National University of Computer and Emerging Sciences

School of Computings | **Spring 2023** | **Islamabad Campus**

Serial No:

CS-4085 MLOps (Section CS-A) Solution

Final Exam
Total Time: 3 Hours
Total Marks: 75

Friday, June 9^{th}, 2023

____

Signature of Invigilator

Course Instructor(s)

Hammad Majeed

______________________ ____________ ______ ______________

Student Name | Roll No | Section | Signature

DO NOT OPEN THE QUESTION BOOK OR START UNTIL INSTRUCTED.

*After asked to commence the exam, please verify that you have **17** different printed pages excluding the cover page. There are total of **6** questions.*

- Attempt on question paper. Attempt all of them. Read the question carefully, understand the question, and then attempt it.
- No additional sheet will be provided for rough work.
- Calculator sharing is strictly prohibited.
- Use permanent ink pens only. Any part done using soft pencil will not be marked and cannot be claimed for rechecking.

Question:	1	2	3	4	5	6	Total
Points:	12	9	16	16	12	10	75
Score:							

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, the research team can effectively manage their scientific computing dependencies, ensure reproducibility of experiments, collaborate seamlessly, and maintain a consistent and controlled development environment. These benefits contribute to the team's ability to conduct rigorous scientific analysis, validate their findings, and advance their understanding of fluid dynamics in aerodynamics using Python.

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

Solution:

```
Porject_name
|-- main.py
|-- requirements.txt
|-- Makefile
|-- Readme.md
|-- .gitignore
```

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

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

Solution:

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

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

# 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?

(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?

(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: Answer: 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.

Answer: 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, minimizing compatibility issues.

Answer: To monitor the performance of the deployed model, you can implement continuous monitoring using MLOps tools and techniques. 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, including data acquisition, data preprocessing, model training, and evaluation. Each stage is carefully designed to handle specific tasks and dependencies using DVC's pipeline tracking capabilities.

To begin, in the data acquisition stage, we have a Python script, get_data.py, which fetches the required data from external sources (you can assume any dummy url). This script is executed as a command in the DVC pipeline. The output of this stage is the data_raw.csv file, representing the raw data obtained.

Next, in the data preprocessing stage, we have the process_data.py script. This script performs various preprocessing steps on the raw data to transform it into a format suitable for model training. The pipeline ensures that this stage is executed only when the data_raw.csv file or the process_data.py script has changed. The output of this stage is the data_processed.csv file, which contains the preprocessed data.

Moving on to the model training stage, we have the train.py script. This script trains a machine learning model using the preprocessed data obtained in the previous stage. The pipeline takes into account changes in the train.py script or the data_processed.csv file to trigger the execution of this stage. During training, the model generates various performance metrics such as accuracy, precision, recall, and F1 score.

Whenever a team member pushes changes to a new branch in our version control system (e.g., Git), the DVC pipeline automatically triggers the model training stage and computes the performance metrics on the newly created branch. These metrics are stored in the metrics.json file and are used for performance comparison and monitoring. By leveraging DVC's pipeline tracking capabilities and automatic performance metric computation on branch push, our team ensures that the models trained in different branches are evaluated consistently. This allows us to detect any performance discrepancies early on and take necessary actions to improve the model's performance and reliability.

You are required to answer the following questions in an incremental way. The answer to the first will be used in the answer to the second and so on.

(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. You are not required to write complete code of the *.py files.

Solution:

```
Project Name
|-- .dvc --> added by dvc
|--.dvcignore --> added dvc
|--.gitignore --> added by github
|--Makefile --> Build script
|--README.md --> Update README.md
|--dvc.yaml --> dvc pipeline
|--get_data.py --> fetches data from external sources
|--metrics.json --> performance metrics stored in the file
|--process_data.py --> performs preprocessing steps
|--requirements.txt --> list of packages required to run
|--train.py --> Training of the data on the 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 the 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?

Content of a typical .dvc file is as follows:

```
 outs:
- md5: c24dc366a0f014d65ef61dcfd7cc7e53
  size: 106635588
  path: data.csv
```

Solution: during 'dvc pull' command, the dvc will report that md5 of the 'data.csv' stored on the remote server does not match with the md5 of the .dvc file pushed on the github. 'Have you forgotten to push the latest data.csv file?'

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

Solution: DVC (Data Version Control) can be leveraged to detect and address model drift in machine learning projects through the following steps:

Versioning data: Start by versioning your training and evaluation datasets using DVC. This allows you to track changes in the data over time and maintain a historical record of dataset versions.

Tracking model performance: Record and track model performance metrics, such as accuracy, precision, recall, or any other relevant metrics, as part of your DVC pipeline. This enables you to compare the performance of different models or model versions.

Regular retraining: Establish a retraining schedule for your models. By periodically retraining the models on the latest version of the training data, you can ensure that the models adapt to potential data drift.

Monitoring predictions: Continuously monitor the predictions made by your model on new, unseen data. Compare the predictions with ground truth labels to identify any discrepancies or deviations that may indicate model drift.

Triggering alerts: Set up automated alerts or notifications to be triggered when model drift is detected. This can be done by defining threshold values for specific performance metrics or using statistical methods to detect significant changes in model behavior.

Retraining and revalidation: When model drift is detected, initiate the retraining process using DVC. This involves using the updated training data and training a new version of the model. After retraining, perform a thorough validation and evaluation of the new model to ensure its performance meets the desired standards.

Documentation and reproducibility: With DVC, ensure that all the steps involved in detecting and addressing model drift, including dataset versions, model versions, and evaluation metrics, are documented and reproducible. This enables better collaboration, transparency, and reproducibility of your machine learning workflows.

Question 4 .... **(16 Marks)**

(a) **(4 Marks)** You have deployed a machine learning model in a production environment using MLflow's model serving capabilities. Suddenly, you start noticing a degradation in the model's performance. How can MLflow help you diagnose the issue and identify potential causes for the performance degradation?

(b) **(4 Marks)** Your team is collaborating on a machine learning project, and you need to ensure that everyone is using the same set of dependencies and reproducing the same environment. How can MLflow assist in managing and reproducing the project's environment?

(c) **(4 Marks)** You are using MLflow's model registry to manage different versions of a trained model. However, you realize that one of the previous versions of the model was performing better than the latest version. How can MLflow assist you in reverting to the previous model version and deploying it?

Solution: Answer: MLflow's tracking capabilities enable you to log various metrics during model serving, such as prediction accuracy, latency, and throughput. By comparing the metrics between different time periods or versions, you can identify if there has been a performance degradation. Additionally, MLflow's ability to log input and output data during serving can help you analyze potential data drift or changes causing the degradation.

Answer: MLflow integrates with package management tools like Conda and Docker to create reproducible environments. By defining the project dependencies in a Conda environment file or Docker image, MLflow ensures that all team members have consistent environments. This facilitates seamless sharing and reproduction of the project's environment across different platforms and systems.

Answer: MLflow's model registry allows you to version and manage different iterations of trained models. In the event of a performance drop in the latest model version, you can use MLflow's model registry to revert to a previous version. By selecting the desired model version and deploying it, you can ensure that the previous version, which demonstrated better performance, is used in production.

(d) **(4 Marks)** Have a look at the followin Python code and explain the code line by line. Please write below the code line in the provided space.

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

EXPERIMENT_NAME = "mlflow-demo"

client = MlflowClient()

EXPERIMENT_ID = client.get_experiment_by_name(EXPERIMENT_NAME).experiment_id

RI = client.search_runs(experiment_ids=EXPERIMENT_ID,
                                order_by=["metrics.accuracy DESC"]).to_list()

br = RI[0]
bmp = br.info.artifact_uri

bm = mlflow.sklearn.load_model(bmp+"/classifier")
```

```
for runs in RI:
    client.delete_run(runs.info.run_id)



client.delete_experiment(EXPERIMENT_ID)
```

Solution:

```
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 (parameter 'depth', metric 'accuracy')
ALL_RUNS_INFO = client.search_runs(experiment_ids=EXPERIMENT_ID, order_by=["metrics.accuracy DESC"]).to_list()
best_run = ALL_RUNS_INFO[0]
best_model_path = best_run.info.artifact_uri
best_model = mlflow.sklearn.load_model(best_model_path+"/classifier")
print(best_run.data.metrics['accuracy'])

# Delete runs (DO NOT USE UNLESS CERTAIN)
for runs in ALL_RUNS_INFO:
    client.delete_run(runs.info.run_id)

# Delete experiment (DO NOT USE UNLESS CERTAIN)
client.delete_experiment(EXPERIMENT_ID)
```

Question 5 . . . . **(12 Marks)**

Consider you have locally implemented Kubernetes system using Kube. To test it, you want to implement and deploy a Flask based coin-changing-problem using a greedy algorithm on this cluster. Python code of this application is mentioned below:

```python
from flask import Flask
from flask import jsonify
app = Flask(__name__)
def change(amount):
  # calculate the resultant change and store the result (res)
  res = []
  coins = [1,5,10,25] # value of pennies, nickels, dimes, quarters
  coin_lookup = {25: "quarters", 10: "dimes", 5: "nickels", 1: "pennies"}
  # divide the amount*100 (the amount in cents) by a coin value
  # record the number of coins that evenly divide and the remainder
  coin = coins.pop()
  num, rem  = divmod(int(amount*100), coin)
  # append the coin type and number of coins that had no remainder
  res.append({num:coin_lookup[coin]})
  # while there is still some remainder, continue adding coins to the result
  while rem > 0:
      coin = coins.pop()
      num, rem = divmod(rem, coin)
      if num:
          if coin in coin_lookup:
              res.append({num:coin_lookup[coin]})
  return res
@app.route('/')
def hello():
  """Return a friendly HTTP greeting."""
  print("I am inside hello world")
  return 'Hello World! I can make change at route: /change'
@app.route('/change/<dollar>/<cents>')
def changeroute(dollar, cents):
  print(f"Make Change for {dollar}.{cents}")
  amount = f"{dollar}.{cents}"
  result = change(float(amount))
  return jsonify(result)
if __name__ == '__main__':
  app.run(host='0.0.0.0', port=8080, debug=True)
```

