**Summary of the MLOps Pipeline for Iris Classification**

This document outlines the architecture and step-by-step process implemented in this assignment to build an end-to-end MLOps pipeline. The goal was to automate the deployment of an Iris species classification model, transforming it from a Python script into a robust, containerized, and monitored web service.

**I. Project Architecture**

The architecture integrates several key technologies to create a continuous, automated loop for model development, deployment, and monitoring.

1. **Version Control (Git & GitHub):** The foundation of the project. All code, including training scripts, API files, and CI/CD workflows, is stored and versioned in a GitHub repository.
2. **Experiment Tracking (MLflow):** Used during the model development phase to systematically log, compare, and manage training runs. It tracks model parameters, performance metrics, and saves the final model artifact, ensuring reproducibility.
3. **API & Application (FastAPI & Pydantic):** The trained model is wrapped in a high-performance web API using FastAPI. Pydantic is used to enforce a strict schema for incoming prediction requests, ensuring data integrity and providing automatic validation.
4. **Containerization (Docker):** The entire application—including the Python environment, all dependencies, the API code, and the trained model—is packaged into a lightweight, portable Docker container. This guarantees that the application runs consistently across any environment.
5. **CI/CD Automation (GitHub Actions):** This is the engine of the pipeline. Two distinct workflows automate the process:
   * **Continuous Integration (CI):** On every code push, this workflow automatically installs dependencies and runs linters and tests to validate code quality.
   * **Continuous Deployment (CD):** On a push to the main branch, this workflow builds the Docker image, pushes it to Docker Hub, and can be configured to automatically deploy the new container to a server.
6. **Monitoring (Prometheus & Grafana):** Once deployed, the API exposes key performance metrics via a /metrics endpoint. Prometheus continuously scrapes these metrics, and Grafana provides a visual dashboard to monitor the API's health, request rate, and other important indicators in real-time.

**II. Step-by-Step Workflow Execution**

The entire process, from training to a live API, follows these automated and manual steps:

**Part 1: Model Development & Experimentation**

* The train.py script is executed. It loads the iris.csv dataset, preprocesses it, and trains both a Logistic Regression and a Random Forest model.
* **MLflow** logs each experiment. The model with the highest accuracy is identified and its serialized file (model.joblib) and the associated scaler are saved to the saved\_model/ directory.

**Part 2: Code Integration and CI Validation**

* Code changes are committed and pushed to a feature branch in the GitHub repository.
* The **CI workflow** is automatically triggered. It runs pytest to ensure the API and other code components are working correctly and flake8 to check for code style issues.

**Part 3: Deployment and CD Automation**

* Once CI passes, the code is merged into the main branch.
* This triggers the **CD workflow**, which performs the following:
  1. Builds a new Docker image using the dockerfile.
  2. Logs into Docker Hub using credentials stored in GitHub Secrets.
  3. Pushes the newly built image (iris-classifier:latest) to the Docker Hub registry.

**Part 4: Serving, Monitoring, and Retraining**

* The Docker container is run on a server, injecting API keys as environment variables. The FastAPI application starts, making the /predict endpoint available.
* **Prometheus** begins scraping the /metrics endpoint, and the **Grafana** dashboard provides live visualizations of the API's performance.
* The /retrain endpoint, protected by a secret API key, can be called to trigger a repository\_dispatch event. This starts the **retraining workflow** on GitHub Actions, which runs train.py again, creating a new model and completing the MLOps loop.

This project successfully demonstrates a modern, automated approach to deploying and managing machine learning models in a production-like environment.