System Design Task

High-Level Design (HLD)

Objective

To build a scalable, real-time data processing pipeline for moderating live chat comments on YouTube using Machine Learning models. The pipeline aims to automate the classification of comments into 'toxic' or 'non-toxic' categories.

System Architecture

Components:

1. Data Ingestion Layer: API Polling Service

- **Why**: This service is responsible for fetching new live chat comments from multiple YouTube videos in real-time.
- **How MLOps Helps**: To ensure scalability, use containerization tools like Docker for easy deployment and Kubernetes for orchestration.

2. Data Processing Layer: Preprocessing and ML Classification

- **Why**: Raw text data usually contains a lot of noise and must be cleaned and transformed into a format that ML models can understand.
- **How MLOps Helps**: Version control ML models and preprocessing code together so that they are always compatible.

3. Database Layer: PostgreSQL Database

- **Why**: Storing the comments and their classifications allows for easy auditing and potential retraining of the model.
- **How MLOps Helps**: Use database migrations to keep the database schema versioned, just like code.

4. Monitoring and Logging

- **Why**: To ensure the system is healthy and to troubleshoot issues.
- **How MLOps Helps**: Use automated monitoring tools that can be configured to trigger redeployments via the CI/CD pipeline if a critical issue arises.

5. MLOps Layer: CI/CD, Model Monitoring, Auto-Scaling

- **Why**: To maintain high availability, scalability, and up-to-date models and code.
- **How MLOps Helps**: Automate as much as possible, from deployments to scaling to issue alerts.

MLOps Practices:

- **Version Control**: Git
- **Continuous Integration/Continuous Deployment**: Jenkins
- **Model Monitoring**: Custom dashboards integrated with machine learning model monitoring tools
- **Auto-Scaling**: Kubernetes
- **Alerts and Notifications**: Prometheus and Grafana

Low-Level Design (LLD)

Data Ingestion Layer

API Polling Service

- **Technology**: Python with **requests**
- **Rate Limit Handling**: Implement a token bucket algorithm or use existing libraries to ensure the API rate limit isn't breached.
- **Error Handling**: In case of an API failure, the service should queue the request and try again.

Data Processing Layer

Comment Cleaning and Preprocessing

- **Technology**: Python with Natural Language Processing libraries like NLTK or spaCy
- **Process**: Tokenization, removing special characters, stemming/lemmatization.

Comment Classification using ML Models

- **Technology**: PyTorch or TensorFlow
- **Hyperparameter Tuning**: Use techniques like Grid Search or Random Search.
- **Model Validation**: Use a validation dataset to evaluate model performance.

Database Layer

PostgreSQL Database

- **Schema Details**:
    - **comments**: Contains columns **id** (Primary Key), **comment** (Text), **label** (Integer), **timestamp** (DateTime).

Batch Operations

- **Why**: To minimize the number of individual INSERT operations, reducing the load on the database.

Monitoring and Logging

Monitoring Service

- **Technology**: Prometheus for gathering metrics and Grafana for visualization.

Logging Service

- **Technology**: Python's **logging** library
- **Logs to Capture**: Errors, warnings, info-level logs for auditing, and debug logs for development.

MLOps Layer

CI/CD Pipeline

- **Technology**: Jenkins or GitHub Actions
- **Stages**: Build -> Test -> Deploy

Model Monitoring

- **Metrics to Monitor**: Accuracy, Precision, Recall, F1-score.

Auto-Scaling

- **Rule-based Scaling**: For instance, if CPU utilization exceeds 70% for 5 minutes, add another instance.

Yasincan Bozkurt