



GL-M326 · LECTURE ACADÉMIQUE

MLOps et Industrialisation des modèles IA

Du modèle au système ML traçable, validé et exploitable



MLOPS
GL-M326

SÉANCE 1 / 6 Comprendre le MLOps : du modèle au système reproductible

ENSEIGNANT

Pr. Youness BOUTYOUR

youness.boutyour@um5.ac.ma

PUBLIC CIBLE

Génie Logiciel

Cycle Ingénieur — 3A

INSTITUTION

Université Mohammed V de Rabat

École Nationale Supérieure d'Informatique et d'Analyse des
Systèmes



GL-M326 · LECTURE ACADÉMIQUE

MLOps et Industrialisation des modèles IA

Du modèle au système ML traçable, validé et exploitable



MLOPS
GL-M326

SÉANCE 1 / 6 Comprendre le MLOps : du modèle au système reproductible

ENSEIGNANT

Pr. Youness BOUTYOUR

youness.boutyour@um5.ac.ma

PUBLIC CIBLE

Génie Logiciel

Cycle Ingénieur — 3A

INSTITUTION

Université Mohammed V de Rabat

École Nationale Supérieure d'Informatique et d'Analyse des
Systèmes

Qui suis-je ?



in

LINKEDIN

id

ORCID



YOUNESS.BOUTYOUR@UM5.AC.MA

ENSEIGNANT • ENSIAS

Pr. Youness BOUTYOUR

Spécialité : Intelligence artificielle

Département • Génie Logiciel

AXES DE RECHERCHE

- 01 **Machine Learning, Deep Learning et représentation des connaissances**
- 02 **Large Language Models (LLM) et Agentic AI**
- 03 **Agents autonomes et systèmes multi-agents**
- 04 **Deep Reinforcement Learning et Multi-Agent Reinforcement Learning (DRL/MARL)**
- 05 **Politiques décentralisées, communication coopérative et décision distribuée**
- 06 **Continual Learning, Federated Learning et environnements adaptatifs**

Plan de la séance

SÉANCE 1 / 6 – COMPRENDRE LE MLOPS : DU MODÈLE AU SYSTÈME REPRODUCTIBLE

01 Cadrage

02 Concepts

03 Démo

04 Synthèse

Objectifs pédagogiques

OBJECTIFS PÉDAGOGIQUES

5 OBJECTIFS

- 01 Expliquer pourquoi un modèle performant ne suffit pas à constituer un système ML exploitable
- 02 Décrire le lifecycle ML et les principaux stages d'un projet MLOps
- 03 Distinguer les pipelines de données, d'entraînement, de validation, de serving et de monitoring
- 04 Identifier les sources de non-reproductibilité d'une expérience ML
- 05 Employer correctement les notions de run, metric, artifact et lineage

CHAPITRE

Comprendre le MLOps : du modèle au système reproductible

01 Cadrage

La question directrice

Pourquoi un modèle de Machine Learning performant n'est-il pas nécessairement prêt pour la production ?

- À la fin de cette séance, vous saurez :
 - expliquer le rôle du **MLOps** ;
 - distinguer les principaux **stages du cycle de vie ML** ;
 - distinguer plusieurs **pipelines MLOps** ;
 - identifier les sources de **non-reproductibilité** ;
 - utiliser correctement : *experiment*, *run*, *parameter*, *metric*, *artifact* et *lineage*.

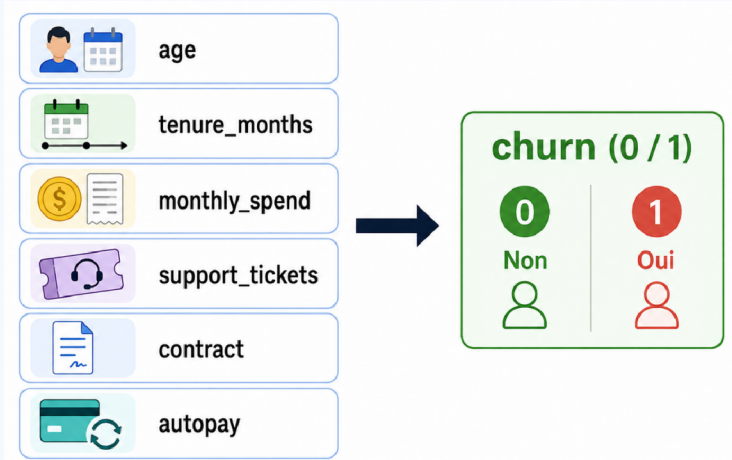
Principes MLOps du module

- Cette séance introduit le vocabulaire commun du module. Les séances suivantes indiquent le sous-ensemble de principes qu'elles approfondissent.

CODE	PRINCIPE	IDÉE CLÉ
P1	Tracer	Relier chaque décision à ses preuves.
P2	Versionner	Identifier code, données, configuration et modèle.
P3	Valider	Vérifier la qualité avant de propager.
P4	Servir	Exposer le modèle approuvé via un contrat exploitable.
P5	Automatiser	Rejouer les étapes et leurs contrôles.
P6	Observer	Mesurer le comportement au runtime.
P7	Décider	Transformer les preuves en action explicite.

Le fil rouge du cours

- **Problème métier** : repérer les clients susceptibles de résilier leur abonnement afin d'agir avant leur départ.
- À partir des informations disponibles au moment de la prédiction — ancienneté, offre, usage et facturation — le modèle estime une **probabilité de churn**.
- Mais un score isolé ne suffit pas : il faut pouvoir expliquer avec quelles données et quel code il a été produit, vérifier sa qualité, le servir et surveiller sa fiabilité.
- Le fil rouge transforme ainsi progressivement ce modèle en **système ML exploitable**, de la séance 1 à la séance 6.



