TP2 – Intégration MLOps complète

(GitHub Actions, DVC, CML, Docker, Google Drive)

Oussama Khouya Radwane Khemisse Achrafe Elalaoui

December 8, 2025

1 Contexte et objectifs

Contexte : projet `ml-dvc-iris` du TP1 (dataset Iris, pipeline DVC).
Objectifs du TP2 :

1. Intégrer le pipeline DVC dans un workflow GitHub Actions.
2. Générer un rapport de métriques avec CML dans les Pull Requests.
3. Configurer un remote DVC Google Drive pour les données et artefacts.
4. Préparer un Dockerfile et la promotion du meilleur modèle vers `models/production_model.pkl`.

2 Architecture MLOps mise en place

L'architecture MLOps repose sur un pipeline DVC défini dans `dvc.yaml` avec plusieurs stages et le suivi des données et artefacts (`data/iris.csv`, `data/iris_preprocessed.csv`, `models/random_forest.pkl`, `metrics/*.json`). Un remote DVC `new_gdrive` sur Google Drive est configuré pour stocker ces artefacts. Un workflow GitHub Actions `mlops-pipeline.yaml` exécute automatiquement `dvc pull`, `dvc repro` et le script `scripts/generate_cml_report.py` afin de produire un rapport CML et un commentaire automatique sur les Pull Requests, en préparation d'une future containerisation Docker et d'un stage `deploy` pour promouvoir le meilleur modèle.

3 Vérification du projet de départ

Les vérifications de base du projet se font en deux étapes :

1. Cloner (ou mettre à jour) le projet `ml-dvc-iris` sur la machine locale.
2. Vérifier que la commande suivante s'exécute localement : `dvc repro` (figure 1).

```
/home/o/p/e/_/D/d/full-MLOps-integration master +4 ?9 ❯ dvc repro
Stage 'prepare' didn't change, skipping
Running stage 'train':
> python src/train.py
Modèle entraîné sauvegardé dans: models/random_forest.pkl
Métriques d'entraînement sauvegardées dans: metrics/train_metrics.json
Accuracy (test): 0.9333
Updating lock file 'dvc.lock'

Running stage 'evaluate':
> python src/evaluate.py
Métriques d'évaluation sauvegardées dans: metrics/eval_metrics.json
Accuracy (données complètes): 0.9867
Updating lock file 'dvc.lock'
```