Docker file is as follows for creating the Pod of the Kubernetes.

```
FROM python:3.10
# Working Directory
WORKDIR /app
# Copy source code to working directory
COPY . app.py /app/
# Install packages from requirements.txt
RUN pip install --upgrade pip &&\
    pip install  -r requirements.txt
EXPOSE 8080
ENTRYPOINT [ "python" ]
CMD [ "app.py" ]
```

Following is the Yaml file written for the deployment the above code.

```
apiVersion: v1
kind: Service
metadata:
  name: coinchange-service
spec:
  selector:
    app: coinchange
  ports:
  - protocol: TCP
    port: 8080
    targetPort: 8080
  type: LoadBalancer
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: coinchange-deployment
spec:
  selector:
    matchLabels:
      app: coinchange
  replicas: 1
  template:
    metadata:
      labels:
        app: coinchange
    spec:
      containers:
      - name: coinchange-server
        image: hammadmajeed/greedycoinchange:latest
        imagePullPolicy: Always
        stdin: true
```

```
        tty: true
        ports:
        - containerPort: 8080
```

(a) **(8 Marks)** Explain each section of the deploymnet Yaml code line by line. Make sure you have described all the importance concept related to Kubernetes. Rewrite the section and then add your explanation below it.

Solution:

```

Explanation:

- `apiVersion: v1`: Specifies the Kubernetes API version to be used for this resource.

- `kind: Service`: Defines a Kubernetes Service, which provides network access to a set of pods.

- `metadata`: Contains metadata for the Service resource.

  - `name: coinchange-service`: Assigns the name "coinchange-service" to the Service.

- `spec`: Specifies the desired state of the Service.

  - `selector`: Defines the labels used to select the pods that the Service will route traffic to.

    - `app: coinchange`: Selects pods with the label `app` set to `coinchange`.

  - `ports`: Specifies the ports on which the Service will listen for incoming traffic.

    - `protocol: TCP`: Defines the protocol to be used for the port (TCP in this case).

    - `port: 8080`: Specifies the port number on which the Service will listen.

    - `targetPort: 8080`: Specifies the port number on the pods to which traffic will be forwarded.

```

```
  - `type: LoadBalancer`: Specifies the type of Service to be created (LoadBalancer in this case), which exposes the Service externally using a cloud providers load balancer.

---

Explanation:

- `apiVersion: apps/v1`: Specifies the Kubernetes API version to be used for this resource.

- `kind: Deployment`: Defines a Kubernetes Deployment, which manages the lifecycle of pods and ensures the desired number of replicas are running.

- `metadata`: Contains metadata for the Deployment resource.

  - `name: coinchange-deployment`: Assigns the name "coinchange-deployment" to the Deployment.

- `spec`: Specifies the desired state of the Deployment.

  - `selector`: Defines the labels used to select the pods controlled by this Deployment.

    - `matchLabels`: Specifies the labels that must match in order for the Deployment to manage the pods.

      - `app: coinchange`: Selects pods with the label `app` set to `coinchange`.

  - `replicas: 1`: Specifies the desired number of pod replicas managed by the Deployment.

  - `template`: Defines the pod template used for creating new replicas.

    - `metadata`: Contains metadata for the pod template.

      - `labels`: Specifies the labels to be applied to the pods.
```

```

          - `app: coinchange`: Assigns the label `app` with the value `coinchange` to the pods.

      - `spec`: Specifies the specification for the pod template.

        - `containers`: Defines the containers to be run in the pods.

          - `name: coinchange-server`: Assigns the name "coinchange-server" to the container.

          - `image: hammadmajeed/greedycoinchange:latest: Specifies the Docker image to be used for the container, pulled from the specified repository.

          - `imagePullPolicy: Always`: Specifies that the container image should always be pulled, even if it already exists.

          - `stdin: true`: Allows for reading from the container`s standard input.

          - `tty: true`: Allocates a terminal for the container.

          - `ports`: Specifies the ports that the container will listen on.

            - `containerPort: 8080`: Defines the port number that the container will listen on.
```

(b) **(4 Marks)** After the successful deploymnet of the code. We had seen in the class that the Flask application was still not accessible by using 8080 port. Can you explain why that happend by drawing the schematic of the Kubernetes system? Also mention that how it can be fixed?