Les six microcycles

MICROCYCLE	DURÉE	QUESTION PRINCIPALE
MC1 — Du modèle au système ML	15 min	Pourquoi le modèle seul ne suffit-il pas ?
MC2 — Définir le MLOps	15 min	Quel problème le MLOps cherche-t-il à résoudre ?
MC3 — Lifecycle et stages	15 min	Que faut-il gérer durant la vie du modèle ?
MC4 — Pipelines MLOps	15 min	Quels flux faut-il distinguer ?
MC5 — Reproductibilité	25 min	Pourquoi deux runs peuvent-ils différer ?
MC6 — Experiment tracking	25 min	Comment rendre une exécution traçable ?

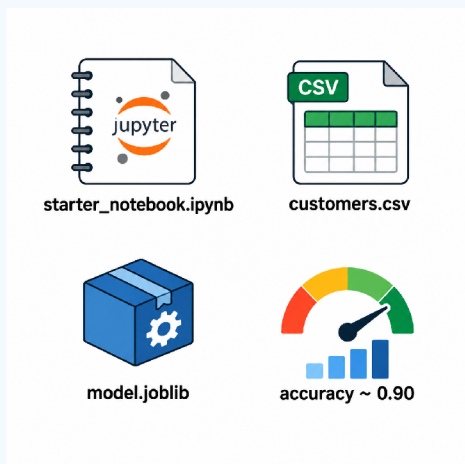
CHAPITRE

Comprendre le MLOps : du modèle au système reproductible

02 Concepts —

Situation initiale - le modèle marche

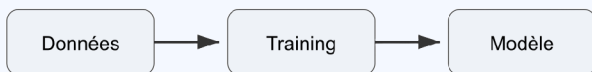
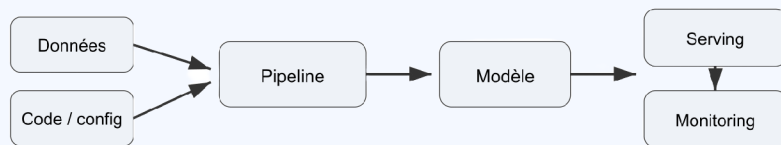
- Une équipe Data Science fournit :



- Avant de parler de MLOps, posez quatre questions :

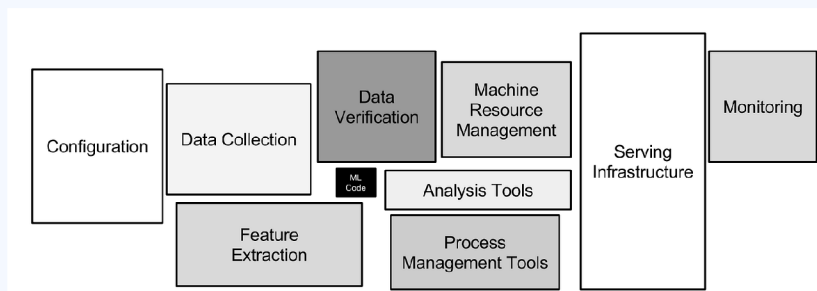
- 01 Peut-on **reproduire** ce résultat ?
- 02 Peut-on identifier exactement les **données** utilisées ?
- 03 Peut-on expliquer d'où vient le `model.joblib` ?
- 04 Que se passera-t-il après le **déploiement** ?

Le changement de perspective



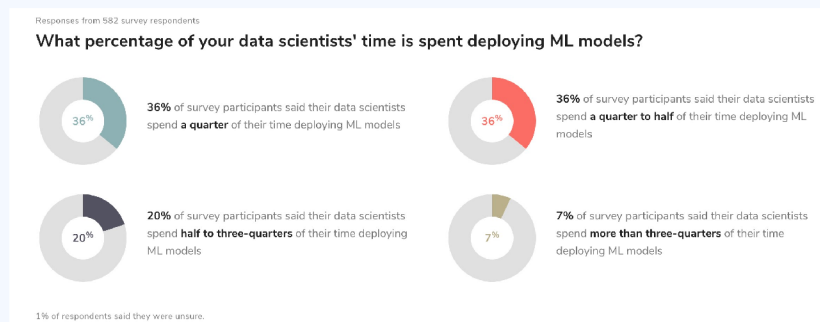
- **Le modèle** répond à une question : « *les prédictions sont-elles suffisamment bonnes ?* »
- **Le système ML** ajoute ce qui manque pour l'utiliser : données et code traçables, pipeline reproductible, service déployable et comportement observable.
- La question devient alors : peut-on **reproduire, déployer, surveiller** et **faire évoluer** ces prédictions avec confiance ?

Le changement de perspective



- Le modèle n'est qu'un composant au centre d'une chaîne plus large.
- En amont**, le système collecte et vérifie les données, gère le code et la configuration, puis prépare les variables d'entrée.
- En aval**, il expose les prédictions via un service et surveille leur comportement dans le temps.
- Le MLOps organise cette chaîne pour rendre le système **reproductible, fiable et maintenable**.

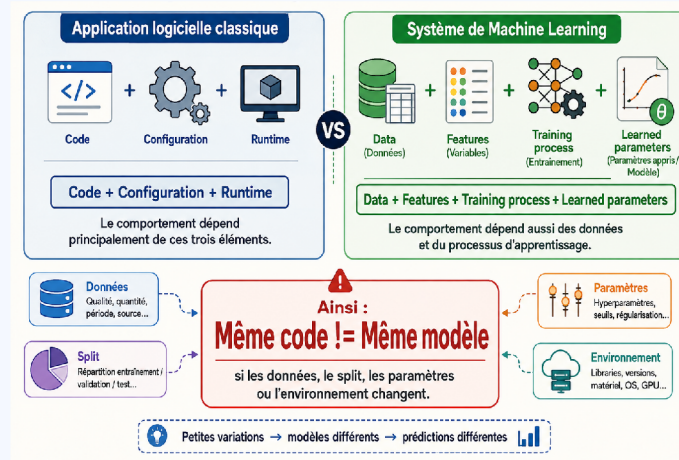
Le constat du terrain



- En production, entraîner le modèle n'est qu'un jalon : l'essentiel du travail se poursuit autour de lui.
- Préparer les données, vérifier la qualité, intégrer le modèle, le déployer et surveiller ses prédictions mobilisent une part majeure de l'effort.
- Sans chaîne outillée, ces opérations restent manuelles, fragiles et difficiles à reproduire.
- Le MLOps transforme ce travail en un processus **traçable, reproductible et industrialisable**.

