TP 6 : GEN IA , MLOPS & RAG

Introduction to Gen IA & RAG

ESME SUDRIA

Luc Marchand - luc.marchand.pro@proton.me
Maxence Talon - maxencetallon@gmail.com

Global Syllabus

01 Introduction and main concepts

02 SQL, set up env and practical work

03 NoSQL world

04 Introduction to Big Data & Data Engineering

05 Kafka & event driven architectures

06 Spark & Delta

07 Warehouse, DBT & BI

08 **IA - MLOps & RAG**

Courses Syllabus

01 Introduction To GEN IA

02 ML Ops

03 Retrieval- Augmented Generation Concepts

04 Practice

01

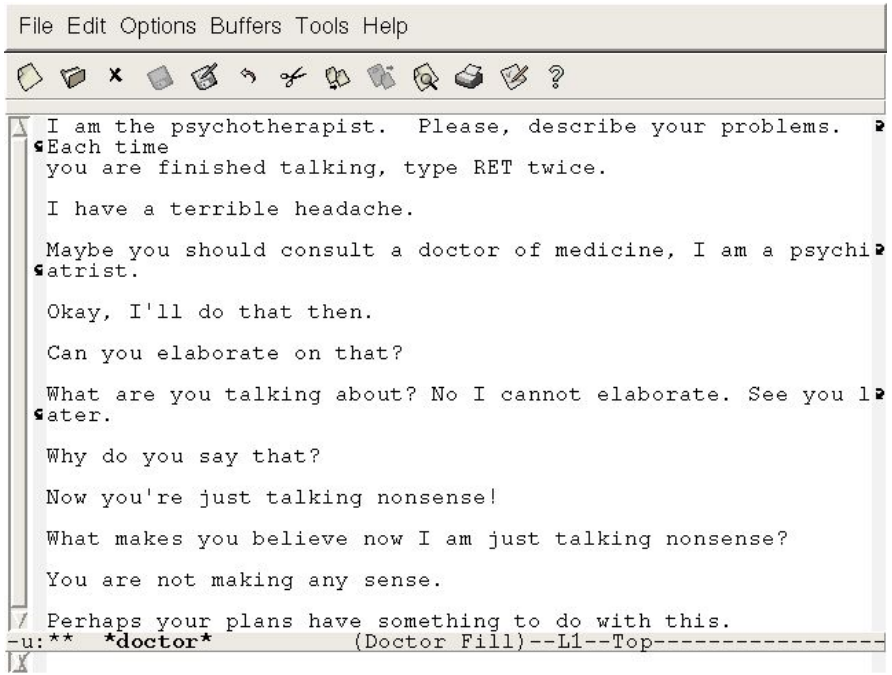
Introduction to Gen IA

L’IA generative

L'IA générative est une branche de l'intelligence artificielle qui utilise des modèles de deep learning pour créer du contenu original à partir de données existantes.

Ces modèles, souvent des réseaux neuronaux profonds, sont entraînés sur de vastes ensembles de données pour identifier des schémas et générer des textes, images, vidéos, ou autres formes de contenu

1966 ELIZA:


File Edit Options Buffers Tools Help
I am the psychotherapist. Please, describe your problems.
Each time
you are finished talking, type RET twice.
I have a terrible headache.
Maybe you should consult a doctor of medicine, I am a psychi
atrist.
Okay, I'll do that then.
Can you elaborate on that?
What are you talking about? No I cannot elaborate. See you l
ater.
Why do you say that?
Now you're just talking nonsense!
What makes you believe now I am just talking nonsense?
You are not making any sense.
Perhaps your plans have something to do with this.
-u:** *doctor* (Doctor Fill)--L1--Top----------

2005 : SCIgen

An Automatic CS Paper Generator :

SCIgen est un générateur de faux articles scientifiques utilisant une grammaire sans contexte pour créer des textes absurdes imitant des recherches en informatique.

Créé en 2005 par des chercheurs du MIT pour dénoncer le manque de rigueur des conférences, il génère des articles complets avec graphiques et citations.

Utilisé principalement par des universitaires chinois pour soumettre des articles frauduleux, il a conduit à la rétractation de 122 articles et à la création de logiciels de détection.

2017: Google Révolutionne tout

Google introduit le premier **Transformer**, qui va devenir rapidement la référence pour traiter les sujets de NLP.

GPT -> T stand for transformer.

