ECOLE NORMALE SUPÉRIEURE DE L'ENSEIGNEMENT TECHNIQUE MOHAMMEDIA
UNIVERSITÉ HASSAN II DE CASABLANCA

ENSET

المدرسة العليا لأساتذة التعليم التقني
المحمدية
جامعة الحسن الثاني بالدار البيضاء

# TP2 – Intégration MLOps complète (GitHub Actions, DVC, CML, Docker, Google Drive)

Pr. Soufiane HAMIDA

## Contexte du TP

Ce TP fait suite au travail réalisé sur le projet `ml-dvc-iris` (TP1 / ateliers précédents) :

- pipeline DVC complet : `prepare` → `train` → `evaluate`,
- dataset `data/iris.csv` et `data/iris_preprocessed.csv`,
- modèle RandomForest + métriques sauvegardées au format JSON.

Dans ce TP2, vous allez transformer ce projet en une **chaîne MLOps complète** en intégrant :

- GitHub Actions + DVC + CML pour la CI,
- Docker pour containeriser l'exécution du pipeline,
- un remote DVC sur Google Drive pour le stockage des données/artefacts,
- une logique simple de *déploiement du meilleur modèle.*

**Format du TP**

- Travail en binôme ou trinôme.
- Durée : **3 heures** en séance encadrée.
- À la fin de la séance, chaque groupe doit déposer :
  1. le **lien du dépôt GitHub** contenant le projet complet,
  2. un **rapport synthétique** avec des captures d'écran.

## Objectifs pédagogiques

À l'issue de ce TP, vous devrez être capables de :

- intégrer un pipeline DVC dans un workflow GitHub Actions,
- générer un rapport automatique de métriques avec CML sur une Pull Request,
- configurer un remote DVC basé sur Google Drive,
- containeriser le pipeline avec Docker,
- mettre en place une stratégie simple de promotion du meilleur modèle.

## Organisation pratique et livrables

**Consignes de rendu**

1. **Dépôt GitHub** :
   - projet `ml-dvc-iris` avec :
     - pipeline DVC fonctionnel (`dvc repro`),
     - workflow GitHub Actions opérationnel,
     - configuration du remote DVC Google Drive,
     - Dockerfile pour exécuter le pipeline.
2. **Rapport PDF** :
   - courte introduction (contexte et objectifs),
   - description de l'architecture MLOps mise en place,
   - captures d'écran :
     - exécution réussie du pipeline GitHub Actions,
     - commentaire CML dans une Pull Request,
     - structure du remote DVC (Google Drive),
     - image Docker construite / exécutée,
     - fichier `models/production_model.pkl` généré.
   - difficultés rencontrées et solutions,
   - courte conclusion.

## Prérequis techniques

- Projet `ml-dvc-iris` déjà initialisé (TP1) avec :
  - `dvc.yaml`, `params.yaml`, dossier `.dvc/`,
  - scripts de prétraitement, d'entraînement et d'évaluation.
- Compte GitHub fonctionnel et dépôt distant.
- Notions de base sur :
  - Git et GitHub (branches, commits, Pull Requests),
  - GitHub Actions,
  - DVC, CML, Docker.

# 1 Vérification du projet de départ (10–15 min)

**Travail demandé**

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

```
dvc repro
```

3) Vérifier que les fichiers suivants existent (ou les recréer si nécessaire) :
   - `data/iris.csv` et `data/iris_preprocessed.csv`,
   - `models/random_forest.pkl`,
   - `metrics/*.json` (métriques d'entraînement / évaluation).

**Capture d'écran à prévoir dans le rapport :** commande `dvc repro` réussie en local.

## 2 Workflow GitHub Actions avec DVC et CML

**Objectif**

Mettre en place un workflow CI qui :

- s'exécute sur chaque `push` et `pull_request`,
- installe Python, DVC, CML et les dépendances du projet,
- exécute le pipeline DVC (`dvc repro`),
- publie un rapport CML dans la Pull Request.

### 1.1. Dépendances Python

a) Créer (ou compléter) un fichier `requirements.txt` qui contient au minimum :
   - les bibliothèques de votre code (ex : `pandas`, `scikit-learn`, `pyyaml`, `joblib`, ...),
   - `dvc[gdrive]` pour la gestion du remote,
   - `cml` pour les rapports.

b) Commiter ce fichier et le pousser sur GitHub.

### 1.2. Script de génération de rapport CML

Créer un script Python `scripts/generate_cml_report.py` qui :

a) lit les métriques dans les fichiers JSON produits par votre pipeline, par exemple :
   - `metrics/train_metrics.json`,
   - `metrics/eval_metrics.json`,
   - éventuellement `metrics/feature_importances.json`, `metrics/class_metrics.json`.

b) génère un fichier Markdown `reports/cml_report.md` contenant au minimum :
   - un résumé des métriques globales (ex : accuracy),
   - un tableau simple (si vous avez des métriques par classe ou par feature).

**Suggestion de squelette (à compléter)**

```
# scripts/generate_cml_report.py
# TODO: charger les fichiers JSON de metrics/
# TODO: construire une liste de lignes Markdown
# TODO: écrire le fichier reports/cml_report.md
```

### 1.3. Fichier GitHub Actions `.github/workflows/mlops-pipeline.yml`

a) Créer le dossier `.github/workflows` à la racine du projet.

b) Ajouter un fichier `mlops-pipeline.yml` qui :
   - utilise un runner `ubuntu-latest`,
   - installe Python (par ex. 3.11),
   - installe les dépendances via `pip install -r requirements.txt`,
   - exécute `dvc pull` puis `dvc repro`,
   - appelle votre script `scripts/generate_cml_report.py`,
   - publie le rapport comme commentaire CML sur la Pull Request.

