        - (c) (4 Marks) How can MLflow assist you in reverting to a previous model version and deploying it?
        - (d) (4 Marks) Explain the following Python code line by line:
    - Question 5 (12 Marks)
        - (a) (8 Marks) Explain each section of the deployment YAML code line by line.
        - (b) (4 Marks) After deployment, the Flask application was still not accessible on port 8080. Why did this happen and how can it be fixed?
    - Question 6 (10 Marks)
        - (a) (6 Marks) Explain the overall flow of the provided Jenkins pipeline.
        - (b) (4 Marks) Rewrite the PushDockerImage stage by adding a check for the existence of the Docker Image before pushing.

Final Exam Spring 2023 - MLOps

Topic	Question	Marks	Percentage	Key Concepts Tested
Environment & Dependency Management	Q1	12	16.0%	Virtual environments, requirements.txt, project structure, Makefile
MLOps Lifecycle & Automation	Q2	9	12.0%	Model retraining, containerization, monitoring, alerting
Data & Model Versioning (DVC)	Q3	16	21.3%	DVC pipelines, reproducibility, data tracking, model drift detection
Experiment Tracking (MLflow)	Q4	16	21.3%	MLflow tracking, model registry, environment management, versioning
Container Orchestration (Kubernetes)	Q5	12	16.0%	K8s YAML, deployments, services, Minikube troubleshooting
CI/CD Pipelines (Jenkins)	Q6	10	13.3%	Jenkins pipeline stages, Docker integration, testing

Topic	Question	Marks	Percentage	Key Concepts Tested
				automation

Question 1 (12 Marks)

A machine learning team is developing a deep learning model for image recognition. They aim to leverage the power of TensorFlow for building and training their models, along with other libraries for data preprocessing and visualization. The project involves multiple developers working on different aspects of the model, such as data preprocessing, model architecture, and evaluation. Build process of the project involves, linting, building and testing phases. The project has following dependencies:

- **TensorFlow (version 2.5.0):** The core deep learning library that provides tools and APIs for building and training neural networks.
- **NumPy (version 1.21.2):** A fundamental library for numerical computations in Python, extensively used for array operations and mathematical functions.
- **Matplotlib (version 3.4.3):** A plotting library that enables visualizing and analyzing data, including model performance metrics and visualization of training results.
- **OpenCV (version 4.5.3):** A computer vision library used for image preprocessing and augmentation tasks, such as resizing, cropping, and applying filters.
- **scikit-learn (version 0.24.2):** A machine learning library that provides various algorithms for data preprocessing, feature selection, and model evaluation.

During development, one developer updates the TensorFlow library to the latest version (2.7.0) to take advantage of new features. However, this update introduces incompatibility issues with other libraries in the project, particularly NumPy and scikit-learn. As a result, the developer encounters the following issues:

- **Compatibility Errors:** The updated TensorFlow version relies on a newer version of NumPy that is not compatible with the existing version (1.21.2) used by other libraries. This leads to compatibility errors and disrupts the functionality of the model, preventing proper data preprocessing and training.
- **Functionality Breakdown:** The scikit-learn library, which depends on the original NumPy version, fails to function correctly due to the incompatible NumPy version introduced by the updated TensorFlow. This issue hampers the team's ability to

utilize scikit-learn's machine learning algorithms for data preprocessing and evaluation.

- **Model Performance Discrepancies:** The inconsistencies introduced by the incompatible library versions result in discrepancies between model performance during development and deployment. The model trained with the updated TensorFlow version may exhibit unexpected behavior or lower performance when deployed in a different environment that relies on the original NumPy version.

(a) (3 Marks) How would you resolve this incompatibility library scenario?

> **Solution:** By utilizing a **virtual environment** (like `venv` or `conda`), the research team can effectively manage their scientific computing dependencies. This ensures:
>
> - Reproducibility of experiments.
> - Seamless collaboration.
> - Maintenance of a consistent and controlled development environment.
>
> These benefits contribute to the team's ability to conduct rigorous scientific analysis, validate their findings, and maintain environment parity.

(b) (3 Marks) Write down project scaffolding.