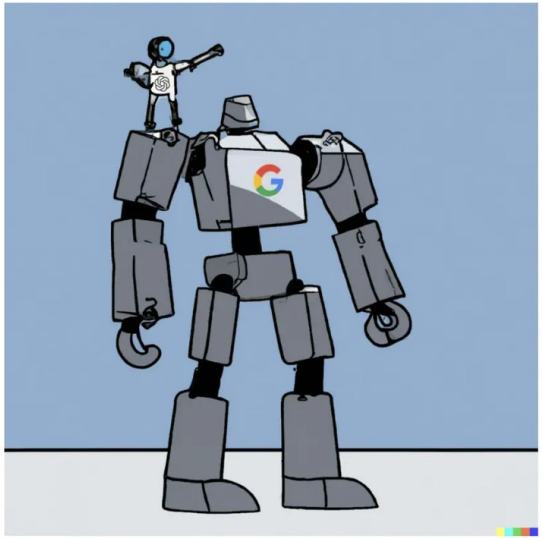
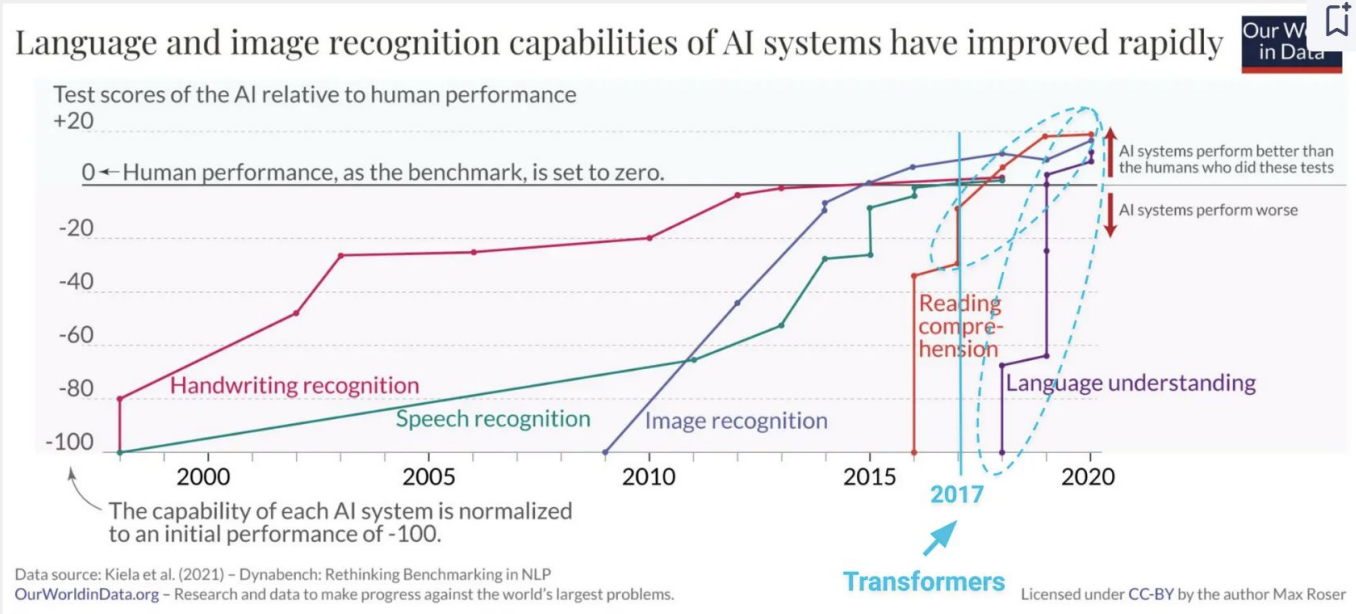


Image generated with DALL.E: "A small robot standing on the shoulder of a giant robot" (and slightly modified with The Gimp)

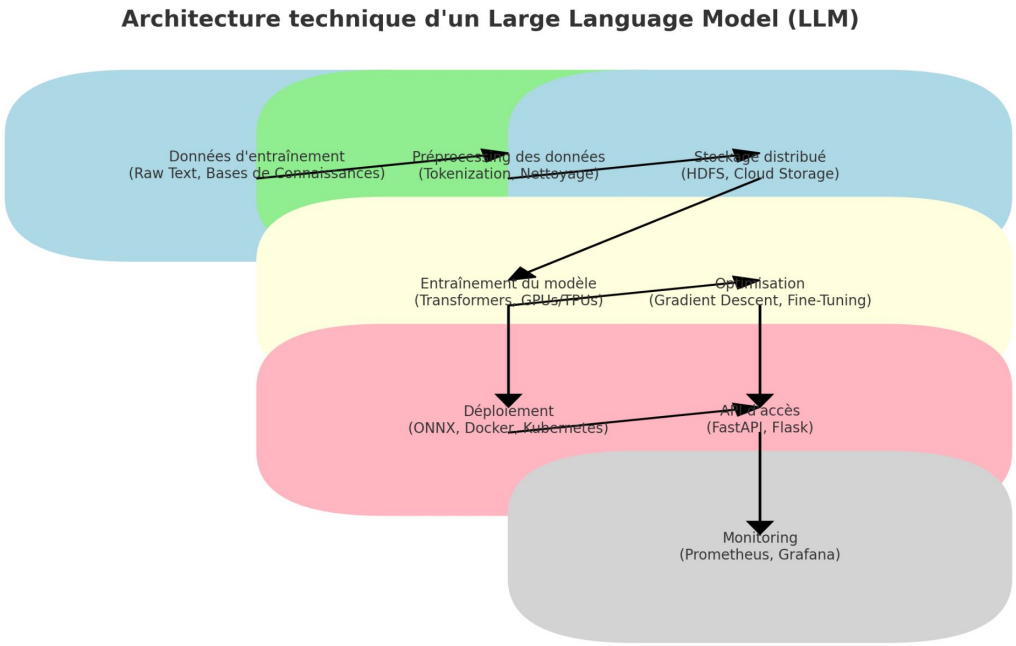
Rapidité de la progression de l'IA gen



- Kiela et al. (2021), Dynabench: Rethinking Benchmarking in NLP: arxiv.org/abs/2104.14337
- Roser (2022), The brief history of artificial intelligence: The world has changed fast – what might be next?: ourworldindata.org/brief-history-of-ai

Text and shapes in blue have been added to the original work from Max Roser.