FIGURE 1 – Exécution locale du pipeline `dvc repro`

4 Workflow GitHub Actions avec DVC et CML

4.1 Exécution réussie du pipeline GitHub Actions

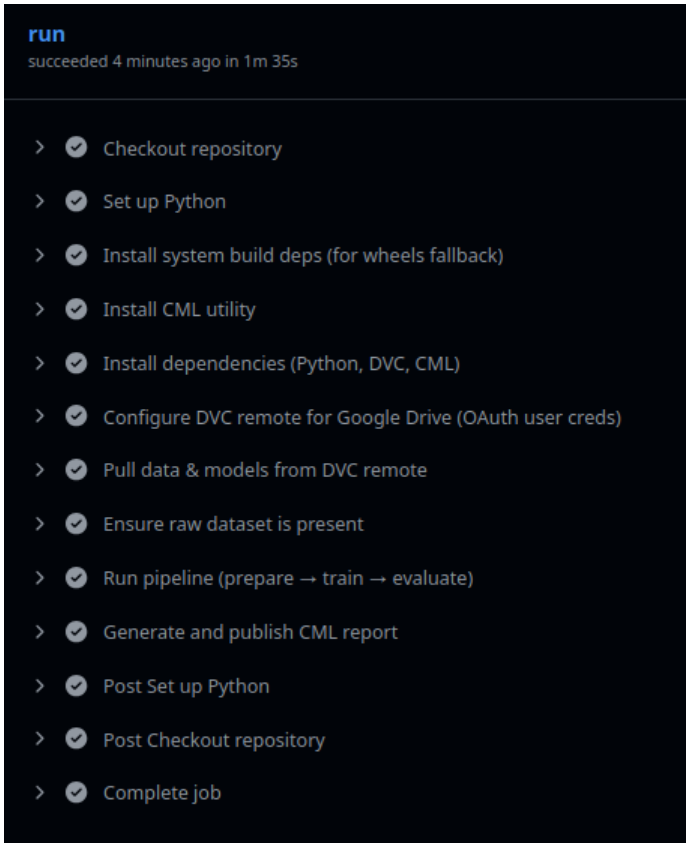

FIGURE 2 – Exécution réussie du pipeline GitHub Actions

4.2 Commentaire CML dans une Pull Request

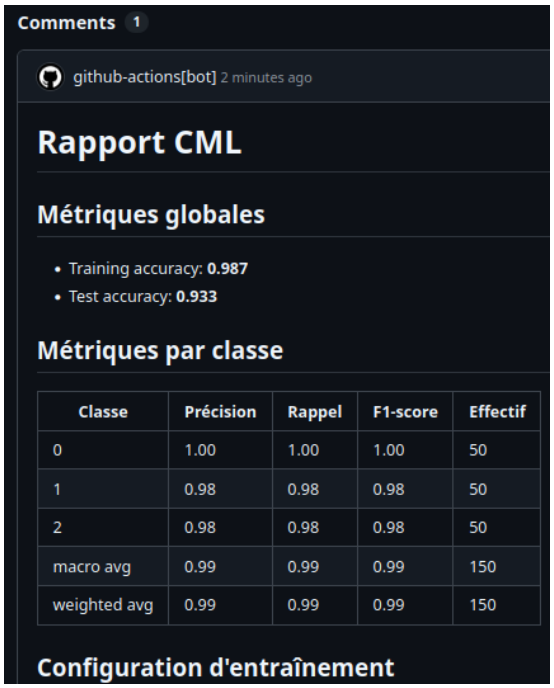

FIGURE 3 – Commentaire CML généré dans la Pull Request

5 Remote DVC Google Drive

— Remote `new_gdrive` pointant vers l'ID Drive, authentifié via secrets OAuth.

Listing 1 – Extrait .dvc/config

```
[core]
    remote = new_gdrive

[remote "new_gdrive"]
    url = gdrive://1qnTG-xYstcnUbljTv94pp2izEqLpBP-o
    gdrive_use_service_account = true
    gdrive_service_account_json_file_path = /home/oldhome/pc/enset/_S3/
       DevOps/dvc/gdrive_user_credentials.json
```

— Synchronisation vérifiée : `dvc pull` (figure 4) et `dvc push` (figure 5).

