

# ML Ops Major Assignment

*An Assignment Report Submitted by*

**Arpit Tomar**

**G24AI2001**

## **Public links -**

**GitHub Repository link** - <https://github.com/arpittomar246/MLops-Major-G24AI2001>

**Docker Hub repository link** - <https://hub.docker.com/r/arpittomar/olivetti>



॥ त्वं ज्ञानमयो विज्ञानमयोऽसि ॥

**Indian Institute of Technology Jodhpur**

**CSE Department**

*November, 2025*

## **Executive overview**

This project implements a production-style MLOps pipeline to build, test, and deploy a face-classification service based on the Olivetti Faces dataset. The pipeline automates training, artifact handling, Docker image build and push, and deployment in Kubernetes. The user-facing component is a Flask web application that accepts an image upload, preprocesses to the dataset format (64×64 grayscale), performs inference, and displays top predictions and probabilities.

### **Key achievements:**

- Model training and test automation (train.py, test.py).
- Containerization of the application with model included.
- CI/CD using GitHub Actions that trains the model and bakes it into the image.
- Docker image pushed to DockerHub.
- Kubernetes deployment with 3 replicas, health probes, NodePort for access.
- UI for inference and explanations for non-expert users.

## **Objectives and success criteria**

### **Primary objectives**

1. Implement and train a classifier on the Olivetti dataset.
2. Wrap the classifier into a Flask inference app with a simple UI.
3. Containerize the app and test locally.
4. Automate training→build→push via GitHub Actions.
5. Deploy to Kubernetes and ensure the app serves requests.
6. Provide documentation, screenshots and a report.

### **Success criteria**

- Reproducible train.py and test.py that save models/savedmodel.pth.
- CI pipeline that uploads the model artifact and includes it into the image.
- Docker image accessible on DockerHub and usable by Kubernetes.
- Kubernetes Deployment with 3 running pods serving inference requests.
- Demonstrable UI that returns predictions and probabilities.

# **Table of Contents -**

<b>1. Executive overview.....</b>	<b>2</b>
<b>2. Objectives and success criteria.....</b>	<b>2</b>
<b>3. Project architecture .....</b>	<b>4</b>
<b>4. Dataset and preprocessing.....</b>	<b>5</b>
<b>5. Model design and training .....</b>	<b>5</b>
-Train python file overview.....	6
<b>6. Model evaluation and results.....</b>	<b>7</b>
-Test python file overview.....	7
<b>7. Flask application explanation.....</b>	<b>8</b>
-App python file overview.....	9
-Flask application overview.....	11
<b>8. Docker image (containerization).....</b>	<b>12</b>
-Docker images.....	13
-Docker containers.....	14
<b>9. CI/CD – GitHub actions: pipeline design and steps.....</b>	<b>15</b>
-ci.yml overview.....	16
-GitHub workflow.....	17
-GitHub Actions.....	18
-GitHub secrets.....	19
<b>10. Kubernetes deployment.....</b>	<b>19</b>
-Deployment and service YAML files.....	20
-Containers using Kubernetes with 3 replicas.....	21
<b>11. Validation, monitoring and testing.....</b>	<b>22</b>
<b>12. Security and secrets handling.....</b>	<b>22</b>
<b>13. Summary and final notes.....</b>	<b>23</b>

## Project architecture (high level)

Developer workstation (git)

└> GitHub (repo + Actions)

– CI job: train.py -> savedmodel.pth (artifact)

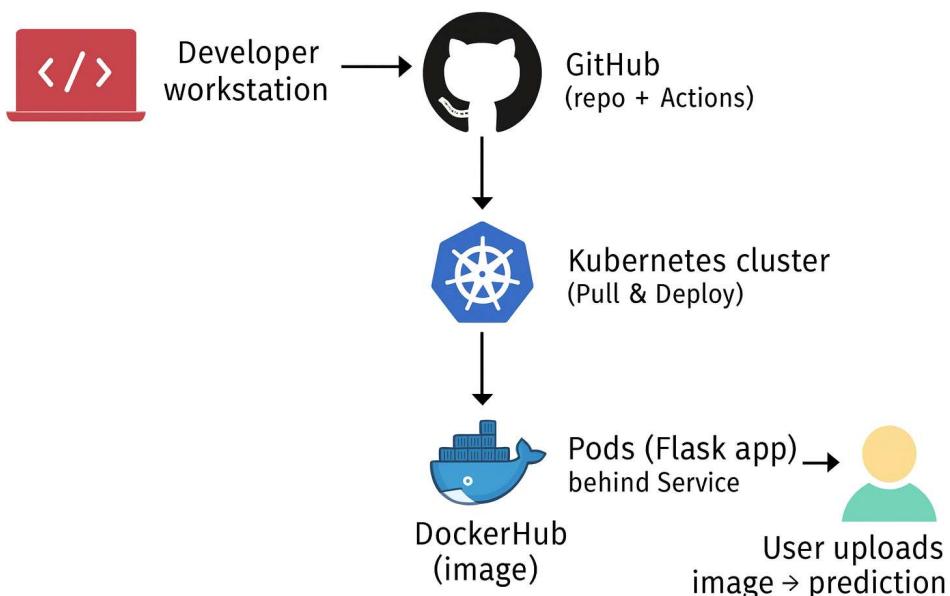
– CD job: download artifact -> docker build -> docker push

└> DockerHub (image)

└> Kubernetes cluster (Pull & Deploy)

└> Pods (Flask app) behind Service

└> User uploads image -> prediction



### Key points:

- The model artifact (savedmodel.pth) is *not* committed to git — it is created during CI and injected into the image in the build job.
- The Docker image is the deployment unit in Kubernetes.
- Kubernetes ensures multi-replica high availability and can perform rolling updates.

# Dataset and preprocessing

## **Dataset**

- Olivetti faces (scikit-learn built-in): 400 images, 40 classes, 10 images per class, grayscale, 64×64 pixels.

## **Preprocessing steps**

1. Load image (uploaded by user or from dataset).
2. Convert to grayscale (if not already).
3. Resize to 64×64.
4. Flatten and scale to the expected format used when training (e.g., X.reshape(1, -1) and normalized if train pipeline used scaling).
5. Feed to the classifier.

## **Why this preprocessing?**

- The dataset images are 64×64 grayscale; the model expects the same shape and intensity normalization used at training. Mismatched preprocessing can drastically reduce accuracy.

# Model design and training

## **Model**

- Chosen algorithm: DecisionTreeClassifier (scikit-learn).
- Rationale: simple baseline, deterministic, easy to serialize. For production-level work, consider RandomForest or simple CNNs for improved performance.

## **train.py (high-level steps)**

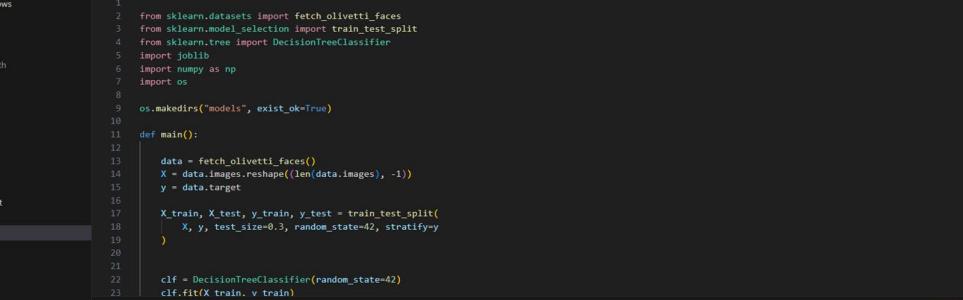
1. Load Olivetti dataset (sklearn.datasets.fetch\_olivetti\_faces).
2. Split into train/test (e.g., 70/30 or use provided split).
3. Optionally scale / normalize.
4. Fit DecisionTreeClassifier (max\_depth tuned or default).
5. Save model to models/savedmodel.pth (or joblib.dump) for inference.
6. Save test metrics (optionally to JSON for UI to show accuracy).

## Train python file code –

The screenshot shows a Visual Studio Code (VS Code) interface with the following details:

- File Explorer:** On the left, it shows a project structure for "MLops Major Assignment G24AI2001". The "train.py" file is selected.
- Code Editor:** The main area displays the "train.py" script. The code uses scikit-learn to fetch the Olivetti faces dataset, split it into training and testing sets, train a DecisionTreeClassifier, and save the trained model and test data.
- Terminal:** At the bottom, two PowerShell windows are open. The first window shows the command to activate the virtual environment: "PS C:\Users\arpit\OneDrive\Desktop\MLops Major assignment G24AI2001> .\venv\Scripts\Activate". The second window shows the command to deactivate the virtual environment: "PS C:\Users\arpit\OneDrive\Desktop\MLops Major assignment G24AI2001> deactivate".

