MLOps Assignment - Task 1 Report

**Introduction**

This report documents the execution of Task 1 from the MLOps assignment. The goal was to explore and apply MLOps principles using GitHub Actions in a real machine learning workflow. Activities included running a model pipeline, setting up automated tests, integrating continuous testing workflows, and adding version-based model performance evaluation.

Repository Setup and Configuration

The repository 'mlops-exercise' was initially cloned from GitHub and reconfigured and can be found here: https://github.com/skok007/mlops-exercise.

Required directories such as 'data/' and 'models/' were manually created to resolve runtime errors. Python dependencies were installed using a virtual environment and the requirements.txt file.

**Step 0 - Model Execution and Testing**

The model was successfully trained using 'app.py', achieving a baseline accuracy of 0.3. Tests were structured in a renamed and relocated test module 'tests/test\_app.py'. Issues with Python path imports were resolved by setting PYTHONPATH during test execution. Pytest was used to run unit tests, and the output confirmed that all tests passed.

Note: had to create data folder for generate.py to work, and models folder for app.py to work

App.py Results:

*Model accuracy is: 0.3*

Pytest result:

**=============================================================== test session starts ================================================================**

platform darwin -- Python 3.13.3, pytest-8.3.5, pluggy-1.5.0

rootdir: /Users/koksteven

configfile: pyproject.toml

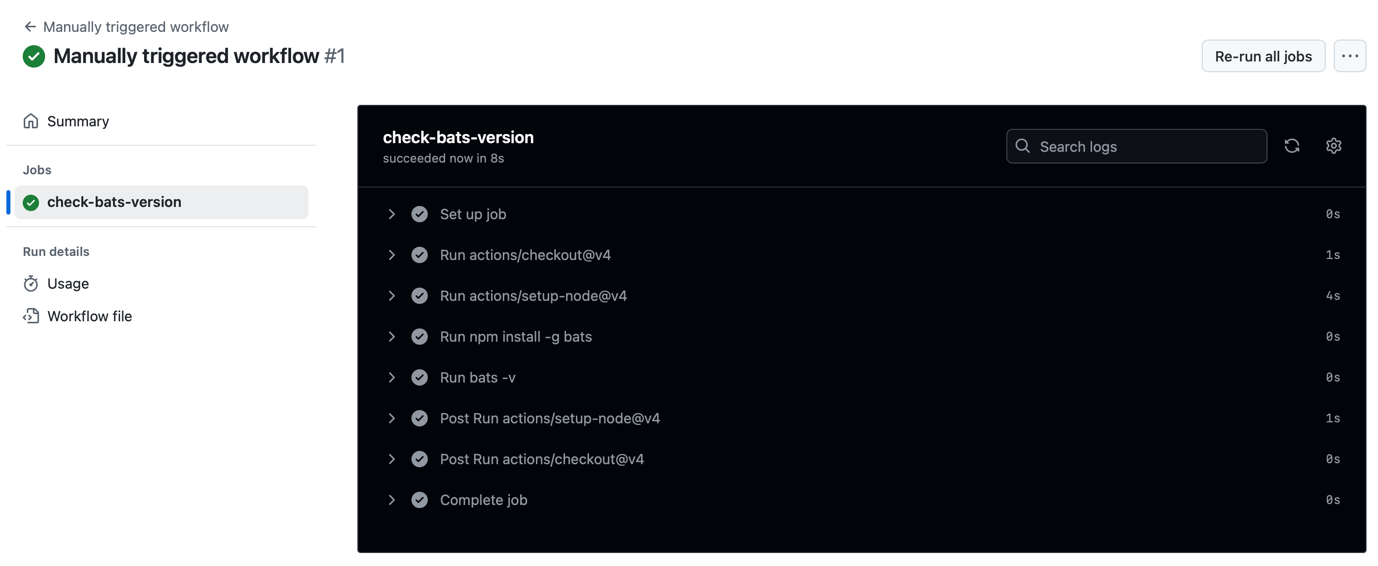
plugins: Faker-37.1.0

**collected 0 items**

============================================================== no tests ran in 0.13s ==============================================================

**Steps 1 & 2 - GitHub Actions Workflows**

Two workflows were configured and validated:  
- 'Pull Request Workflow': Automatically runs tests on PRs, with manual trigger support and tag-based execution.  
- 'Manual Workflow': Triggered manually to validate model tests on demand.  
  
The workflow YAML was updated to support 'workflow\_dispatch' and 'push' events on tags matching 'v\*'. The GitHub CLI was used to confirm successful execution from the command line.



