Grille d'Évaluation – Déploiement de Modèles ML sur Git

Critères	Description	Note (1-5)
Organisation du dépôt Git	Structure claire (dossier `src/`, `data/`, `notebooks/`, `models/`, `docs/`, etc.).	
Versionnement du code ML	Scripts Python bien versionnés (prétraitement, entraînement, évaluation).	
Tracking des notebooks	Jupyter Notebooks bien nommés et versionnés. Utilisation d'outils comme `nbdime` si besoin.	
Gestion des dépendances	Fichier `requirements.txt`, `environment.yml` ou `pyproject.toml` bien défini.	
Documentation du code	Présence de commentaires et README explicatif (usage, structure, entraînement, prédiction).	
Gestion des données	Datasets non inclus directement dans Git (liens externes ou gestion via DVC / scripts de download).	
Tracking des modèles	Modèles sauvegardés dans un dossier `models/` ou suivis avec DVC/MLflow.	
Reproductibilité	Présence de scripts ou instructions pour reproduire l'entraînement et l'évaluation.	
Gestion des versions de modèles	Historique clair des versions de modèles (tags Git, DVC, MLflow, nommage).	
Utilisation des branches	Utilisation de branches pour les features, expérimentations ou déploiement.	
Messages de commit explicites	Les commits sont clairs, précis, et en lien avec les changements apportés.	
Pipeline d'entraînement	Script ou pipeline clair pour l'entraînement (`train.py`, `run_pipeline.sh`, etc.).	
Tests de performance du modèle	Présence de métriques (accuracy, F1, confusion matrix...) avec courbes et interprétation.	
Déploiement local ou web (Flask/FastAPI)	Application de test simple (API Flask, notebook de prédiction, interface Streamlit...).	
Automatisation (optionnel avancé)	Présence de CI/CD (GitHub Actions, GitLab CI) pour tests automatiques, build ou déploiement.	
Gestion des secrets (optionnel avancé)	Les clés API, tokens ou mots de passe ne sont pas poussés dans Git (utilisation de `.env`).	

Grille d'Évaluation – Déploiement de Modèles ML sur Git