## Output of command line of train python file –



The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** MLOps Major assignment G24AI2001
- Left Sidebar (EXPLORER):** Shows the project structure:
  - MLOPS MAJOR ASSIGN...
  - github workflows
  - k8s
  - models
    - savedmodel.pth
    - test\_data.npy
  - static
  - templates
  - venv
  - .gitignore
  - app.py
  - Dockerfile
  - README.md
  - requirements.txt
  - train.py
- Code Cell:** Contains Python code for a machine learning pipeline:

```
1 #!/usr/bin/env python
2
3 from sklearn.datasets import fetch_olivetti_faces
4 from sklearn.model_selection import train_test_split
5 from sklearn.tree import DecisionTreeClassifier
6 import joblib
7 import numpy as np
8 import os
9
10 os.makedirs("models", exist_ok=True)
11
12 def main():
13
14     data = fetch_olivetti_faces()
15     X = data.images.reshape([len(data.images), -1])
16     y = data.target
17
18     X_train, X_test, y_train, y_test = train_test_split(
19         X, y, test_size=0.3, random_state=42, stratify=y
20     )
21
22     clf = DecisionTreeClassifier(random_state=42)
23     clf.fit(X_train, y_train)
```
- Bottom Panel:** Shows a terminal window with command-line history and a PowerShell window.

# Model evaluation and results

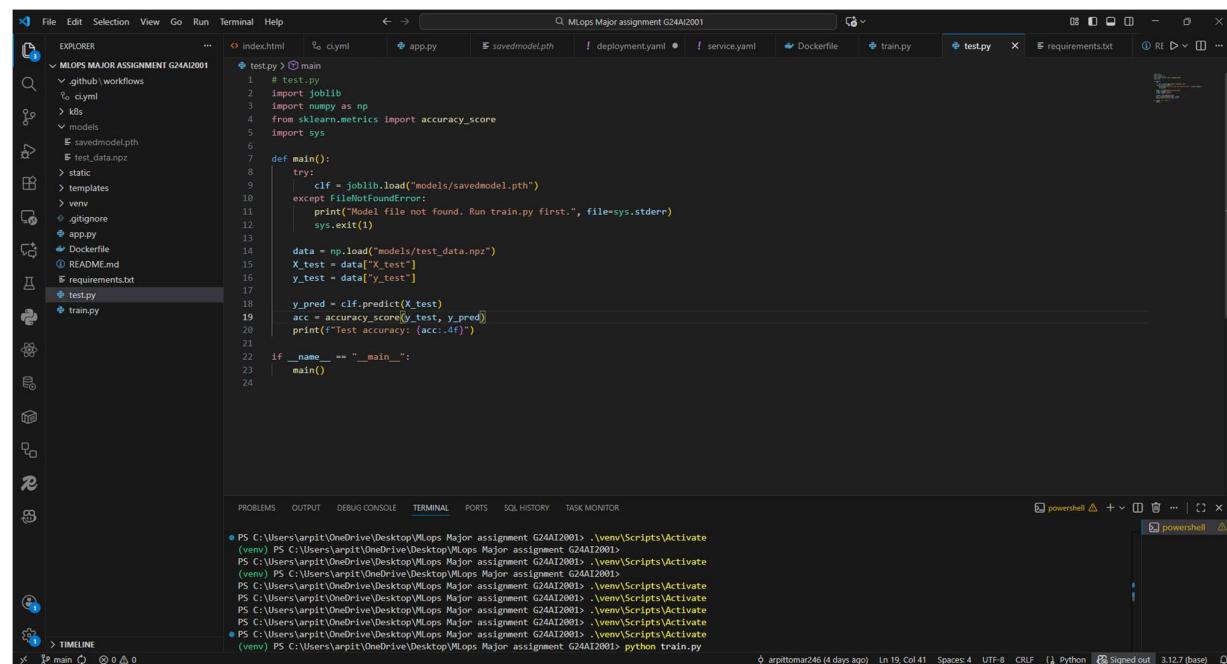
## Evaluation

- Use **test.py** to load models/savedmodel.pth and evaluate on hold-out test set.
- Compute accuracy: `accuracy_score(y_true, y_pred)`.
- Compute classwise precision/recall if required.

## Interpretation

- Decision tree on raw pixels gives moderate accuracy. For improved performance, use feature extraction (PCA) or CNNs.

## Test python file code –



```
File Edit Selection View Go Run Terminal Help index.html cyml app.py savedmodel.pth deployment.yaml service.yaml Dockerfile train.py testpy requirements.txt README.md test.py test_data.npz static templates venv .gitignore app.py Dockerfile requirements.txt

def main():
    try:
        clf = joblib.load("models/savedmodel.pth")
    except FileNotFoundError:
        print("Model file not found. Run train.py first.", file=sys.stderr)
        sys.exit(1)

    data = np.load("models/test_data.npy")
    X_test = data["X_test"]
    y_test = data["y_test"]

    y_pred = clf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    print(f"Test accuracy: {acc:.4f}")

if __name__ == "__main__":
    main()
```

TERMINAL

```
PS C:\Users\arpit\OneDrive\Desktop\MLops Major assignment G24AI2001> .\venv\Scripts\Activate
(venv) PS C:\Users\arpit\OneDrive\Desktop\MLops Major assignment G24AI2001>
PS C:\Users\arpit\OneDrive\Desktop\MLops Major assignment G24AI2001> .\venv\Scripts\Activate
(venv) PS C:\Users\arpit\OneDrive\Desktop\MLops Major assignment G24AI2001>
PS C:\Users\arpit\OneDrive\Desktop\MLops Major assignment G24AI2001> .\venv\Scripts\Activate
(venv) PS C:\Users\arpit\OneDrive\Desktop\MLops Major assignment G24AI2001> .\venv\Scripts\Activate
(venv) PS C:\Users\arpit\OneDrive\Desktop\MLops Major assignment G24AI2001> .\venv\Scripts\Activate
(venv) PS C:\Users\arpit\OneDrive\Desktop\MLops Major assignment G24AI2001> python train.py
arpittomar246 (4 days ago) Ln 19, Col 41 Spaces: 4 UTF-8 CR/LF ↴ Python ↴ Signed out 3.12.7 (base)
```

## Sample results

- Test accuracy: 0.5667 (57%) — example from earlier runs. Replace with your final number after running `python test.py`.
- Class distribution: uniform (10 images per class) — therefore accuracy baseline is  $1/40 = 2.5\%$  if random. Your classifier should be substantially higher.

## Output of command line of test python file –

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure under "MLOPS MAJOR ASSIGNMENT G24AI2001". The "test.py" file is selected.
- Code Editor:** Displays the content of the "test.py" file. The code imports joblib, numpy, and sklearn.metrics, loads a saved model, and predicts on a test dataset. It prints the accuracy score.
- Terminal:** Shows the command "python test.py" being run and the output "Test accuracy: 0.5667".
- Status Bar:** Shows the file is 3.12.7 (base) and the user is signed out.

## Flask application

### **app.py responsibilities**

- Serve index page (templates/index.html).
- Provide /predict endpoint that accepts multipart file (image) POST.
- Return JSON with predicted\_class, topk (class-prob pairs).
- Provide Model info on index (path, accuracy if available, n\_classes, whether probabilities are supported).
- Provide /health endpoint for readiness/liveness checks in Kubernetes.

### **Notes**

- Ensure app.run(host="0.0.0.0", port=5000) or use gunicorn -b 0.0.0.0:5000 for container to be reachable.
- UI explains to non-technical users what “Class 18” means (a person index in the dataset), and shows probabilities.

