Capstone: Deployment Architecture

Project Overview

This project forecasts Wikipedia page views using a deep learning model trained on time series data. The model takes the last 30 days of traffic and predicts the next day's value. This document outlines a simple deployment architecture that can grow over time.

Inputs + Outputs

Since my dataset isn't live, my model will take the last 30 days of available traffic data for a page selected by the user and return the next day's prediction. This will allow the model to be evaluated using real past data.
Because the dataset is static, monitoring will be done by evaluating model predictions against known values within the dataset. A script compares the predicted and actual values for the last 30 days and logs the MAE. In a real-world deployment, this would be replaced by a live script that calculates error as new data comes in.

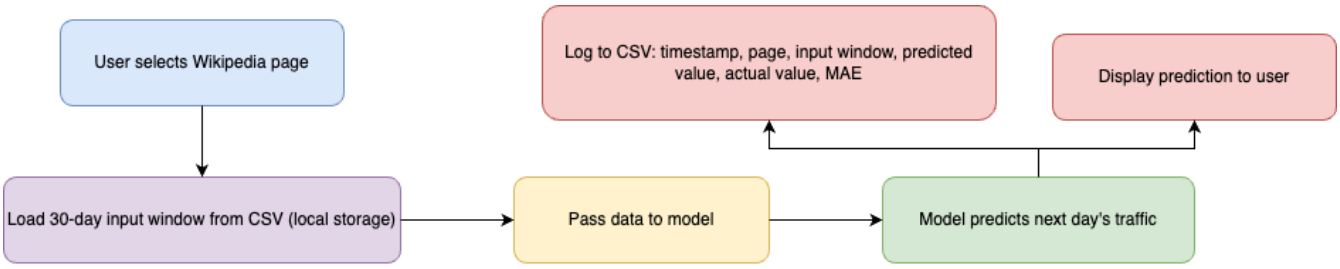
Input: The last 30 days of daily page views for a selected Wikipedia page.
Output: The predicted number of page views for the next day in the dataset. (Example: "Predicted page views for Tour_de_France on Nov 14, 2016: 142")

Data Storage

This project uses a pre-cleaned CSV file containing daily page views for multiple Wikipedia pages. The file is stored locally, which the Flask app will read and store as a variable when the user selects a page. The model will use this data to create the slice needed for prediction-no new external data will be fetched, except for saving prediction results. Prediction results will be saved in a local CSV including the timestamp, input window, predicted value, actual value, and MAE. In this project, the actual value and MAE are available immediately because the dataset is static and complete. With live data, these values would only become available after the forecasted date, and MAE would need to be calculated later as part of the monitoring process.

How Data moves through the System

When a user selects a Wikimedia page in the Flask app, the app loads the relevant time series data from the CSV and extracts the most recent 30 days (the input window). This input is passed into the trained model, which returns a prediction for the next day. The prediction, input data, actual value and MAE is saved in a local CSV file. The page, date and predicted value will be displayed to the user. This process happens each time a new prediction is requested.



Model Lifecycle + Maintenance

My model would need retraining as new data becomes available over time to keep up with traffic trends. For example, on a monthly schedule. Data needed for retraining would include daily page views per Wikipedia page. Now, that data is stored in a local CSV but, when retraining, I would update that CSV with the most recent 30 days.
The retrained model will be evaluated by calculating the MAE on a validation split of the data. If the new model performs as well or better than the previous one, it will replace the old .h5 file and automatically update the model in Flask. A separate folder will save a copy of each retrained model with a timestamp for version tracking.
The CSV log includes the prediction, actual value, and MAE. This allows me to manually track model performance over time and assess when retraining is needed. I also plan to add a column that labels each prediction as “on target,” “drifting,” or “poor” based on the MAE. This would make it easier to monitor the model’s performance over time and identify when retraining is needed.

Tools & Technologies Used

- Python
- Pandas, NumPy
- Keras / TensorFlow
- Flask
- Google Colab

Estimated Cost

This project is being developed and tested locally using free tools (Google Colab, Flask, CSV storage) with no implementation costs. Optional cloud storage (like AWS S3) for model versions or logs could add up to $5–$20 per month depending on usage. In terms of time, an estimated 2+ hours/month would be needed for retraining and manual monitoring. Memory and processing usage is light, making this approach accessible even on a laptop.