Architecture de l’IA gen



Les transformers

Les Transformers sont des modèles d'intelligence artificielle basés sur une architecture de deep learning qui utilise des mécanismes d'attention pour traiter et générer des séquences de données, comme du texte ou des images.

Fonctionnement :

- **Mécanisme d'attention** : Permet au modèle de se concentrer sur différentes parties de l'entrée pour mieux comprendre le contexte et les relations entre les éléments.
- **Architecture** : Composée de couches empilées de transformateurs, incluant des encodeurs et des décodeurs, qui permettent de traiter les séquences de manière parallèle et efficace.

Les transformers

Applications :

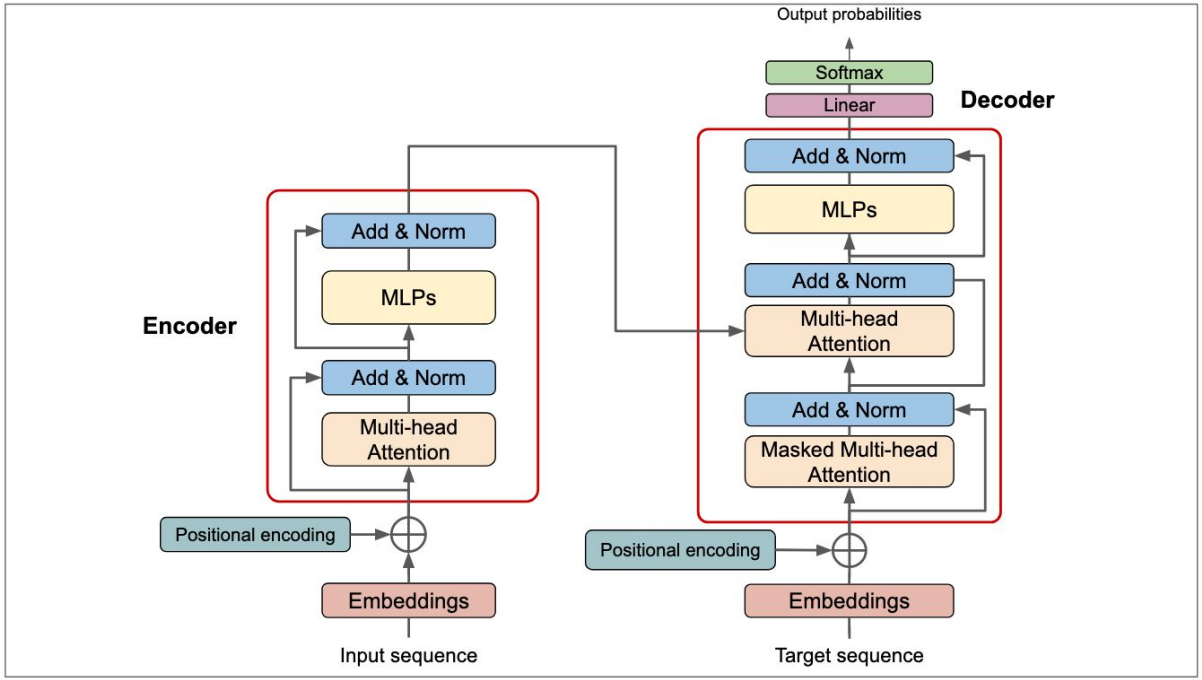
- **Traitement du langage naturel (NLP)** : Traduction automatique, génération de texte, chatbots.
- **Vision par ordinateur** : Reconnaissance d'images, génération d'images.
- **Autres domaines** : Bioinformatique, génération de musique, analyse de séries temporelles.

Les transformers

Avantages :

- **Efficacité** : Capables de traiter de grandes quantités de données en parallèle.
- **Polyvalence** : Utilisables dans divers domaines et pour différentes tâches.

Architecture Transformers



Le LLM

Large Language Models (LLM)

Définition :

- Les LLM sont des modèles d'intelligence artificielle qui utilisent des réseaux neuronaux profonds pour traiter et générer du langage naturel. Ils sont entraînés sur de vastes ensembles de données textuelles pour comprendre et produire du texte de manière cohérente et contextuelle.

Le LLM

Large Language Models (LLM)

Fonctionnement :

- **Entraînement** : Formés sur des milliards de mots, les LLM apprennent les schémas et les relations linguistiques.
- **Architecture** : Utilisent des architectures avancées comme les Transformateurs (Transformers) pour gérer de grandes quantités de données et de paramètres.
- **Génération** : Capables de produire du texte, traduire des langues, répondre à des questions, et bien plus encore.

Le LLM

Large Language Models (LLM)

Applications :

- **Traitement du langage naturel (NLP)** : Chatbots, assistants virtuels, analyse de sentiments.
- **Création de contenu** : Rédaction d'articles, génération de code, création artistique.
- **Automatisation** : Automatisation de tâches complexes, aide à la décision.

Le LLM

Large Language Models (LLM)

Avantages :

- **Polyvalence** : Peuvent être appliqués à une variété de tâches linguistiques.
- **Efficacité** : Améliorent la productivité et l'efficacité dans divers domaines.