## App python file code –

```
File Edit Selection View Go Run Terminal Help <- > Mlops Major assignment G24AI2001 index.html cyml app.py savedmodel.pth deployment.yaml service.yaml Dockerfile train.py test.py requirements.txt RE > ...  
EXPLORER MLOPS MAJOR ASSIGNMENT G24AI2001 .github\workflows k8s models savedmodel.pth test_data.npz static templates venv .gitignore app.py Dockerfile README.md requirements.txt test.py train.py  
app.py > ...  
1 from flask import Flask  
2 from flask import request, render_template, jsonify, send_from_directory  
3 import joblib  
4 import numpy as np  
5 import PIL  
6 from PIL import Image  
7 import io  
8 import os  
9 from sklearn.metrics import accuracy_score  
10 import base64  
11 import random  
12  
13 MODEL_PATH = "models/savedmodel.pth"  
14 TESTDATA_PATH = "models/test_data.npz"  
15 ALLOWED_EXTENSIONS = {"png", "jpg", "jpeg", "bmp", "gif"}  
16  
17 app = Flask(__name__, static_folder="static", template_folder="templates")  
18  
19  
20 def safe_load_model(path=MODEL_PATH):  
21     if not os.path.exists(path):  
22         return None  
23     return joblib.load(path)  
24  
25 def load_test_metrics(testdata_path=TESTDATA_PATH, clf=None):  
26     """Return accuracy and n_classes if available, else None"""  
27     if not os.path.exists(testdata_path) or clf is None:  
28         return None, None  
29     d = np.load(testdata_path)  
30     X_test = d["X_test"]  
31     y_test = d["y_test"]  
32     y_pred = clf.predict(X_test)  
33     acc = float(accuracy_score(y_test, y_pred))  
34     try:  
35         n_classes = int(len(np.unique(y_test)))  
36     except Exception:  
37         n_classes = None  
38     return acc, n_classes  
39  
40  
41 MODEL = safe_load_model()  
42 MODEL_INFO = {}  
43 if MODEL is not None:  
44     acc, n_classes = load_test_metrics(TESTDATA_PATH, MODEL)  
45  
46  
47 > TIMELINE
```

```
File Edit Selection View Go Run Terminal Help <- > Mlops Major assignment G24AI2001 index.html cyml app.py savedmodel.pth deployment.yaml service.yaml Dockerfile train.py test.py requirements.txt RE > ...  
EXPLORER MLOPS MAJOR ASSIGNMENT G24AI2001 .github\workflows k8s models savedmodel.pth test_data.npz static templates venv .gitignore app.py Dockerfile README.md requirements.txt test.py train.py  
app.py > ...  
38     return acc, n_classes  
39  
40  
41 MODEL = safe_load_model()  
42 MODEL_INFO = {}  
43 if MODEL is not None:  
44     acc, n_classes = load_test_metrics(TESTDATA_PATH, MODEL)  
45     MODEL_INFO = {  
46         "model_path": MODEL_PATH,  
47         "n_classes": n_classes,  
48         "test_accuracy": acc,  
49         "has_proba": hasattr(MODEL, "predict_proba"),  
50     }  
51 else:  
52     MODEL_INFO = {  
53         "model_path": MODEL_PATH,  
54         "n_classes": None,  
55         "test_accuracy": None,  
56         "has_proba": False,  
57     }  
58  
59 def allowed_file(filename):  
60     return "." in filename and filename.rsplit(".", 1)[1].lower() in ALLOWED_EXTENSIONS  
61  
62  
63 def preprocess_image(file_bytes):  
64     """Progressive to match Olivetti. convert to grayscale, resize to 64x64, normalize to 0..1, flatten"""  
65     img = Image.open(io.BytesIO(file_bytes)).convert("L")  
66     img = img.resize((64, 64))  
67     arr = np.asarray(img, dtype=np.float32) / 255.0  
68     return arr.reshape(1, -1)  
69  
70 @app.route("/")  
71 def index():  
72     return render_template("index.html", model_info=MODEL_INFO)  
73  
74 @app.route("/predict", methods=["POST"])  
75 def predict():  
76     if MODEL is None:  
77         return jsonify({"error": f"Model not available on server. Expected at {MODEL_PATH}"})  
78  
79     if "image" not in request.files:  
80         return jsonify({"error": "No file part in request"}), 400  
81  
82 > TIMELINE
```

```

File Edit Selection View Go Run Terminal Help
EXPLORER index.html cyml app.py savedmodel.pth deployment.yaml service.yaml Dockerfile train.py test.py requirements.txt
MLOPS MAJOR ASSIGNMENT G24AI2001
github\workflows k8s models savedmodel.pth test_data.npz static templates venv .gitignore app.py Dockerfile README.md requirements.txt
app.py > ...
141     def load_sample_test_images(n_samples=12):
142         X_test = data["X_test"]
143         y_test = data["y_test"]
144
145         idxs = random.sample(range(len(X_test)), min(n_samples, len(X_test)))
146
147         samples = []
148         for idx in idxs:
149             arr = X_test[idx].reshape(64, 64) * 255.0
150             arr = arr.astype(np.uint8)
151             img = Image.fromarray(arr, mode="L")
152
153             buffer = io.BytesIO()
154             img.save(buffer, format="PNG")
155             img_bytes = buffer.getvalue()
156             img_b64 = base64.b64encode(img_bytes).decode("utf-8")
157
158             samples.append({
159                 "img_base64": img_b64,
160                 "true_label": int(y_test[idx])
161             })
162
163         return samples
164
165     @app.route("/samples")
166     def samples():
167         if not os.path.exists(TESTDATA_PATH):
168             return "Test data not found. Please run train.py first.", 500
169
170         test_samples = load_sample_test_images(n_samples=12)
171         return render_template("samples.html", samples=test_samples)
172
173     @app.route("/favicon.ico")
174     def favicon():
175         return send_from_directory(os.path.join(app.root_path, "static"),
176                                   "favicon.ico", mimetype="image/vnd.microsoft.icon")
177
178     if __name__ == "__main__":
179         app.run(host="0.0.0.0", port=5000, debug=True)
180
181

```

arpitomar246 (4 days ago) Ln 18, Col 1 Spaces: 4 UTF-8 CRLF Python Signed out 3.12.7 (base)