Pourquoi le ML crée-t-il plus de dépendances ?

- Un logiciel classique dépend principalement de son code et de sa configuration.
- Un système ML dépend aussi des **données**, des variables calculées, du processus d'entraînement et de l'environnement d'exécution.
- Modifier un seul de ces éléments peut changer le modèle produit et ses prédictions.
- Il faut donc relier et versionner les artefacts, de la donnée source jusqu'au service en production.



Dette technique ML : un exemple

- **Situation** : le modèle est réentraîné chaque semaine, avec de nouvelles données et parfois une nouvelle configuration.
- **Après trois mois** : 12 jeux de données, 12 modèles, plusieurs versions du code et plusieurs environnements coexistent.



QUESTION CRITIQUE

Quel modèle est en production, avec quelles données et quel code a-t-il été produit ?

- Sans lien entre ces artefacts, une correction ou un retour arrière devient une enquête manuelle.
- La traçabilité donne à chaque modèle une **identité** et permet de suivre son **cycle de vie**, de l'entraînement à la production.

Une définition opérationnelle

DÉFINITION Le **MLOps** désigne les pratiques qui rendent le cycle de vie d'un système ML **fiable, traçable et exploitable**, de la donnée au service en production.

- Il couvre le cycle complet : **préparer, entraîner, valider, déployer, observer** et **faire évoluer**.
- Pour produire des résultats fiables, les pratiques doivent être :
 - **reproductibles** ;
 - **traçables** ;
 - **testables**.
- Pour exploiter le système dans la durée, elles doivent le rendre :
 - **automatisable** ;
 - **déployable** ;
 - **observable** ;
 - **maintenable**.
- Il articule quatre disciplines : *Machine Learning, Data Engineering, Software Engineering* et *DevOps*.

MLOps n'est pas une collection d'outils

VISION CENTRÉE SUR LES OUTILS

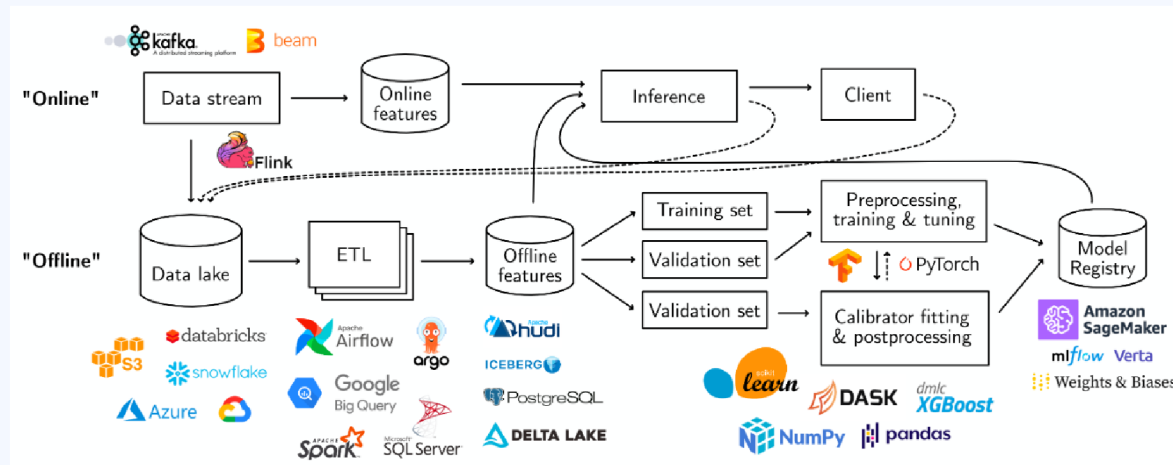
- MLflow
- DVC
- Docker
- GitHub Actions
- Kubernetes
- ...
- Un catalogue répond à la question « **avec quoi ?** », mais pas à « **pour quel besoin ?** ».
- Risque : empiler des outils sans une chaîne cohérente ni un critère de réussite.

VISION D'INGÉNIERIE

- On part du besoin à garantir, puis on construit la capacité nécessaire pour faire vivre le système :
 - Tracer
 - Versionner
 - Valider
 - Servir
 - Automatiser
 - Observer
 - Décider

PRINCIPE **Besoin → capacité → preuve → outil adapté**

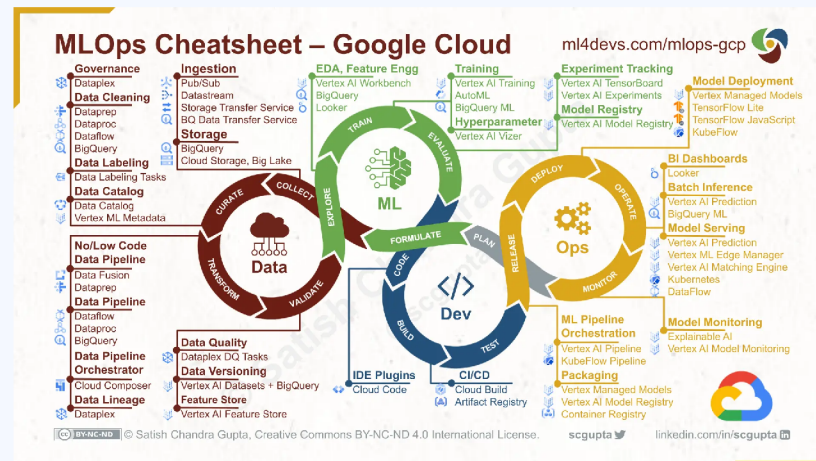
MLOps n'est pas une collection d'outils



Architecture générale d'un pipeline générique de machine learning de bout en bout. Les logos représentent un échantillon d'outils utilisés pour construire les composants du pipeline, illustrant l'hétérogénéité de la pile technologique. Shankar et al., 2021.