Défis :

- **Ressources** : Nécessitent des ressources computationnelles importantes pour l'entraînement et l'inférence.
- **Éthique** : Risques de biais et de désinformation.

La big Picture des LLM

Google

PaML (540b), **LaMDA** (137b) and others (Bard relies on LaMDA)

Meta

OPT-IML (175b), **Galactica** (120b), **BlenderBot3** (175b), **Llama 2** (70b)

OpenAI

GPT-3 (175b), **GPT-3.5** (?b), **GPT-4** (?b)

Baidu 百度

ERNIE 3.0 Titan (260b)

BigScience BLOOM

BLOOM (176b)

HUAWEI

PanGu-α (200b)

NVIDIA

Megatron-Turing NLG (530b)

LG AI Research

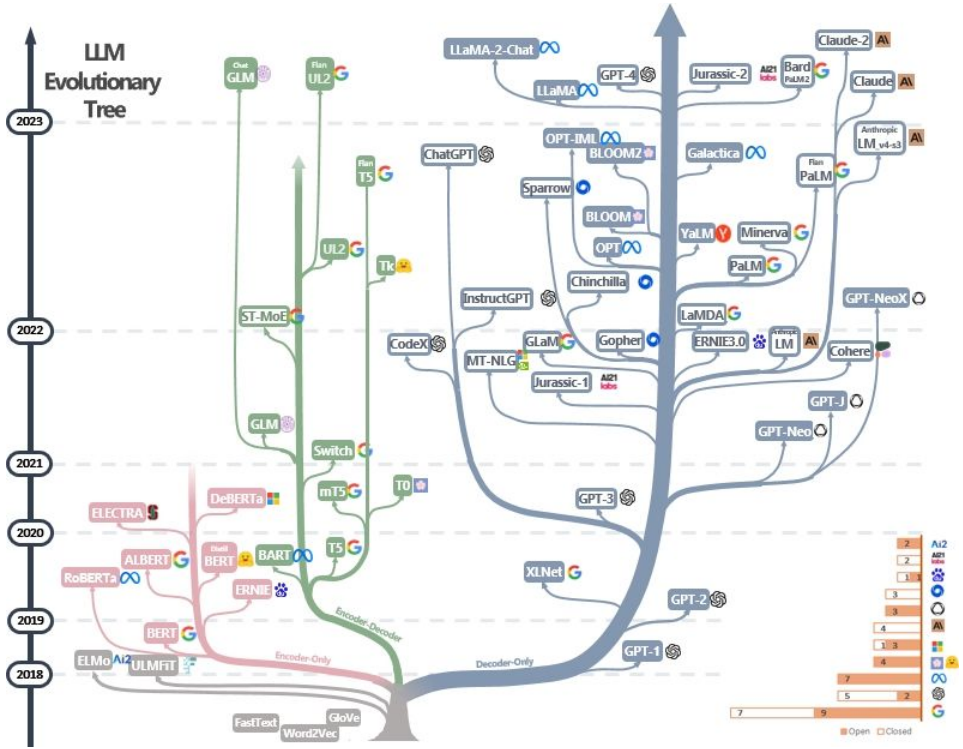
Exaone (300b)

AI21labs

Jurassic-1 (178b), **Jurassic-2** (?b)

(It appears that all those models rely only on transformer-based decoders)

La big Picture des LLM



Limitations of the GEN IA : Hallucinations

Les hallucinations en IA générative se produisent lorsque le modèle génère des informations incorrectes ou fictives, présentées de manière convaincante comme des vérités.

Causes :

- **Données d'entraînement insuffisantes** : Manque de données pertinentes ou biaisées.
- **Modèles complexes** : Difficulté à interpréter et à vérifier les sorties du modèle.
- **Génération probabiliste** : Le modèle prédit les mots suivants basés sur des probabilités, ce qui peut entraîner des erreurs.

Exemples :

- Un chatbot affirmant faussement que le télescope spatial James Webb a capturé des images d'une planète en dehors de notre système solaire.
- Un assistant virtuel générant des faits historiques incorrects ou des citations inventées.

Limitations of the GEN IA :
Non-déterministe et bias d'entraînement

L'IA générative peut produire des résultats différents même avec les mêmes entrées, ce qui peut entraîner une imprévisibilité dans les résultats.

Les modèles peuvent reproduire et amplifier les biais présents dans les données d'entraînement, ce qui peut entraîner des résultats discriminatoires ou injustes

Limitations of the GEN IA : Le phénomène de Drift

Le drift en IA se réfère à la dégradation progressive des performances d'un modèle au fil du temps en raison de changements dans les données ou l'environnement.