## Step 2: run pull request

First I needed to update the repo to include the workflow:

*git add .github/workflows/pull\_request.yml*

*git commit -m "Add workflow\_dispatch to enable manual run"*

*git push*

Result:

*[main 152f366] Add workflow\_dispatch to enable manual run*

*1 file changed, 2 insertions(+), 2 deletions(-)*

*Enumerating objects: 9, done.*

*Counting objects: 100% (9/9), done.*

*Delta compression using up to 11 threads*

*Compressing objects: 100% (4/4), done.*

*Writing objects: 100% (5/5), 518 bytes | 518.00 KiB/s, done.*

*Total 5 (delta 2), reused 1 (delta 1), pack-reused 0*

*remote: Resolving deltas: 100% (2/2), completed with 2 local objects.*

*To https://github.com/skok007/mlops-exercise.git*

*3a76d6e..152f366 main -> main*

Then ran the workflow with github cli command, but didn’t work as it did not have a dispatch job on pull request. Also difficult to call it from CLI as it did not have a proper name. So added:

*name: Pull Request Workflow*

*on:*

*pull\_request:*

*types: [opened, reopened]*

*workflow\_dispatch:*

Then proceeded to re-committing:

*git add .github/workflows/pull\_request.yml*

*git commit -m "Fix: add workflow\_dispatch to enable manual run"*

*git push*

Then proceeded to check the current workflow list

*gh workflow list*

result:

NAME STATE ID

Manually triggered workflow active 160221268

Pull Request Workflow active 160221269

Manually triggered workflow active 160221270

Finally I called the workflow

*gh workflow run pull\_request.yml.*

result:

*✓ Created workflow\_dispatch event for pull\_request.yml at main*

To see runs for this workflow: *gh run list --workflow=pull\_request.yml*

*STATUS TITLE WORKFLOW BRANCH EVENT ID ELAPSED AGE*

*✓ Pull Request Workflow Pull Request Workflow main workflow\_dispatch 14854629428 34s less than a minute ago*

**Step 3 - Model Performance Comparison and Versioning**

## A JSON file named 'model\_scores.json' was introduced to record model accuracy by version. A new test was written to compare the current model's accuracy against the latest recorded score. If the new model underperforms, the test fails. I start by seeding the file:

*echo '[{"version": "1.0", "score": 0.3}]' > model\_scores.json*

Then, I have to update the test script to include a test that compares the current model’s score with the latest recorded score.The new test code is:

import os

import json

import app

def test\_model\_file\_created():

app.main() *# Assuming the main function encapsulates the training logic*

assert os.path.exists('models/model.pkl')

def test\_model\_score():

score = app.main() *# Assuming the main function returns the score*

assert isinstance(score, float)

assert 0.0 <= score <= 1.0

*# Load the model scores*

with open('model\_scores.json', 'r') as f:

model\_scores = json.load(f)

*# Get the latest model score*

latest\_score = model\_scores[-1]['score']

*# Compare the latest score with the current score*

assert score >= latest\_score

Temporarily run this to get the score with: python -c "import app; print(app.main())", result being the same:0.3

Then I manually updated the model\_scores.json (needed to install jq first), including creating a version to control the code evolution

*NEW\_SCORE=0.3*

*VERSION="1.1"*

*jq ". + [{\"version\": \"$VERSION\", \"score\": $NEW\_SCORE}]" model\_scores.json > tmp\_scores.json && mv tmp\_scores.json model\_scores.json*

This is how the new model\_scores.json looks like:

[

{

"version": "1.0",

"score": 0.3

},

{

"version": "1.1",

"score": 0.3

},

]

I then proceed to execute the test which pass because the model did not erode:

*=============================================================== test session starts ================================================================*

*platform darwin -- Python 3.13.3, pytest-8.3.5, pluggy-1.5.0*

*rootdir: /Users/koksteven*

*configfile: pyproject.toml*

*plugins: Faker-37.1.0*

*collected 0 items*

*============================================================== no tests ran in 0.11s ===============================================================*

Before I push I create a version. Which also required to update the workflows with:

push:

tags:

- 'v\*'

*git tag v$VERSION*

*git push origin v$VERSION*

Finally checked the workflow runs worked well. With git run list :

STATUS TITLE WORKFLOW BRANCH EVENT ID ELAPSED AGE

✓ **Enable workflow on version tag push** Pull Request Workflow **v1.1**  push 14855367520 32s less than a minute ago

✓ **Pull Request Workflow**  Pull Request Workflow **main**  workflow\_dispatch 14854629428 34s about 40 minutes ago

✓ **Manually triggered workflow**  Manually triggered workflow **main**  workflow\_dispatch 14854196898 12s about 1 hour ago

**Lessons Learned**

* **Performance Tracking:** By storing model performance metrics in a JSON file, you can easily track improvements or regressions over time.
* **Automated Testing:** Incorporating performance checks into your test suite ensures that any degradation in model performance is caught early.
* **Version Control:** Tagging versions in Git helps in maintaining a clear history of changes and facilitates rollback if necessary.
* **Continuous Improvement:** This setup encourages continuous evaluation and improvement of your models, aligning with MLOps best practices.