FINAL PROJECT

1. Project goal:

Build a production-ready end-to-end batch ML system to forecast energy consumption levels for the next 24 hours across multiple consumer types (e.g., residential, commercial, industrial) from Denmark. The system must be robust, scalable, and real-time, consisting of at least three main components:

- Feature Engineering Pipeline: ingests data from one or multiple data sources and transforms it into valuable features for the model.
- Training Pipeline: takes the features as input and trains and evaluates the model on them. At this step, before finding the best model & hyperparameters, you experiment with multiple configurations. After, you can automate the training process by leveraging the best configuration you found during the experimentation step. The output of this step is a model artifact.
- Batch Prediction Pipeline: Takes as input the features and model artifacts and runs the model on batches (big chunks) of data. Afterward, it saves the predictions into storage, from where various apps can consume the predictions in real-time.

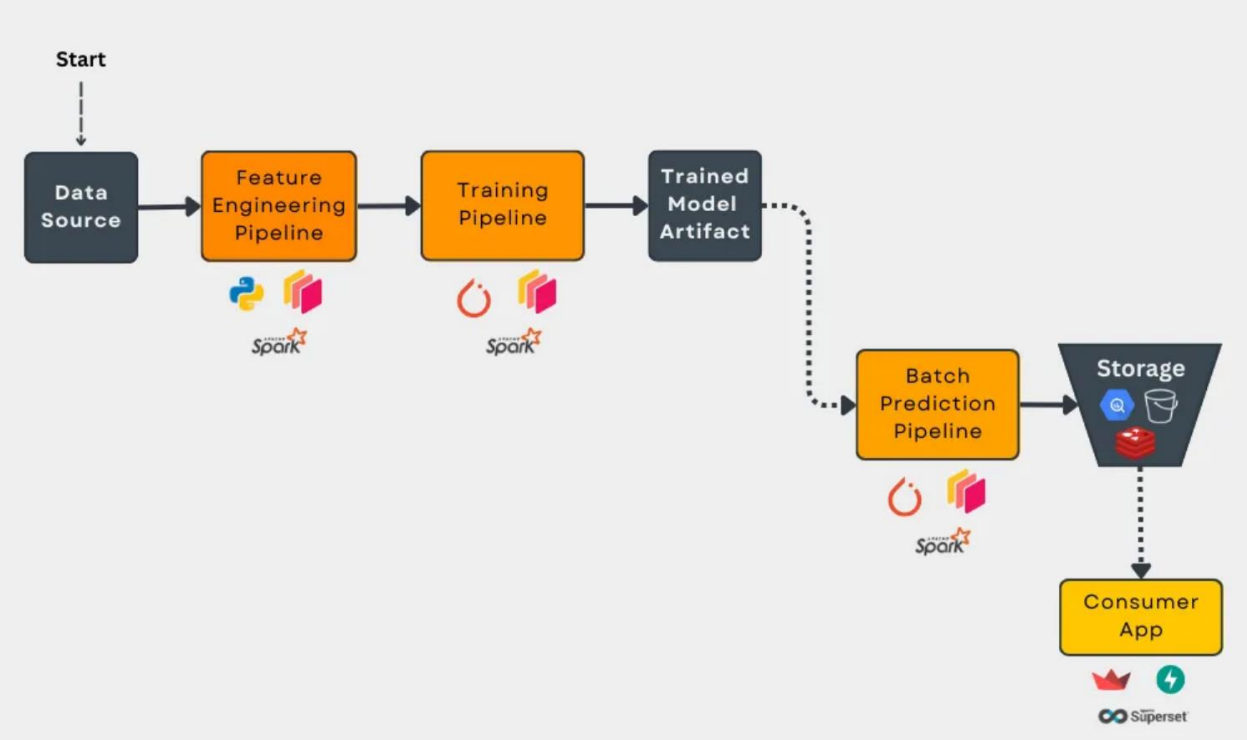

Figure 1: The overview

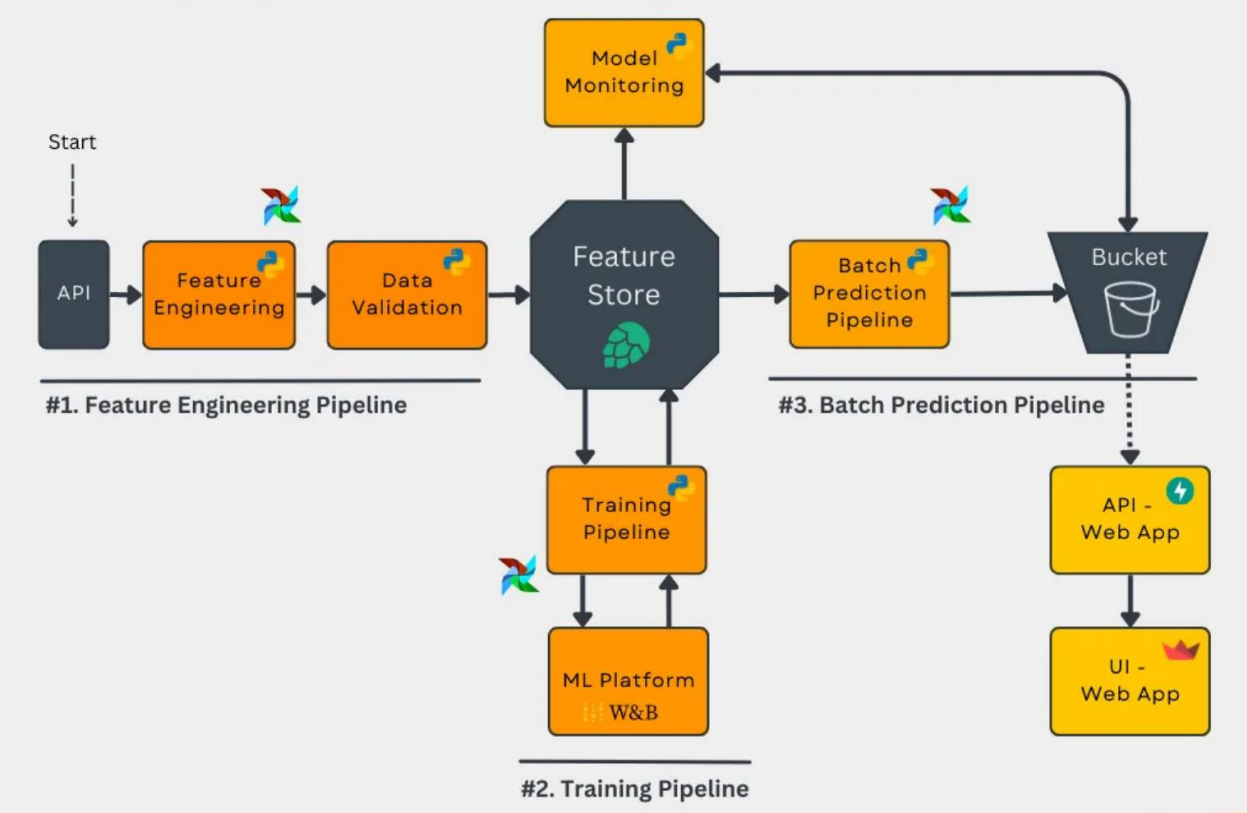

Figure 2: Suggested workflow diagram.

2. Minimum Project requirements for each step:

2.1. Feature Engineering Pipeline

In the FE pipeline, you will do the standard data preprocessing, such as:

- Cleaning data.
- Standardize the data.
- Vectorize the data.
- Aggregations → create new features.

Tools FE: Pandas, Spark, Polars, PyTorch, TF

Feature Stores: Hopsworks, Feast

2.2. Training Pipeline

In this pipeline, you will:

- Experiment with various transformations
- Experiment to find the best model & hyperparameters
- Train the model
- Evaluate & test the model

→ Save the best config and model artifact in the model registry.

When automatically training, you will:

- train the model using the best configuration found at the experimentation step.
- evaluate & test the model.

→ Save the new model artifact in the model registry.

Tools Training: Sklearn, PyTorch, TF

ML Platforms: W&B, MLFlow, Comet ML

2.3. Inference / Batch Prediction Pipeline

This step is relatively straightforward:

- Ingest the given version of the features from the feature
- Load the version of the model artifact from the model registry
- Run the model in batch mode on all the loaded samples
- Save the predictions to storage

Tools Inference: Sklearn, PyTorch, TF

Model Registry: W&B, MLFlow, Comet ML, Hopsworks

2.4. Orchestration

In the orchestration step, you will wrap all your tasks (in our case, the three pipelines) into a DAG (Directed Acyclic Graph). A DAG is a fancy term for a sequence of steps (=tasks) that cannot form cycles. Note that a DAG supports branches (e.g., start a plain training job or do hyperparameter tuning). You will be fine as long it is unidirectional (e.g., from left to right). A DAG will support:

- Run the whole pipeline with a single call
- Easily pass the versions of your resources (features, dataset, model, etc.) between the tasks

Tools: Airflow, Metaflow, Prefect, ZenML, Dagster

2.5. ML Monitoring

Mainly ML monitoring is all about alerting you when you must retrain the model to ensure constant performance. You can do that by:

- Computing real-time metrics, in case you can access real-time GT (which rarely happens).
- Computing data, labels, and concept drift act as proxy metrics.
- Detect outliers and edge cases that might mess up your model.

Tools Monitoring: Arize, Evidently, Seldon

2.6. Consumer App

We built an API server using FastAPI that reads the predictions from the model through a RESTful API. Also, we consumed the RESTful API using Streamlit and displayed them in a simple dashboard that shows the past observations and predictions for every time series.

