

# MLOps, déploiement de Machine Learning en production

Redha Moulla

Paris, 19 - 21 mars 2025

# Plan de la formation

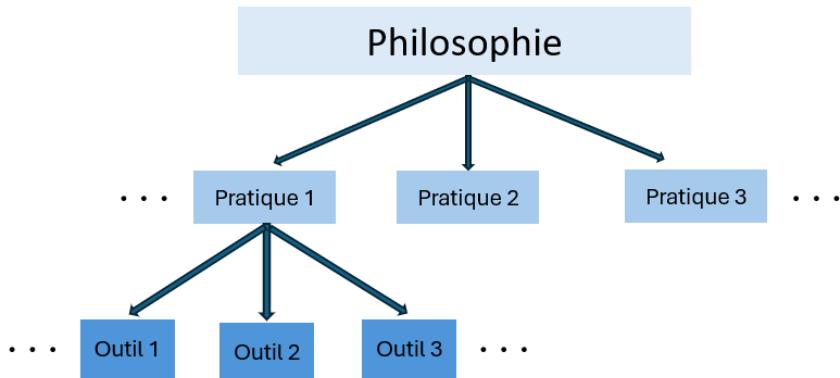
- Principes du MLOps
- APIisation avec Flask
- Virtualisation avec Docker
- CI/CD
- Monitoring
- LLMs

# Qu'est-ce que le MLOps ?

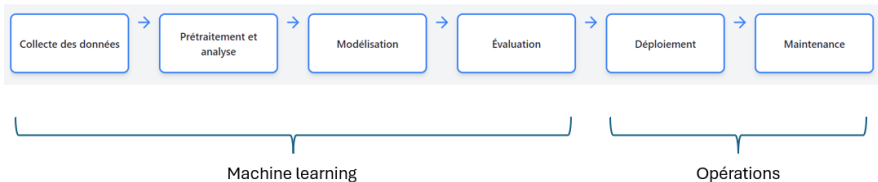
# Définition

*Le MLOps est un ensemble de pratiques et d'outils mis en oeuvre pour automatiser et optimiser le déploiement, la gestion et la maintenance des modèles de machine learning en production.*

# Qu'est-ce que le MLOps



# Cycle de vie d'un projet de machine learning



*Mais ce processus est en réalité très itératif.*

# Pourquoi le MLOps ?

Les différents problèmes que l'on peut rencontrer à l'interface entre la phase machine learning et la phase opérations.

- Problèmes de communication
- Problèmes techniques
- Problèmes de surveillance et de maintenance des modèles
- Problèmes de gestion des données
- Problèmes liés au déploiement et à l'automatisation
- Problèmes de collaboration organisationnelle
- Problèmes liés à la sécurité et à la conformité
- Problèmes de compétences et de formation

# Différence de nature entre les projets Data et Dev

## Projets data

- Nature exploratoire, résultats incertains
- Estimation des délais difficile
- "Produit fini" flou, amélioration continue
- Attentes parfois irréalistes
- Métriques de succès statistiques
- Cycle de vie : mises à jour fréquentes
- Confort avec l'ambiguïté, exploration

## Projets dev

- Nature déterministe, spécifications claires
- Estimations plus précises
- "Produit fini" défini par les spécifications
- Attentes mieux alignées techniquement
- Métriques liées aux fonctionnalités
- Cycle de vie plus traditionnel
- Tendance à réduire l'incertitude

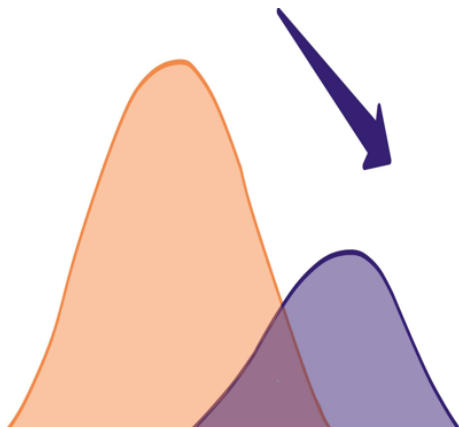


# 0. Problème de data drift

Le data drift est le phénomène de changement dans la distribution des données.

Il apparaît d'une manière fréquente dans les modèles mis en production. Il peut être dû à plusieurs facteurs :

- Différence entre les données de test et les données en production.
- Changement de l'environnement (avant et après le Covid par exemple).
- Changement dans le comportement des utilisateurs ou clients.
- Effets de l'algorithme lui-même sur les utilisateurs.
- Etc.



# 1. Problèmes de communication

- **Terminologie et langage différents :**

Les data scientists et les développeurs parlent souvent des "mêmes" concepts avec des terminologies différentes. Par exemple, un développeur peut parler d'API ou de scalabilité, tandis qu'un data scientist peut parler de modèles prédictifs et de dérive des données. Cette différence peut créer des incompréhensions.

- **Objectifs mal alignés :**

Les priorités des deux équipes peuvent différer, avec les data scientists axés sur la précision des modèles et les développeurs concentrés sur la stabilité et l'optimisation du code en production. Cette divergence d'objectifs peut entraîner des tensions si les équipes ne communiquent pas clairement leurs attentes respectives.

- **Manque de documentation :**

Les data scientists peuvent fournir des modèles ou des scripts sans documentation complète, ce qui rend difficile pour les développeurs de comprendre le contexte ou les paramètres du modèle lors du déploiement en production.

## 2. Problèmes techniques

- **Prototypage vs production :**

Le code écrit dans les notebooks n'est pas directement utilisable en production. Il manque souvent de structure, de gestion des erreurs et de tests nécessaires pour être fiable en environnement de production.

- **Scalabilité et performance :**

Les modèles et pipelines développés par les data scientists ne sont pas toujours adaptés pour une exécution à grande échelle en production. Les problèmes de performance (latence, mémoire) peuvent survenir lors du déploiement à grande échelle.

- **Compatibilité des outils :**

Les outils utilisés en exploration (Pandas, scikit-learn) ne sont pas toujours compatibles avec les environnements de production (Kubernetes, Docker, TensorFlow Serving), nécessitant des ajustements ou réécritures du code.

### 3. Problèmes de gestion des données

- **Qualité et gouvernance des données :**

Les data scientists ont besoin de données propres et de haute qualité, mais si la gouvernance des données n'est pas solide, cela peut entraîner des modèles biaisés ou des erreurs.

- **Accès et gestion des données :**

Il peut y avoir des problèmes de gestion des droits d'accès aux données entre les équipes, ce qui ralentit les projets ou complique les échanges de données entre les équipes data et dev.

- **Versioning des données :**

Les modèles peuvent être entraînés sur une version des données et mis en production avec une autre, ce qui entraîne des incohérences. La gestion des versions des données n'est pas toujours bien synchronisée entre les équipes.

## 4. Problèmes liés au déploiement et à l'automatisation

- **Déploiement manuel :**

Les modèles sont souvent développés dans des environnements isolés et il peut être difficile de les déployer manuellement en production sans automatisation.

- **MLOps immature :**

Si les pratiques MLOps (intégration et déploiement continus pour les modèles) ne sont pas mises en place, il peut être difficile de maintenir les modèles et de les réentraîner régulièrement en production.

- **Maintenance des modèles :**

Une fois en production, les modèles doivent être surveillés et mis à jour. Sans une automatisation du monitoring et de la reformation, les modèles peuvent devenir obsolètes et imprécis.

## 5. Problèmes de collaboration organisationnelle

- **Silos organisationnels :**

Les équipes data et dev travaillent souvent dans des silos, ce qui entraîne une mauvaise communication et une collaboration limitée sur les projets communs.

- **Problèmes de priorisation :**

Les équipes data peuvent prioriser des améliorations de modèles, tandis que les équipes dev se concentrent sur la stabilité du système, entraînant des désaccords sur la priorisation des projets.

- **Propriété et responsabilité :**

Il peut être difficile de définir qui est responsable de la performance continue des modèles en production. Les développeurs s'occupent souvent de l'infrastructure, mais les data scientists sont responsables de la précision des modèles.

## 6. Problèmes liés à la sécurité et à la conformité

- **Protection des données :**

Les data scientists et les développeurs doivent gérer des données sensibles, ce qui impose des contraintes de sécurité pour éviter les violations de la confidentialité (ex : RGPD).

- **Gestion des accès :**

Il est important de s'assurer que seuls les membres autorisés peuvent accéder aux données critiques. Une mauvaise gestion des accès peut entraîner des fuites de données ou des violations de conformité.

