LEETSUMMARIZER

PROJECT REPORT

of

Machine Learning Operations

BY

GROUP

DEEPANSH GANDHI

RAHUL ODEDRA

BRIJESH SAVAJ

SANKET SHAH

RISHABH MADANI

DEPARTMENT OF COLLEGE OF ENGINEERING

NORTHEASTERN UNIVERSITY

BOSTON, MASSACHUSETTS – 022115

JUNE 2024

CHAPTER 1: INTRODUCTION

1.0 INTRODUCTION

We are developing a Chrome extension that summarizes users' code on LeetCode in plain English. It breaks down the code into simple steps, explaining the logic and functionality in clear, easy-to-understand language.

CHAPTER 2: DATA SOURCING AND CLEANING

1. Dataset Introduction:

We are creating our own dataset which has the leetcode question, code and the plain text on which we want to train the language model on.

2. Dataset Size:

The total size of our dataset is 36 KB, ensuring a comprehensive collection of questions, code, and explanations for effective model training.

3. Data Card:

Our dataset is a collection of code solutions to programming questions, along with plain text summaries describing the solutions.

Number of examples:

Year Created: 2024

Domain: Computer Science, Programming, Natural Language Processing

Each example contains the following fields:

Question: The programming question from Leetcode

Code: User's code solution written in Python

Plain Text: A plain text summary describing the approach and logic of the code

4. Data Sources:

All code in our dataset is from Neetcode, a popular platform for coding interview preparation. Neetcode provides a comprehensive collection of coding problems along with detailed solutions. The questions are from Leetcode, which is a well-known platform for coding challenges, often used by software developers to practice their programming skills.

5. Data Rights and Privacy:

Users can include personal names or identifiers in the comments of their code.To preserve user privacy, we are ignoring all comments within the code:

- **Permissions:** When users download our Chrome extension, we will obtain their explicit permission to collect their data. This data will be used to generate plain English text from their code and for future retraining of our model

- **Anonymization:** We ensure user privacy by using their Leetcode user ID for storing information in our database. This ID is not utilized in our model training process, safeguarding the anonymity of users

- **Data Access Controls:** We will implement access controls and authentication mechanisms to restrict access to user data to authorized personnel only

CHAPTER 3: DATA PLANNING AND SPLITS

Getting Data from Firestore:

- We gather data securely from Firestore, where we store LeetCode questions and their code.

Protecting Privacy and Checking Quality:

- To respect user privacy, we exclude all comments when we convert code into plain English. This keeps personal information out of our dataset.
- We make sure every piece of data matches our set structure: Problem Description, Code, and Plain English Explanation.

Ensuring It's Python:e

- We check that all code is written in Python. This ensures our dataset stays consistent and reliable.

Managing Changes:

- We use data version control to keep track of changes in our machine-learning models and datasets. This helps us keep everything organized, backtrack if needed, and improve over time.

Splitting the Dataset:

- We divide our dataset into 20 parts to train the model effectively. Additionally, we reserve 5 questions to thoroughly test and evaluate how well our model works.

