AIA Bloc_02: stripe

Architecture Machine Learning

1. Feature Store (Feast)

Rôle : Centraliser et gérer les features (variables d'entrée) utilisées pour l'entraînement et l'inférence.

- **Online Store (Redis)** :
    - Stocke les features en temps réel pour les requêtes d'inférence (ex : évaluation instantanée d'une transaction).
    - Permet un accès ultra-rapide (latence faible) pour les prédictions en production.
- **Offline Store (S3 Parquet)** :
    - Stocke les features historiques pour l'entraînement des modèles.
    - Format Parquet optimisé pour les requêtes analytiques (ex : Spark).

Flux :

- Les données brutes (ex : transactions, utilisateurs) sont transformées en features et stockées dans les deux stores.

---

2. Model Registry (MLflow)

Rôle : Gérer le cycle de vie des modèles.

- **Versioning** : Suivi des versions des modèles (ex : v1.0, v1.1).
- **Experiment Tracking** : Enregistrement des métriques (ex : précision, F1) et des hyperparamètres pour chaque entraînement.
- **Metadata** : Documentation des modèles (ex : auteur, date, données utilisées).

Flux :

- Après entraînement, le modèle est enregistré dans MLflow avec ses métriques et artefacts.

---

3. Training Pipeline

Rôle : Préparer les données, entraîner et valider les modèles.

- **Feature Engineering (Spark)** :
    - Transformation des données brutes en features (ex : agrégation, normalisation).
    - Exécution distribuée pour traiter de gros volumes de données.
- **Model Training (Python/SKlearn/XGBoost)** :
    - Utilisation d'algorithmes comme XGBoost pour entraîner des modèles de classification (ex : détection de fraude).
- **Validation & Testing** :
    - Évaluation des performances sur des jeux de test (ex : AUC-ROC, précision/rappel).
    - Validation croisée pour éviter le surapprentissage.

Flux :

- Les données de l'Offline Store sont utilisées pour entraîner le modèle, qui est ensuite validé avant déploiement.

---

4. Serving Layer

Rôle : Déployer les modèles et servir les prédictions.

- **Real-time API (FastAPI)** :
    - Expose le modèle via une API REST pour les prédictions en temps réel (ex : validation d'une transaction).
    - Latence critique : réponse en <100ms.
- **Batch Predictions (Airflow)** :
    - Prédictions en lot pour des analyses périodiques (ex : scoring de risque pour tous les utilisateurs).

- **A/B Testing Framework** :
    - Comparaison de deux versions d'un modèle en production pour mesurer l'impact (ex : taux de conversion).

Flux :

- Le modèle est déployé dans l'API ou Airflow, et les prédictions sont servies aux applications Stripe.

---

5. Monitoring

Rôle : Surveiller la santé du système et des modèles.

- **Model Performance** :
    - Suivi des métriques (ex : précision, F1) pour détecter une dégradation.
- **Data Drift Detection** :
    - Détection des changements dans la distribution des données (ex : nouvelle tendance de fraude).
- **Prediction Latency** :
    - Surveillance des temps de réponse pour garantir une expérience utilisateur fluide.

Flux :

- Alertes en cas de dérive ou de baisse de performance, déclenchant une réentraînement ou une investigation.

Exemple de code pour Features avec Feast

```
from feast import FeatureStore

# Initialiser le Feature Store
fs = FeatureStore(repo_path=".")

# Récupérer des features pour l'entraînement (offline)
training_df = fs.get_historical_features(
    entity_df=entity_df,  # DataFrame avec les entities (ex: user_id)
    features=[
```

```
        "transaction_stats:avg_transaction_amount",
        "transaction_stats:transaction_count",
    ],
).to_df()

# Récupérer des features en temps réel (online)
features = fs.get_online_features(
    entity_rows=[{"user": 123}],  # Dictionnaire avec les entities
    features=[
        "transaction_stats:avg_transaction_amount",
    ],
)
```

Exemple définition centralisée des Features

```
# Définition centralisée dans Feast
user_stats = FeatureView(
    name="user_transaction_stats",
    entities=[user],
    schema=[
        Field(name="avg_amount", dtype=Float34),
        Field(name="count", dtype=Int64),
    ],
    source=FileSource("data/transactions.parquet"),
    # Transformation centralisée
    transform=SqlTransformation("""
        SELECT
            user_id,
            AVG(amount) as avg_amount,
            COUNT(*) as count
        FROM transactions
        GROUP BY user_id
    """)
)
```

Online Store et Offline Store

Critère	Offline Store	Online Store
Usage	Entraînement, analyse batch	Inférence en temps réel

Volume de données	Très grand (To/Po)	Petit (Go/To, seulement les données actives)
Fréquence de mise à jour	Batch (quotidien/hebdomadaire)	Temps réel ou quasi-temps réel
Latence acceptable	Secondes/minutes	Millisecondes
Coût	Optimisé pour le stockage (peu coûteux)	Optimisé pour la vitesse (plus cher)
Exemple de requête	"Donne-moi toutes les features de 2023"	"Donne-moi les features de l'utilisateur 123 NOW"

Principe de l'A/B test ML

Tester deux versions d'un modèle en production sur une fraction du trafic, pour comparer leurs performances selon des métriques métier et techniques, avant un déploiement global.

Exemple : Nouveau modèle de fraude (v2.1)

- 90 % du trafic → modèle actuel (v2.0)
- 10 % → nouveau modèle (v2.1, avec features "nouvel appareil")
- Objectif : réduire les faux positifs sans perdre en détection

```
ab_test = {
    'traffic_split': {'v2.0': 0.9, 'v2.1': 0.1},
    'success_criteria': {
        'false_positive_rate < 1.5%',  # ✅ 1.42%
        'latency_p95 < 300ms'          # ⚠️ 218ms
    }
}
```

Résultat : ✅ Promotion en production → gain estimé à +5,1 M$/an.

Déploiement : rollout progressif (10 % → 100 %) avec surveillance.