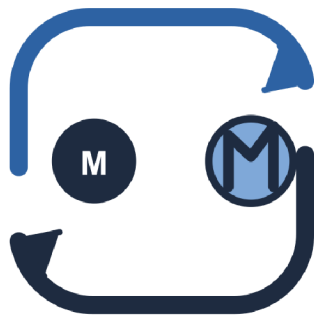


MLOps et Industrialisation des modèles IA

TP 4

Superviser, détecter la dérive et décider

Du modèle au système ML traçable, validé et exploitable

**MLOPS**
GL-M326

TP · SOMMAIRE

Table des matières

Mission et question directrice	3
Compétences et ressources	3
Contrat de progression TP3 → TP4	3
Consignes de travail	4
Installer les outils et diagnostiquer l'environnement	4
Préparer le workspace	5
Architecture cible	6
Parcours guidé et preuves	6
1. Charger le champion	6
2. Journaliser les inférences	6
3. Définir les vues et la référence	6
4. Comparer week 1 et week 2	7
4 bis. Mesurer l'état opérationnel du service	7
5. Révéler les labels en différé	8
6. Décider et documenter	8
Questions techniques et ouvertes	8
Compte rendu obligatoire et preuves visuelles	9
Minimum requis	10
Observabilité de référence et innovation	10
Livrable et réussite	11

Modalité : travail individuel. Chaque étudiant analyse les signaux, justifie ses décisions et remet son propre compte rendu avec des preuves issues de sa machine.

Mission et question directrice

Le modèle champion sert des prédictions, mais les capteurs de la deuxième semaine ne ressemblent plus à ceux de l'entraînement. Une alerte de drift apparaît. L'équipe doit éviter deux erreurs : ignorer un signal utile ou réentraîner automatiquement sans preuve de baisse de performance.

Comment transformer un signal de production en décision opérationnelle proportionnée, alors que les labels arrivent en retard ?

Compétences et ressources

- charger un modèle par alias Registry ;
- journaliser les prédictions et les erreurs ;
- distinguer infrastructure, service, données, modèle et métier ;
- mesurer le drift sans utiliser prématurément les labels ;
- évaluer la performance lorsque les labels arrivent ;
- choisir entre surveillance, investigation, collecte, retraining et rollback ;
- rédiger un runbook et un postmortem.

Le kit `student/` consomme le service et le champion produits par le TP3. Il fournit `week_1.csv`, `week_2.csv`, les configurations de monitoring et les fonctions d'observation. Le Registry et les aliases restent sous la responsabilité du TP3.

Contrat de progression TP3 → TP4

Le TP4 consomme l'identité du champion, le contrat HTTP et le journal de prédictions du TP3. Il ne se limite pas au drift : la décision doit croiser état du service, qualité des données, performance différée et contexte métier. Les sorties transmissibles sont `drift_report_week_1.json`, `drift_report_week_2.json`, `operational_metrics.json`, `performance_week_2.json`, `action.json`, `tp4_handoff.json`, le runbook et le postmortem.

Le `release_manifest.json` du TP3 est la source de vérité de la version effectivement portée par `champion`. Il est vérifié avec les décisions de promotion/rollback avant toute analyse. Le handoff produit doit ensuite porter `schema_version: tp3.handoff.v1`, `quality_gate:`

ACCEPT, alias `champion`, version effective, `run_id`, `data_sha256` et métriques cohérents. Le TP4 produit à son tour un handoff `tp4.handoff.v1` pour le TP5, avec l'identité du champion, l'action, les preuves disponibles/manquantes et le SHA-256 des quatre rapports de monitoring.

Consignes de travail

Travaillez dans une copie `TP4_NOM_PRENOM` et gardez séparés les résultats de `week_1` et `week_2`. Respectez la séquence suivante : observer, mesurer, qualifier, attendre les labels, décider. Ne concluez pas à une baisse de performance uniquement à partir d'un drift de distribution.

Chaque décision doit distinguer quatre éléments : le signal observé, la preuve disponible, la preuve manquante et l'action proportionnée. Toute commande qui modifie un rapport ou un fichier de décision doit être reproduite dans le compte rendu.