CHAPTER 4: GITHUB REPOSITORY

Link: https://github.com/gandhidee/LeetSummarizer

GitHub Repository Folder Structure:

```
├── .dvc
│   ├── .gitignore
│   └── config
├── .git
│   ├── config
│   ├── description
│   └── sourcetreeconfig
├── .github
│   └── workflows
│       └── build-and-deploy.yaml
├── Logging
│   ├── DataFlow_pipeline.py
│   ├── Load_to_bigquery.py
│   └── Logging.py
├── Scraping
│   └── problem_scrape.py
├── assets
│   ├── Chrome_Extension-1.png
│   ├── Chrome_Extension-2.png
│   ├── Chrome_Extension-3.png
│   ├── DVC_eg.jpeg
│   ├── pipeline_execution.jpeg
│   ├── pipeline_execution.png
│   └── pipeline_flow.jpg
├── backend
│   ├── node_modules
│   │   └── .package-lock.json
```

```
│   ├── .dockerignore
│   ├── dockerfile
│   ├── index.js
│   ├── package-lock.json
│   └── package.json
├── config
│   └── airflow.cfg
├── dags
│   ├── src
│   │   ├── __pycache__
│   │   ├── data_preprocessing
│   │   ├── .DS_Store
│   │   └── __init__.py
│   ├── .DS_Store
│   └── airflow.py
├── data_drift
│   ├── code_complexity.py
│   ├── compare_complexities.py
│   ├── drift_detection.py
│   ├── gcp_alert.py
│   ├── load_data_from_firestore.py
│   ├── requirements.txt
│   └── service_account_key.json
├── data_pipeline
│   ├── Leet_Summarizer_train_data.csv
│   ├── data.json.dvc
│   └── write_to_firestore.py
├── frontend
│   ├── icons
│   ├── background.js
│   ├── content.js
```

```
│   ├── manifest.json
│   ├── normalizedBody.json
│   ├── package-lock.json
│   ├── package.json
│   ├── popup.html
│   ├── popup.js
│   └── server.js
├── logs
│   ├── dag_processor_manager
│   │   └── dag_processor_manager.log
│   └── scheduler
│       ├── 2024-06-26
│       └── latest -> 2024-06-26
├── mlflow
│   └── dockerfile
├── model
│   ├── tests
│   │   ├── __init__.py
│   │   ├── test_remove_comments.py
│   │   └── test_validate_code.py
│   ├── __init__.py
│   ├── model.ipynb
│   └── updated_Model_func.ipynb
├── model_endpoint
│   ├── dashboard.py
│   ├── dockerfile
│   ├── main.py
│   └── requirements.txt
├── model_training
│   ├── dockerfile
│   ├── requirements.txt
```

```
│   └── train.py
├── plugins
├── .DS_Store
├── .dvcignore
├── .env
├── .gitignore
├── LeetSummarizer
├── README.md
├── __init__.py
├── cloudbuild.yaml
├── docker-compose.yaml
├── dockerfile
├── requirements.txt
```

CHAPTER 5: PROJECT SCOPE

Problems:

- Users face challenges in efficiently revisiting summarized approaches to problems they've solved, hindering their interview preparation process.
- Users may encounter difficulties in understanding their own code and problem-solving approaches, impeding their learning and skill improvement.

Current Solutions:

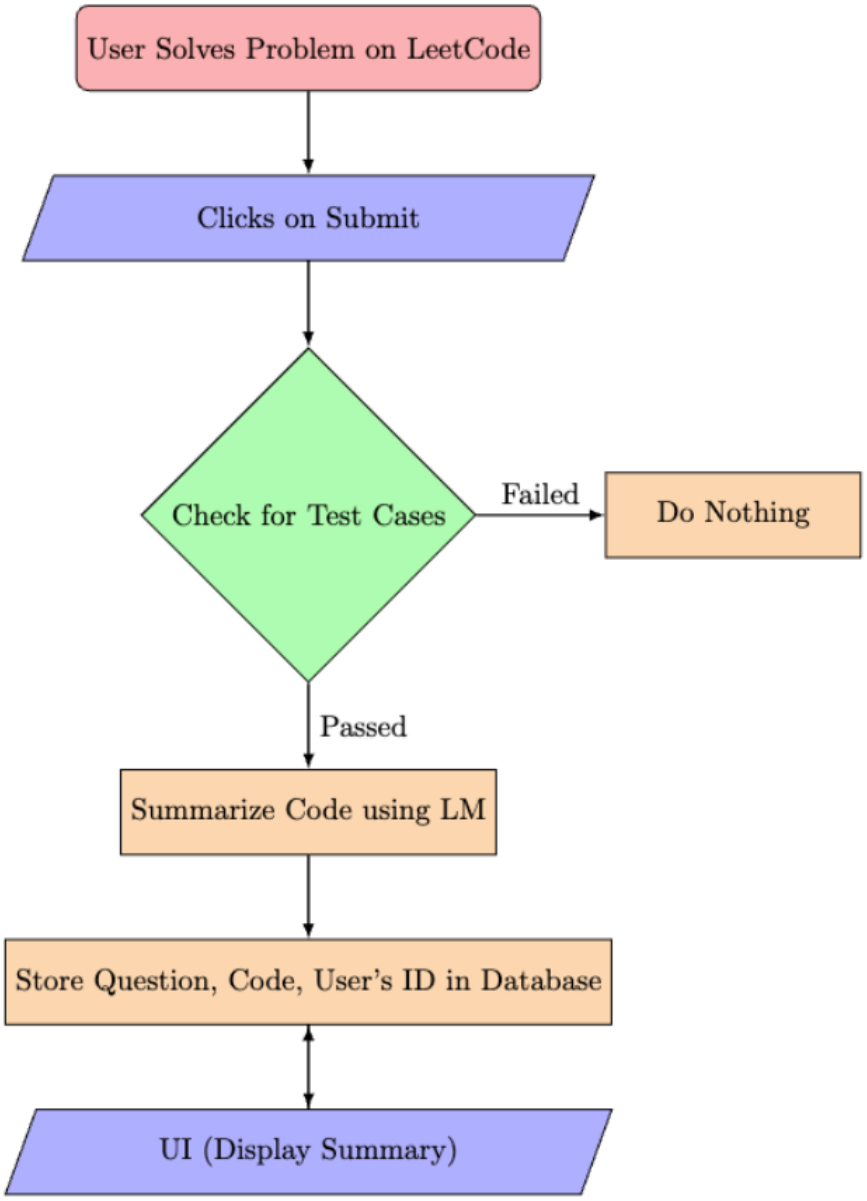
- Users currently rely on manual methods such as reviewing their code or notes, which can be time-consuming and inefficient.
- They may seek assistance from online forums or tutorials, but these resources may not always provide tailored summaries or explanations specific to their own solutions.