# Les différentes organisations des équipes Data et Dev

Dans le développement et le déploiement des projets de Machine Learning, plusieurs types d'organisation sont possibles pour maximiser la collaboration et l'efficacité entre les équipes de data science, de développement et d'ingénierie.

- ➊ **Approche en silos (traditionnelle)** : Les équipes travaillent de manière indépendante, avec transfert de responsabilités des data scientists aux devs.
- ➋ **Approche collaborative (cross-fonctionnelle)** : Les équipes collaborent dès le début du projet et tout au long du cycle de vie du modèle.
- ➌ **Approche DevOps/MLOps** : Automatisation des processus d'entraînement, de déploiement et de surveillance des modèles à travers des pipelines CI/CD.
- ➍ **Équipe centralisée (Center of Excellence)** : Une équipe dédiée centralise les compétences et les processus pour tous les projets de ML de l'entreprise.
- ➎ **Squads dédiés ou équipes produits (Product Teams)** : Des équipes autonomes et multidisciplinaires sont responsables du développement de produits spécifiques intégrant des modèles ML.
- ➏ **Approche par chapitre et guildes (Spotify Model)** : Une combinaison d'équipes autonomes (squads) et de groupes d'experts (chapitres) partageant des pratiques et des standards communs.



# Approche en silos (traditionnelle)

Dans cette organisation, les équipes de data science et de développement fonctionnent de manière indépendante. Les data scientists développent leurs modèles en isolation, puis passent le modèle aux développeurs pour le déploiement.

## Avantages :

- Chaque équipe se concentre sur son expertise principale.
- Les processus peuvent être optimisés indépendamment pour chaque équipe.

## Inconvénients :

- Manque de communication, ce qui peut entraîner des problèmes d'intégration en production (incompatibilité de l'infrastructure, erreurs dans le pipeline de données).
- Les retours d'expérience des développeurs arrivent trop tard, après la phase de conception du modèle.
- Problèmes de duplication d'efforts et de perte de temps lors des itérations.

## Quand l'utiliser :

- Pour des projets simples et bien définis où les étapes sont clairement séparées.

# Approche collaborative (cross-fonctionnelle)

Les équipes de data science et de développement travaillent ensemble dès le début du projet. Les développeurs participent aux étapes de conception des modèles, et les data scientists collaborent pendant le développement des systèmes de production.

## Avantages :

- Meilleure communication entre les équipes.
- Les problèmes d'intégration sont détectés plus tôt.
- Les modèles sont conçus pour être déployés plus rapidement, avec moins de risques d'incompatibilités techniques.

## Inconvénients :

- Plus de coordination et de gestion de projet nécessaire.
- Peut ralentir le développement initial en raison des échanges constants.

## Quand l'utiliser :

- Pour des projets complexes où l'intégration du modèle dans le système de production est cruciale et nécessite une optimisation fine.
- Lorsque des itérations fréquentes entre le développement du modèle et le déploiement sont nécessaires.

# Approche DevOps/MLOps

Cette organisation intègre les principes du DevOps dans la data science sous le terme de MLOps. L'idée est de créer des pipelines automatisés pour entraîner, tester, déployer, surveiller et maintenir les modèles de machine learning.

## Avantages :

- Automatisation des processus, ce qui réduit les erreurs humaines.
- Déploiement plus rapide et plus fréquent des modèles en production.
- Les équipes de data science et de développement partagent les mêmes outils et processus d'intégration continue (CI/CD).
- Meilleure gestion des versions de modèles et surveillance en temps réel pour réentraîner si nécessaire.

## Inconvénients :

- Nécessite une mise en place initiale complexe des outils et des processus.
- Peut nécessiter des compétences supplémentaires pour les équipes data.

## Quand l'utiliser :

- Pour des projets à long terme où de nombreux modèles sont mis à jour fréquemment.
- Dans les environnements où la qualité, la performance et la réactivité du modèle en production sont critiques

# Équipe centralisée (Center of Excellence)

Dans cette organisation, une équipe centralisée est responsable à la fois du développement des modèles de machine learning et de leur déploiement. Elle regroupe des compétences en data science, en développement et en MLOps.

## Avantages :

- Unité dans la gestion des processus et des technologies.
- Facilite la collaboration entre les disciplines.
- Meilleure standardisation des pratiques et des outils utilisés dans toute l'entreprise.

## Inconvénients :

- Peut manquer de flexibilité si les besoins spécifiques des équipes sont très variés.
- Risque d'une surcharge de travail pour l'équipe centralisée si l'organisation est trop large.

## Quand l'utiliser :

- Pour des entreprises qui veulent centraliser l'expertise en machine learning et l'appliquer à plusieurs départements.

# Squads dédiés ou équipes produits (Product Teams)

Les équipes de data science et de développement sont intégrées au sein de petites équipes produits, appelées "squads". Chaque squad est autonome et responsable d'un produit spécifique ou d'une fonctionnalité (ex. : un modèle de recommandation, un algorithme de détection de fraude).

## Avantages :

- Les équipes sont alignées sur un objectif commun : le produit.
- Collaboration directe et quotidienne entre data scientists, développeurs, et autres membres du produit.
- Grande réactivité et agilité pour itérer rapidement sur le produit.

## Inconvénients :

- Risque de duplication d'efforts si chaque squad développe des solutions similaires sans coordination.
- Peut être difficile à mettre en place dans des grandes organisations avec des équipes réparties.

## Quand l'utiliser :

- Dans des organisations avec une forte culture d'agilité ou des entreprises tech où la personnalisation de chaque produit est clé.

# Approche par chapitre et guilde

Inspirée du modèle d'organisation de Spotify, cette approche combine des squads (équipes produits) avec des chapitres (groupes d'experts dans un domaine, comme la data science ou le développement) et des guildes (groupes transversaux autour de pratiques communes).

## Avantages :

- Maintient une expertise forte au sein des équipes tout en encourageant la collaboration inter-équipes.
- Favorise l'innovation et l'échange de bonnes pratiques.
- Flexibilité pour adapter les équipes aux besoins du projet.

## Inconvénients :

- Complexité organisationnelle si mal gérée.
- Peut entraîner des conflits de priorités entre les squads et les chapitres.

## Quand l'utiliser :

- Dans des entreprises de taille moyenne à grande qui veulent favoriser l'innovation tout en maintenant une cohérence entre les équipes.

# Quels problèmes le MLOps résout-il ?

- **Déploiement** : Les modèles développés en laboratoire ne sont souvent pas directement utilisables en production en raison de différences d'environnement, d'outils, et de processus.
- **Répétabilité** : Les processus manuels dans le développement et le déploiement des modèles sont souvent source d'erreurs, de lenteurs, et de difficultés à reproduire les résultats.
- **Cycle de vie** : Une fois déployés, les modèles ML peuvent perdre en performance au fil du temps (en raison de la dérive des données, par exemple). Sans une gestion appropriée, cela peut entraîner des décisions erronées ou des inefficacités.
- **Collaboration** : Les équipes de data science, d'ingénierie, et d'opérations travaillent souvent en silos, ce qui complique la collaboration et entraîne des frictions lors du déploiement et de la gestion des modèles en production.
- **Scalabilité** : Le déploiement de modèles ML à grande échelle peut être complexe, surtout lorsque plusieurs modèles doivent être gérés, surveillés, et mis à jour simultanément.

# Collaboration

Data scientist



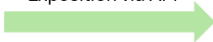
Passage du code



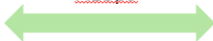
Opérations



Exposition via API

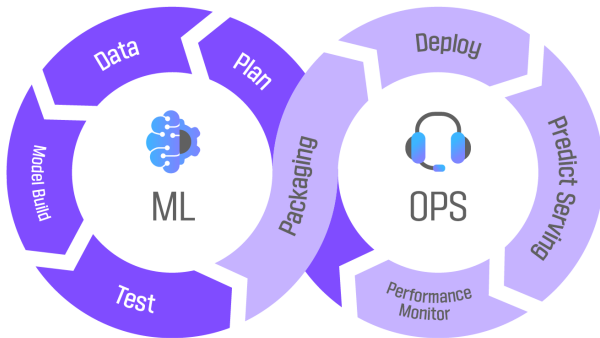


MLOps





# Machine learning + Opérations



# Principes du MLOps

- Automatisation du cycle de vie des modèles.
- Intégration continue et déploiement continu (CI/CD).
- Collaboration entre les équipes ML (data scientists) et les équipes opérations (Data engineer, ML engineers, Développeurs).
- Monitoring des modèles et des données en production.
- Versioning et traçabilité du code et des données.
- Optimisation et gestion des ressources, notamment sur le Cloud.

