Design and Implementation of MLOps for an ML Application

Assignment 1 Report

Data Engineering Course

Assignment Github Repository:
https://github.com/JADS-GROUP8/DE

Berkay Kulak, Diogo Rio, Doruk Karakas, Krishna Teja Atluri

and Mikolaj Hilgert

Group 8 | Oct 30, 2024

Table of contents

1. Overview of the ML Application

1.1 Problem Overview

In the United States in the year 2022, 702,880 people have died from heart disease which means that once every 33 seconds a person has succumbed to death due to cardiovascular complications (Centers for Disease Control and Prevention, 2024). Also, heart disease was the leading cause of death which was responsible for 1 in every 5 deaths. This resulted in a cost of about $252.2 billion between 2019 and 2020. This makes heart disease a pressing issue in the healthcare sector.

In this project, we have created an application that assists the General Practitioners (GPs) in identifying the initial complications of heart diseases using the basic information obtained from the results of a general health check. Based on these results, the patients will be diagnosed with early signs of heart disease, allowing for timely interventions and preventive measures. This application can help identify patients who may be at risk and in need of further testing or specialised treatment. This potential for early detection could save lives and reduce healthcare costs by prioritising preventive care over expensive treatments.

1.2 ML application goals and data description

The dataset used is available on Kaggle: Heart Failure Prediction Dataset.

Our machine learning application predicts the likelihood of heart disease, the base ML code is based on an existing notebook by Tanmay Deshpande: Heart Failure Prediction:CV Score(90%+)| 5 Models. We extended it into an MLOps pipeline using Vertex AI, incorporating CI/CD to build and deploy a Flask application for user interaction. Below are the primary goals and requirements of the application:

- **Detection of Heart Disease Risk:** The main objective is to provide a reliable application that predicts heart disease risk based on patient data.
- **Support Clinical Decision-Making:** The application aims to complement GPs' assessments with data-driven predictions, serving as an additional resource for informed decision-making.
- **Maintain High Accuracy and Reliability:** By using continuous monitoring and retraining, the application is expected to maintain its accuracy over time, accounting for new data and any changes in patterns.

The dataset from Kaggle includes essential predictive variables, such as age, blood pressure, cholesterol levels, and other health metrics that contribute to cardiovascular health insights. This

application processes these variables to assess each individual's heart disease risk, facilitating timely interventions. Key features within the dataset include:

- **Age:** Age of the individual in years.
- **Sex:** Biological sex of the individual.
- **Chest Pain Type:** Categorised information on chest pain (Angina, Asymptomatic, etc.).
- **Resting Blood Pressure:** Measured in mmHg.
- **Cholesterol Levels:** Total serum cholesterol level.
- **Fasting Blood Sugar:** Indicator if fasting blood sugar > 120 mg/dl.
- **Resting ECG Results:** Categorised ECG results.
- **Maximum Heart Rate Achieved** and other relevant physiological indicators.

1.3 Functional Requirements

Below, a list of functional requirements for the application have been listed:

- **Retraining Requirements:**
  Retraining occurs on a scheduled basis or can be manually triggered. This ensures that the model remains up-to-date with recent data. The new data is appended to the main dataset file in the bucket for use if the retrained model outperforms the current model in production.

- **Model Monitoring Requirements:**
  The monitoring process identifies significant shifts in data distributions i.e. data drift by observing the model's performance on new data. This is done by logging a 'critical' warning if the model's accuracy drops below the *baseline* set during the initial deployment of that model i.e. 0.8.

- **Model Deployment Constraints:**
  The CI/CD pipeline automatically initiates deployment of the retrained model using a trigger only when the newly trained model's results surpass the existing one's. This ensures stability by not replacing models that perform better.

- **User Anonymity:**
  User anonymity is essential in health-related applications because data it deals with is sensitive personal data, such as medical history or demographic details. Therefore, any process that may reveal a person's identity should be avoided. Thus, the system must not log personally identifiable information regarding the user.

2. Design and Implementation of the MLOps System

The MLOps system design is illustrated in Figure 1, providing a visual overview of how the system is operationalised. At a high level, the system comprises two main components:

1. **Vertex AI ML Pipeline**: Handles the end-to-end machine learning workflow.
2. **Prediction Serving Application**: A user-facing website for presenting predictions.

Both components are also managed and deployed through CI/CD pipelines. This section will provide a detailed explanation of the flow and interactions, and how they are operationalised.

Starting with the Vertex AI ML Pipeline, the first step consists of pulling the raw csv data file from the designated Google Storage bucket. This data is then cleaned (removal of rows with NAs), and the categorical features are also encoded in this step. The mappings are hardcoded to ensure that label encoding is consistent. Then, a 70-30 train-test split is applied to the data. The pipeline then stores the data splits as outputs so that the following steps can make use of them.