**Indication**

Votre fichier YAML doit contenir au moins :

- un bloc `on:` pour déclencher sur `push` et `pull_request`,
- un job avec les `steps` suivants : checkout, setup python, install deps, dvc pull, dvc repro, génération du rapport, commentaire CML.

**Captures d'écran à prévoir :**

- exécution réussie du workflow (onglet `Actions`),
- commentaire CML généré dans la Pull Request.

# 3 Remote DVC Google Drive

**Objectif**

Configurer un remote DVC basé sur Google Drive et l'utiliser dans votre pipeline CI GitHub Actions.

## 2.1. Ajout du remote en local

1) Créer un dossier Google Drive dédié au projet et récupérer son identifiant (ID).
2) En local, ajouter un remote DVC par défaut :

```
dvc remote add -d gdrive_remote gdrive://<ID_DOSSIER_GDRIVE>
```

3) Commiter la configuration DVC (`.dvc/config`).

## 2.2. Test du remote

1) Exécuter :

```
dvc push
```

pour envoyer les données/artefacts vers Google Drive.

2) Supprimer localement un fichier de sortie (par ex. `data/iris_preprocessed.csv`) puis vérifier que :

```
dvc pull
```

le récupère correctement.

## 2.3. Intégration dans GitHub Actions

- Créer un secret GitHub (par ex. `GDRIVE_CREDENTIALS_DATA`) contenant les informations de compte de service Google nécessaires à `dvc[gdrive]`.
- Adapter le workflow pour que `dvc pull` et `dvc push` puissent utiliser ce secret.

**Capture d'écran à prévoir :** contenu du dossier Google Drive (liste des fichiers DVC) et succès de `dvc push`/`dvc pull`.

# 4 Containerisation du pipeline DVC avec Docker

**Objectif**

Créer une image Docker capable d'exécuter le pipeline `dvc repro` pour le projet `ml-dvc-iris`.

### 3.1. Création du fichier `Dockerfile`

À la racine du projet, créer un fichier `Dockerfile` qui :

a) part d'une image Python (par ex. `python:3.11-slim`),

b) copie `requirements.txt` et installe les dépendances,

c) copie le reste du projet dans `/app`,

d) définit la commande par défaut pour lancer `dvc repro`.

**Suggestion de structure minimale**

```
FROM python:3.11-slim
WORKDIR /app
# TODO: copier requirements.txt et installer les deps
# TODO: copier le code du projet
# TODO: definir CMD ["dvc", "repro"]
```

### 3.2. Construction et test local

1) Construire l'image :

```
docker build -t ml-dvc-iris:latest .
```

2) Lancer un conteneur pour tester :

```
docker run --rm ml-dvc-iris:latest
```

**Captures d'écran à prévoir :** commande `docker build` et `docker run` réussies.

## 5 Test end-to-end du workflow sur Pull Request

**Objectif**

Vérifier que toute la chaîne CI s'exécute automatiquement sur une Pull Request : DVC, CML et (optionnellement) Docker.

### 4.1. Création d'une branche de test

1) Créer une nouvelle branche de fonctionnalité :

```
git checkout -b feature/test-mlops
```

2) Modifier un hyperparamètre dans `params.yaml` (par exemple `n_estimators` du RandomForest).

3) Commiter et pousser la branche sur GitHub.

### 4.2. Ouverture de la Pull Request et vérification

1) Ouvrir une Pull Request de `feature/test-mlops` vers `main`.

2) Vérifier que :

- le workflow GitHub Actions s'exécute automatiquement,
- le pipeline DVC se termine sans erreur,
- un commentaire CML est ajouté dans la PR avec les métriques.

**Captures d'écran à prévoir :**

- vue du pipeline dans l'onglet `Actions`,
- commentaire CML dans la PR.

# 6 Déploiement automatique du meilleur modèle

**Objectif**

Mettre en place un stage DVC `deploy` qui promeut automatiquement le meilleur modèle en fonction de l'accuracy.

### 5.1. Script `scripts/deploy.py`

Créer un script Python `scripts/deploy.py` qui :

a) lit les métriques d'évaluation (par ex. `metrics/eval_metrics.json`),

b) compare l'accuracy courante à la meilleure accuracy connue (dans `metrics/best_metrics.json`),

c) si l'accuracy est meilleure :

- copie `models/random_forest.pkl` vers `models/production_model.pkl`,
- met à jour `metrics/best_metrics.json`.

**Indication**

Vous pouvez vous inspirer de la logique suivante :

- fonction utilitaire `load_json(path)` qui renvoie un dict ou `None`,
- comparaison `current_acc > best_acc`,
- utilisation du module `shutil` pour copier le fichier modèle.

### 5.2. Ajout du stage `deploy` dans `dvc.yaml`

1) Ajouter à la fin de votre `dvc.yaml` un stage `deploy` qui :
   - dépend de `scripts/deploy.py`, `models/random_forest.pkl` et `metrics/eval_metrics.json`,
   - produit `models/production_model.pkl` et `metrics/best_metrics.json`.
2) Relancer `dvc repro` pour vérifier que le nouveau stage est bien exécuté.

**Capture d'écran à prévoir :** arborescence du projet montrant `models/production_model.pkl` et `metrics/best_metrics.json` après exécution.

## Conclusion (à rédiger dans votre rapport)

Dans votre rapport, vous devrez conclure en :

- résumant la chaîne MLOps mise en place (DVC + GitHub Actions + CML + Docker + Google Drive),
- expliquant en quelques lignes votre stratégie de promotion du meilleur modèle,
- discutant des limites de votre solution et des pistes d'amélioration (tests supplémentaires, monitoring, déploiement sur un vrai service, etc.).