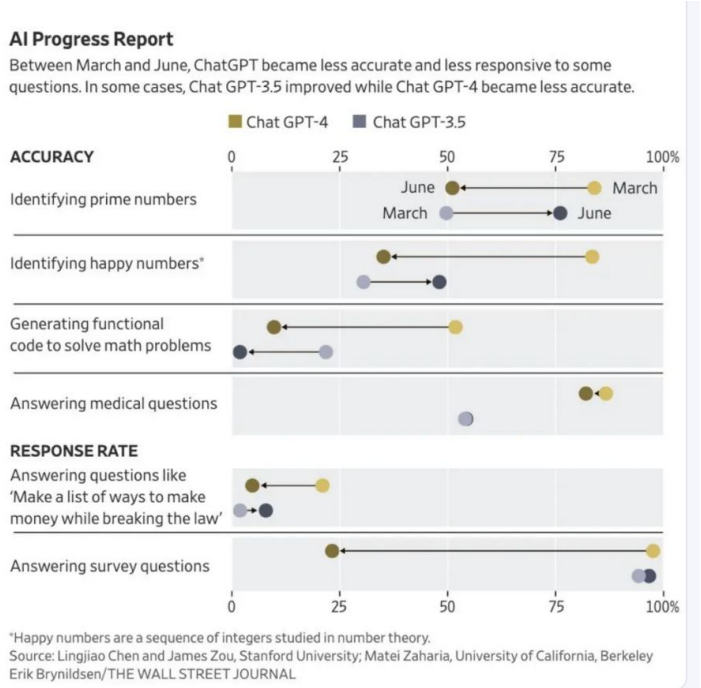
Causes :

- **Évolution des données** : Les données d'entraînement deviennent obsolètes ou ne reflètent plus les nouvelles tendances.
- **Changements dans l'environnement** : Modifications des contextes d'utilisation, des préférences des utilisateurs ou des normes linguistiques.
- **Biais accumulés** : Amplification des biais présents dans les données d'entraînement.

Conséquences :

- **Précision réduite** : Diminution de la pertinence et de l'exactitude des prédictions du modèle.
- **Décisions erronées** : Risque accru de prises de décisions basées sur des informations incorrectes.
- **Perte de confiance** : Réduction de la confiance des utilisateurs dans les systèmes d'IA.

Limitations of the GEN IA : Le phénomène de Drift



02

ML Ops

Qu’est-ce que le MLOps ?

MLOps (Machine Learning Operations) désigne un ensemble de pratiques et de processus qui combinent le Machine Learning (ML) et les principes de DevOps pour automatiser et améliorer la gestion des modèles d’IA tout au long de leur cycle de vie.

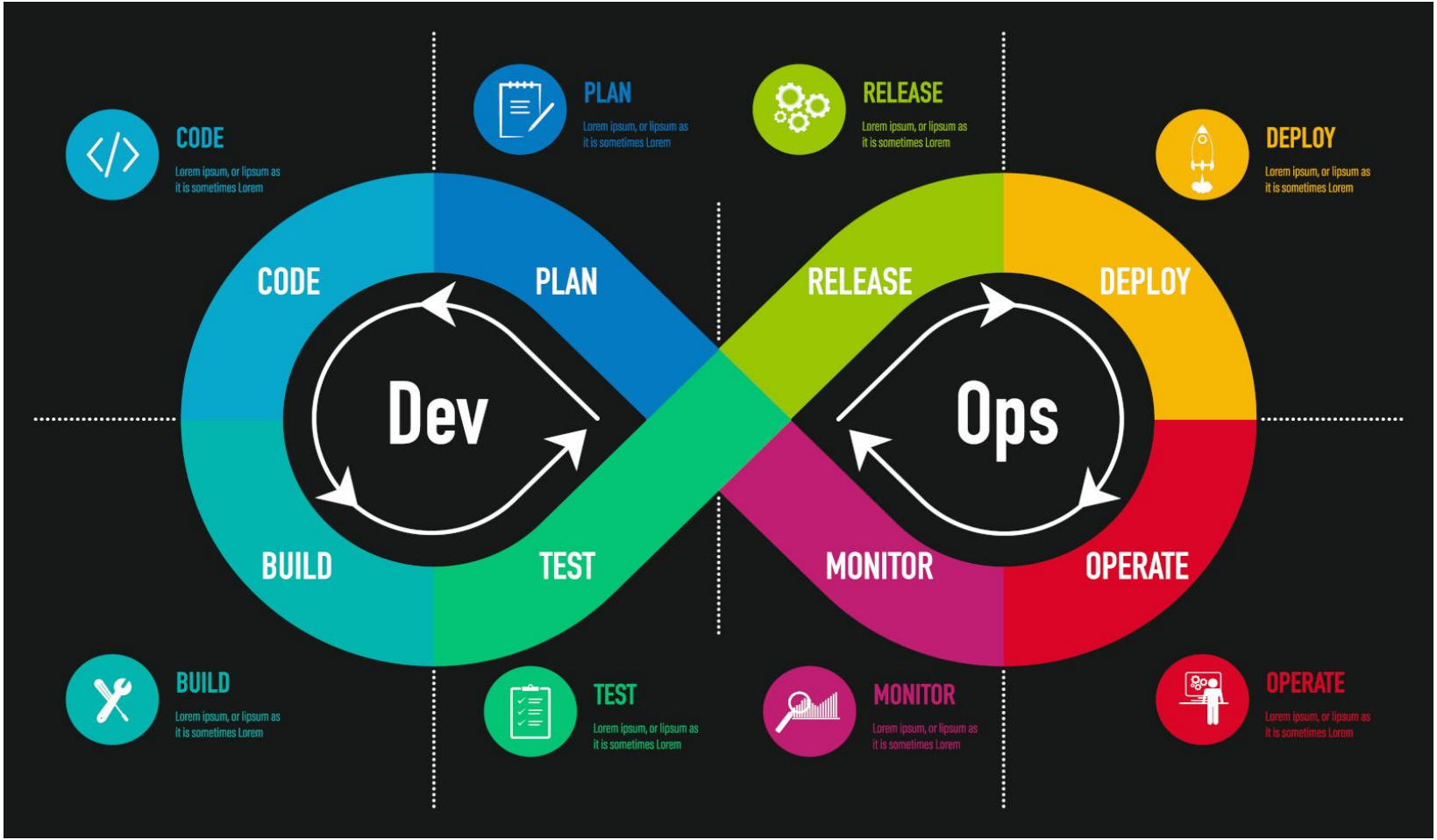
Objectif : rendre les modèles ML **scalables**, **reproductibles**, et **fiables** en production.