Solution: Minikube does not allocate external ip dy default. To assign an external ip you need to execute the following:

```
minikube service my-app-service
```

Question 6 .... **(10 Marks)**

Consider following Yaml code for the Jenkins CI/CD pipleline.

```yaml
pipeline:
  agent:
    label: 'your-jenkins-agent-label'
  environment:
    PYTHON_VERSION: '3.9.6'
  stages:
    - stage: Checkout
      steps:
        - checkout: scm
    - stage: SetupEnvironment
      steps:
        - sh 'python3 -m venv venv'
        - sh 'source venv/bin/activate'
        - sh 'pip install --upgrade pip'
    - stage: InstallDependencies
      steps:
        - sh 'pip install -r requirements.txt'
    - stage: RunTests
      steps:
        - sh 'python -m pytest'
    - stage: BuildDockerImage
      steps:
        - sh 'docker build -t my-app:${env.BUILD_NUMBER} .'
    - stage: PushDockerImage
      steps:
        - sh 'docker login -u your-docker-username -p your-docker-password'
        - sh 'docker tag my-app:${env.BUILD_NUMBER} your-docker-username/my-app:${env.BUILD_NUMBER}'
        - sh 'docker push your-docker-username/my-app:${env.BUILD_NUMBER}'
```

(a) **(6 Marks)** Analyzing the provided Jenkins pipeline YAML file for a Python project, explain the overall flow of the pipeline, including the stages and their respective steps.

Solution:

```
`pipeline:`: This line indicates the start of the Jenkins pipeline definition.
`agent:`: Specifies the agent or executor on which the pipeline will run.
`label: 'your-jenkins-agent-label'`: Assigns a label to the agent that will execute the pipeline. Replace `'your-jenkins-agent-label'` with the desired agent label.
`environment:`: Defines environment variables that will be available throughout the pipeline.
`PYTHON_VERSION: '3.9.6'`: Sets the `PYTHON_VERSION` environment variable to `3.9.6`. Replace `'3.9.6'` with the desired Python version.
`stages:`: Indicates the start of the stages block, which contains the different stages of the pipeline.
`- stage: Checkout`: Defines a stage named "Checkout" within the pipeline.
`steps:`: Indicates the start of the steps block within a stage, which contains the individual steps to be executed.
`- checkout: scm`: Performs a checkout of the source code from the configured source code management system (SCM).
`- stage: SetupEnvironment`: Defines a stage named "SetupEnvironment" within the pipeline.
`- sh 'python3 -m venv venv'`: Executes a shell command to create a Python virtual environment named `venv` using `python3 -m venv`.
`- sh 'source venv/bin/activate'`: Activates the virtual environment by sourcing the activation script located at `venv/bin/activate`.
`- sh 'pip install --upgrade pip'`: Upgrades the pip package manager to the latest version.
`- stage: InstallDependencies`: Defines a stage named "InstallDependencies" within the pipeline.
`- sh 'pip install -r requirements.txt'`: Installs the project's dependencies listed in the `requirements.txt` file using pip.
`- stage: RunTests`: Defines a stage named "RunTests" within the pipeline.
`- sh 'python -m pytest'`: Executes the project's tests
```

```
using the pytest framework.
`- stage: BuildDockerImage`: Defines a stage named "BuildDockerImage" within the pipeline.
`- sh 'docker build -t my-app:${env.BUILD_NUMBER} .'`: Builds a Docker image tagged with `my-app` and the unique build number using the Dockerfile located in the current directory (`.`).
`- stage: PushDockerImage`: Defines a stage named "PushDockerImage" within the pipeline.
`- sh 'docker login -u your-docker-username -p your-docker-password'`: Logs into a Docker registry using the provided Docker username and password.
`- sh 'docker tag my-app:${env.BUILD_NUMBER} your-docker-username/my-app:${env.BUILD_NUMBER}'`: Tags the Docker image with both the unique build number and the Docker username.
`- sh 'docker push your-docker-username/my-app:${env.BUILD_NUMBER}'`: Pushes the tagged Docker image to the Docker registry.
```

(b) **(4 Marks)** Rewrite the PushDockerImage stage of the above file by adding a test to check the availability of the valid Docker Image before pushing it.

Solution:

```
  - stage: PushDockerImage
      steps:
        - sh '[ -e my-app:${env.BUILD_NUMBER} ] && echo "File exists." || exit'
        - sh 'docker login -u your-docker-username -p your-docker-password'
        - sh 'docker tag my-app:${env.BUILD_NUMBER} your-docker-username/my-app:${env.BUILD_NUMBER}'
        - sh 'docker push your-docker-username/my-app:${env.BUILD_NUMBER}'
```