MLOps n'est pas une collection d'outils

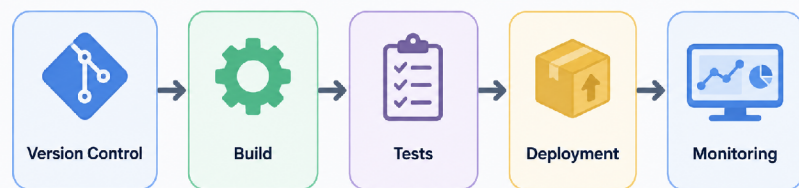
- Une plateforme cloud propose des services répartis entre les domaines **Data**, **ML**, **Dev** et **Ops**.
- Chaque service répond à une capacité particulière : collecter, entraîner, déployer, observer ou automatiser.
- Ce schéma GCP est un **exemple d'écosystème**, pas une recette à reproduire ni une définition du MLOps.
- La démarche consiste à partir du besoin, puis à choisir une pile cohérente avec les contraintes du projet.



MLOps vs DevOps

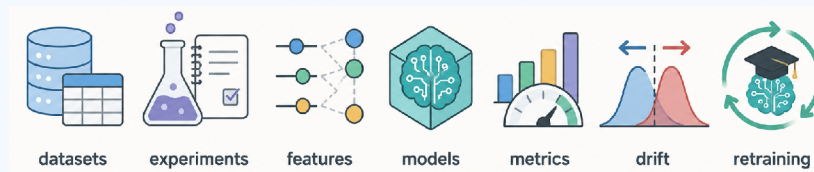
CE QUE DEVOPS INDUSTRIALISE

- Le cycle du **code** : développer, tester, intégrer, livrer et opérer une application.



CE QUE LE ML AJOUTE

- Des objets qui évoluent avec le temps : **données**, variables, labels, modèles, métriques et prédictions.



CONCLUSION MLOps étend DevOps au cycle de vie des données et des modèles.

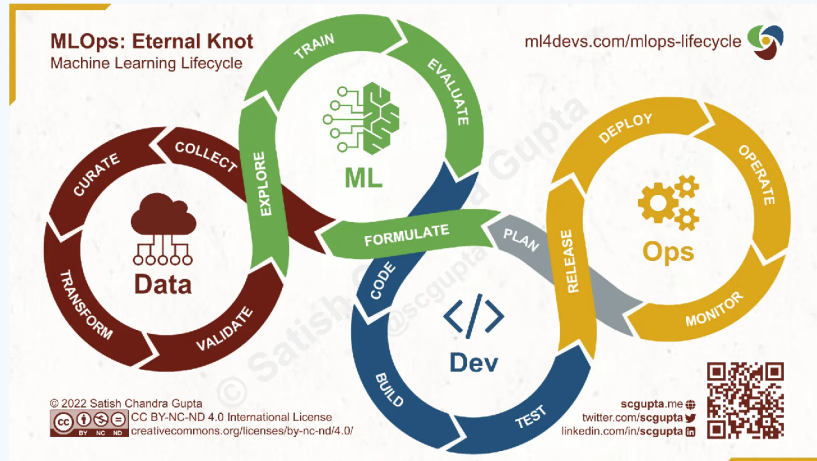
MLOps vs DevOps

Git + CI/CD sécurisent le code et l'exécution du pipeline, mais ne suffisent pas à reproduire un modèle.

- 01 **Git versionne le code, pas nécessairement les données** Un même code exécuté sur un autre dataset peut produire un modèle différent.
- 02 **La CI/CD rejoue les étapes, mais pas forcément le même contexte expérimental** Seed, partitionnement, hyperparamètres, dépendances et environnement doivent aussi être maîtrisés.
- 03 **Le modèle doit rester traçable après l'entraînement** Son identité doit relier données, code, configuration, paramètres, métriques et environnement.

À RETENIR **Reproduire un modèle exige de versionner et de tracer plus que le code.**

Le cycle vu comme des boucles



- Le cycle ML n'est pas une chaîne linéaire : plusieurs boucles avancent en parallèle et se nourrissent mutuellement.
- Data** collecte, transforme et valide les données ; **ML** explore, entraîne et évalue les modèles.
- Dev** construit et teste le logiciel ; **Ops** déploie, opère et surveille le service.
- Le monitoring crée le retour d'information qui permet de corriger, réentraîner ou faire évoluer le système.

Du cycle aux stages

LECTURE Le cycle précédent montre les boucles qui coopèrent. Les quatre stages suivants proposent une lecture opérationnelle de ce même cycle, de la donnée jusqu'au feedback.

Stage 1 — Data & Experimentation

FLUX DONNÉES

- Définir la cible, la fenêtre d'observation et les sources autorisées.
- Contrôler la qualité, préparer les variables et séparer les jeux sans fuite de données.
- Versionner le snapshot réellement consommé par chaque exécution.

FLUX EXPÉRIMENTATION

- Fixer un protocole pour tester variables, algorithmes et hyperparamètres.
- Comparer les exécutions avec les mêmes données, les mêmes métriques et des conditions maîtrisées.
- Enregistrer le code, la configuration, les métriques et le modèle produit.

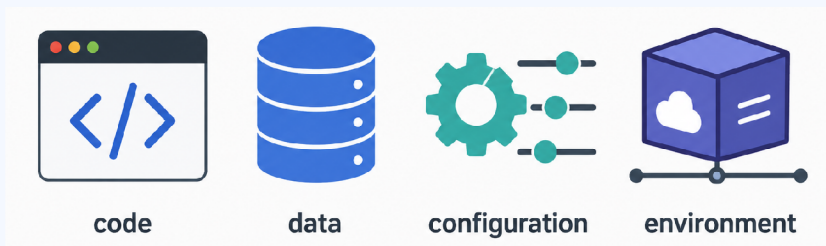
PREUVE ATTENDUE

Un modèle candidat accompagné d'une expérience traçable et comparable.