Pourquoi le MLOps est essentiel ?

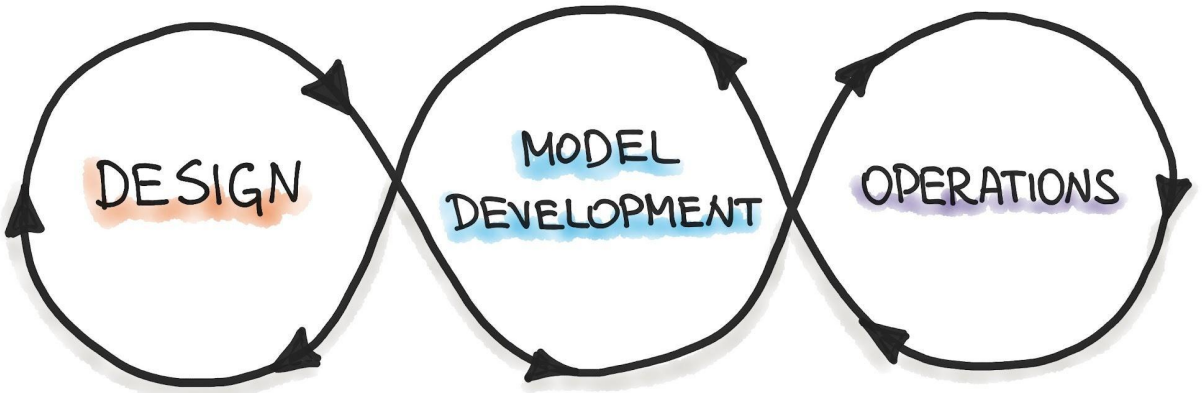
Contexte industriel : Les modèles ML sont souvent créés dans des environnements isolés (ex : notebooks). Sans un pipeline structuré, il est difficile de passer de la phase de prototypage à la production.

Défis :

- Gestion des données massives (Big Data).
- Maintenance des modèles en production.
- Collaboration entre équipes de Data Scientists, Data Engineers, et DevOps.
- Possibilité de réentrainement à chaud en continu


CODE
Lorem ipsum, or lipsum as it is sometimes Lorem
PLAN
Lorem ipsum, or lipsum as it is sometimes Lorem
RELEASE
Lorem ipsum, or lipsum as it is sometimes Lorem
DEPLOY
Lorem ipsum, or lipsum as it is sometimes Lorem
CODE
PLAN
RELEASE
DEPLOY
Dev
Ops
BUILD
TEST
MONITOR
OPERATE
BUILD
Lorem ipsum, or lipsum as it is sometimes Lorem
TEST
Lorem ipsum, or lipsum as it is sometimes Lorem
MONITOR
Lorem ipsum, or lipsum as it is sometimes Lorem
OPERATE
Lorem ipsum, or lipsum as it is sometimes Lorem

MLOps



- Requirements Engineering
- ML Use-Cases Priorization
- Data Availability Check

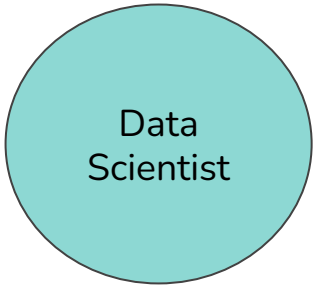
- Data Engineering
- ML Model Engineering
- Model Testing & Validation

- ML Model Deployment
- CI/CD Pipelines
- Monitoring & Triggering

Différence entre le DevOps et le MLOps ?

DevOps	MLOps
Gestion du cycle de vie logiciel	Gestion du cycle de vie des modèles ML.
CI/CD pour déploiement rapide	CI/CD + **monitoring des performances des modèles**
Versionnage du code	Versionnage du code, **des données, et des modèles**
Logs et monitoring d'applications	Logs, monitoring, et **gestion des dérives des modèles (data drift)**

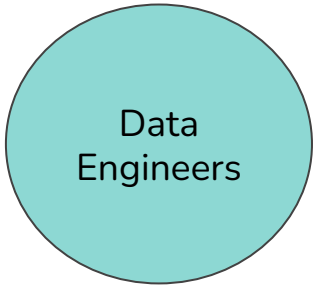
Acteurs principaux



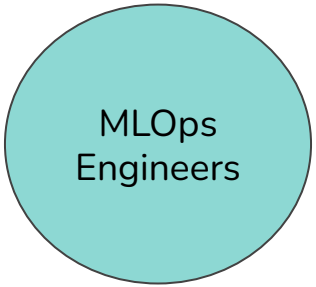
développement et

expérimentation de modèles

gestion des pipelines de données et mise en place des infrastructures Big Data.



intégration des modèles dans des environnements scalables et automatisation des workflows



Contexte big data

- **Volume** : Gérer des datasets massifs qui ne tiennent pas en mémoire (ex : logs, clicstreams).
- **Vélocité** : Traiter des flux de données en temps réel (Kafka, Spark Streaming).
- **Variété** : Manipuler des formats hétérogènes (images, textes, JSON, Parquet).
- **Variabilité** : Garantir que les modèles restent performants malgré des données qui évoluent dans le temps (concept drift).