> **Solution:**
>
> ```
> Project_name/
> |-- main.py
> |-- requirements.txt
> |-- Makefile
> |-- README.md
> |-- .gitignore
> ```

(c) (3 Marks) Write down requirements.txt file for this project.

> **Solution:**

```
tensorflow==2.5.0
numpy==1.21.2
matplotlib==3.4.3
opencv-python==4.5.3
scikit-learn==0.24.2
```

(d) (3 Marks) Write down Makefile for this project.

Solution:

```
# Linting phase
lint:
        @echo "Running linting..."
        # Add linting command here, e.g., pylint or flake8
        flake8 .

# Building phase
build:
        @echo "Building the project..."
        # Add build commands here, e.g., compiling code or packaging assets

# Testing phase
test:
        @echo "Running tests..."
        # Add test commands here, e.g., running unit tests or integration
tests
        pytest

# Default target
default: lint build test
```

Question 2 (9 Marks)

Suggest solution by using different MLOps techniques for the following cases.

(a) (3 Marks) You are working on an MLOps project where you have implemented a machine learning model for a

recommendation system. During deployment, you notice that the model's performance starts to degrade over time. How would you address this issue using MLOps principles and practices?

> **Solution:** To address the model performance degradation issue, you can implement an **automated retraining pipeline** using MLOps. By continuously collecting new data, retraining the model periodically, and deploying the updated version, you can ensure that the model adapts to changing patterns and maintains its performance over time.

(b) (3 Marks) As part of an MLOps project, you are responsible for automating the model deployment process. However, the deployment environment differs from the development environment, causing compatibility issues with the model and its dependencies. How would you ensure seamless deployment using MLOps approaches?

> **Solution:** To ensure seamless deployment in a different environment, you can utilize **containerization** technologies like **Docker** and container orchestration platforms like **Kubernetes**. By packaging the model and its dependencies into a container image, you can ensure consistency between the development and deployment environments (Environment Parity), minimizing compatibility issues.

(c) (3 Marks) In an MLOps project, you are tasked with monitoring the performance of a deployed machine learning model. Over time,

you observe that the model’s accuracy gradually decreases, affecting its effectiveness. How would you implement model monitoring and alerting mechanisms using MLOps principles?

> **Solution:** To monitor the performance of the deployed model, you can implement **continuous monitoring** using MLOps tools (like Prometheus/Grafana or MLflow). This includes tracking key performance metrics, setting up automated alerting systems for anomalies or performance degradation, and periodically re-evaluating the model’s effectiveness to ensure its ongoing reliability.

Question 3 (16 Marks)

In the Machine Learning Performance Monitoring project, our team is working on building a robust pipeline for training and evaluating machine learning models using DVC (Data Version Control). The goal of this project is to ensure reliable and consistent model performance across different iterations and branches.

The pipeline consists of several stages: data acquisition (`get_data.py`), data preprocessing (`process_data.py`), model training (`train.py`), and evaluation.

(a) (4 Marks) Generate project scaffolding by using the above description and the norms taught in the class. Add one line description of each file.

> **Solution:**
>
> ```
> Project_Name/
> |-- .dvc/                # Internal DVC configuration and cache
> |-- .dvcignore           # Files to be ignored by DVC
> |-- .gitignore           # Files to be ignored by Git
> |-- Makefile             # Build script for automating tasks
> ```

```
  |-- README.md          # Project documentation and overview
  |-- dvc.yaml           # DVC pipeline definition (stages, deps, outs)
  |-- get_data.py        # Script to fetch data from external sources
  |-- metrics.json       # File where performance metrics are stored
  |-- process_data.py    # Script for data preprocessing steps
  |-- requirements.txt   # List of Python packages required
  |-- train.py           # Script for training the model on input data
```

(b) (4 Marks) Write Yaml file for the above mentioned DVC pipeline. Use the file names mentioned in the above description.