Les journaux `reports/monitoring/http_events.jsonl` et `reports/monitoring/predictions.jsonl` sont des preuves obligatoires. Chaque événement HTTP doit contenir un horodatage ISO-8601, un statut, une latence et `input_valid`. Le validateur recalcule le volume, le taux d'erreur et la disponibilité depuis le journal ; un tableau rempli manuellement ne suffit pas.

Installer les outils et diagnostiquer l'environnement

Le TP requiert Python 3.10 ou plus récent, Git, `uv`, `curl` et un serveur MLflow accessible. Docker est recommandé si vous reprenez le service dans un conteneur. Vérifiez :

TERMINAL · BASH

```
python --version
git --version
uv --version
curl --version
```

Si `uv` manque sur Linux ou macOS :

TERMINAL · BASH

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

Les dépendances Python sont isolées et verrouillées par `uv.lock`. Installez Docker Engine/Desktop uniquement pour l'extension de serving conteneurisé.

Préparer le workspace

Téléchargez l'**énoncé PDF** et l'**archive Code de départ (.zip)** depuis la page du TP. Le ZIP contient directement le checkpoint étudiant complet (code, configuration et données), sans sous-dossier `student/`. Extrayez-le dans un nouveau dossier personnel et gardez le PDF accessible.

Linux/macOS, depuis le dossier contenant le ZIP :

TERMINAL · BASH

```
mkdir -p "$HOME/ensias-tps/MLOps/TP4_NOM_PRENOM"
unzip "<nom-exact-du-kit-TP4>.zip" -d "$HOME/ensias-tps/MLOps/TP4_NOM_PRENOM"
cd "$HOME/ensias-tps/MLOps/TP4_NOM_PRENOM"
pwd
ls -la
```

Windows PowerShell :

CODE

```
$work = "$HOME\ensias-tps\MLOps\TP4_NOM_PRENOM"
New-Item -ItemType Directory -Force $work
Expand-Archive -LiteralPath ".\<nom-exact-du-kit-TP4>.zip" -DestinationPath $work
Set-Location $work
Get-ChildItem
```

Vérifiez `pyproject.toml`, `uv.lock`, `Makefile`, `data/` et `README.md`. Si vous avez réalisé l'extension Registry du TP3, vous pouvez plutôt continuer dans une copie de votre workspace TP3 afin de conserver la base MLflow et l'alias `champion` ; ne fusionnez pas les deux versions sans comparer leurs fichiers.

TERMINAL · BASH

```
make install
make report
make test
make doctor
```

`make doctor` vérifie Python, `pytest`, MLflow, Git et `curl`. Pour les tests HTTP, démarrez l'API dans un autre terminal ; une erreur de connexion à `127.0.0.1:8000` signifie généralement que le service n'est pas lancé.

Vérifiez que l'alias `champion` existe. La colonne `machine_failure` des fichiers pédagogiques est considérée comme retardée : elle ne doit pas servir à détecter le drift.

Architecture cible

CODE

```
models:/predictive-maintenance@champion
  → API → validation → prédiction → journal JSONL
  → lots de production
    → drift sans labels → alerte → investigation/collecte
    → labels disponibles → performance → action
```

Parcours guidé et preuves

1. Charger le champion

Dans `maintenance_ml/api/main.py`, utilisez `lifespan` pour charger :

CODE

```
models:/predictive-maintenance@champion
```

`/model-info` doit retourner nom, alias et numéro réel de version. `/health` ne doit annoncer `model_loaded: true` qu'après un chargement réussi.

Preuve : réponses JSON de `/health` et `/model-info`.

2. Journaliser les inférences

Après chaque requête valide, écrivez dans `reports/monitoring/predictions.jsonl` :

- timestamp UTC ;
- version du modèle ;
- features reçues ;
- prédiction et probabilité.

Effectuez trois requêtes valides et une invalide. Vérifiez qu'une entrée HTTP 422 n'est pas envoyée au modèle et ne produit pas de ligne de prédiction.

Consignes : vérifiez le nombre de lignes avant et après les quatre requêtes, contrôlez le format UTC et comparez l'identité du modèle journalisée à `/model-info`. Ne capturez pas de données personnelles dans les figures.