Volume

Description

- Les projets Big Data nécessitent de manipuler des datasets bien plus volumineux que ce que peut gérer un ordinateur individuel.

- Les données peuvent provenir de multiples sources (logs web, capteurs IoT, transactions financières, etc.) et s'accumuler rapidement.

Impact sur le MLOps

- Les pipelines doivent être conçus pour traiter des données stockées de manière distribuée (ex : Hadoop Distributed File System - HDFS).

- Les étapes d'entraînement des modèles nécessitent un traitement distribué (par ex : Apache Spark MLlib ou TensorFlow on Kubernetes).

Vélocité

Description

- Les données sont souvent générées en flux continus et nécessitent une ingestion et un traitement en temps réel.

- Exemples : flux de clics (web), transactions bancaires, données de capteurs dans des usines ou des villes connectées.

Impact sur le MLOps

- Les pipelines doivent intégrer des outils de streaming comme Kafka ou Spark Streaming.

- L’évaluation et la mise à jour des modèles doivent être capables de fonctionner en quasi-temps réel (low-latency).

Variété

Description

- Les données peuvent être structurées (tables SQL), semi-structurées (JSON, XML), ou non structurées (textes, images, vidéos).

- Par exemple, un projet de classification d'articles e-commerce peut devoir analyser à la fois des descriptions textuelles et des images des produits.

Impact sur le MLOps

- Les pipelines doivent être capables de gérer plusieurs types de données simultanément.

- Les étapes de preprocessing doivent être adaptées au format de chaque type de données (ex : NLP pour textes, transformation FFT pour signaux audio).

Variabilité

Description

- Les données évoluent dans le temps, ce qui peut entraîner des problèmes de **concept drift** (le modèle devient obsolète face à des données qui changent).

- Exemple : un modèle de recommandation peut devenir moins performant à mesure que les préférences des utilisateurs changent.

Impact sur le MLOps

- Nécessité d'implémenter des systèmes de monitoring pour détecter les dérives de données.

- Mise en place d'un pipeline automatisé pour réentraîner régulièrement les modèles.

Exemple d’applications ML avec Big Data

E-commerce : systèmes de recommandation

- **Données**
  - Historique d’achats, clic streams (flux de navigation), avis des clients, informations sur les produits.

- **Défis**
  - Traiter en temps réel des flux d’interactions utilisateurs pour adapter les recommandations dynamiques.
  - Gérer les datasets massifs (plusieurs millions d’articles et clients).

- **Solution typique**
  - Spark MLlib pour la phase d’entraînement, Kafka pour l’ingestion des flux, et Redis pour servir les recommandations en temps réel.

Exemple d’applications ML avec Big Data

Finance : détection de fraude

- **Données**
  - Transactions bancaires, logs des connexions utilisateurs, données démographiques.

- **Défis**
  - Identifier des comportements frauduleux dans des millions de transactions en temps réel.
  - Gérer des ensembles de données très déséquilibrés (très peu de fraudes par rapport aux transactions normales)

- **Solution typique**
  - Spark Streaming pour l’analyse en temps réel, entraînement d’un modèle supervisé avec MLflow pour gérer les versions.

Exemple d’applications ML avec Big Data

Santé : modèles prédictifs

- **Données**
  - Dossiers médicaux électroniques, images médicales, données de capteurs connectés.

- **Défis**
  - Fusionner des données provenant de sources variées tout en respectant les réglementations sur la confidentialité (ex : GDPR, HIPAA).
  - Traiter des images massives ou des signaux en temps réel.

- **Solution typique**
  - Hadoop ou Amazon S3 pour stocker les données massives, TensorFlow pour entraîner des modèles DL.

Outils couramment utilisés dans un contexte Big Data

Ingestion des données

- **Apache Kafka**
  - Plateforme de messagerie distribuée.
  - Idéal pour l'ingestion de flux de données en temps réel.
  - Utilisé pour connecter des sources de données hétérogènes à des pipelines ML.

- **Apache NiFi**
  - Permet de concevoir des pipelines de données flexibles avec des connecteurs prêts à l'emploi.
  - Utile pour orchestrer des flux entre différents systèmes (bases de données, services cloud).

Outils couramment utilisés dans un contexte Big Data

Traitement et stockage

- **Apache Spark :**
  - Framework pour le traitement distribué.
  - Inclut des bibliothèques ML (Spark MLlib) et des API pour le traitement des données en batch ou en streaming.

- **Hadoop / Hive**
  - Hadoop : stockage distribué des données.
  - Hive : interface SQL pour interroger des données massives stockées dans HDFS.

Outils couramment utilisés dans un contexte Big Data

Analyse et monitoring

- **Elasticsearch**
  - Moteur de recherche distribué rapide.
  - Peut être utilisé pour indexer et rechercher des logs ou des métadonnées.

- **Prometheus / Grafana**
  - Prometheus : collecte et gestion des métriques en temps réel.
  - Grafana : visualisation des métriques pour le monitoring des performances.