# Dans la pratique...

Les approches MLOps peuvent être différentes d'une entreprise à une autre et dépendent de plusieurs facteurs :

- Infrastructure de l'entreprise.
- Organisation de l'entreprise.
- Ressources disponibles en termes de compétences.
- Projets mis en oeuvre.
- Maturité de l'entreprise du point de vue du ML.
- Etc. etc.

# Outils

## Versioning



GitHub



GitLab



Data version Control

## Virtualisation



Docker



Podman



Kubernetes

## CI/CD



Jenkins



GitHub Actions



Pytest

## APIs



Flask



FastAPI



Postman

## Cloud



AWS SageMaker



GCP VertexAI

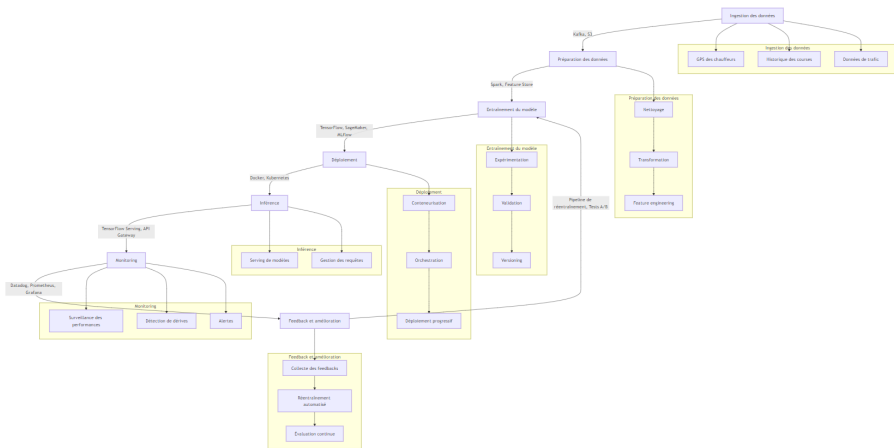


Azure ML

# Plate-formes end-to-end



## Exemple d'architecture MLOps



# Organisation : principaux métiers de l'IA

Les métiers de l'IA sont en constante évolution et ont subi une évolution assez importantes ces dernières années. On peut distinguer cependant les métiers suivants :

- **Data scientist** : il est au cœur du projet IA. Son principal rôle consiste à traduire le besoin métier en une problématique technique pour élaborer un modèle de ML ou autre, qu'il doit tester et valider avec les métiers. Il est donc normalement amené à travailler étroitement avec les métiers.
- **Machine Learning Engineer** : son rôle consiste à adapter le code développé par le data scientist pour le déployer, souvent sur une infrastructure Cloud (AWS, GCP, Azure, etc.). Il doit ensuite veiller au bon fonctionnement de l'application.
- **Data Engineer** : il est responsable de l'infrastructure liée au projet (pipelines de données, etc.).
- **Data analyst** : son rôle est d'analyser les données pour répondre à des questions métier.
- **Data architecte** : il conçoit l'architecture de la solution qui intègre éventuellement de l'IA (bases de données, back-end, front-end, services Cloud, etc.), notamment quand celle-ci est complexe.

# Virtualisation avec Docker



# Introduction à Docker

## ● Qu'est-ce que Docker ?

- Docker est une plateforme open-source qui permet de développer, expédier et exécuter des applications dans des conteneurs.
- Un conteneur est une unité standardisée de logiciel qui emballe le code et toutes ses dépendances pour que l'application s'exécute rapidement et de manière fiable d'un environnement informatique à un autre.

## ● Pourquoi utiliser Docker ?

- Portabilité : Les conteneurs peuvent s'exécuter sur n'importe quel système ayant Docker installé, qu'il s'agisse de votre machine locale, d'un serveur dans le cloud, ou d'une machine virtuelle.
- Isolation : Les conteneurs isolent les applications, ce qui permet d'éviter les conflits de dépendances et de bibliothèques.
- Scalabilité : Docker facilite le déploiement et la mise à l'échelle des applications grâce à des orchestrateurs comme Kubernetes.

# Fonctionnement Dev et Ops avant les conteneurs

## Équipe Développement (Dev)

- Focus sur l'écriture et la livraison de nouvelles fonctionnalités.
- Environnements de développement local souvent différents de la production.
- Difficultés à assurer la portabilité du code entre les environnements (local, test, production).
- Confiance limitée dans les infrastructures des opérations.

## Équipe Opérations (Ops)

- Focus sur la stabilité, la performance et la maintenance des serveurs.
- Gestion des configurations et des dépendances complexes dans différents environnements.
- Problèmes liés aux applications qui "fonctionnent en local" mais échouent en production.
- Confiance limitée dans la stabilité des nouvelles versions livrées par les développeurs.

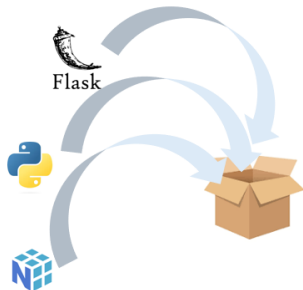
# Empaquetage avec Docker

## Qu'est-ce que l'empaquetage dans Docker ?

L'empaquetage dans Docker consiste à encapsuler une application et ses dépendances dans une image Docker.

## Avantages de l'empaquetage

- Garantir la cohérence entre les environnements de développement, de test et de production.
- Simplifier le déploiement sur n'importe quel environnement où Docker est installé.
- Réduire les erreurs liées aux différences de configuration ou de dépendances.



# Différence entre Machine Virtuelle et Docker

## Machine Virtuelle (VM)

- Chaque VM inclut un système d'exploitation complet (OS invité).
- Les VMs sont lourdes en ressources (CPU, mémoire, stockage).
- Nécessite un hyperviseur pour gérer les VMs (ex : VMware, Hyper-V).
- Démarrage et exécution plus lents à cause du démarrage de l'OS.
- Isolation complète entre les VMs.



## Docker (Conteneurs)

- Les conteneurs partagent le noyau du système d'exploitation hôte.
- Les conteneurs sont légers et utilisent moins de ressources.
- Pas besoin d'un hyperviseur, tout est géré par le moteur Docker.
- Démarrage quasi-instantané grâce au partage du noyau de l'OS.
- Isolation légère, mais moins stricte que celle des VMs.



# Concepts de base de Docker

- **Images Docker**

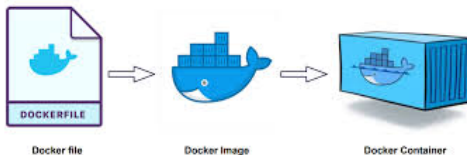
- Une image est un modèle immuable contenant tout ce dont une application a besoin pour s'exécuter : le code, les bibliothèques, les variables d'environnement, et les fichiers de configuration.
- Les images sont construites à partir de fichiers Dockerfile.

- **Conteneurs Docker**

- Un conteneur est une instance d'une image Docker en cours d'exécution.
- Les conteneurs peuvent être créés, démarrés, arrêtés, déplacés et supprimés en utilisant l'interface de ligne de commande Docker ou l'API REST de Docker.

- **Dockerfile**

- Un Dockerfile est un fichier texte contenant une série d'instructions pour construire une image Docker.
- Il définit les étapes nécessaires pour configurer l'environnement dans lequel votre application va s'exécuter.



# Exemple de Dockerfile

```
# Utiliser une image Python comme base
FROM python:3.9-slim

# Définir le répertoire de travail à l'intérieur du conteneur
WORKDIR /app

# Copier le fichier requirements.txt dans le conteneur
COPY requirements.txt .

# Installer les dépendances Python nécessaires
RUN pip install --no-cache-dir -r requirements.txt

# Copier tout le contenu du projet dans le répertoire de travail
COPY . .

# Exposer le port sur lequel l'application va tourner
EXPOSE 5000

# Spécifier la commande pour lancer l'application Flask
CMD ["python", "app.py"]
```

# Docker Hub

- **Qu'est-ce que Docker Hub ?** Docker Hub est un service de registre d'images Docker public. Il permet aux développeurs de stocker, partager et distribuer des images Docker.
- **Fonctionnalités principales**
  - Hébergement d'images publiques et privées.
  - Recherche d'images officielles et communautaires (par ex. `nginx`, `python`).
  - Gestion des équipes et accès pour collaborer autour des images Docker.
  - Intégration avec des outils CI/CD pour automatiser la création et le déploiement d'images.
- **Utilisation courante**
  - `docker pull <image>` : Télécharger une image depuis Docker Hub.
  - `docker push <image>` : Envoyer une image vers Docker Hub.