Avant de retenir le meilleur score, répondre à quatre questions :

- | | |
|--|--------------------------------------|
| • Quelles données ont été utilisées ? | • Quels paramètres ont été retenus ? |
| • Quelles métriques ont été obtenues ? | • Quel modèle en résulte ? |

Stage 2 — Training



PRODUIRE UN MODÈLE CANDIDAT

- Le pipeline de **Training** assemble les données préparées, le code et la configuration.
- Chaque **run** conserve son identité : données, paramètres, code, environnement et artefacts.
- Le pipeline produit un modèle candidat et les preuves nécessaires pour rejouer l'exécution.

CONCEPT CLÉ

Un *run* est une exécution identifiable du pipeline de Training.

Il relie le modèle produit à ses données, son code, sa configuration, ses métriques et son environnement.

Stage 2 — Evaluation

MESURER LE MODÈLE

- **Performance prédictive** : évaluer le modèle sur un jeu de test indépendant.
- **Performance opérationnelle** : mesurer la latence, le coût et la consommation de ressources.
- **Robustesse et métier** : vérifier la stabilité selon les segments et les objectifs importants.

COMPARER LES RUNS

- Utiliser un protocole fixé à l'avance : mêmes splits, métriques et règles de calcul.
- Comparer les **runs** à une référence et à des critères explicites, par exemple :
 - `F1 >= seuil` ;
 - `latence <= seuil` ;
 - tests critiques = `PASS` .
- Conserver les résultats et les écarts pour justifier le choix du candidat.

CONCEPT CLÉ

L'*Evaluation* mesure la qualité d'un modèle dans des conditions contrôlées.

Un score n'est comparable que si les données, le protocole et les métriques sont identiques.

Stage 2 — Validation

VÉRIFIER AVANT PROMOTION

- Contrôler les métriques, les artefacts et les tests du pipeline.
- Vérifier la qualité des données et le respect des contraintes de sécurité et de performance.
- Relier chaque résultat à l'identité du **run** et conserver les preuves.

PRÉPARER LA DÉCISION

- La **Validation** transforme les résultats de l'**Evaluation** en une décision explicite.
- Les critères doivent être définis avant la promotion, lisibles et automatisables.
- Un modèle n'est pas promu uniquement parce qu'il obtient le meilleur score.
- La décision doit être reproductible et auditable.

CONCEPT CLÉ

Le *Quality Gate* est une règle de décision qui autorise ou bloque la promotion.

ACCEPT permet de poursuivre le cycle ; **REJECT** conserve les preuves et arrête la promotion.

Stage 3 — Registry & Deployment

REGISTRY – IDENTIFIER ET PROMOUVOIR

- Le **Model Registry** conserve des versions identifiables et immuables des modèles produits.
- Chaque version référence son **run**, son snapshot de données, son code, sa configuration et ses métriques.
- Les statuts et alias indiquent son rôle : `candidate`, `staging`, `champion`.
- Une promotion doit être explicite, justifiée par les preuves du **Quality Gate** et historisée.

DEPLOYMENT – SERVIR LA BONNE VERSION

- Choisir un mode adapté au besoin et au SLA : **batch**, API online, stream ou edge.
- Déployer le modèle avec son preprocessing, ses dépendances et son contrat d'inférence.
- Vérifier la version réellement servie au moyen de tests de santé et de **smoke tests**.
- Rendre la promotion réversible pour permettre un **rollback** vers une version connue.

CONCEPT CLÉ

Le Registry gère l'identité et la promotion ; le Deployment rend la version approuvée disponible.

Chaîne attendue : modèle validé → version enregistrée → alias promu → modèle servi et vérifié.

Stage 4 — Inference

SERVIR UNE PRÉDICTION

- Servir les prédictions selon le contrat d'inférence : schéma, format et délai attendus.
- Charger la version approuvée du modèle avec son preprocessing et ses dépendances.
- Gérer les entrées invalides, les erreurs et les indisponibilités avec un comportement défini.

COLLECTER LES SIGNAUX UTILES

- Journaliser la version du modèle, la latence, les erreurs, le volume et les prédictions produites.
- Respecter la confidentialité : ne collecter que les données nécessaires au suivi.
- Préparer les informations qui permettront de relier une anomalie à une version et à un **run**.

CONCEPT CLÉ

Le contrat d'inférence décrit ce que le service accepte et garantit.

Il précise le schéma d'entrée, le format de sortie, les erreurs attendues et le SLA.

Stage 4 — Monitoring

OBSERVER LE SERVICE ET LES DONNÉES

- **Service** : disponibilité, erreurs, latence, coût et volume des requêtes.
- **Données et sorties** : distributions d'entrée et de prédiction, valeurs manquantes et anomalies.
- Comparer les signaux à une référence et suivre leur évolution dans le temps.

ATTENDRE LES LABELS

- La performance réelle ne peut être mesurée qu'une fois les labels disponibles.
- Suivre la qualité globale et les écarts entre segments importants.
- Distinguer un **data drift** d'une baisse démontrée de performance.

CONCEPT CLÉ

Le monitoring transforme les observations en signaux exploitables.

Un signal de *drift* indique un changement ; il ne prouve pas à lui seul que le modèle est moins performant.

Stage 4 — Feedback & Action

DIAGNOSTIQUER AVANT D'AGIR

- Relier le signal au modèle servi, aux données, au trafic et au contexte métier.
- Rechercher la cause : changement de population, défaut de données, incident de service ou évolution du besoin.
- Documenter le diagnostic et les preuves associées.

