**MLOps Assignment 1**

**Group 39**

1. PRIYANKA KUMARI (2023ac05927)
2. GAGANDEEP SINGH KHURANA (2023ac05806)
3. ABHISHEK DUTT (2023aa05473)
4. TAPAS RAO NALGE (2023ac05381)

**Summary**

Dataset : Iris

The provided code defines a complete MLOps pipeline for classifying Iris flowers, taking a structured approach to machine learning model development and deployment. The pipeline is broken down into distinct, modular components, including data processing, model training, experiment tracking, API creation, and containerization.

**1. Data Processing and Model Training**

The project utilizes a GitHub repository for code management, providing a centralized system for tracking changes, collaborating with a team, and maintaining a complete history of the codebase.

The pipeline begins with data loading and preprocessing, handled by the src/preprocess.py script. This script fetches the raw Iris dataset (loaded via sklearn.datasets.load\_iris()) and saves it as a CSV file (data/iris.csv) for standardized use across the project.

The updated src/train.py script is now capable of training **multiple models** to find the best fit. It loads the pre-processed data, splits it into training and testing sets, and trains both a **Logistic Regression** model and a **Random Forest** on the data. After training, it compares the accuracy of both models and saves the best-performing one as models/model.pkl.

**2. Experiment Tracking with MLflow**

A core component of the architecture is the use of **MLflow** for experiment tracking. The src/train.py script leverages MLflow to log the parameters and accuracy score for each model. This practice ensures that all model development iterations are recorded, making it easy to compare and select the best-performing model.

**3. API & Docker Packaging**

The pipeline deploys the trained model as a REST API, providing a standardized and scalable way to make predictions. The API is built using **FastAPI**, and its single endpoint, /predict, accepts a JSON object with the required flower measurements (sepal\_length, sepal\_width, petal\_length, and petal\_width) as defined by a Pydantic schema (app/schema.py). This ensures that incoming data is automatically validated. The API loads the model.pkl file and returns the predicted Iris class.

For portability and consistent environments, the entire service is containerized using **Docker**. The Dockerfile specifies a Python 3.9 environment and installs all dependencies from requirements.txt.

**4. CI/CD with GitHub Actions**

The ci.yml file defines an automated workflow using **GitHub Actions**. This workflow is triggered on every push to the main branch, ensuring code quality and a continuously updated Docker image. The pipeline includes steps to:

* Lint the code with flake8.
* Build the Docker image for the FastAPI application.
* Push the newly built image to Docker Hub using secure secrets for authentication.
* The workflow does not include a deployment step.

**5. Logging and Monitoring**

The API includes logging to record incoming prediction requests and their corresponding model outputs to logs/prediction\_logs.txt. This log provides a simple but effective way to monitor API usage and model behavior over time.

Overall, this architecture provides a robust, reproducible, and scalable foundation for deploying a machine learning model, covering the essential stages of a modern MLOps pipeline.