3. Définir les vues et la référence

Dans `configs/monitoring.yaml`, rendez explicites :

- dataset de référence ;
- features numériques et catégorielles ;

- mesure par type de variable ;
- seuil par mesure ;
- nombre minimal de variables en drift ;
- responsable et fréquence de revue.

Expliquez pourquoi `machine_failure` est exclue du drift non supervisé.

Consigne : justifiez chaque seuil choisi et indiquez ce qui déclenche une alerte globale. Une variable isolée en drift ne doit pas être assimilée automatiquement à un incident modèle.

4. Comparer week 1 et week 2

Exécutez ou complétez `maintenance_ml/monitoring/drift.py`. Pour chaque feature, le rapport contient mesure, seuil, valeur observée et booléen `drifted`.

TERMINAL · BASH

```
make monitor
```

Produisez :

- `drift_report_week_1.json` ;
- `drift_report_week_2.json` ;
- `drift_report.json` ;
- `action.json`.

Week 1 est la première observation de production ; week 2 est une observation plus dégradée. Avec les données du starter, week 1 peut déjà déclencher une alerte : comparez alors le nombre de variables dérivantes et l'intensité des scores, sans transformer une alerte de drift en preuve de baisse de performance.

Preuve attendue : un tableau comparatif par feature pour les deux semaines, le seuil appliqué, la décision booléenne et une conclusion séparant contrôle négatif et scénario incident.

4 bis. Mesurer l'état opérationnel du service

Créez `reports/monitoring/http_events.jsonl` à partir des requêtes réellement observées. Chaque ligne doit conserver au minimum `timestamp`, `status_code`, `latency_ms` et `input_valid`. Calculez puis interprétez dans `operational_metrics.json` :

- latence p50 et p95 ;
- taux d'erreurs HTTP (5xx) et disponibilité ;
- volume de requêtes et taille du lot analysé ;
- taux d'entrées invalides (422 ou validation métier) ;
- délai d'arrivée des labels entre l'heure du lot et leur disponibilité.

Utilisez `make operational-metrics`, comparez les résultats aux seuils de `configs/monitoring.yaml` et expliquez toute donnée manquante. Un drift sans latence, volume, erreurs et délai de labels ne permet pas de qualifier l'état réel du service.

5. Révéler les labels en différé

Après avoir figé votre conclusion de drift, simulez l'arrivée des labels :

TERMINAL · BASH

```
make performance-week2  
make decide
```

`performance_week_2.json` doit distinguer date du lot, date de disponibilité des labels, baseline et métriques. Comparez le champion à la baseline sans réécrire la conclusion de l'étape précédente.

Consigne : archivez le rapport de drift avant `make performance-week2`, puis expliquez si les labels confirment, nuancent ou contredisent le signal initial.

6. Décider et documenter

Choisissez une action :

CODE

NO_ACTION | INVESTIGATE | COLLECT_LABELS | RETRAIN | ROLLBACK

`action.json` contient signaux, preuves disponibles, preuves manquantes, responsable, action, justification et condition de réévaluation. TP4 ne lance pas le retraining et ne déplace pas `champion` ; le TP5 transformera cette décision en demande et release contrôlées.

Rédigez ensuite :

- un `runbook.md` pour l'opérateur ;
- une `incident-timeline.md` ;
- un `postmortem.md` ;
- un indicateur mesurable d'amélioration.

Critère de réussite : `action.json`, le runbook et le postmortem racontent le même incident, avec les mêmes dates, métriques, responsable et condition de réévaluation. Une action `RETRAIN` doit préciser les données et le gate qui autoriseront le nouveau modèle.

Questions techniques et ouvertes

Le rapport répond aux neuf questions ci-dessous. Les questions 1 à 5 vérifient la boucle de monitoring ; les questions 6 à 9 vérifient le handoff vers TP3, la séparation avec TP5 et la séparation entre produit métier et observabilité.

1. Pourquoi un drift ne prouve-t-il pas une baisse de performance ?
2. Pourquoi les labels ne doivent-ils pas entrer dans la détection initiale ?
3. Quand faut-il collecter des labels avant de décider ?
4. Donnez un scénario de rollback, de retraining et de conservation du champion.
5. Quelles limites présente un journal JSONL local ?
6. Quels SLO, métriques système et mécanismes de collecte ajouteriez-vous ?
7. Comment éviter l'alert fatigue lorsque plusieurs variables dérivent ?
8. Quels champs du contrat TP3 sont nécessaires pour rattacher un signal à la version réellement servie et à ses données ?
9. Pourquoi Prometheus et Grafana doivent-ils être séparés de l'application métier, et que devra automatiser TP5 ?
10. Quelle décision prendriez-vous si le drift est fort mais la performance différée reste stable ?