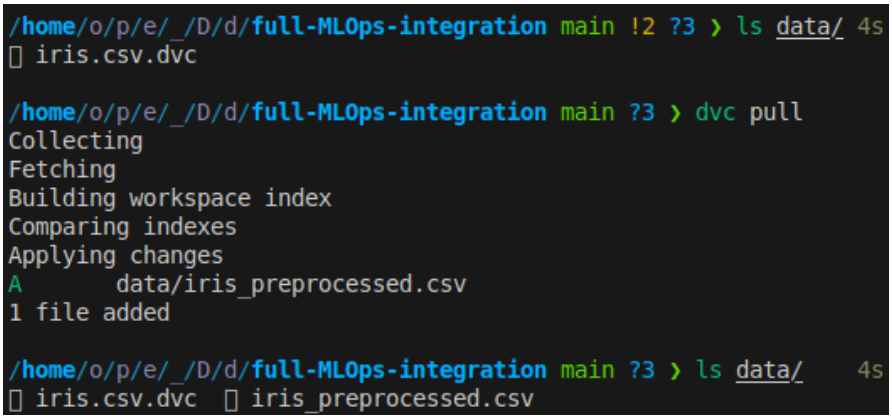

FIGURE 4 – Récupération des artefacts par `dvc pull`

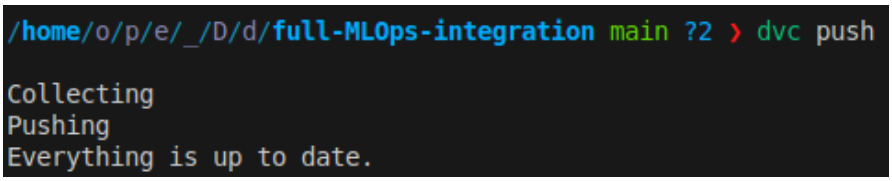

FIGURE 5 – `dvc push` indiquant un cache distant à jour

— Structure de Remote DVC (google drive) : (figure 6).

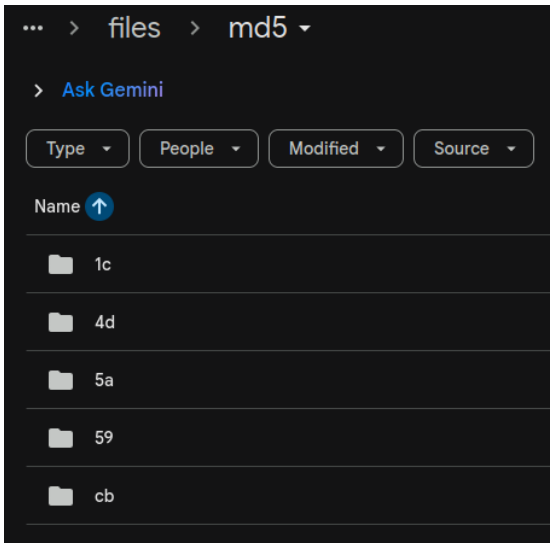

FIGURE 6 – structure dossier dvc (`google drive`)

6 Containerisation Docker

6.1 Création du fichier Dockerfile

À la racine du projet, nous créons un fichier `Dockerfile` qui part de l'image `python:3.11-slim`, installe les dépendances et lance `dvc repro` par défaut. Extrait :

Listing 2 – Extrait du fichier Dockerfile

```
FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install --no-cache-dir --upgrade pip && \
    pip install --no-cache-dir -r requirements.txt

COPY . .

CMD ["dvc", "repro"]
```

6.2 Construction de l'image et test local sur un conteneur

Étapes :

1. Construire l'image : `docker build -t ml-dvc-iris:latest .` (figure 7).
2. Lancer un conteneur de test : `docker run -rm ml-dvc-iris:latest` (figure 8).