Proposed Solutions:

- Our project offers a solution by providing users with a Chrome extension that generates concise summaries of their LeetCode solutions.
- Our model analyzes the problem statement and user's solution to create summaries.
- All summaries are stored and can be accessed by the user at any time, providing a convenient repository for interview preparation and ongoing learning.
- This tailored approach sets our solution apart from existing manual methods and generic online resources, offering users a more efficient and effective way to prepare for interviews and improve their coding skills.

CHAPTER 5: CURRENT APPROACH FLOW CHART AND BOTTLENECK DETECTION

Flowchart:



Bottleneck:

Given the unavailability of the desired dataset online, we have manually created it for this problem. We have selected a subset of problems from LeetCode covering a range of topics and difficulty levels and manually gathered the corresponding questions and code solutions for each problem.

This approach poses challenges concerning both the quality and quantity of data available for training our language model. Our model does not perform well for problems involving complex data structures like tries, graphs, etc. and it's essential to acknowledge these limitations upfront.

CHAPTER 7: METRICS, OBJECTIVES, AND BUSINESS GOALS

Model Accuracy:

- The goal is to generate accurate and relevant summaries of LeetCode solutions. This accuracy is evaluated using metrics such as Rouge-L score and Similarity score.

Automated Pipeline for Data:

- The project fully automates data preprocessing steps using Airflow, ensuring seamless and efficient handling of data.

Automated CI/CD and Continuous Training (CT) Workflow:

- The project automates data ingestion, model retraining, evaluation, and deployment processes to continuously adapt to new LeetCode problem descriptions and code, minimizing manual intervention.

Continuous Monitoring via Logging and Dashboards:

- Model logs are saved and monitored to detect data drift and anomalies in real-time through dedicated dashboards. This monitoring includes system health checks, memory utilization analysis, and log assessments.

Scraping Data Efficiently:

- Problem descriptions, user-submitted code, and user IDs are scraped to store summaries generated by the model in a database. The project focuses on efficiently integrating model outputs with scraped data via API integration.