# Docker Compose

- **Qu'est-ce que Docker Compose ?**

- Un outil permettant de définir et gérer des applications multi-conteneurs.
- Utilise un fichier YAML ('docker-compose.yml') pour configurer les services, réseaux et volumes.
- Permet de lancer tous les conteneurs avec une seule commande :  
`docker-compose up`.

- **Principales fonctionnalités**

- Démarrage, arrêt et gestion de plusieurs conteneurs à la fois.
- Configuration des réseaux et volumes partagés entre les conteneurs.
- Facile à utiliser pour les environnements de développement et de test.





# Pourquoi utiliser Docker Compose ?

- **Gestion multi-conteneurs simplifiée**

- Facilite le démarrage d'applications nécessitant plusieurs services (ex. : une API, une base de données, un cache).

- **Environnements reproductibles**

- Tout l'environnement est décrit dans un fichier YAML, assurant que tout le monde utilise la même configuration.

- **Isolation des services**

- Chaque service fonctionne dans son propre conteneur avec ses propres dépendances.

- **Commandes simples**

- Démarrer tous les services : `docker-compose up`.
- Arrêter et nettoyer les services : `docker-compose down`.

# Structure d'un fichier docker-compose.yml

## Exemple de fichier docker-compose.yml

- Déclaration des services (ex : web, base de données).
- Définition des réseaux et volumes partagés.

```
version: '3.8'

services:
  api:
    build:
      context: .
      dockerfile: Dockerfile.api
    ports:
      - "8080:5000"
    environment:
      DATABASE_URL: "postgresql://user:password@db:5432/ml_db"
    depends_on:
      - db
    volumes:
      - ./api:/app

  preprocess:
    build:
      context: .
      dockerfile: Dockerfile.preprocess
    command: python preprocess.py
    volumes:
      - ./preprocess:/app
    depends_on:
      - db

  db:
```

# Introduction à Kubernetes

- **Qu'est-ce que Kubernetes ?**

- Kubernetes (K8s) est une plateforme open-source d'orchestration de conteneurs.
- Conçu pour automatiser le déploiement, la mise à l'échelle et la gestion des applications conteneurisées.
- Fonctionne sur des clusters de machines (physiques ou virtuelles).

- **Principales fonctionnalités**

- Mise à l'échelle automatique des applications.
- Distribution des charges de travail entre les différents nœuds du cluster.
- Tolérance aux pannes et redémarrage automatique des conteneurs défectueux.
- Gestion des secrets, des volumes persistants, et des configurations.



# Concepts clés de Kubernetes

- **Pod**

- Unité de base de déploiement dans Kubernetes, un pod contient un ou plusieurs conteneurs.
- Les conteneurs d'un même pod partagent le même réseau et les mêmes volumes.

- **Node**

- Un nœud est une machine (virtuelle ou physique) qui exécute des pods.
- Kubernetes répartit les pods sur les différents nœuds du cluster.

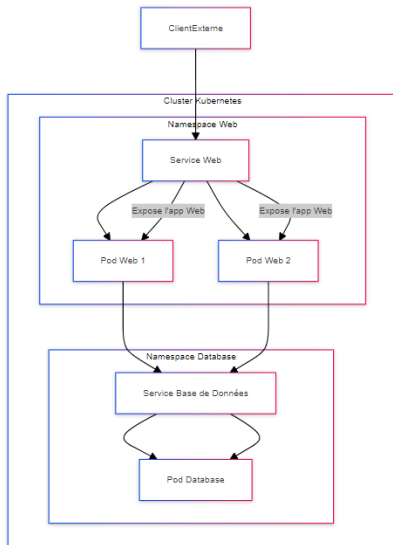
- **Service**

- Abstraction qui expose un ensemble de pods sous forme de service réseau stable.
- Permet la communication entre les composants internes ou externes au cluster.

- **Deployment**

- Un Deployment gère le déploiement et la mise à jour de groupes de pods.
- Permet de garantir qu'un certain nombre de pods sont en cours d'exécution à tout moment.

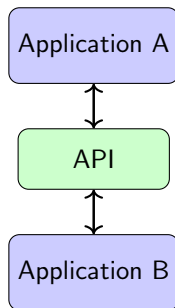
# Exemple d'orchestration avec Kubernetes



# Qu'est-ce qu'une API ?

## API : Interface de Programmation d'Application

- Ensemble de définitions, protocoles et outils
- Permet la communication entre différents logiciels
- Agit comme un "contrat" entre deux applications
- Facilite l'intégration et la modularité



## Exemples :

- APIs Web (REST, GraphQL)
- APIs de systèmes d'exploitation
- APIs de bibliothèques logicielles

# Avantages de l'APlisation

- **Interopérabilité** : Facilite la communication entre différents systèmes et technologies, permettant une intégration fluide.
- **Modularité et réutilisabilité** : Permet de développer des composants indépendants et réutilisables, réduisant la duplication de code.
- **Scalabilité** : Supporte plus facilement la croissance et l'évolution des systèmes, adaptable aux changements de charge.
- **Amélioration de l'expérience développeur** : Simplifie l'intégration et accélère le développement, offrant des interfaces claires et documentées.
- **Innovation et nouveaux modèles économiques** : Permet la création de nouveaux services et produits basés sur les APIs, ouvrant de nouvelles opportunités.
- **Sécurité et contrôle d'accès** : Offre une couche de sécurité supplémentaire et une gestion fine des accès aux ressources et fonctionnalités.

# Types d'APIs

- API Web : REST, SOAP, GraphQL
  - Communication entre applications via Internet
- API Système : Windows API, POSIX
  - Interaction avec les systèmes d'exploitation
- API de Bibliothèque : Java API, .NET Framework
  - Accès aux fonctionnalités de bibliothèques logicielles
- API Hardware : OpenGL, DirectX
  - Communication entre logiciels et matériel



# API REST : Définition et principes

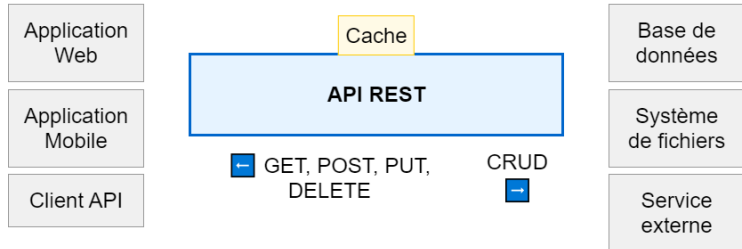
**Définition :** REST (Representational State Transfer) est un style d'architecture pour la conception d'applications en réseau. Une API REST utilise les méthodes HTTP pour effectuer des opérations CRUD (Create, Read, Update, Delete) sur des ressources, identifiées par des URIs.

## Principes clés :

- **Client-Serveur** : Séparation des préoccupations, améliorant la portabilité et la scalabilité.
- **Sans état** : Chaque requête contient toutes les informations nécessaires, aucune session côté serveur.
- **Mise en cache** : Les réponses doivent être explicitement marquées comme cachables ou non-cachables.
- **Interface uniforme** : Simplicité et découplage de l'architecture, basée sur les ressources, les représentations et les messages auto-descriptifs.
- **Système en couches** : Hiérarchie des composants, chaque couche ne peut voir que la couche adjacente.

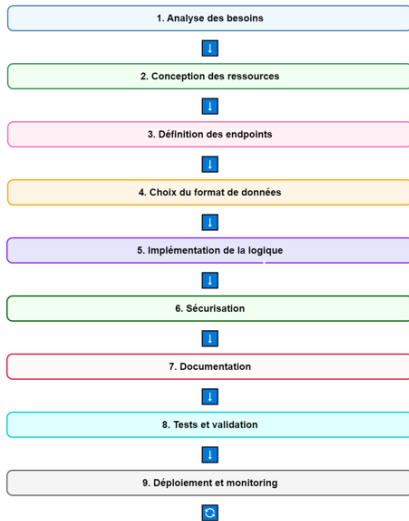
L'API REST favorise la simplicité, la scalabilité et l'interopérabilité, en utilisant les standards web existants (HTTP, URI, JSON/XML) pour faciliter l'intégration entre systèmes hétérogènes.