Solution:

```
stages:
  get_data:
    cmd: python get_data.py
    deps:
      - get_data.py
    outs:
      - data_raw.csv
  process_data:
    cmd: python process_data.py
    deps:
      - data_raw.csv
      - process_data.py
    outs:
      - data_processed.csv
  train:
    cmd: python train.py
    deps:
      - train.py
      - data_processed.csv
    outs:
      - by_region.png
    metrics:
      - metrics.json:
          cache: false
```

(c) (4 Marks) Consider the following scenario:

I am working on the cloned copy of a Git repository. I have stored `data.csv` file in my local machine which will be input to my ML model. I have started tracking this data using DVC and pushed the `data.csv` file to the remote server (not on Github). Later on the local copy of `data.csv` file is appended with new information and at the end updated repository is pushed on the Github, but I forgot to push the updated `data.csv` to the remote server. What will happen when my new partner pulls the Github repository and `data.csv` from the remote server?

> **Solution:** When the partner runs the `dvc pull` command, DVC will check the MD5 hash in the `.dvc` file (which reflects the *new* updated data) against the file available on the remote storage (which is the *old* data). DVC will report a hash mismatch or that the file is missing from the remote, effectively saying: *"The MD5 of 'data.csv' on the remote server does not match the one in the .dvc file. Have you forgotten to push the latest data.csv file?"*

(d) (4 Marks) Suggest steps to track model drift using DVC.

> **Solution:**
>
> 1. **Versioning Data:** Version training and evaluation datasets using DVC to track changes over time and maintain a historical record.
> 2. **Tracking Model Performance:** Record performance metrics (accuracy, precision, etc.) in the DVC pipeline (using `metrics` field). Use `dvc metrics diff` to compare across versions.
> 3. **Regular Retraining:** Establish a retraining schedule/pipeline. Periodically run `dvc repro` on new data.
> 4. **Monitoring Predictions:** Continuously monitor predictions on new production data and compare them against ground truth to identify deviations.
> 5. **Triggering Alerts:** Set up automated alerts when performance metrics drop below a certain threshold.
> 6. **Retraining and Revalidation:** Once drift is detected, update the training data, retrain using DVC, and validate the new model.
> 7. **Documentation and Reproducibility:** Ensure all versions (data, code, model) are linked via Git and DVC for full auditability.

Question 4 (16 Marks)

(a) (4 Marks) You have deployed a machine learning model using MLflow's model serving. Suddenly, you notice a degradation in performance. How can MLflow help you diagnose the issue?

> **Solution:** MLflow's **tracking** capabilities allow you to log metrics during serving (e.g., accuracy, latency). By comparing these metrics across different time periods or model versions in the MLflow UI, you can pinpoint when the degradation started. Furthermore, logging "artifacts" like input/output data histograms can help identify **data drift** or schema changes.

(b) (4 Marks) How can MLflow assist in managing and reproducing the project's environment?

> **Solution:** MLflow uses **MLproject** files to define the software environment (using Conda, virtualenv, or Docker). By specifying dependencies in a `Conda.yaml` or `Dockerfile`, MLflow ensures that any team member can run the project with the exact same environment using `mlflow run`, facilitating reproducibility.

(c) (4 Marks) How can MLflow assist you in reverting to a previous model version and deploying it?

> **Solution:** The **MLflow Model Registry** provides versioning for models. It allows you to transition models through stages (e.g., `Staging`, `Production`, `Archived`). If the latest version performs poorly, you can simply re-assign the "Production" tag to a previous, better-performing version and update your serving endpoint to point to that version.

(d) (4 Marks) Explain the following Python code line by line:

```python
import os
import mlflow
import pandas as pd
from mlflow.tracking import MlflowClient

EXPERIMENT_NAME = "mlflow-demo"
client = MlflowClient()

# Retrieve Experiment information
EXPERIMENT_ID = client.get_experiment_by_name(EXPERIMENT_NAME).experiment_id

# Retrieve Runs information, ordered by accuracy descending
RI = client.search_runs(experiment_ids=EXPERIMENT_ID, order_by=["metrics.accuracy
DESC"]).to_list()

# Get the best run (top of the list)
br = RI[0]
bmp = br.info.artifact_uri

# Load the model from the best run's artifacts
bm = mlflow.sklearn.load_model(bmp + "/classifier")

# Delete all runs in the experiment
for runs in RI:
    client.delete_run(runs.info.run_id)

# Delete the experiment itself
client.delete_experiment(EXPERIMENT_ID)
```

Detailed Explanation:

1. **Imports:** Imports OS, MLflow, Pandas, and the `MlflowClient` for manual tracking control.
2. **Client Init:** Initializes the `MlflowClient` to interact with the tracking server.
3. **Get ID:** Finds the `experiment_id` for the experiment named "mlflow-demo".
4. **Search Runs:** Searches for all runs in that experiment, sorting them by `accuracy` in descending order (highest first).
5. **Best Run:** Identifies the best run (`RI[0]`) and retrieves its `artifact_uri` (location where model files are stored).
6. **Load Model:** Uses `mlflow.sklearn.load_model` to load the trained classifier into memory.
7. **Cleanup Loop:** Iterates through all retrieved runs and deletes them using their `run_id`.
8. **Delete Experiment:** Removes the entire experiment metadata from the tracking server.

Question 5 (12 Marks)

Consider you have locally implemented Kubernetes using Minikube. You want to deploy a Flask-based coin-changing application.

(a) (8 Marks) Explain each section of the deployment YAML code line by line.

> **Solution:**
>
> **Service Section:**
>
> - `apiVersion: v1`: Specifies the API version for the Service resource.
> - `kind: Service`: Defines the resource as a Service (for networking).
> - `metadata.name`: "coinchange-service" - unique name for the service.
> - `spec.selector`: Links the service to pods labeled `app: coinchange`.
> - `ports`: Maps incoming traffic on port 8080 to the pod's `targetPort` 8080.
> - `type: LoadBalancer`: Requests an external IP to expose the service outside the cluster.
>
> **Deployment Section:**
>
> - `apiVersion: apps/v1`: Specifies the API version for the Deployment resource.
> - `kind: Deployment`: Defines the resource as a Deployment (manages Pods).
> - `spec.replicas: 1`: Specifies that one instance of the pod should be running.
> - `spec.selector.matchLabels`: Tells the deployment which pods it is responsible for.
> - `template`: The blueprint for the pods to be created.
> - `template.spec.containers`: Definition of the container (image: `hammadmajeed/greedycoinchange:latest`).
> - `imagePullPolicy: Always`: Ensures the latest image is pulled from the registry every time.
> - `ports.containerPort: 8080`: The port the app inside the container listens on.

(b) (4 Marks) After deployment, the Flask application was still not accessible on port 8080. Why did this happen and how can it be fixed?

> **Solution: Reason:** In a local **Minikube** environment, the `LoadBalancer` type does not automatically get an external IP address like it would on a cloud provider (AWS/GCP). The service remains in a `<pending>` state for the External-IP.
>
> **Fix:** You need to use the Minikube tunnel or the service command to expose it. Run:
>
> ```
> minikube service coinchange-service
> ```
>
> This command will create a local tunnel and open the application in your browser.

Question 6 (10 Marks)

(a) (6 Marks) Explain the overall flow of the provided Jenkins pipeline.

> **Solution:**
>
> 1. **Agent:** Runs on a specific Jenkins node labeled 'your-jenkins-agent-label'.
> 2. **Environment:** Sets a global variable `PYTHON_VERSION`.
> 3. **Checkout:** Pulls the source code from the Git repository.
> 4. **SetupEnvironment:** Creates a clean Python virtual environment and upgrades `pip`.
> 5. **InstallDependencies:** Installs required libraries from `requirements.txt`.
> 6. **RunTests:** Executes unit tests using `pytest`. If tests fail, the pipeline stops.
> 7. **BuildDockerImage:** Builds a Docker image using the current build number as a tag.
> 8. **PushDockerImage:** Logs into Docker Hub and pushes the newly built image.

(b) (4 Marks) Rewrite the `PushDockerImage` stage by adding a check for the existence of the Docker Image before pushing.

Solution:

```
stage('PushDockerImage') {
    steps {
        sh 'docker image inspect my-app:${env.BUILD_NUMBER} > /dev/null 2>&1
|| (echo "Image not found!" && exit 1)'
        sh 'docker login -u your-docker-username -p your-docker-password'
        sh 'docker tag my-app:${env.BUILD_NUMBER} your-docker-username/my-
app:${env.BUILD_NUMBER}'
        sh 'docker push your-docker-username/my-app:${env.BUILD_NUMBER}'
    }
}
```