Chaque réponse doit s'appuyer sur un rapport, une métrique, une réponse HTTP ou un fichier de décision.

Compte rendu obligatoire et preuves visuelles

Le compte rendu individuel doit contenir :

- les commandes et requêtes exécutées ;
- les réponses `/health` et `/model-info` ;
- le journal JSONL ;
- les rapports de drift et de performance ;
- la décision opérationnelle ;
- les réponses à toutes les questions.

Les preuves visuelles doivent être placées dans `reports/figures/` et doivent :

- montrer le terminal, l'API, le JSON ou le Registry utile ;
- avoir un nom explicite, par exemple `tp4-04-drift-week2.png` ;
- comporter une légende, la commande et une limite d'interprétation ;
- fournir au minimum :
 - installation et `make doctor` ;
 - `/health` ;
 - `/model-info` ;
 - requête valide et requête invalide ;
 - journal JSONL ;
 - comparaison week 1/week 2 ;
 - performance différée ;
 - décision finale.

Les captures fabriquées, téléchargées ou sans contexte ne sont pas recevables.

Le rapport doit distinguer signal observé, preuve manquante, interprétation et action choisie.

Minimum requis

Le travail est recevable si les éléments suivants sont présents :

- champion servi par alias ;
- journal de prédictions ;
- contrôles week 1 et week 2 ;
- rapport de drift sans labels ;
- indicateurs opérationnels p50/p95, erreurs, disponibilité, volume, entrées invalides, lots et délai de labels ;
- performance différée ;
- décision argumentée ;
- runbook ;
- stack Prometheus/Grafana séparée, cible UP et requêtes PromQL documentées.

Checklist avant remise : les labels n'ont pas été utilisés pour l'alerte initiale, les deux semaines sont comparées, la décision est traçable et le rapport distingue clairement observation, interprétation et action.

Observabilité de référence et innovation

La stack de référence est fournie dans `student/observability/`. Elle doit être utilisée pour séparer l'API de serving du dashboard MLOps :

TERMINAL · BASH

```
export GRAFANA_ADMIN_PASSWORD='mot-de-passe-local'
make observability-up
make observability-check
```

Pour un hôte Linux dont le bridge Docker ne joint pas `host.docker.internal`, utilisez l'overlay fourni après avoir lancé l'API sur toutes les interfaces :

TERMINAL · BASH

```
API_HOST=0.0.0.0 make api
make observability-up-host
make observability-check
```

Le mot de passe Grafana doit être injecté par l'environnement ou par une copie locale de `student/observability/.env.example`; aucun mot de passe ne doit être commité dans le starter.

Lorsque Prometheus tourne dans Docker et scrute l'API depuis le réseau de l'hôte, démarrez le service avec `API_HOST=0.0.0.0 make api`. La valeur par défaut reste `127.0.0.1` pour un usage local non exposé.

Documentez la cible `/metrics`, deux requêtes PromQL, un panneau Grafana et la limite de cette observation. Choisissez ensuite une extension :

- autre mesure de drift ;
- OpenTelemetry ou une alerte Prometheus ;
- segmentation par type de machine ;
- mécanisme de feedback ;
- seuils adaptatifs ;
- simulation d'un incident différent.

Présentez le signal observé, la décision prise, la preuve et la limite.

Livrable et réussite

Avant de compresser, vérifiez que l'archive contient :

- code du monitoring ;
- configuration des références et seuils ;
- rapports week 1 et week 2 ;
- performance différée ;
- `action.json` ;
- runbook, chronologie et postmortem ;
- rapport et figures.



Le rapport doit distinguer distribution, performance observée, disponibilité des preuves et action opérationnelle. Une alerte qui conclut automatiquement `RETRAIN` ne satisfait pas le TP.

Le nom exact de l'archive est `MLOps_TP4_Nom_Prenom.zip`.