CHOISIR UNE ACTION

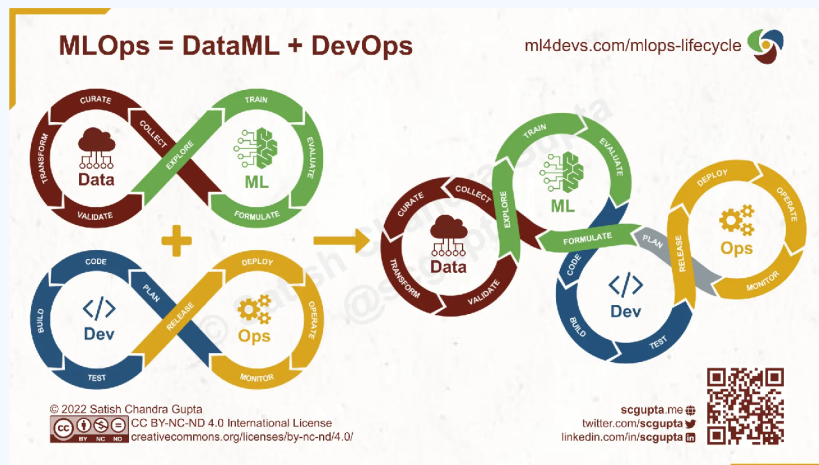
- Continuer à observer, enquêter, corriger les données ou ajuster le service.
- Réentraîner seulement si les preuves et les critères le justifient.
- Autoriser l'action, conserver la décision et vérifier son effet après intervention.

FEEDBACK LOOP

Le monitoring produit un signal ; il ne décide pas seul d'un réentraînement.

Chaîne attendue : signal → diagnostic → décision autorisée → action → nouvelle observation.

Un seul cycle, deux développements



DEUX BOUCLES À SYNCHRONISER

- **Data-ML** : données, features, entraînement, évaluation et modèle.
- **Dev-Ops** : code, tests, déploiement, infrastructure et exploitation.
- Le **MLOps** relie ces boucles pour qu'une version du modèle puisse être livrée, observée et remplacée de façon maîtrisée.

IDÉE CLÉ Un modèle performant ne devient un service fiable que si les deux boucles restent synchronisées.

- Un système ML fait évoluer deux objets à la fois : le **modèle** et le **service** qui le met à disposition.

Maturité MLOps : une progression par capacités

- La maturité répond à une question : **que pouvons-nous garantir à chaque nouveau modèle ?**
- Elle se mesure par le niveau de contrôle du **lifecycle ML**, pas par le nombre d'outils installés.
- Chaque palier remplace une opération fragile ou manuelle par une capacité vérifiable.

UNE PROGRESSION INDICATIVE

- **Niveau 0 — Manuel** : chacun reproduit difficilement le résultat.
- **Niveau 1 — Reproductible** : les données, le code et l'environnement sont identifiés.
- **Niveau 2 — Automatisé** : le Training, les tests et la Validation sont rejouables.
- **Niveau 3 — Déployable** : la promotion et le Deployment sont contrôlés.
- **Niveau 4 — Opérationnel** : le service est observé et la Feedback loop est maîtrisée.

À RETENIR

La maturité MLOps décrit ce que l'équipe sait garantir de façon répétable.

Les capacités — reproduire, valider, promouvoir, servir et observer — se construisent progressivement.

Maturité MLOps : comparer les référentiels

- Les fournisseurs proposent des **modèles de maturité** pour structurer cette progression, mais aucun ne constitue une norme universelle.

RÉFÉRENTIEL	NIVEAUX	ACCENT PRINCIPAL
Google Cloud	3	Automatiser le pipeline ML puis le CI/CD du système ML
AWS	3	Automatiser le pipeline ML et le continuous training
Microsoft Azure	5	Étendre progressivement DevOps au Training, au Deployment et aux opérations

LECTURE

Les découpages diffèrent, mais la direction est commune.

Utiliser un référentiel pour situer les capacités actuelles, choisir le prochain risque à réduire et définir une preuve de progrès.

Plusieurs pipelines coexistent en MLOps

Les capacités MLOps sont portées par plusieurs pipelines spécialisés. Chacun automatise une partie du **lifecycle ML** et transmet des artefacts contrôlés au suivant.

CONSTRUIRE ET VALIDER

- **Data pipeline** : produire des données fiables et versionnées.
- **Training pipeline** : produire un modèle candidat évalué.
- **CI pipeline** : vérifier le code, les tests et les critères qualité.

LIVRER ET OPÉRER

- **Deployment pipeline** : promouvoir et livrer un artefact approuvé.
- **Inference pipeline** : transformer une observation en prédiction.
- **Monitoring / Feedback pipeline** : observer la production et guider l'action.

CONTRAT COMMUN

Chaque pipeline possède un déclencheur, des entrées, des sorties et des critères de réussite.

Une sortie validée devient l'entrée contrôlée d'une autre étape du cycle de vie du modèle.

Diagnostic des pipelines

Pour chaque situation, identifiez le pipeline principalement concerné, puis vérifiez votre réponse avec la justification.

EXERCICE – SITUATION	RÉPONSE	POURQUOI ?
La source contient soudainement des valeurs manquantes et un schéma différent.	Data	Le pipeline contrôle et prépare les données.
Un nouveau dataset validé déclenche un entraînement.	Training	Il produit un nouveau modèle candidat.
Une dépendance Python incompatible apparaît après un commit.	CI	La CI vérifie le code, l'environnement et les tests.
Une version approuvée doit être mise en production.	Deployment	Le pipeline promeut et livre l'artefact.
Une requête contient une catégorie inconnue.	Inference	Le service traite l'entrée et applique le contrat d'inférence.
Un changement statistique est observé en production.	Monitoring / Feedback	Le signal doit être observé, diagnostiqué puis traité.

CHAPITRE

Comprendre le MLOps : du modèle au système reproductible

03 Démo —

Démonstration live — le protocole

Le kit enseignant est disponible dans `code/seance1/`.

1 • OBSERVER

Notebook + dataset

`accuracy`, `F1`, `model.joblib`

2 • REPRODUIRE

Seed + configuration

Même entrée → même sortie

3 • TRACER

Hash + manifeste + MLflow

Un run identifiable

Faire exécuter les cellules par la classe avant d'expliquer la solution.

