

Cahier des Charges – Mini-projet MLOps

1 Introduction

Ce mini-projet MLOps a pour but de construire un **workflow de bout en bout** :

- gestion du code (Git),
- conteneurisation (Docker / Docker Compose),
- versioning des données (DVC),
- suivi d'expériences (MLflow),
- pipeline CI/CD/CT (ZenML + GitLab CI (optionnel)),
- optimisation (Optuna),
- déploiement (API d'inférence) ; monitoring et retrain en bonus.

2 Objectifs du projet

- Partir d'un cas d'usage ML simple (CPU ou petit GPU).
- Rendre l'entraînement et l'inférence reproductibles (code + data + config).
- Assurer la traçabilité : version Git + version data + runs MLflow + exécution pipeline.
- Déployer une API d'inférence et simuler une mise à jour ($v1 \rightarrow v2$) + rollback.

3 Contenu du projet

3.1 Cas d'usage, données et modèle

- Le dataset doit être public, léger et entraînable rapidement.
- Un modèle baseline doit être proposé (simple mais mesurable).
- Une métrique principale doit être définie (accuracy, F1, RMSE, mAP, etc.).

3.2 Gestion du code (Git)

- Repo GitLab propre : README, structure claire, branches (main/dev), tags ($v1$, $v2$).

3.3 Conteneurisation (Docker / Docker Compose)

- Dockerfile(s) pour entraîner et/ou servir.
- docker-compose.yml pour exécuter la stack localement.

3.4 Versioning des données (DVC)

- Dataset tracké avec DVC (pas de gros fichiers dans Git).
- Remote DVC fonctionnel (push/pull) et démonstration de reproductibilité.

3.5 Experiment Tracking (MLflow)

- Au moins : 1 run baseline + plusieurs runs comparables (variations simples).
- Log des paramètres, métriques, et artefacts (modèle, figures, etc.).

3.6 Pipeline MLOps (ZenML) + CT

- Pipeline ZenML : data → train → eval → export.
- Plusieurs exécutions de pipeline (baseline + variations).

3.7 Optimisation (Optuna)

- Étude Optuna courte (ex. 5–10 trials) sur quelques hyperparamètres.
- Comparaison simple : baseline / variations / meilleur run Optuna.

3.8 CI/CD (GitLab CI)

- Pipeline CI minimum : tests/lint + build image(s) + push registry.
- (Option CT) : job planifié qui lance un run “smoke” (epochs=1 ou subset).

3.9 Déploiement (Serving)

- Une API d'inférence stable (ex. `/predict`) indépendante du backend.
- Déploiement via Docker Compose.
- Simulation v1 → v2 + rollback (preuve par test/capture).

4 Bonus (Optionnel)

- Monitoring (latence, nb requêtes, erreurs, métriques custom).
- Retrain (déclenchement simple : planifié ou conditionnel).

5 Livrables

- Lien vers Github/GitLab.
- Dockerfiles + docker-compose.yml.
- Configuration DVC (fichiers `.dvc` / `dvc.yaml` si utilisé) + preuve push/pull.
- Captures MLflow (liste + comparaison) et ZenML (runs/pipeline).
- `.gitlab-ci.yml`. (optionnel)
- Démo déploiement : test d'inférence + update (v2) + rollback.
- Documentation : README détaillant exécution, commandes, et structure.

A Annexe – Exemples de projets

- **Tabulaire (classification multi-classe)** : UCI Iris + Baseline : Logistic Regression / SVM (scikit-learn).
- **Tabulaire (classification binaire, santé)** : UCI Breast Cancer (Diagnostic) + Baseline : RandomForest / SVM (scikit-learn).
- **Tabulaire (régression)** : California Housing (scikit-learn) + Baseline : RandomForestRegressor.
- **Série temporelle (prévision / régression)** : UCI Bike Sharing + Baseline : modèle TF (Dense/Conv/RNN) pour forecasting.
- **Détection d'anomalies (streaming)** : Numenta Anomaly Benchmark (NAB) + Baseline : IsolationForest (scikit-learn).
- **Recommandation (collaborative filtering)** : MovieLens 100K + Baseline : SVD (Surprise).
- **NLP (sentiment analysis)** : SST-2 (Hugging Face) + Baseline : DistilBERT (Transformers, sequence classification).
- **NLP (NER / token classification)** : CoNLL-2003 (Hugging Face) + Baseline : DistilBERT (Transformers, token classification).
- **Audio (keyword spotting)** : TFDS Speech Commands + Baseline : petit CNN (TensorFlow tutorial).
- **Graph ML (node classification)** : Cora (via PyTorch Geometric) + Baseline : GCN (PyG tutorial).