Then, two candidate machine learning algorithms are trained on the same training set: a decision tree model and a logistic regression model. Each model is in its own designated component, which independently builds and trains the model. After training, these models are tested on the test set to gauge their performance, allowing them to be compared based on performance metrics such as accuracy and ROC score. Simultaneously, the pipeline fetches the current production model from the Model Repository. This production model is evaluated on the same test set, providing a baseline for comparison with the newly trained models. The result of this is saved for later use. Additionally, if the results of the training are lower than the saved accuracy metric, this is logged as it indicates model drift.

Next, the model performance comparison component compares the accuracy scores of the two newly trained models with each other. This comparison allows the system to determine which model performs better. Depending on which model performed better, the specific choice flow is initiated. To ensure only good performing models are being deployed, a check whether the model itself reaches a ROC threshold (i.e 0.8) is made, next another check is made regarding whether the existing production model is not already better. If not, then the new model meets the criteria to be deployed/promoted. The chosen model is then tagged and saved in Model Repo, and this action is followed by a trigger to update for the prediction serving component, bringing the new model live.

The prediction serving component fetches the model from the Model Repo Google Storage bucket, and embeds it into the API application. This application is implemented using Flask, and serves to provide an endpoint that can be used for predictions. This API is used by another Flask application which is the Prediction-UI component, which serves as an usable interface for users

to be able to input the prediction input parameters and then view the prediction results. This part also only logs if errors occur, but does not store any of the input values, inline with the user anonymity requirements. This prediction UI and API are deployed using Cloud Run, which allows for deployment of applications on Google's infrastructure.

The CI/CD part of this MLOps system consists of some distinct flows and triggers. The first pipeline is responsible for the *model retraining and evaluation* with Vertex AI, while the second handles the *application deployment* to Cloud Run. Each has its own role and use case in maintaining model performance and keeping the application updated and automated.

The *retraining and evaluation* triggers are activated in three ways:

1. Manual invocation: allowing for the pipeline to be executed at any time, which can be useful during development.
2. Changes to pipeline code in git repository: triggering the pipeline automatically when changes are made to the vertex-ai path.
3. Monthly scheduled execution: initiating the pipeline periodically, even if no changes have been made, to retrain the model with fresh data. This can aid in the spotting of model drift.

In relation to the *application deployment* flows, they are triggered by:

1. Changes to application code in git repository: ensuring the deployed application reflects changes as soon as they are made. If proper branching rules are followed, this allows for main to be treated as production.
2. New model is promoted to production: deploying the latest and best-performing model to production.

Together, these CI/CD pipelines create a complete workflow that allows to continuously train and deploy new iterations of the model and application, with minimal manual effort.

Apart from the Cloud Build CI/CD pipelines, this project also uses Github Actions to apply linting checks on the Python code, ensuring that the code that is put in the code-base is high quality and adheres to standards. The CI/CD pipeline also uses pytest to run the tests, currently only one exists, and it was written to check whether the backend endpoint works, as this was a pain-point when developing the application, and ensures code will work when deployed. This CI/CD triggered on every push and pull-request to main.