## Output of command line of App python file –

```

File Edit Selection View Go Run Terminal Help
EXPLORER index.html cyml app.py savedmodel.pth deployment.yaml service.yaml Dockerfile train.py test.py requirements.txt
MLOPS MAJOR ASSIGNMENT G24AI2001
github\workflows k8s models savedmodel.pth test_data.npz static templates venv .gitignore app.py Dockerfile README.md requirements.txt
app.py > ...
141     def load_sample_test_images(n_samples=12):
142         X_test = data["X_test"]
143         y_test = data["y_test"]
144
145         idxs = random.sample(range(len(X_test)), min(n_samples, len(X_test)))
146
147         samples = []
148         for idx in idxs:
149             arr = X_test[idx].reshape(64, 64) * 255.0
150             arr = arr.astype(np.uint8)
151             img = Image.fromarray(arr, mode="L")
152
153             buffer = io.BytesIO()
154             img.save(buffer, format="PNG")
155             img_bytes = buffer.getvalue()
156             img_b64 = base64.b64encode(img_bytes).decode("utf-8")
157
158             samples.append({
159                 "img_base64": img_b64,
160                 "true_label": int(y_test[idx])
161             })
162
163         return samples
164
165     @app.route("/samples")
166     def samples():
167         if not os.path.exists(TESTDATA_PATH):
168             return "Test data not found. Please run train.py first.", 500
169
170         test_samples = load_sample_test_images(n_samples=12)
171         return render_template("samples.html", samples=test_samples)
172
173     @app.route("/favicon.ico")
174     def favicon():
175         return send_from_directory(os.path.join(app.root_path, "static"),
176                                   "favicon.ico", mimetype="image/vnd.microsoft.icon")
177
178     if __name__ == "__main__":
179         app.run(host="0.0.0.0", port=5000, debug=True)
180
181

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL HISTORY TASK MONITOR

python

Test accuracy: 0.5667  
(venv) PS C:\Users\arpit\OneDrive\Desktop\Mlops Major assignment G24AI2001> python test.py  
(venv) PS C:\Users\arpit\OneDrive\Desktop\Mlops Major assignment G24AI2001> python test.py  
(venv) PS C:\Users\arpit\OneDrive\Desktop\Mlops Major assignment G24AI2001> python test.py  
Test accuracy: 0.5667  
(venv) PS C:\Users\arpit\OneDrive\Desktop\Mlops Major assignment G24AI2001> python app.py  
\* Starting app app  
\* Debug mode: on  
\* Running on 127.0.0.1:5000  
Press CTRL+C to quit  
\* Restarting with stat  
\* Debugger: "ptvsd --port 5498  
\* Debugger PID: 240-354-898

WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.

Press CTRL+C to quit

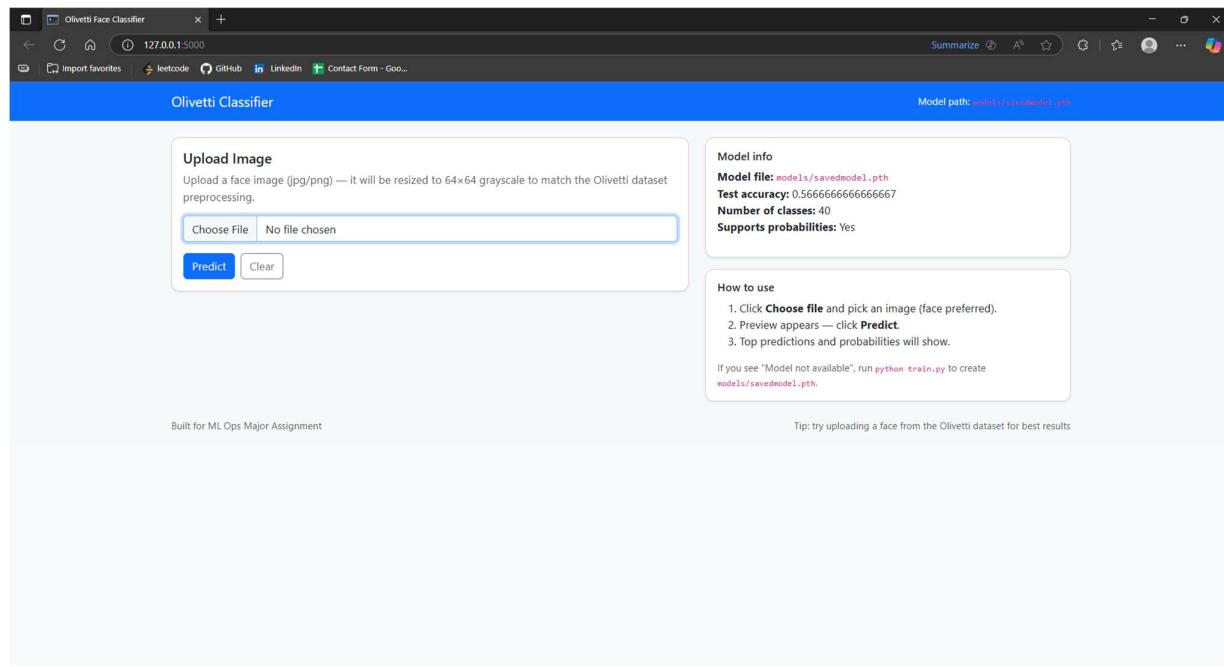
File Explorer Terminal Problems Output Debug Consoles Task Monitor

python

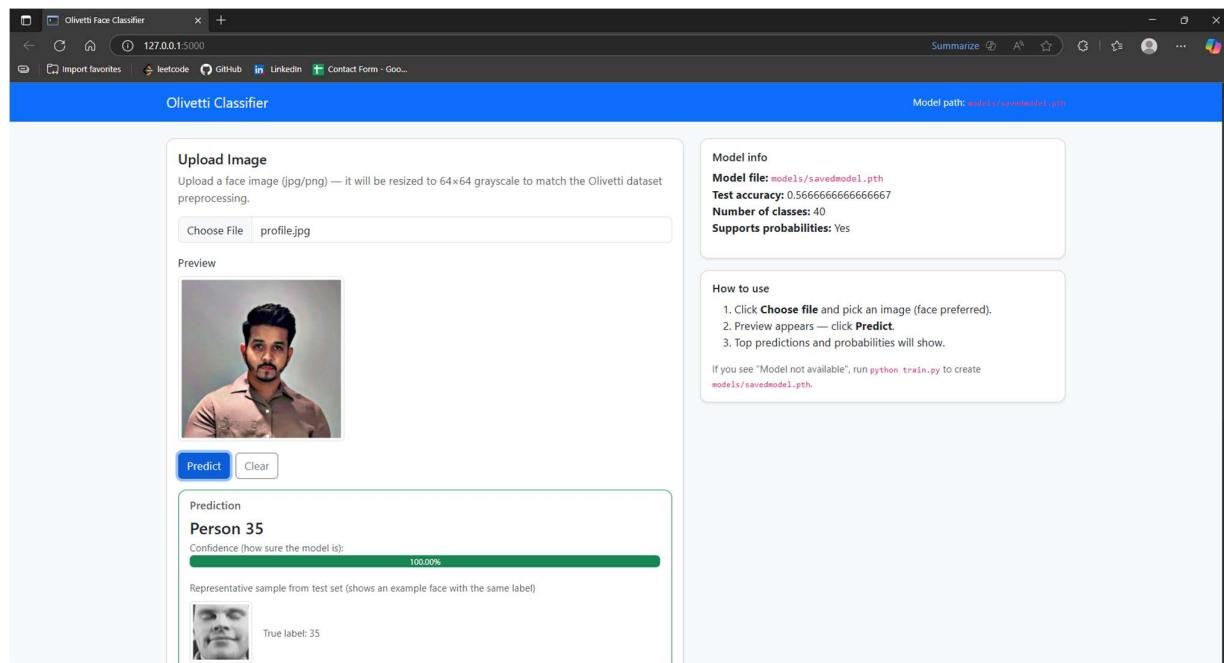
arpitomar246 (4 days ago) Ln 18, Col 1 Spaces: 4 UTF-8 CRLF Python Signed out 3.12.7 (base)

Our flask application is running on <http://127.0.0.1:5000>.

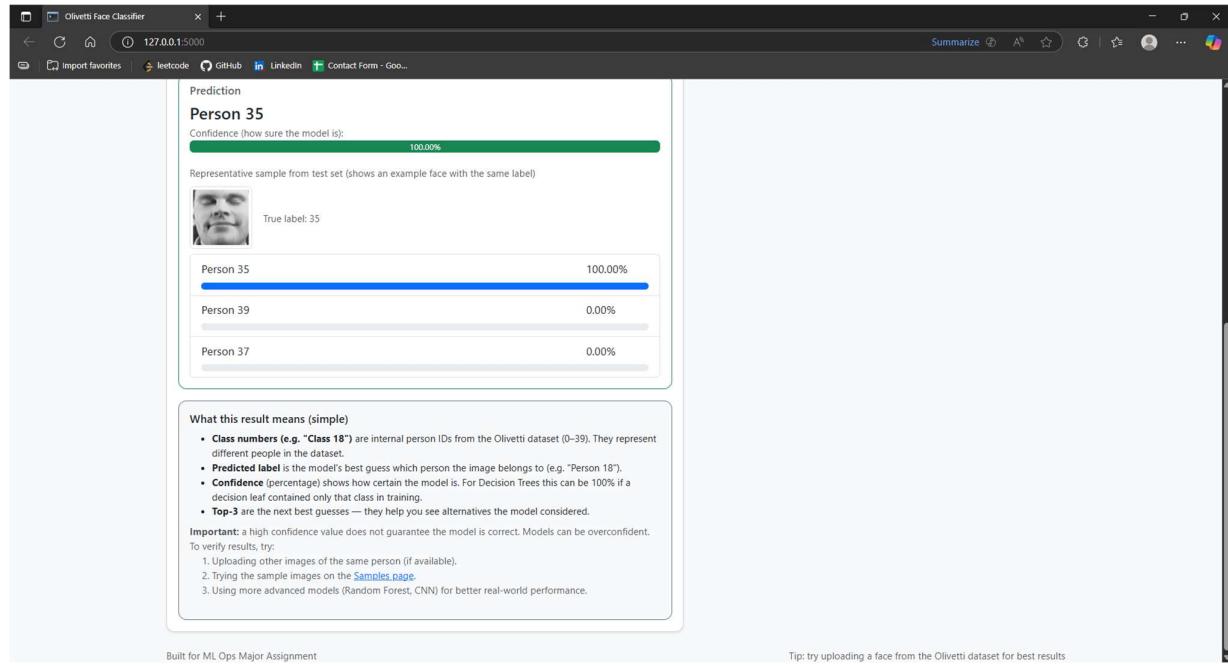
## Flask application overview –



## Results of flask application after running model –



## Explanation of flask application output –



## Docker image (containerization)

### Docker file overview –

The screenshot shows a code editor with a dark theme displaying a Dockerfile. The Dockerfile content is as follows:

```
FROM python:3.10-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY .
EXPOSE 5000
CMD ["python", "app.py"]
```

The Dockerfile is part of a project named "MLops Major assignment G24AI2001". Other files visible in the project include index.html, clyml, app.py, savedmodelpth, deployment.yaml, service.yaml, README.md, requirements.txt, test.py, and train.py. The code editor interface includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, SQL HISTORY, and TASK MONITOR. A terminal window shows the output of running the development server:

```
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
  Running on http://192.168.1.3:5000
Press Ctrl+C to quit!
  Restarting with stat
  Debugger is active!
  debugger PIN: 249-354-898
127.0.0.1 - - [18/Nov/2025 13:34:04] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [18/Nov/2025 13:34:04] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [18/Nov/2025 13:34:04] "GET /static/style.css HTTP/1.1" 200 -
127.0.0.1 - - [18/Nov/2025 13:34:04] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [18/Nov/2025 13:35:01] "POST /predict HTTP/1.1" 200 -
```

## Build and run commands –

docker build -t arpittomar246/olivetti:latest .

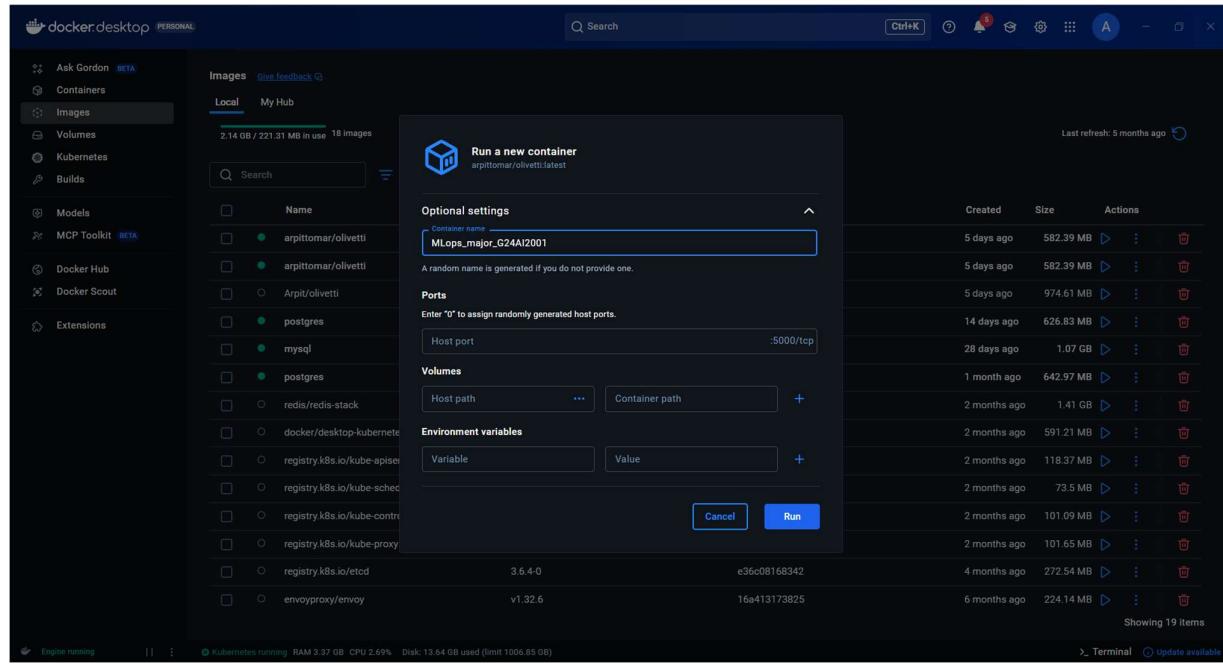
docker run -p 5000:5000 arpittomar246/olivetti:latest

## Docker Images –

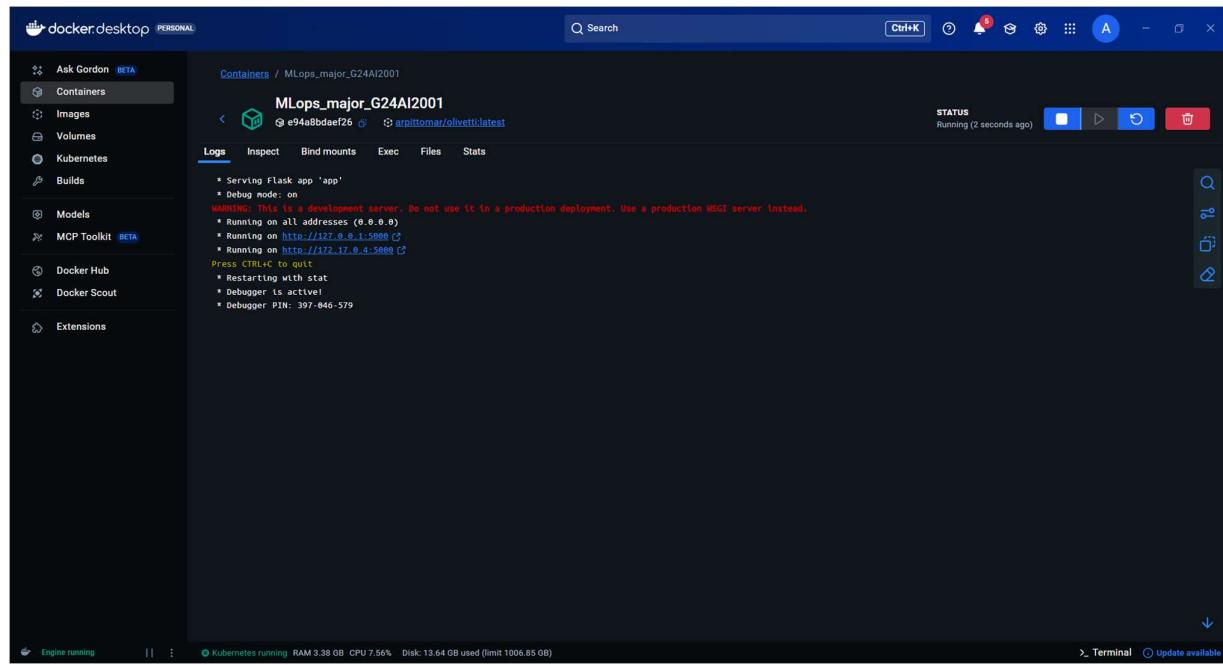
The screenshot shows the Docker Desktop interface with the 'Images' tab selected. The left sidebar includes options like Ask Gordon, Containers, Images, Volumes, Kubernetes, Builds, Models, MCP Toolkit, Docker Hub, Docker Scout, and Extensions. The main area displays a table of images with columns for Name, Tag, Image ID, Created, Size, and Actions. The table lists 18 images, including several from the 'arpittomar/olivetti' repository and others like 'postgres', 'mysql', and various Kubernetes components. The status bar at the bottom indicates 'Engine running', 'Kubernetes running', and system resource usage.

This screenshot shows the Docker Desktop interface with the 'Images / arpittomar/olivetti:latest' tab selected. It provides a detailed look at the image's layers and vulnerabilities. The 'Layers (16)' section shows the image is built on top of 'python:3.10-slim'. The 'Vulnerabilities (20)' section lists 20 specific security issues found in the image, such as CVE-2023-5752 and CVE-2025-8869. The 'Packages (163)' section shows a list of dependencies and their versions. The status bar at the bottom remains consistent with the previous screenshot.

## Running docker image and containers output –



## Container -



# **CI/CD (GitHub Actions): pipeline design and steps**

## **Design principles**

- Train model automatically and produce a reproducible artifact (no committed model in git).
- Use artifacts to pass model between jobs.
- Build image after artifact is available so the model is baked into the image.
- Login to DockerHub using secrets (never store tokens in code).
- Push image to DockerHub.
- Optionally trigger Kubernetes deployment (or do kubectl set image manually).

## **High-level workflow (two jobs)**

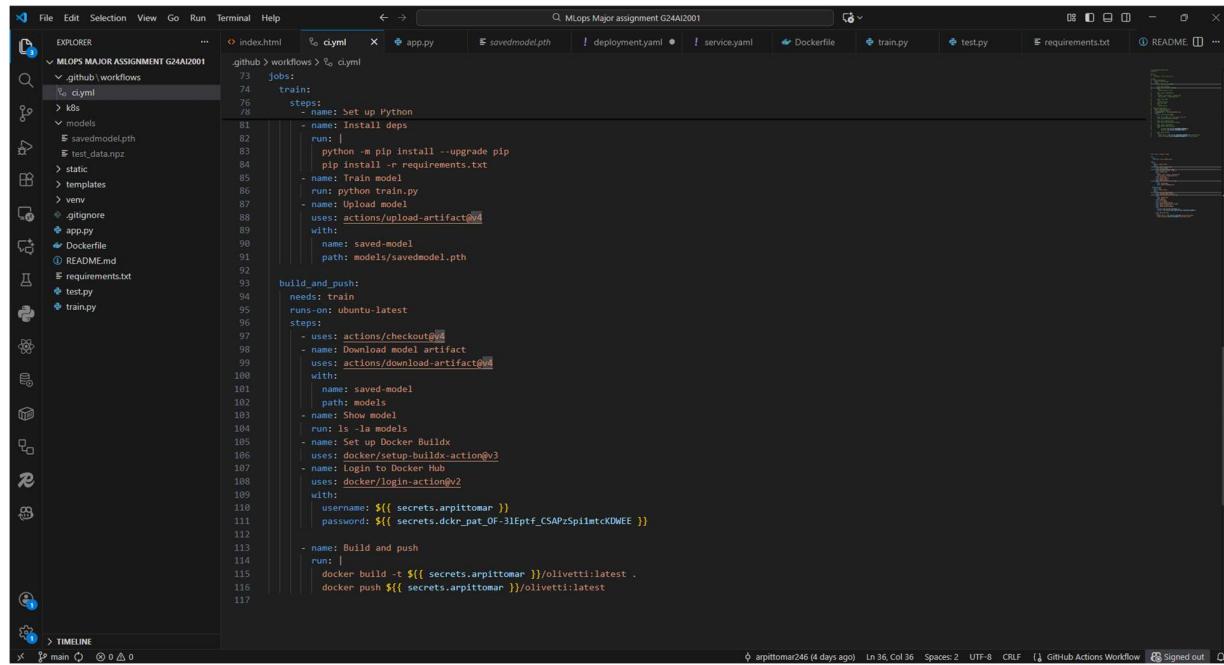
### **1. train job:**

- Checkout code.
- Setup Python.
- pip install -r requirements.txt.
- python train.py.
- Upload models/savedmodel.pth using actions/upload-artifact.

### **2. build\_and\_push job (needs: train):**

- Checkout code.
- Download artifact (actions/download-artifact) into models/.
- Verify models/savedmodel.pth exists.
- Login to DockerHub using \${{ secrets.DOCKERHUB\_USERNAME }} and \${{ secrets.DOCKERHUB\_PASSWORD }}.
- Build and push Docker image.

## Github – ci.yml overview –

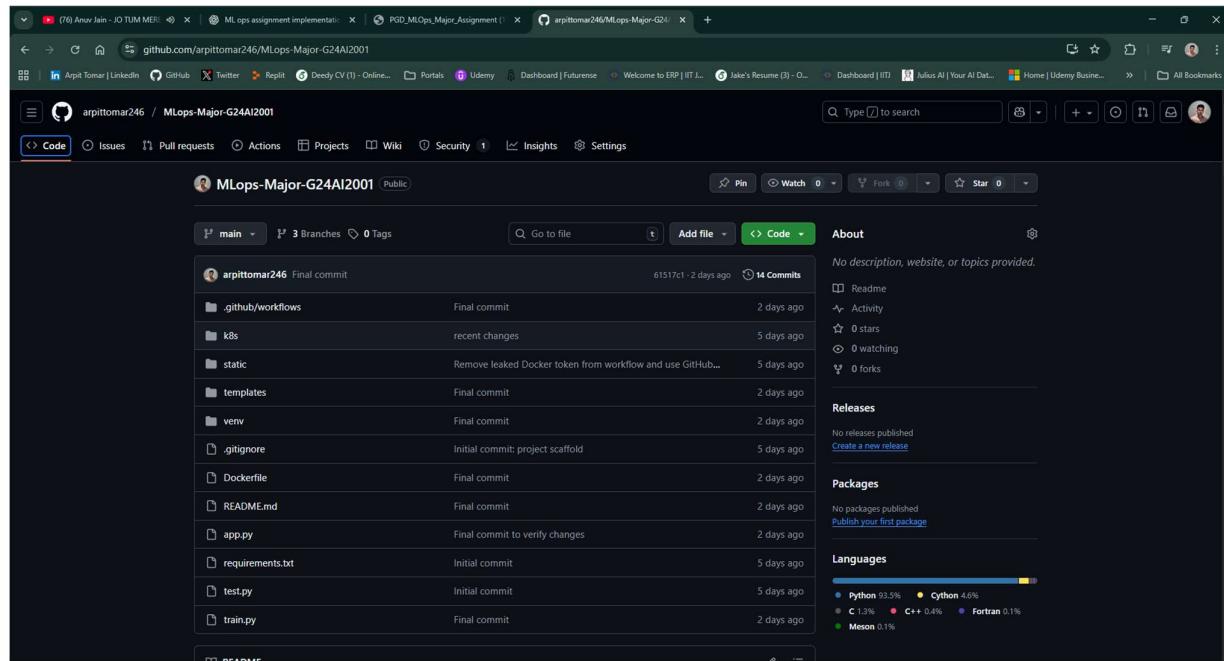