Feuille de route — code projeté

ÉTAPE	SUPPORT	PREUVE AJOUTÉE
1 · Observer	<code>code/seance1/steps/01-observer/train.py</code>	scores qui varient
2 · Configurer	<code>code/seance1/steps/02-configuration/</code>	seed, YAML et pipeline
3 · Tracer	<code>code/seance1/finish/src/train.py</code>	manifeste, hash et MLflow

Le passage d'un fichier au suivant doit être commenté : chaque étape ajoute une preuve et répond à une question précise. Les fragments sont dans `code/seance1/steps/`.

Étape 1 — Observer : obtenir un premier résultat

```
cd code/seance1/starter
python -m pip install -r requirements.txt
jupyter notebook starter_notebook.ipynb
```

OBJECTIF

- Exécuter un premier **Training** de bout en bout et identifier les artefacts produits.

LE NOTEBOOK RÉALISE

- charge `customers.csv` ;
- sépare les données en **train** et **test** ;
- entraîne une **Random Forest** ;
- affiche `accuracy` et `F1` ;
- sauvegarde le modèle dans `model.joblib`



QUESTION DE TRAÇABILITÉ

Après fermeture du notebook, peut-on encore relier `model.joblib` aux données, au split et aux paramètres qui l'ont produit ?

Le résultat est visible ; sa provenance ne l'est pas encore.

Étape 1 — Rejouer : observer l'écart

CONSIGNE

- Relancer le même notebook trois fois, sans modifier le code ni les données.
- Noter **accuracy** et **F1** pour chaque **run**.

EXTRAIT OBSERVÉ

```
train_test_split(X, y, test_size=0.20)
RandomForestClassifier(n_estimators=100)
```

EXÉCUTION	ACCURACY	F1
A	0,75	0,53
B	0,72	0,50
C	0,67	0,38

QUESTION

Qu'est-ce qui peut expliquer cette variation ?

Les valeurs sont illustratives : l'observation importante est qu'un même code peut produire des résultats différents.

Étape 1 — Diagnostiquer : tester les hypothèses

AVANT DE CONCLURE

HYPOTHÈSE	PREUVE À VÉRIFIER
Données différentes	fichier, taille, date ou hash
Split différent	indices du jeu <code>test</code> et <code>random_state</code>
Modèle différent	<code>random_state</code> et hyperparamètres
Code différent	preprocessing et version du commit

CONCLUSION On observe d'abord, on formule ensuite une hypothèse, puis on cherche la preuve.

La suite de la démonstration ajoutera progressivement ces garanties : configuration, seed, manifeste et tracking.

CE QUE RÉVÈLE LE STARTER

Le starter ne fixe pas les deux générateurs aléatoires :

```
train_test_split(X, y, test_size=0.20)
RandomForestClassifier(n_estimators=100)
```

- le **split** peut changer ;
 - les arbres de la **Random Forest** peuvent changer.
- La variation observée est donc une hypothèse à vérifier, pas encore une preuve complète.

- Une métrique seule décrit un résultat ; elle ne décrit pas l'expérience qui l'a produit.

Étape 2 — Stabiliser : rendre le protocole déterministe

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, random_state=42, stratify=y
)
model = RandomForestClassifier(
    n_estimators=100, max_depth=8, random_state=42
)
```

DÉCISIONS RENDUES EXPLICITES

- `random_state=42` fixe le split et l'aléa de la **Random Forest**.
- `stratify=y` conserve la proportion de la classe `churn` dans les jeux.
- `test_size=0.20` définit la règle de séparation train/test.
- Ces choix rendent les **runs** comparables dans le même environnement.

RÉSULTAT ATTENDU

Avec la configuration de référence, le résultat devient stable :

```
accuracy = 0.760
f1       = 0.538
```

Le seed stabilise l'exécution ; il ne versionne ni les données, ni le code, ni l'environnement.

Étape 2 — Configurer : externaliser les décisions

```
# configs/train.yaml
seed: 42
test_size: 0.20
model:
  type: random_forest
  n_estimators: 100
  max_depth: 8
```

POURQUOI UNE CONFIGURATION SÉPARÉE?

- Elle rend les décisions lisibles pendant la revue.
- Elle permet de comparer deux runs sans fouiller le notebook.
- Elle est versionnable avec le code et réutilisable par le pipeline.
- Elle évite de dupliquer les mêmes choix dans plusieurs scripts.

POINT TECHNIQUE

La configuration devient une preuve réutilisable de l'expérience.

Dans `code/seance1/steps/02-configuration/train.py`, le `Pipeline` ajuste le preprocessing pendant `fit`, après le split : les transformations apprennent uniquement sur le jeu `train`.

Étape 3 — Construire l'identité du run

DE L'EXÉCUTION À LA PREUVE

IDENTIFIER L'EXPÉRIENCE

- Le dataset est identifié par une empreinte **SHA-256** de son contenu :

SHA-256(customers.csv) = <64 caractères hexadécimaux>

- Deux fichiers nommés `customers.csv` peuvent donc être différents.
- Le **Run ID** identifie l'exécution qui a produit les métriques et les artefacts.

CONCEPT CLÉ

Un fichier et un score ne suffisent pas à identifier une expérience.

L'empreinte du dataset et le **Run ID** relie le résultat à son exécution.

Étape 3 — Produire les preuves locales

```
{
  "run_id": "...",
  "data_sha256": "...",
  "seed": 42,
  "test_size": 0.20,
  "metrics": {"accuracy": 0.76, "f1": 0.538},
  "model_artifact": "artifacts/model.joblib"
}
```

VÉRIFIER LE MANIFESTE

- **Entrées** : dataset, configuration et code.
- **Décisions** : seed, split et hyperparamètres.
- **Sorties** : métriques, modèle et autres artefacts.
- Le manifeste est lisible par une personne et exploitable par un outil.

CE QUE L'ON VÉRIFIE

- le fichier JSON est créé après le **Training** ;
- les valeurs correspondent au run exécuté ;
- `model_artifact` pointe vers un artefact réellement présent.

CONCEPT CLÉ Le manifeste est la preuve locale du run.

Il relie entrées, décisions et sorties dans un fichier transmissible, vérifiable et archivable.

Étape 3 — Centraliser avec MLflow

LANCER LE TRACKING SERVER