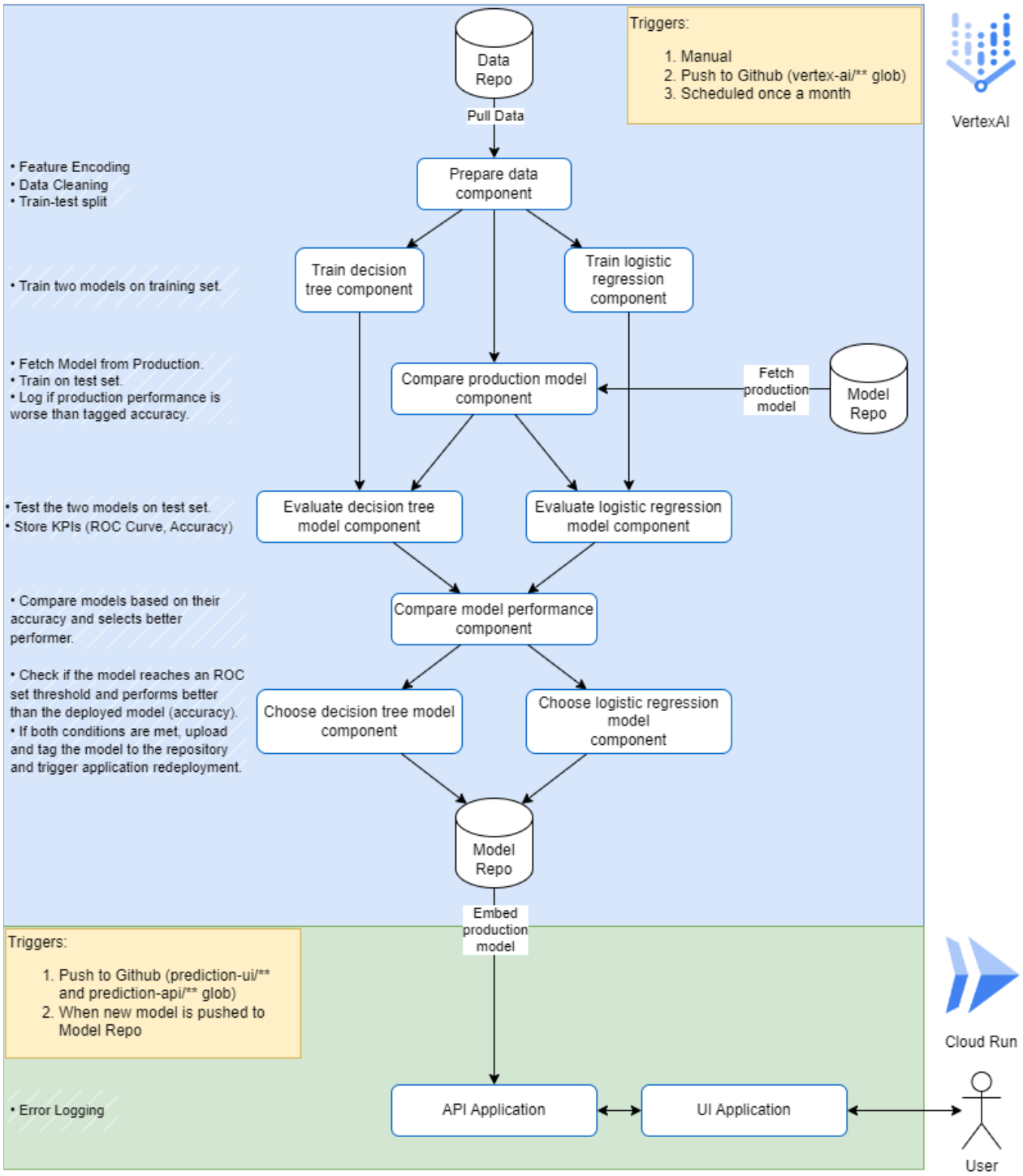


Figure 1. MLOps Application Design

3. Reflection on the Design and Implementation of the MLOps System

3.1 Alternative design discussion

To explore potential enhancements to our MLOps system, the section below evaluates some alternative design approaches that can boost the quality of this project.

One key area that can be improved is the triggering strategies for the pipeline. Currently, there is no automated retraining when the source data file is updated. While we do schedule a retrain once a month, it could be reworked to be event driven (pub-sub) when the dataset is updated with new data. This approach would ensure that the model is trained with the latest data without the rigid time-based scheduling, and this solution can also aid in the issue of data-drift by making sure the model is up-to-date.

Regarding deployment, the current setup replaces the model in the prediction API when performance improvements are confirmed. However, implementing a phased rollout could improve reliability. In such a case, we would only deploy the new model to a few users to gauge performance. It is also beneficial to store the predictions that the system is making, which could eventually be used to tune the model, using the predictions as training data if actual correct labels can be gathered. Moreover, general improvements to monitoring could also be considered, such as tracking model latency and resource usage.

In relation to the pipeline implementation itself, we could consider testing a wider range of ML models and comparing them using additional metrics beyond just accuracy and ROC scores.

3.2 Implementation reflection

Overall, the implementation of this project was successful, and all of the functional requirements were delivered. However, it has its limitations. For example, the development of the VertexAI pipeline was quite cumbersome, as there is no effective way to run these pipelines locally and this as a result increased the development time.

Collaboration on this project was a major pain due to the various dependencies needed to set up to get the application running. For example, all of the storage buckets and triggers needed to be recreated in each team member's cloud environment. It could be beneficial to either introduce infrastructure-as-code or create a single account for the group to use. Moreover, due to the time limitations for this project, no extensive testing was done, as a single integration test is not sufficient. A multitude of unit and integration tests could improve the reliability of the application.

4. Individual Contributions of Students

Mikolaj: I worked as the de facto lead for this project. I took charge as the maintainer of the code, overseeing the project's progress and quality. I worked with the group to develop the VertexAI pipeline, then took steps to operationalise it within VertexAI with various extensions. I also worked on the prediction-serving application side of the project, connecting the backend with the frontend, and created the CI/CD pipelines. For the report, I contributed to the design and reflection sections.