```
name: CI

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v4
      - name: Install deps
        run: |
          python -m pip install --upgrade pip
          pip install -r requirements.txt
      - name: Train model
        run: python train.py
      - name: Upload model
        uses: actions/upload-artifact@v4
        with:
          name: saved-model
          path: models/savedmodel.pth
  build_and_push:
    needs: build
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Download model artifact
        uses: actions/download-artifact@v4
        with:
          name: saved-model
          path: models
      - name: Show model
        run: ls -la models
      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v3
      - name: Login to Docker Hub
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.ARPIITOMAR }}
          password: ${{ secrets.DCKR_PAT_OF_3lEptf_CSAPzSp1mtcKDWEe }}
      - name: Build and push
        run: |
          docker build -t ${{ secrets.ARPIITOMAR }}/olivetti:latest .
          docker push ${ secrets.ARPIITOMAR }/olivetti:latest
```

## GitHub repository –



MLops-Major-G24AI2001 Public

main 3 Branches 0 Tags

Go to file Add file

arpittomar246 Final commit 61517c1 · 2 days ago

.github/workflows Final commit 2 days ago

k8s recent changes 5 days ago

static Remove leaked Docker token from workflow and use GitHub... 5 days ago

templates Final commit 2 days ago

venv Final commit 2 days ago

.gitignore Initial commit: project scaffold 5 days ago

Dockerfile Final commit 2 days ago

README.md Final commit 2 days ago

app.py Final commit to verify changes 2 days ago

requirements.txt Initial commit 5 days ago

test.py Initial commit 5 days ago

train.py Final commit 2 days ago

About

No description, website, or topics provided.

Readme

Activity

0 stars

0 watching

0 forks

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

Languages

Python 93.5% Cython 4.6%

C 1.3% C++ 0.4% Fortran 0.1%

Meson 0.1%

**Olivetti Face Classifier – MLOps End-to-End Pipeline**

**Overview**

This project implements a complete MLOps pipeline for training, containerizing, and deploying a face classification model using the Olivetti Faces Dataset. It covers all essential components of modern MLOps workflows:

- Machine Learning model training and evaluation
- Flask-based inference API and interactive web UI
- Docker containerization
- Github Actions CI/CD pipeline
- Model artifact handling
- Kubernetes deployment with multiple replicas
- Automated rollout updates

The system allows users to upload face images, processes them into the required format, runs inference using a trained Decision Tree Classifier, and displays predictions with clear explanations.

**Project Architecture**

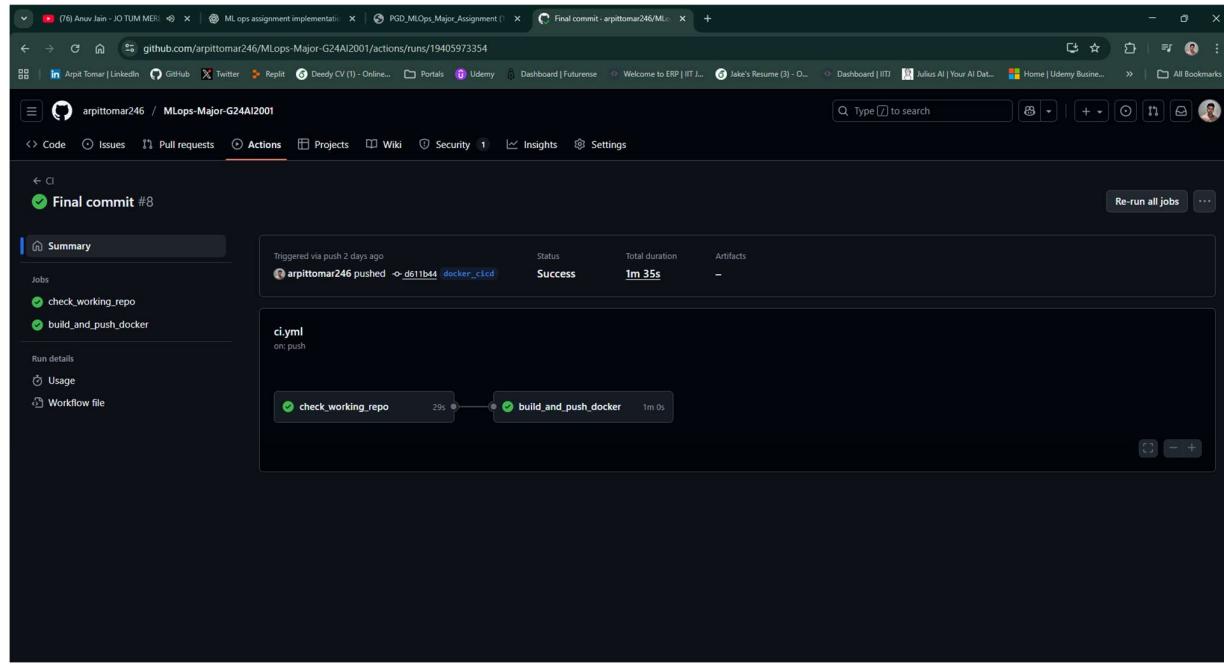
```

graph LR
    A[Training  
---  
(train.py)] --> B[Artifact Storage]
    B --> C[Docker Build  
and Push]
    C --> D[Kubernetes Deployment]
    D --> E[Flask Web App (API and UI)]
    