# Schéma de principe d'une API REST



# Conception d'une API REST

## Processus d'API REST



# Outils et technologies Python d'APIs

- **Frameworks de développement** : Flask, FastAPI, Django, Streamlit, etc.
- **Outils de documentation** : Sphinx, MkDocs, pydoc, FastAPI (documentation automatique avec Swagger/OpenAPI)
- **Sécurité et authentification** : Flask-JWT, Django-oauth-toolkit, Python-jose (pour JWT), Authlib
- **Tests et monitoring** : Pytest, unittest, Locust (tests de charge), Django Debug Toolbar, Flask-MonitoringDashboard
- **Formats de données** : JSON (json module), XML (xml.etree.ElementTree), Protocol Buffers (protobuf), YAML (PyYAML)
- **Analyse et analytics** : Django Silk, Flask-Monitoring, OpenTelemetry-Python

# CI/CD

# CI/CD pour ML : Définition et principes

**Définition :** La CI/CD (Intégration Continue/Déploiement Continu) pour ML est l'application des pratiques d'automatisation et de surveillance continue au cycle de vie complet des modèles de machine learning, de l'intégration et des tests à la livraison et au déploiement.

## Principes clés :

- **Reproductibilité :** Garantir la reproduction fiable des expériences et résultats ML.
- **Versioning :** Gérer les versions du code, des données et des modèles.
- **Automatisation :** Tests, évaluation et déploiement des modèles ML.
- **Gestion des données :** Intégrer le contrôle de version et la validation des données.
- **Déploiement contrôlé :** Utiliser des techniques comme le blue/green ou canary release.
- **Monitoring continu :** Surveiller les performances et détecter les dérives du modèle.
- **Traçabilité :** Maintenir un historique complet pour l'audit et la gouvernance.

# Spécificités de la CI/CD pour les projets de ML

- **Dépendance aux données** : Intégration de pipelines de données dans le processus CI/CD, gestion des versions des datasets.
- **Expérimentation itérative** : Adaptation du CI/CD pour gérer de multiples expériences et hyperparamètres.
- **Reproductibilité complexe** : Nécessité de versionner non seulement le code, mais aussi les environnements, les données et les seeds aléatoires.
- **Tests spécifiques au ML** : Inclusion de tests de performance du modèle, de biais, et de robustesse en plus des tests unitaires classiques.
- **Déploiement des modèles** : Gestion du déploiement de modèles en tant qu'artefacts distincts du code applicatif.
- **Monitoring post-déploiement** : Surveillance continue des performances du modèle et détection de la dérive conceptuelle.
- **Réentraînement automatisé** : Intégration de boucles de feedback pour le réentraînement automatique des modèles.
- **Gestion des ressources** : Orchestration de ressources de calcul variables pour l'entraînement et l'inférence.

# Défis courants de la CI/CD pour les projets ML

- **Reproductibilité des résultats** : Garantir que les expériences ML produisent des résultats cohérents à chaque exécution.
- **Gestion des données volumineuses** : Intégrer efficacement de grands ensembles de données dans les pipelines CI/CD.
- **Versionnage complexe** : Gérer simultanément les versions du code, des données, des modèles et des environnements.
- **Temps d'exécution longs** : Optimiser les pipelines pour gérer les longs temps d'entraînement des modèles.
- **Dépendances instables** : Maintenir la stabilité avec des bibliothèques ML en évolution rapide.
- **Tests automatisés adaptés** : Concevoir des tests pertinents pour évaluer la qualité et les performances des modèles ML.
- **Dérive des données et des modèles** : Détecter et gérer les changements dans les patterns de données et les performances des modèles.
- **Environnements hétérogènes** : Assurer la cohérence entre les environnements de développement, de test et de production.
- **Gouvernance et conformité** : Intégrer les exigences de traçabilité et d'explicabilité des modèles dans le pipeline CI/CD.



# Bonnes pratiques de CI/CD en ML

- **Versionnage holistique** : Utiliser DVC ou MLflow pour versionner code, données et modèles ensemble.
- **Conteneurisation** : Employer Docker pour encapsuler l'environnement ML complet, assurant la reproductibilité.
- **Tests multicouches** : Implémenter des tests unitaires, d'intégration, et de performance des modèles.
- **Validation des données** : Intégrer des vérifications automatiques de la qualité et de la cohérence des données.
- **Pipeline modulaire** : Concevoir des pipelines CI/CD flexibles, séparant l'entraînement du déploiement.
- **Métriques de surveillance** : Définir et suivre des KPIs clairs pour les performances des modèles en production.
- **Déploiement progressif** : Utiliser des techniques comme le blue/green deployment ou le canary release.
- **Automatisation du réentraînement** : Mettre en place des triggers pour le réentraînement basé sur les performances.
- **Documentation as Code** : Maintenir la documentation du modèle et du pipeline dans le repository.
- **Gestion des secrets** : Utiliser des coffres-forts pour les credentials et les données sensibles.

# Configuration de l'environnement virtuel pour ML

- **Choix de l'outil :**

- virtualenv ou venv pour Python
- Conda pour une gestion plus large (Python, R, etc.)
- Docker pour une isolation complète

- **Création de l'environnement :**

- Python : `python -m venv myenv`
- Conda : `conda create --name myenv python=3.x`
- Docker : Créer un Dockerfile approprié

- **Gestion des dépendances :**

- Utiliser `requirements.txt` ou `environment.yml`
- Spécifier les versions exactes des packages

- **Bonnes pratiques :**

- Un environnement par projet
- Versionner les fichiers de configuration
- Utiliser `.gitignore` pour exclure l'environnement du versioning
- Documenter les étapes de configuration

- **Intégration CI/CD :**

- Automatiser la création de l'environnement dans les pipelines
- Utiliser des caches pour accélérer les builds

# Outils de CI/CD pour ML

- **Gestion de version** : Git, GitHub, GitLab, Bitbucket, DVC (Data Version Control), MLflow
- **Orchestration de pipelines** : Apache Airflow, Kubeflow, Argo Workflows, Prefect
- **Plateformes CI/CD** : Jenkins, GitLab CI/CD, GitHub Actions, CircleCI, Travis CI
- **Conteneurisation** : Docker, Kubernetes, OpenShift
- **Gestion d'environnements** : Conda, virtualenv, Docker
- **Gestion de packages** : pip, conda, Poetry
- **Tests et validation** : pytest, unittest, Great Expectations
- **Monitoring et logging** : Prometheus, Grafana, ELK Stack (Elasticsearch, Logstash, Kibana)
- **Gestion d'expériences** : MLflow, Weights & Biases, Neptune.ai

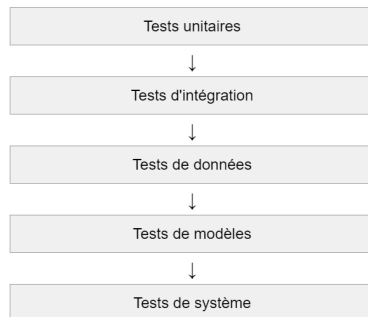
# Tests automatisés pour ML

## Types de tests :

- **Tests unitaires** : Fonctions individuelles, composants
- **Tests d'intégration** : Interactions entre composants
- **Tests de données** : Qualité, cohérence, intégrité
- **Tests de modèles** : Performance, biais, robustesse
- **Tests de système** : Pipeline ML complet

## Outils :

- pytest, unittest pour tests unitaires et d'intégration
- Great Expectations pour tests de données
- MLflow, scikit-learn pour tests de modèles



# Validation des données et des features

## Importance :

- Garantit la qualité et la cohérence des données
- Préviend les erreurs en aval dans le pipeline ML
- Assure la fiabilité et la reproductibilité des modèles

## Techniques de validation :

- **Vérifications de schéma** : Types de données, contraintes
- **Contrôles statistiques** : Distribution, outliers, corrélations
- **Vérifications d'intégrité** : Valeurs manquantes, doublons
- **Validation métier** : Règles spécifiques au domaine
- **Tests de dérive** : Comparaison avec données historiques

## Intégration dans CI/CD :

- Automatiser les tests de données
- Définir des seuils de qualité
- Alertes en cas d'anomalies
- Versionnage des données validées