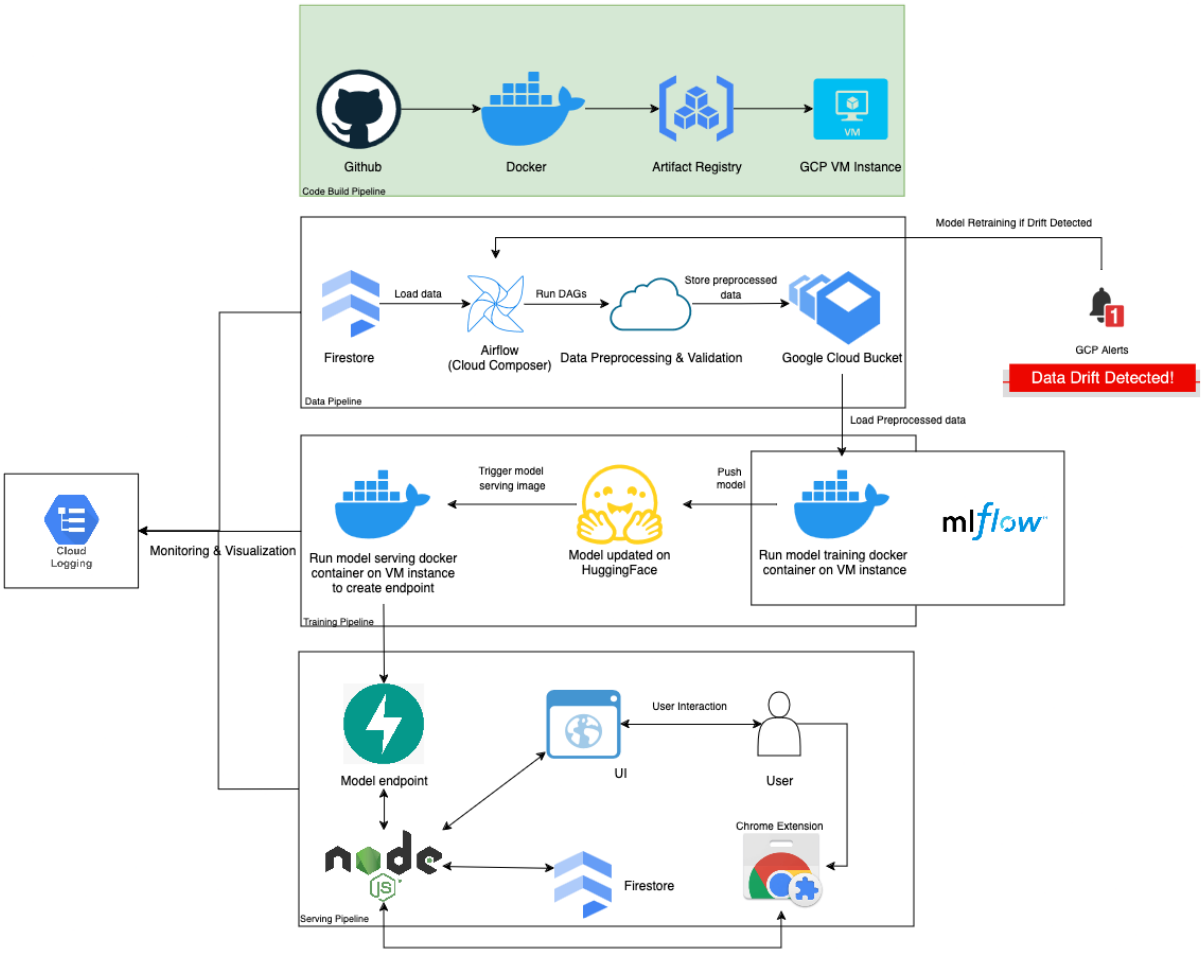
Project Objectives and Alignment with Business Goals:

- The primary objective is to enhance user learning by helping users understand their code and improve problem-solving skills. This aligns with the business goal of providing educational tools that enhance user competency and confidence in technical interviews.

CHAPTER 8: FAILURE ANALYSIS

- The model is specifically trained for codes written in Python, which means it encounters failures when dealing with codes written in languages other than Python. If the input code is not in Python, we raise an error.
- Due to its training on simple data structures, the model's performance is limited for problems involving complex data structures such as tries or graphs.
- In cases where there is missing or incomplete data due to data scraping errors, the model starts hallucinating. This can be improved by using RAGs.
- If the system lacks GPU compute capabilities, the model cannot be loaded.
- Failure to regularly update the model with new data or to retrain it to adapt to new LeetCode problems could lead to performance degradation over time.

CHAPTER 9: DEPLOYMENT INFRASTRUCTURE



CHAPTER 10: MONITORING PLAN

The following is our monitoring plan for LeetSummarizer :-

Model Performance Metrics:

- The primary metrics tracked are Rouge-L score and Similarity score.
- Alerts are triggered if the absolute difference between the training data's code complexity metric and new data exceeds a predefined threshold, indicating potential data drift.

Resource Utilization:

- Metrics tracked include VM total memory and VM CPU utilization usage of the deployed model containers (Virtual Machine).
- These metrics are monitored in real-time using a Google Cloud logging dashboard.

Data Quality Checks:

- Alerts are flagged if the code data is not in the Python programming language.

MLFlow Tracking:

- Metrics tracked include various experiment metrics, managed and monitored through MLFlow.

Log Management:

- Logs tracked include deployment logs, model logs, and error logs.
- These logs are monitored in real-time using Google Cloud Logging dashboard.

CHAPTER 11: SUCCESS AND ACCEPTANCE CRITERIA

Accurate Summarization:

- The model must generate accurate and relevant summaries for at least 90% of LeetCode solutions, as measured by Rouge-L score and Similarity score.

Efficiency and Automation:

- The entire data pipeline, including data preprocessing, ingestion, model retraining, evaluation, and deployment, must be fully automated using Airflow and CI/CD workflows.

Robust Monitoring and Management:

- Continuous monitoring of model performance, resource utilization, and data quality must be in place, with real-time alerts for data drift, anomalies, and system health issues.

CHAPTER 12: TIMELINE PLANNING

May Last Week: Create the model and test performance.

June First Week: Establish the ML pipeline.

Mid-June: Deploy ML pipeline and monitor performance.

June End: Tweak model as per requirement and evaluate performance.