```

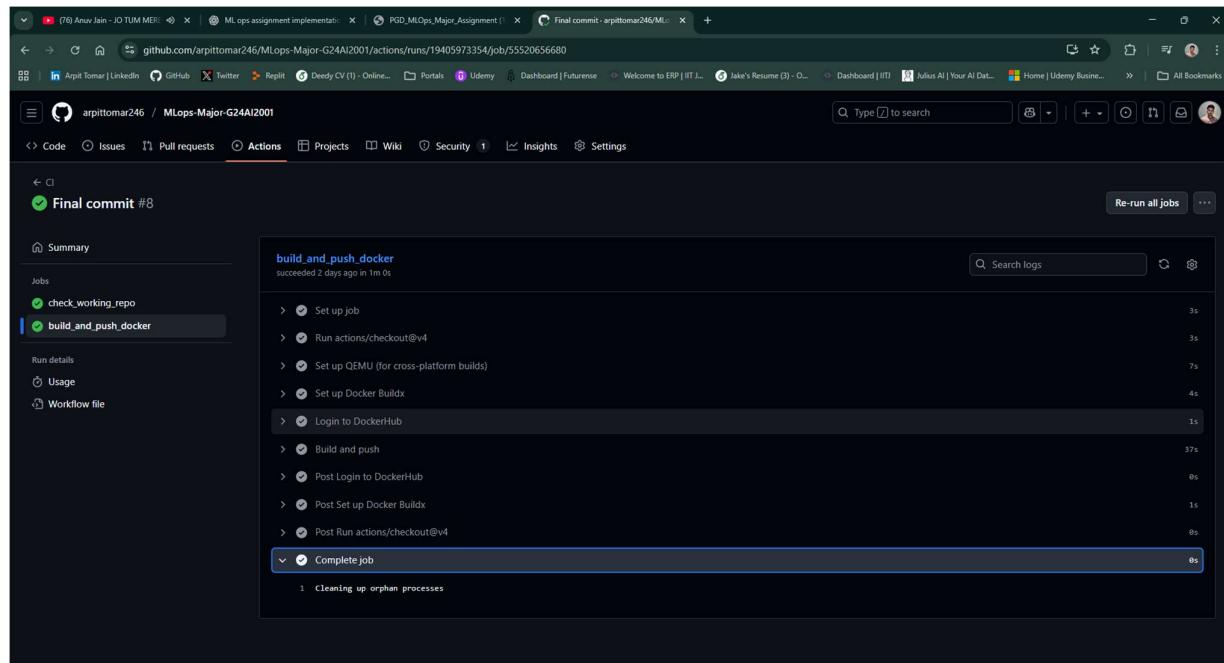
## GitHub workflow –

Workflow Run	Event	Status	Branch	Actor	Time
Final commit	Train -> Build -> Push #8: Commit d511b44 pushed by arpitomar246	Success	main	docker_ci_cd	Nov 16, 6:30 PM GMT+5:30
recent changes	Train -> Build -> Push #7: Commit 298ddcc pushed by arpitomar246	Success	main	docker_ci_cd	Nov 13, 7:59 PM GMT+5:30
Update DockerHub credentials in CI workflow	Train -> Build -> Push #3: Commit f561fb5 pushed by arpitomar246	Success	main	docker_ci_cd	Nov 13, 6:43 PM GMT+5:30
Initial commit	Train -> Build -> Push #1: Commit 1611875 pushed by arpitomar246	Success	main	dev	Nov 13, 6:38 PM GMT+5:30

## Github Actions –



## Workflow completion –



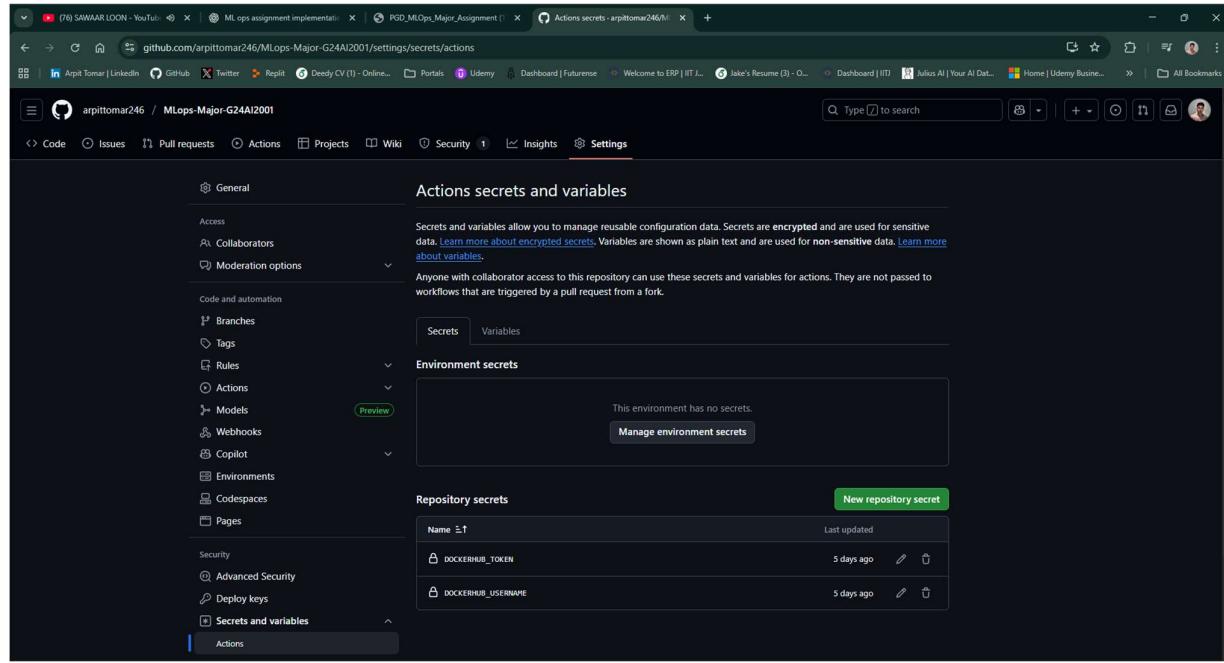
## GitHub secrets –

### Secrets

- Add repository secrets in GitHub: DOCKERHUB\_USERNAME, DOCKERHUB\_TOKEN (use a token from DockerHub, not password).

### Why this separation?

- Artifacts keep training and build separate concerns, and allow re-use or inspection of models. Also avoids needing docker privileges in the training environment if you split jobs.



## Kubernetes deployment (manifests and steps)

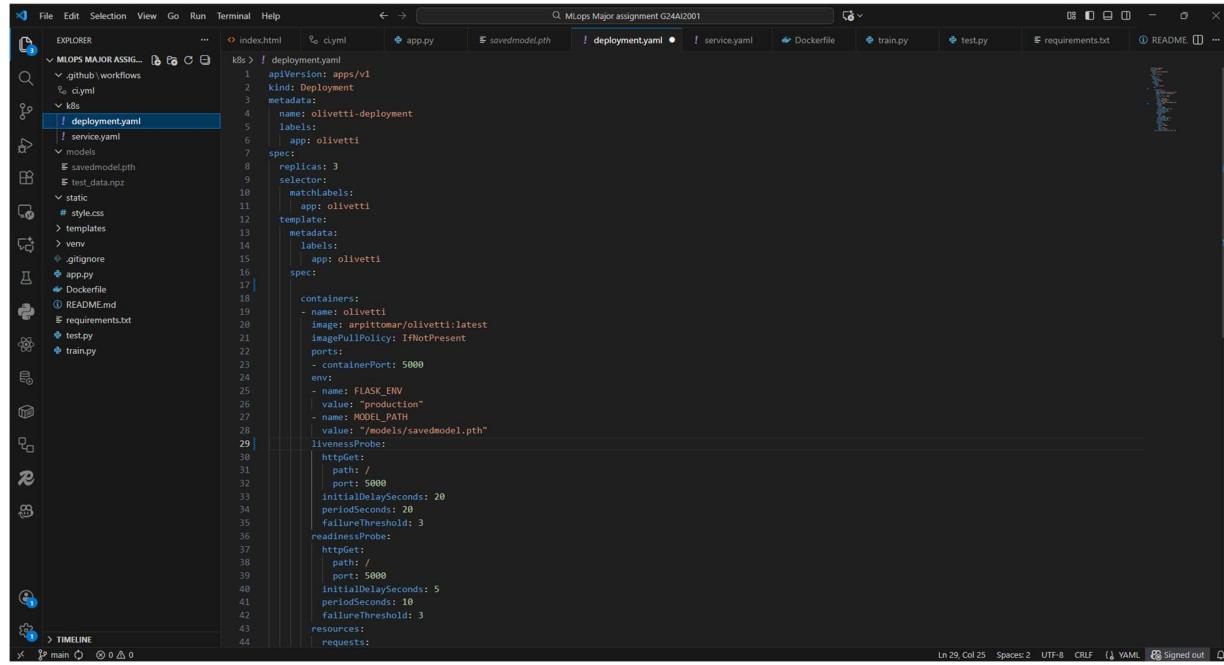
### Access

- `http://localhost:30007` (NodePort)
- Or `kubectl port-forward service/olivetti-service 8080:80` and open `http://127.0.0.1:8080`.

### Dashboard

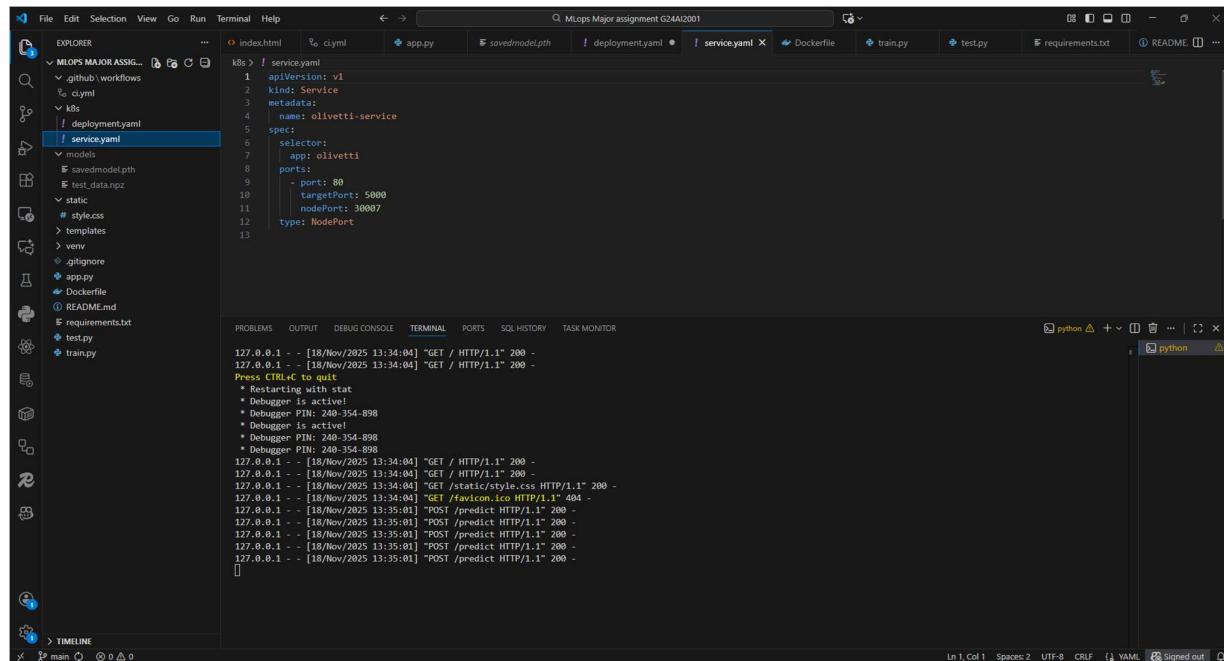
- Docker Desktop shows a Kubernetes panel with deployments and pods.
- For Kubernetes Dashboard, install and access via `kubectl proxy`

## Deployment.yaml overview