# Évaluation automatique des modèles ML

## Objectifs :

- Assurer la qualité et la fiabilité des modèles
- Détecter les régressions de performance
- Faciliter la comparaison entre versions
- Automatiser la prise de décision pour le déploiement

## Métriques courantes :

- Classification : Précision, Rappel, F1-score, AUC-ROC
- Régression : MSE, RMSE, MAE,  $R^2$
- Spécifiques au domaine : temps d'inférence, taille du modèle

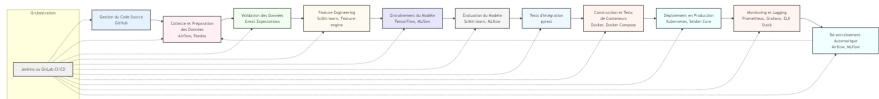
## Techniques d'évaluation :

- Validation croisée
- Évaluation sur des sous-ensembles spécifiques
- Tests de robustesse (données bruitées, adversariales)

## Intégration dans CI/CD :

- Exécution automatique après l'entraînement
- Définition de seuils de performance acceptables
- Comparaison avec le modèle de référence (baseline)
- Génération de rapports automatiques

# Exemple de CI/CD



# Outils pour l'intégration continue

- **Jenkins** Un serveur d'automatisation open-source permettant de configurer et d'exécuter des pipelines CI/CD.
- **GitHub Actions** Intégré à GitHub, permet de créer des workflows automatisés pour tester et déployer le code.
- **GitLab CI** Intégré à GitLab, fournit des fonctionnalités CI/CD robustes avec une configuration simple en YAML.



# Pipeline de CI/CD pour modèles ML

## Étapes d'un pipeline CI/CD

- Extraction du code et des configurations de modèle depuis le dépôt.
- Construction de l'environnement d'entraînement (par exemple, via Docker).
- Entraînement et évaluation des modèles.
- Tests unitaires et tests de validation sur les modèles.
- Déploiement des modèles validés en production.

# Déploiement des modèles

## Méthodes de déploiement

- Déploiement en tant que service web (API RESTful).
- Déploiement dans des pipelines de données pour traitement batch.
- Intégration dans des systèmes de recommandation ou des applications en temps réel.

## Outils de déploiement

- **Docker** : conteneurisation des modèles pour un déploiement reproductible.
- **Kubernetes** : orchestration des conteneurs pour une scalabilité et une gestion efficaces.
- **TensorFlow Serving** : serveur dédié pour le déploiement de modèles TensorFlow.
- **TorchServe** : serveur pour déployer des modèles PyTorch.

# Surveillance des modèles en production

- **Importance de la surveillance**

- Assurer que les modèles fonctionnent comme prévu.
- Détecter les dérives de données et les baisses de performance.
- Réagir rapidement aux incidents en production.

- **Outils de surveillance**

- **Prometheus** : système de surveillance et de collecte de métriques open-source.
- **Grafana** : outil de visualisation pour afficher les métriques collectées.
- **MLflow** : suivi des expériences, gestion des modèles et surveillance des performances.

# Maintenance des modèles

- **Gestion des versions de modèles**

- Importance de versionner les modèles pour assurer la traçabilité et la reproductibilité.
- Outils comme DVC (Data Version Control) pour gérer les versions des données et des modèles.

- **Mise à jour des modèles**

- Stratégies de mise à jour pour minimiser l'impact en production (par exemple, déploiement bleu-vert, canary deployment).
- Validation continue des nouveaux modèles avant leur déploiement complet.

- **Ré-entraînement des modèles**

- Planification de ré-entraînements périodiques pour s'assurer que les modèles restent performants face à des données changeantes.
- Surveillance des dérives de données pour déclencher des ré-entraînements automatiques.

# Monitoring

# Types de monitoring

## **Performance :**

- mesures courantes : précision, rappel, F1-score, AUC
- comparaison des prédictions actuelles avec les données historiques pour identifier les tendances

## **Dérives :**

- surveillance de la dérive des données (distributions des features en entrée)
- analyse des dérives des prédictions (ex. distribution des probabilités de classe)

## **Latence et disponibilité :**

- temps de réponse du modèle : analyse des variations dans le temps
- suivi de la consommation des ressources (CPU, RAM)
- détection des pannes ou des indisponibilités

# Méthodes pour surveiller les dérives

## Dérive des données :

- analyse statistique des variables en entrée (KS-test,  $\chi^2$ )
- surveillance des variables critiques (features clés influençant fortement les prédictions)

## Dérive des prédictions :

- analyse des distributions des sorties du modèle (probabilités des classes, etc.)
- comparaison entre les étiquettes réelles et les prédictions pour détecter les anomalies

## Approches :

- seuils prédéfinis pour les alertes (ex. baisse de l'AUC sous un seuil critique)
- intégration d'outils comme Prometheus pour la collecte des métriques et des alertes

# Alertes et actions automatisées

- mise en place d'alertes dès qu'un seuil critique est dépassé
- outils d'intégration comme Slack ou e-mail pour notifier en temps réel les anomalies détectées
- actions automatiques possibles : déclencher un retrain, désactiver temporairement le modèle ou ajuster les paramètres
- suivi des logs pour comprendre les causes des anomalies



# Outils de monitoring

## **Prometheus et Grafana :**

- collecte et stockage de métriques système (temps de réponse, taux d'erreur, etc.)
- visualisation des métriques en temps réel via Grafana
- alertes configurables via Prometheus Alertmanager

## **Evidently.ai :**

- détection des dérives des données et des modèles
- suivi de la performance des modèles en production
- analyse détaillée des changements de distributions de features

## **Seldon Core :**

- framework pour déployer, monitorer et gérer les modèles dans Kubernetes
- inclut des fonctionnalités pour le monitoring des performances et la gestion des dérives

# Étapes clés pour une surveillance efficace

- définir les métriques de performance critiques (ex. accuracy, recall)
- surveiller la dérive des données en continu
- mettre en place des alertes en cas de déviation par rapport aux performances attendues
- suivre les logs et traces pour diagnostiquer les causes des dégradations
- évaluer et réajuster le modèle et le pipeline régulièrement

# Cas d'usage : modèle de prédiction du churn

- surveillance des performances : AUC, F1-score, précision, rappel
- suivi des dérives des variables clés : âge, transactions, durée d'abonnement
- intégration avec Prometheus pour suivre les métriques en temps réel
- alertes configurées pour détecter une baisse de performance
- actions automatisées : redéploiement du modèle ou réentraînement automatique en cas de dérive

# Conclusion

- le monitoring des modèles est crucial pour maintenir la performance en production
- la détection des dérives doit être rapide et efficace, avec des alertes automatisées
- choisir les bons outils et stratégies de surveillance est essentiel pour garantir une réactivité optimale

# Transformers et LLMs

---

## Attention Is All You Need

---

**Ashish Vaswani\***  
Google Brain  
avaswani@google.com

**Noam Shazeer\***  
Google Brain  
noam@google.com

**Niki Parmar\***  
Google Research  
nikip@google.com

**Jakob Uszkoreit\***  
Google Research  
usz@google.com

**Llion Jones\***  
Google Research  
llion@google.com

**Aidan N. Gomez\*<sup>†</sup>**  
University of Toronto  
aidan@cs.toronto.edu

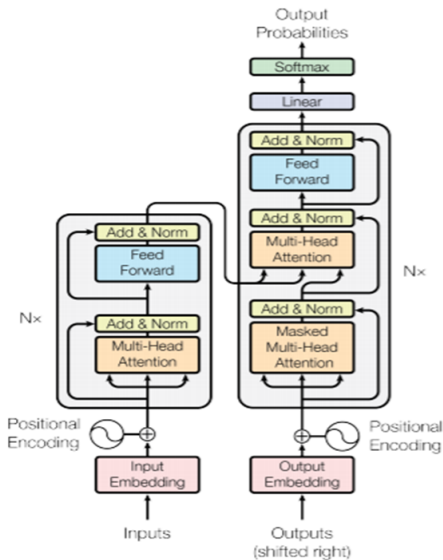
**Lukasz Kaiser\***  
Google Brain  
lukaszkaier@google.com