```
# Terminal 1 – depuis la racine du kit
cd code/seance1/finish
make mlflow-server
```

JOURNALISER LES PREUVES

```
# Terminal 2
make train

with mlflow.start_run() as run:
    mlflow.log_params(params)
    mlflow.log_metrics(metrics)
    mlflow.log_artifacts("artifacts",
artifact_path="outputs")
```

VÉRIFICATION

Ouvrir <http://127.0.0.1:5000> et retrouver le run par son `run_id`.

Le tracking centralise les paramètres, métriques et artefacts pour les comparer après l'exécution.

Le résultat de la démonstration

AVANT – RÉSULTAT ISOLÉ

Provenance implicite : le fichier existe, mais ses données, paramètres et conditions d'exécution ne sont pas reliés.

APRÈS – EXPÉRIENCE TRAÇABLE

Provenance consultable : chaque run peut être retrouvé, comparé et expliqué.

À RETENIR

Le seed stabilise une exécution ; la traçabilité identifie l'expérience complète.

La reproductibilité ne consiste donc pas seulement à obtenir deux fois le même score.

Experiment tracking : une pièce de la reproductibilité

CE QUE FOURNIT MLFLOW

- Un **Run ID** pour retrouver une exécution.
- Les paramètres, métriques et artefacts associés.
- Une comparaison entre plusieurs runs.
- Un point d'accès partagé pour consulter les résultats.

CE QU'IL FAUT ENCORE GARANTIR

- les données et leur version exacte ;
- le preprocessing et l'absence de fuite de données ;
- le code et l'environnement d'exécution ;
- le protocole d'évaluation et la validité statistique ;
- l'orchestration et les dépendances du pipeline.

À RETENIR

Experiment Tracking enregistre les preuves d'un run ; la reproductibilité MLOps permet de reconstruire le protocole qui l'a produit.

CHAPITRE

Comprendre le MLOps : du modèle au système reproductible

04 Synthèse

1. Les idées essentielles

PENSER EN SYSTÈME

- **Un modèle n'est qu'un composant.** Le service inclut aussi les données, le code de preprocessing, l'API de serving et le monitoring.
- **Le lifecycle est continu.** Data, training, validation, deployment et monitoring forment une chaîne avec des retours en arrière.
- **Les pipelines sont complémentaires.** Training produit une version ; validation la contrôle ; deployment et monitoring l'opèrent.

PENSER EN PREUVES

- **Un score ne décrit pas l'expérience.** Il manque le dataset, le split, le protocole et les conditions d'exécution.
- **Le seed ne suffit pas.** Il contrôle une source d'aléa, mais ne fixe ni la version des données ni l'environnement.
- **La provenance relie les preuves.** Elle associe données, code, configuration, run, métriques et artefacts.

IDÉE DIRECTRICE

Le MLOps transforme une expérimentation ponctuelle en système ML reproductible, traçable et opérable.

2. Le cycle MLOps en une vue

DE LA DONNÉE AU MODÈLE

- 01 **Data** : acquérir, contrôler, préparer et versionner les données réellement consommées.
- 02 **Experimentation** : tester features, modèles et hyperparamètres dans des conditions comparables.
- 03 **Training** : produire un modèle, ses métriques et les artefacts du run.
- 04 **Validation** : vérifier qualité, performances et contraintes avant le [Quality Gate](#).

DU MODÈLE AU SERVICE

- 01 **Deployment** : promouvoir une version validée et la servir avec un contrat d'inférence.
 - 02 **Monitoring** : observer latence, erreurs, drift et performance après mise en production.
 - 03 **Feedback** : investiguer, corriger ou réentraîner selon des signaux et une décision autorisée.
- À chaque étape, les entrées, les sorties et la décision doivent rester traçables ; le déploiement ne clôt donc pas le cycle.

LECTURE DU CYCLE

Le cycle ne s'arrête pas au déploiement : la production produit un retour vers les données et l'expérimentation.

3. La règle d'or : pouvoir expliquer chaque décision

POUR CHAQUE RUN

- Identifier le dataset par sa version ou son empreinte, ainsi que le code et l'environnement.
- Conserver configuration, seed, split et preprocessing appliqué après le découpage des données.
- Enregistrer métriques, artefacts, manifeste et `Run ID` dans un emplacement consultable.

POUR CHAQUE DÉCISION

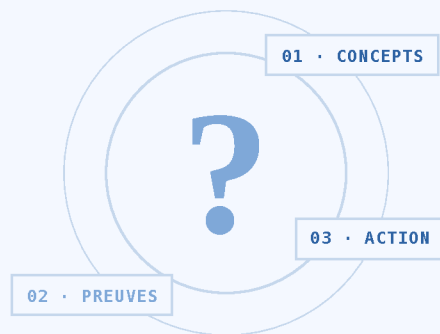
- Comparer les runs selon un protocole et des métriques définis à l'avance.
- Accepter, rejeter ou promouvoir avec un `Quality Gate` et une justification.
- Pouvoir rejouer, auditer et revenir à une version antérieure sans reconstruire l'historique à la main.

OUVERTURE – SÉANCE 2

Comment garantir l'intégrité des données et des dépendances lorsque le pipeline évolue ?

Versionnement des données · intégrité expérimentale · pipeline reproductible.

MLOPS ET INDUSTRIALISATION DES MODÈLES IA



Questions et discussion

Qu'est-ce qui mérite d'être vérifié, discuté
ou mis en pratique ?

POUR PROLONGER

Pr. Youness BOUTYOUR youness.boutyour@um5.ac.ma