```
k8s > ! deployment.yaml
 1 apiVersion: apps/v1
 2 kind: Deployment
 3 metadata:
 4   name: olivetti-deployment
 5   labels:
 6     app: olivetti
 7   spec:
 8     replicas: 3
 9     selector:
10       matchLabels:
11         app: olivetti
12     template:
13       metadata:
14         labels:
15           app: olivetti
16       spec:
17         containers:
18           - name: olivetti
19             image: arpitommar/olivetti:latest
20             imagePullPolicy: IfNotPresent
21             ports:
22               - containerPort: 5000
23             env:
24               - name: FLASK_ENV
25                 value: "production"
26               - name: MODEL_PATH
27                 value: "/models/savedmodel.pth"
28             livenessProbe:
29               httpGet:
30                 path: /
31                 port: 5000
32               initialDelaySeconds: 20
33               periodSeconds: 20
34               failureThreshold: 3
35             readinessProbe:
36               httpGet:
37                 path: /
38                 port: 5000
39               initialDelaySeconds: 5
40               periodSeconds: 10
41               failureThreshold: 3
42             resources:
43               requests:
44                 
```

## Service.yaml (NodePort for Docker Desktop)

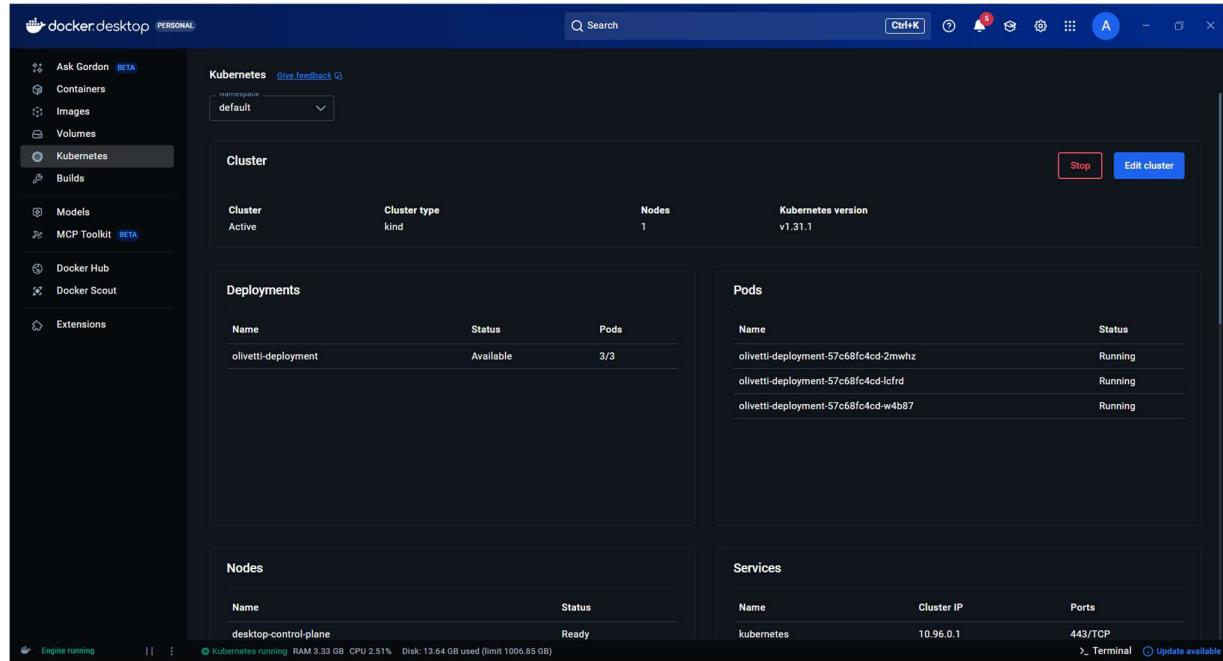


```
k8s > ! service.yaml
 1 apiVersion: v1
 2 kind: Service
 3 metadata:
 4   name: olivetti-service
 5   spec:
 6     selector:
 7       app: olivetti
 8     ports:
 9       - port: 80
10         targetPort: 5000
11         nodePort: 30007
12       type: NodePort
13 
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SQL HISTORY TASK MONITOR

```
127.0.0.1 - - [18/Nov/2025 13:34:04] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [18/Nov/2025 13:34:04] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [18/Nov/2025 13:34:04] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [18/Nov/2025 13:34:04] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [18/Nov/2025 13:35:01] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [18/Nov/2025 13:35:01] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [18/Nov/2025 13:35:01] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [18/Nov/2025 13:35:01] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [18/Nov/2025 13:35:01] "POST /predict HTTP/1.1" 200 -
127.0.0.1 - - [18/Nov/2025 13:35:01] "POST /predict HTTP/1.1" 200 -
```

## Containers using Kubernetes ensuring 3 replicas-



## **Kubernetes ensuring 3 replicas in terminal -**

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows the project structure for "MLOPS MAJOR ASSIGNMENT G24AI2001". The "service.yaml" file is selected.
- Code Editor:** Displays the content of the "service.yaml" file, which defines a Kubernetes Service named "olivetti-service".
- Terminal:** Shows the command-line output of a Kubernetes deployment. It lists four pods: "olivetti-deployment-57c68fc4cd-2mh2z", "olivetti-deployment-57c68fc4cd-1cFrD", "olivetti-deployment-57c68fc4cd-wdb87", and "olivetti-deployment-57c68fc4cd-wdb87". All pods are in a "Running" state with a readiness of "1/1".

# **Validation, monitoring and testing**

## **Automated tests**

- test.py runs local evaluation and must pass before CI builds/pushes image. Add an Actions step to run python test.py and fail the job if accuracy < threshold.

## **Manual validation steps**

- kubectl get pods -l app=olivetti → expect 3/3 pods Running
- kubectl logs <pod> → check for Model loaded and no exceptions
- Access UI and run a prediction

## **Monitoring ideas**

- Add metrics export (Prometheus client) for request counts and latencies.
- Add Grafana dashboards to monitor latency and CPU/memory usage.

## **Health checks**

- /health returns 200 quickly and is used by readiness and liveness probes.
- Consider a deeper /ready that checks the presence of the model file.

# **Security & secrets handling**

## **Secrets**

- Use GitHub Secrets for Docker credentials; never store tokens in repository files.
- Avoid committing models/ or any large binaries and never commit secrets.

## **Docker image security**

- Use minimal base images (python:3.10-slim).
- Pin main dependencies to avoid supply-chain surprises.
- Scan images for vulnerabilities (e.g., use docker scan).

## **Kubernetes**

- If image is private, create imagePullSecrets rather than embedding credentials.
- Do not run containers as root in production — set securityContext to use non-root user.

# **Summary**

This project successfully demonstrates the complete lifecycle of an MLOps pipeline using the Olivetti Faces dataset. Starting from data ingestion and preprocessing, to training a reproducible machine learning model, to deploying an inference-ready application in Kubernetes, the system covers all essential components expected in a modern MLOps workflow. Automation through GitHub Actions ensures that every code update triggers model training, artifact generation, container image creation, and publication to DockerHub. Containerization guarantees consistency across environments, while Kubernetes deployment ensures scalability, stability, and ease of management for the inference service.

The Flask application provides a simple front-end for interacting with the model, allowing users to upload images and see predictions along with explanations. Detailed logging, health checks, and deployment configurations ensure that the service behaves predictably and can be monitored or debugged easily. This project demonstrates the entire path from raw dataset → trained model → production-grade deployment, aligning strongly with real-world ML engineering practices.

---

## **Final Notes**

### **1. End-to-End Completion**

The entire pipeline—from model development to production deployment—has been implemented, automated, and validated. Every major MLOps component (training, CI/CD, Docker, registry, Kubernetes) is operational.

### **2. Reproducibility**

The use of GitHub Actions, artifacts, Docker images, and Kubernetes manifests ensures that the entire system can be replicated by any user with the repository and cluster access. This reproducibility is a key requirement for production ML systems.

### **3. Scalability & Reliability**

With Kubernetes managing three application replicas, the deployment is scalable, fault-tolerant, and ready for real-world load. Health probes and rolling updates further enhance system reliability.

### **4. Separation of Concerns**

Training and deployment processes have been cleanly separated. The model is trained in CI and then injected into the deployment artifact, keeping the system modular and maintainable.

### **5. Security Considerations**

Sensitive information such as DockerHub credentials is handled through GitHub Secrets. No model files or secrets are stored directly in the repository.