Tools Server: FastAPI, BentoML, Modal

Tools Client: Streamlit, Gradio, Dash

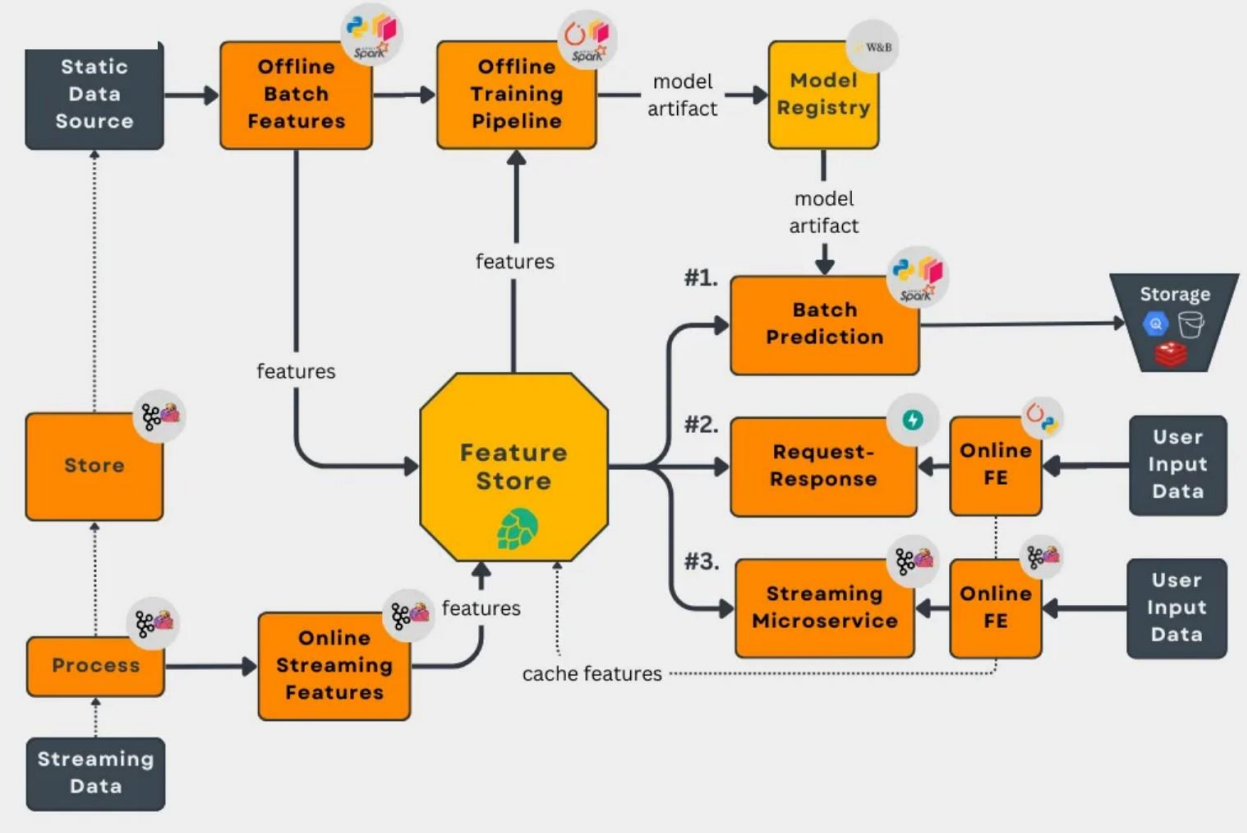

Figure 3: Expected system design.

Data

We used an open API that provides hourly energy consumption values for all the energy consumer types within Denmark. They provide an intuitive interface where you can easily query and visualize the data. You can access the data here.

The data has 4 main attributes:

- **Hour UTC**: the UTC datetime when the data point was observed.
- **Price Area**: Denmark is divided into two price areas: DK1 and DK2 - divided by the Great Belt. DK1 is west of the Great Belt, and DK2 is east of the Great Belt.
- **Consumer Type**: The consumer type is the Industry Code DE35, owned and maintained by Danish Energy.
- **Total Consumption**: Total electricity consumption in kWh

Note: The observations have a lag of 15 days! But for our demo use case, that is not a problem, as we can simulate the same steps as it would be in real time.

IMPORTANT OBSERVATION

Thus, if you cannot query the API, we will mock the same behavior by loading the static data (ConsumptionDE35Hour.csv) from the MS Team.

Additional Requirements:

1. **Documentation**: Provide detailed documentation of your work, including the code, explanation of the designed system, and the results obtained.
2. **Presentation**: Prepare a presentation summarizing your work and the project.
3. **Code Quality**: Ensure the code is well-structured and well-commented to ensure readability and maintainability.