**Illia Polosukhin\*<sup>‡</sup>**  
illia.polosukhin@gmail.com

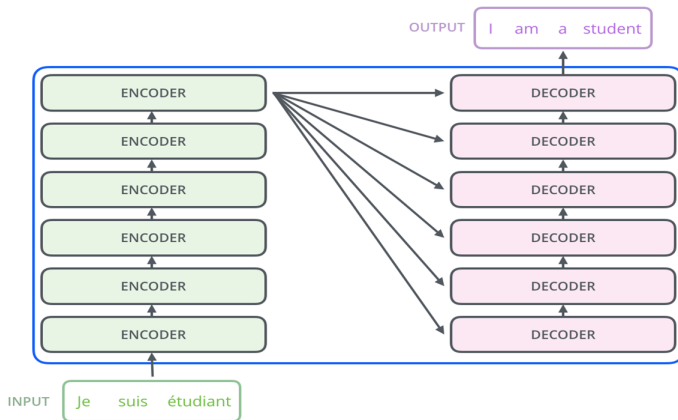
### Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to

# Transformer originel

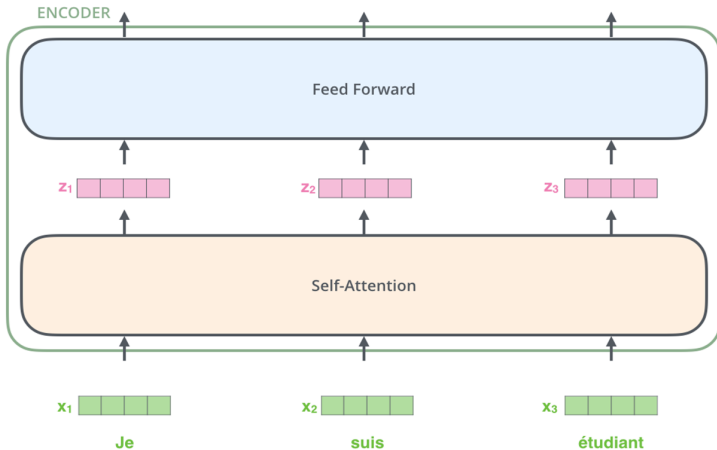


# Mécanisme de cross-attention

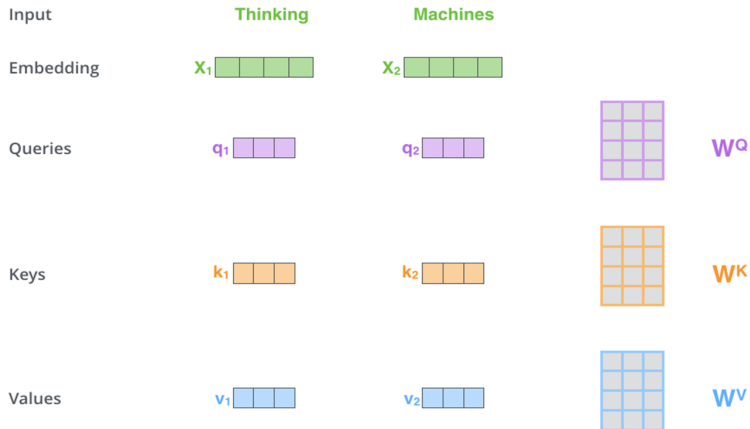




# Mécanisme de self-attention



# Fonctionnement du mécanisme de self-attention



# Calcul des scores de self-attention

Input

Embedding

Queries

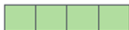
Keys

Values

Score

Thinking

$x_1$



$q_1$



$k_1$



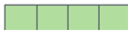
$v_1$



$$q_1 \cdot k_1 = 112$$

Machines

$x_2$



$q_2$



$k_2$

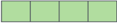
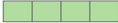








$v_2$



$$q_1 \cdot k_2 = 96$$

# Calcul des scores de self-attention

Input	Thinking	Machines
Embedding	$x_1$ 	$x_2$ 
Queries	$q_1$ 	$q_2$ 
Keys	$k_1$ 	$k_2$ 
Values	$v_1$ 	$v_2$ 
Score	$q_1 \cdot k_1 = 112$	$q_2 \cdot k_2 = 96$
Divide by $8 (\sqrt{d_k})$	14	12
Softmax	0.88	0.12

# Calcul de la nouvelle représentation

Input

Embedding

Queries

Keys

Values

Score

Divide by 8 (  $\sqrt{d_k}$  )

Softmax


Softmax  
X  
Value

Sum

Thinking

$x_1$  

$q_1$  


$k_1$  

$v_1$  

$q_1 \cdot k_1 = 112$

14

0.88

$v_1$  

$z_1$  

Machines

$x_2$  

$q_2$  


$k_2$  

$v_2$  

$q_1 \cdot k_2 = 96$

12

0.12

$v_2$  

$z_2$  

# Performances du tranformer originel

Entraîné sur WMT 2014 English- German dataset, comprenant près de 4.5 millions de phrases sentence, le WMT 2014 English-French dataset, comprenant près de 36 millions de phrases.

- 8 NVIDIA P10 GPUs
- **Base** : 12 heures
- **Large** : 3.5 jours

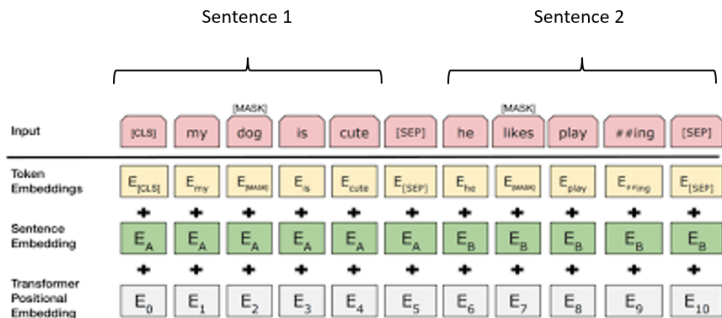
Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	<b>41.29</b>	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	<b><math>3.3 \cdot 10^{18}</math></b>	
Transformer (big)	<b>28.4</b>	<b>41.8</b>	$2.3 \cdot 10^{19}$	

# BERT

BERT (Bidirectional Encoder Representations from Transformers) représente une avancée majeure dans le NLP, introduisant une approche novatrice pour la modélisation de langage. Il utilise la bidirectionnalité pour comprendre le contexte des mots, permettant une compréhension plus fine du langage.

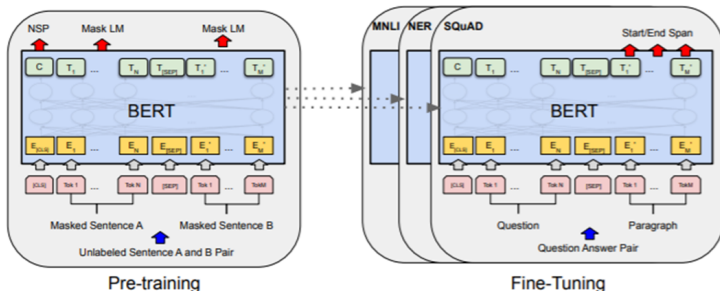
## Pré-entraînement



# Fine-tuning de BERT pour des tâches spécifiques

Après pré-entraînement, BERT est affiné pour des tâches spécifiques:

- Ajout d'une couche de sortie spécifique à la tâche (classification, NER, QA).
- Fine-tuning de tous les paramètres du modèle pré-entraîné sur le corpus de la tâche.





# Performances de BERT

BERT a été pré-entraîné sur le BookCorpus (800 millions de mots) et English Wikipedia (2,500 millions de mots).

## Deux modèles :

- **BERT Base** : 12 couches avec 110 millions de paramètres.
- **BERT Large** : 24 couches avec 340 millions de paramètres.

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Average
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
BERT <sub>BASE</sub>	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT <sub>LARGE</sub>	<b>86.7/85.9</b>	<b>72.1</b>	<b>92.7</b>	<b>94.9</b>	<b>60.5</b>	<b>86.5</b>	<b>89.3</b>	<b>70.1</b>	<b>82.1</b>

# Introduction aux modèles autorégressifs - GPT-3

GPT-3 (Generative Pre-trained Transformer 3) est le modèle de langue développé par OpenAI, représentant la troisième génération de la série GPT. Avec 175 milliards de paramètres, GPT-3 pousse les limites de la génération de texte et de la compréhension du langage naturel.