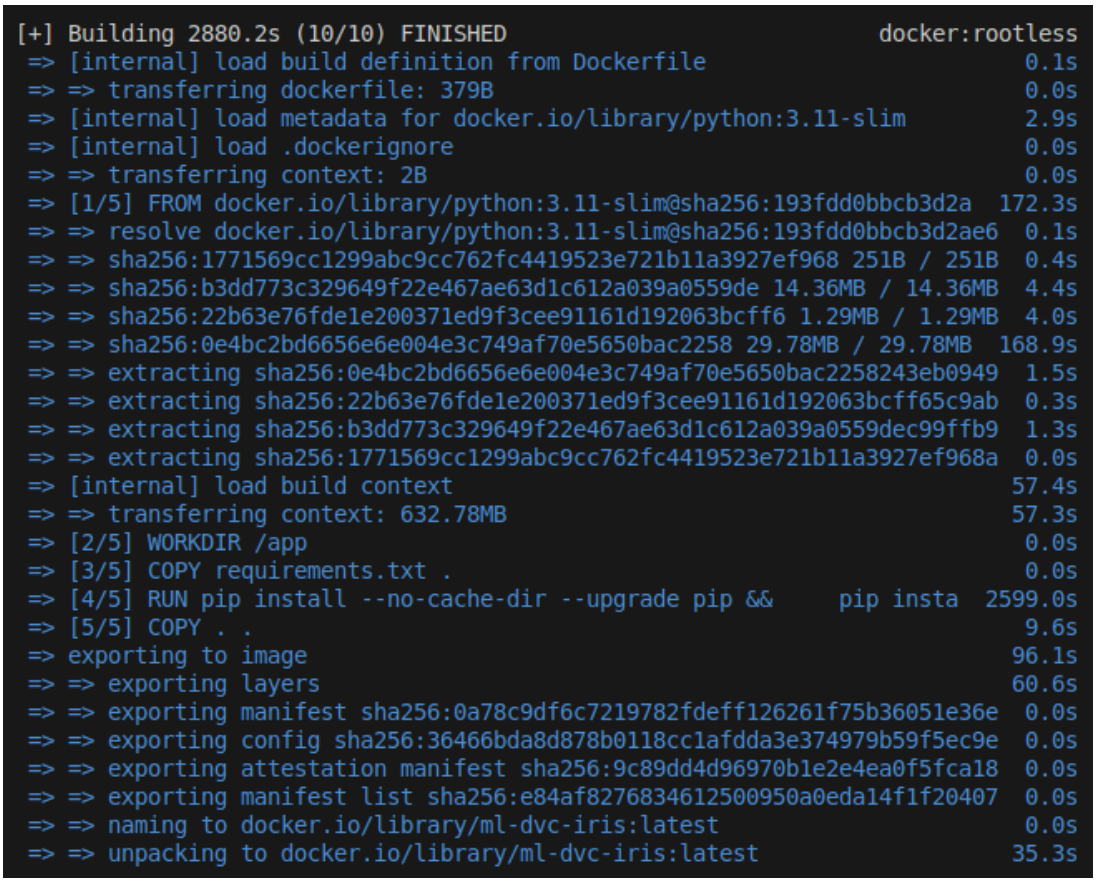

FIGURE 7 – docker buid output (`google drive`)

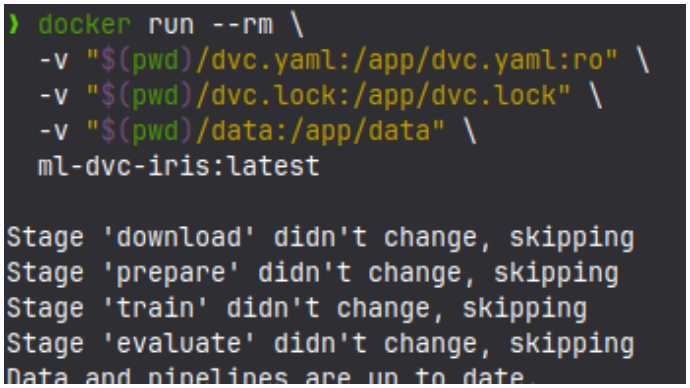

FIGURE 8 – Exécution du conteneur Docker

7 Difficultés rencontrées et solutions

— Docker build très long (48 minutes) pour construire l'image `ml-dvc-iris:latest` (voir figure 7).
— `iris.csv` non versionné par DVC car seulement *deps* et non *out*. Solution : ajouter `iris.csv` comme *out* dans `dvc.yaml` et faire `dvc push`.

Listing 3 – Extrait du nouveau fichier dvc.yaml

```
stages:
  download:
    cmd: python scripts/download_iris.py
    deps:
      - scripts/download_iris.py
    outs:
      - data/iris.csv

  prepare:
    cmd: python scripts/preprocess.py
    deps:
      - scripts/preprocess.py
      - data/iris.csv
    outs:
      - data/iris_preprocessed.csv
```

8 Conclusion

En résumé, la chaîne MLOps mise en place combine DVC (pipeline et versionnement des données/modèles), un remote Google Drive pour les artefacts, un workflow GitHub Actions avec CML pour l'exécution automatique et le rapport dans les PR, ainsi qu'un Dockerfile pour rejouer le pipeline dans un conteneur de façon reproductible.