extend its capabilities by incorporating new locations and additional data. This iterative improvement will ensure the model remains dynamic and adaptable to various regions and conditions.