Krishna Teja: I took the responsibility of designing and implementing a trigger within the VertexAI pipeline. This trigger retrains the model and deploys it if the conditions mentioned in the metadata are satisfied. This helps in tackling the data drift ensuring that our model remains up-to-date and performs optimally. Alongside this technical work, I contributed to the overview section of the project report.

Berkay: I contributed ideas for the ML pipeline and the UI component. I have designed the frontend using HTML and CSS, to ensure a user-friendly experience. During development I have made use of pull requests in GitHub for safety and to ensure quality.

Diogo: Accompanying work on the initial VertexAI notebook. Set up CI/CD pipeline with workflow actions to perform Python linting checks and automated API testing.

Doruk: Due to being sick for the majority of the time period of the given assignment, I have not been able to contribute a lot in this assignment. I have only been able to contribute to the preparation of the report.

5. References

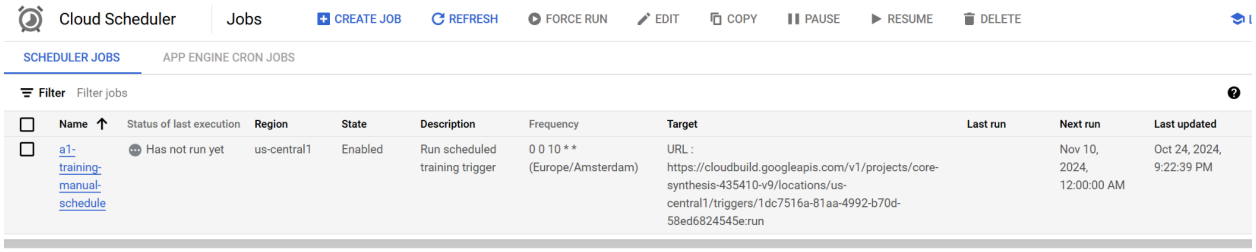
Centers for Disease Control and Prevention. (2024). Heart disease facts. U.S. Department of Health & Human Services. Retrieved from https://www.cdc.gov/heart-disease/data-research/facts-stats/index.html

6. Appendix

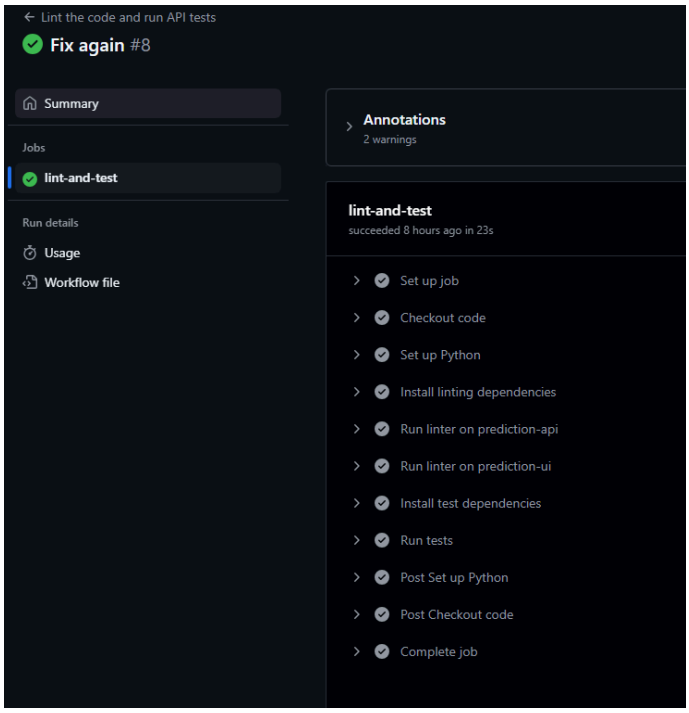
6.1 Triggers

Name	Description	Repository	Event	Build configuration	Status	
a1-app-deployment	-	JADS-GROUP8/DE	Push to branch	cloud-build/cloud_build_app_deployment_execution.json	Enabled	Run
a1-build-executor	-	JADS-GROUP8/DE	Manual	cloud-build/cloud_build_pipeline_executor_tool.json	Enabled	Run
a1-training-code-changed	-	JADS-GROUP8/DE	Push to branch	cloud-build/cloud_build_training_execution.json	Enabled	Run
a1-training-manual	-	JADS-GROUP8/DE	Manual	cloud-build/cloud_build_training_execution.json	Enabled	Run

6.2 Scheduled Trigger



6.3 Github CI/CD



6.4 Vertex AI Pipeline