- **Capacités générales** : GPT-3 excelle dans une variété de tâches de TALN sans fine-tuning spécifique, grâce à sa puissance de modélisation du langage.
- **Architecture autorégressive** : Utilise un modèle Transformer pour prédire le mot suivant dans un texte, apprenant des patterns complexes sur des données massives.
- **Approche few-shot learning** : Capable d'adapter ses réponses à des tâches spécifiques avec peu ou pas d'exemples d'entraînement.
- **Impact sur IA et le NLP** : GPT-3 a significativement avancé les capacités des systèmes d'IA dans la compréhension et la génération de langage naturel, ouvrant de nouvelles voies pour l'application de l'intelligence artificielle.

# Architecture de GPT-3

GPT-3 repose sur une architecture Transformer améliorée, optimisée pour traiter efficacement de grandes quantités d'informations et générer des réponses cohérentes et contextuellement pertinentes.

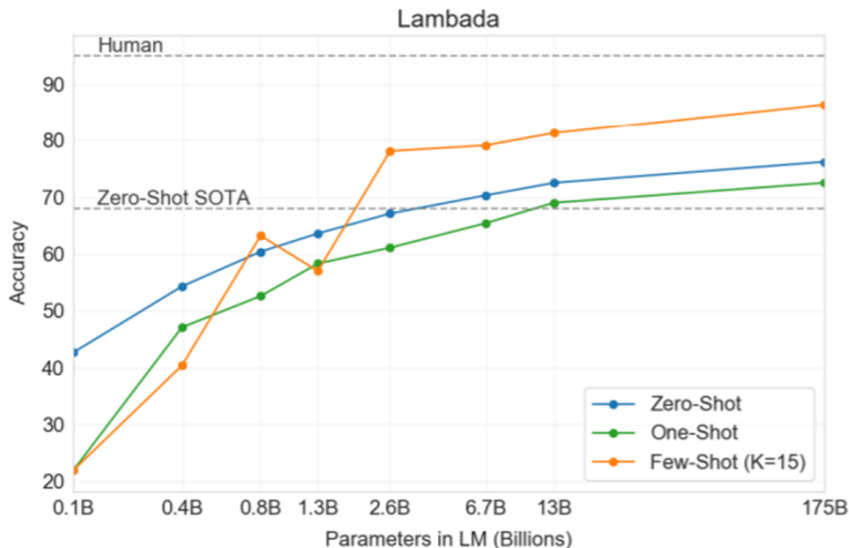
- **Taille du modèle** : Avec 175 milliards de paramètres, GPT-3 est l'un des modèles de langue les plus grands et les plus complexes à ce jour.
- **Mécanisme d'attention** : L'attention multi-têtes permet au modèle de pondérer différemment les parties d'un input, améliorant la compréhension du contexte.
- **Optimisation et entraînement** : Techniques d'optimisation avancées pour gérer la taille du modèle et l'efficacité de l'entraînement.

# Few-Shot Learning avec GPT-3

L'une des innovations les plus remarquables de GPT-3 réside dans sa capacité à effectuer du few-shot learning, permettant au modèle de comprendre et d'exécuter des tâches spécifiques avec très peu d'exemples.

- **Définition** : Le few-shot learning désigne la capacité d'un modèle à apprendre une nouvelle tâche à partir d'un très petit nombre d'exemples d'entraînement, souvent seulement quelques-uns.
- **Mécanisme dans GPT-3** :
  - GPT-3 utilise des prompts contenant quelques exemples de la tâche désirée pour guider le modèle sur ce qui est attendu, avant de présenter la question ou la tâche à résoudre.
  - Le modèle généralise ensuite à partir de ces exemples pour générer des réponses ou des solutions aux problèmes posés, montrant une compréhension étonnante de la tâche avec un minimum de guidance.
- **Exemples d'application** :
  - Classification de texte, génération de résumés, réponse à des questions spécifiques, et plus, avec seulement quelques exemples pour chaque tâche.
- **Impact** : Cette capacité émergente réduit considérablement le besoin de vastes ensembles de données d'entraînement spécifiques à la tâche, ouvrant la voie à des applications plus flexibles et accessibles de modèles de langage.

# Performances de GPT-3 en few-shot learning



# ChatGPT

ChatGPT, développé par OpenAI, est un modèle de langage basé sur l'architecture GPT (Generative Pre-trained Transformer) optimisé pour comprendre et générer des dialogues naturels. L'entraînement de ChatGPT se décline selon les étapes

suivantes :

- ➊ **Pré-entraînement sur un corpus volumineux :** Comme GPT-3, ChatGPT est d'abord pré-entraîné sur un vaste ensemble de données textuelles, englobant un large éventail de la littérature disponible sur Internet, pour apprendre une compréhension générale du langage.
- ➋ **Fine-tuning supervisé :** Ensuite, ChatGPT est affiné sur des dialogues spécifiques pour améliorer ses compétences conversationnelles. Cette étape utilise des paires question-réponse et des conversations pour enseigner au modèle des structures de dialogue et des réponses contextuellement appropriées.
- ➌ **Reinforcement Learning from Human Feedback (RLHF):** Utilisation de techniques de renforcement pour ajuster les réponses du modèle basées sur les préférences et les corrections fournies par des évaluateurs humains, raffinant davantage la pertinence et la naturalité des réponses.

# LLMs propriétaires vs Open Source

## ● Modèles propriétaires

- ChatGPT, Claude, Gemini...
- Poids non disponibles au public.
- Performances élevées, accès limité par API payante
- Protection intellectuelle stricte (code non accessible)
- Support technique assuré, documentation avancée
- Dépendance aux fournisseurs et coûts potentiellement élevés

## ● Modèles open source

- Llama, Mistral, Gemma...
- Poids disponibles au public
- Transparence du code, flexibilité d'adaptation
- Performances parfois inférieures, mais amélioration continue
- Gratuit, mais coût de l'infrastructure à gérer
- Exposition aux failles potentielles de sécurité, support communautaire

**Conclusion :** Choisir entre contrôle, flexibilité et performances optimales.

# Quantization des modèles de deep learning

**Motivation :** Réduire la taille et la consommation énergétique des modèles en diminuant la précision des poids et activations.

**Principe :** Représenter les poids et activations avec une précision inférieure (ex: 8 bits au lieu de 32 bits).

- Conversion des poids et activations de flottants (32 bits) en entiers (8, 4 ou 2 bits).
- Peut être appliquée à l'inférence uniquement (post-training quantization) ou lors de l'entraînement (quantization-aware training).
- Balance entre compression et performance : plus la quantization est forte, plus la perte de précision peut être importante.

**Avantages :**

- Réduction de l'empreinte mémoire et du temps d'inférence.
- Accélération des calculs sur hardware spécialisé (TPU, NPU, CPU avec SIMD).
- Meilleure efficacité énergétique, utile pour l'embarqué et les mobiles.



# Low-Rank Adaptation (LoRA)

**Motivation :** Adapter efficacement de grands modèles de langage sans nécessiter un fine-tuning complet.

**Principe :** Approximer les matrices de poids d'un modèle pré-entraîné en les décomposant en une somme de deux matrices de rang inférieur.

- Soit  $W_0 \in \mathbb{R}^{d \times k}$  une matrice de poids d'un modèle pré-entraîné.
- La nouvelle matrice du modèle fine-tuné  $W = W_0 + \Delta W$ .
- LoRA introduit une perturbation  $\Delta W = AB$ , avec  $A \in \mathbb{R}^{d \times r}$  et  $B \in \mathbb{R}^{r \times k}$ , où  $r \ll \min(d, k)$ .
- Seules les matrices  $A$  et  $B$  sont entraînées, réduisant ainsi le nombre de paramètres mis à jour.

**Avantages :**

- Réduction drastique des besoins en mémoire et en calcul.
- Adaptation efficace à différentes tâches sans modifier  $W_0$ .
- Compatible avec les architectures transformer.

# LoRA : Impact et Applications

## Pourquoi LoRA est efficace ?

- La majorité des poids d'un réseau de neurones pré-entraîné n'ont pas besoin d'être modifiés complètement.
- Les modèles pré-entraînés possèdent une structure de redondance exploitable via l'approximation bas rang.
- Permet d'intégrer rapidement de nouvelles tâches à un modèle sans nécessiter un réentraînement coûteux.

## Applications :

- Fine-tuning de modèles de langage (ex: adaptation GPT-3 à un domaine spécifique).
- Réduction de la consommation mémoire sur des dispositifs limités (ex: edge computing).
- Adaptation rapide de modèles multimodaux (ex: CLIP, vision et langage).

# Merci de votre attention

[redha.moulla@axia-conseil.com](mailto:redha.moulla@axia-conseil.com)