Liens entre BigData et MLOps

Collecte des données à grande échelle

- Les workflows MLOps doivent commencer par des pipelines robustes pour collecter et stocker les données massives.

- Exemple : utiliser Kafka pour l’ingestion, Spark pour le nettoyage et la transformation, et Hadoop/S3 pour le stockage.

Liens entre BigData et MLOps

Entraînement distribué

- Les frameworks comme TensorFlow et PyTorch doivent s’intégrer avec Spark ou Kubernetes pour gérer des volumes massifs de données.

- Les modèles peuvent être entraînés sur des clusters avec GPUs pour accélérer les calculs.

Liens entre BigData et MLOps

Monitoring des pipelines ML

- Environnements Big Data nécessitent un monitoring constant pour détecter des pannes ou des dérives de données.

- Les outils comme MLflow ou Neptune.ai permettent de suivre les versions des modèles et de monitorer leurs performances.

En résumé

- Les défis du Big Data (volume, vélocité, variété, variabilité) imposent des contraintes spécifiques aux pipelines MLOps.

- Les outils comme Kafka, Spark, et Hadoop s’intègrent dans les pipelines pour gérer les données massives.

- Les modèles ML doivent être conçus pour évoluer avec les données tout en restant performants.

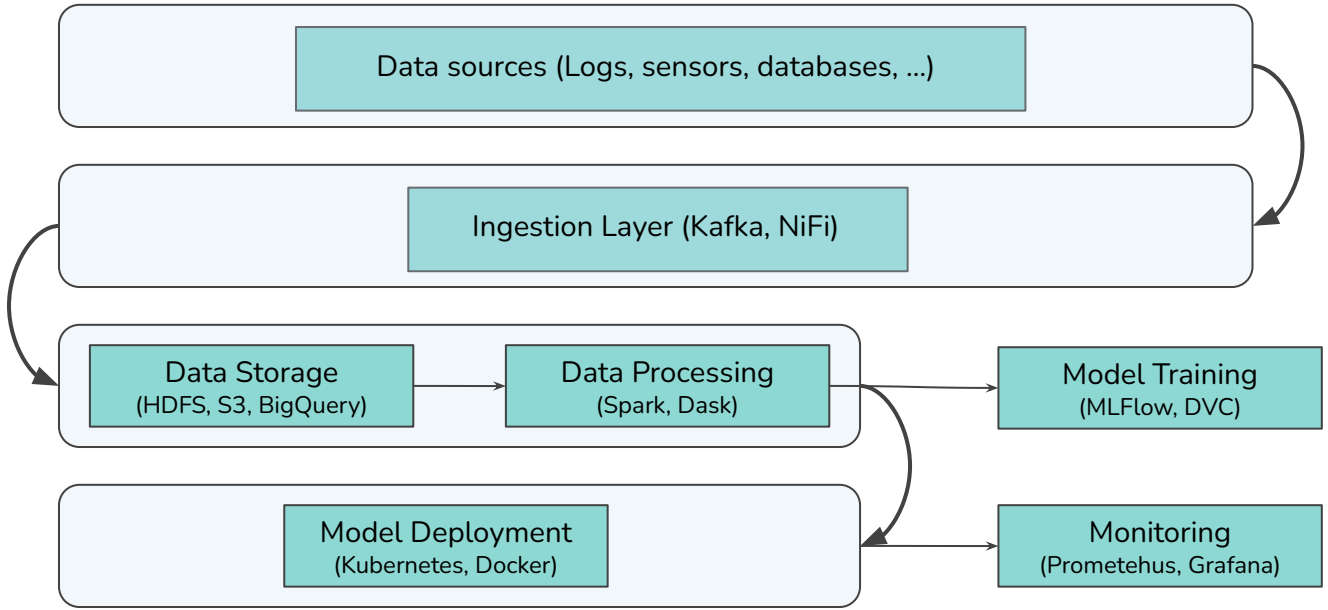
Exemple d’un workflow ML appliqué au Big Data

Phase 1 **data collection**

Phase 2 **data ingestion**

Phase 3 **data storage**

Phase 4 **model lifecycle**



Bonnes pratiques pour le data scientist

Écriture de code pour le MLOps

Structuration des projets

- Utilisation de standards pour organiser les projets afin de garantir la clarté, la réutilisabilité et la collaboration.

- Organisation recommandée

```
├── data/
│   ├── raw/            # Données brutes
│   ├── processed/      # Données transformées
├── notebooks/          # Notebooks pour les explorations
├── src/                # Scripts principaux
│   ├── data/           # Scripts de preprocessing
│   ├── models/         # Scripts d'entraînement
│   └── utils/          # Fonctions utilitaires
├── tests/              # Scripts de test
├── Dockerfile          # Définition de l'image Docker
└── README.md           # Documentation du projet
```

Bonnes pratiques pour le data scientist

Écriture de code pour le MLOps

Notebooks reproductibles et scripts batch

- Les notebooks sont utiles pour la phase d'expérimentation mais doivent être **reproductibles** :
- Versionner les notebooks avec **nbstripout** (enlever les outputs avant commit).
- Sauvegarder les états intermédiaires (modèles, métriques) pour faciliter la reprise.
- Les scripts batch sont essentiels pour les pipelines en production :
- Convertir les notebooks validés en scripts modulaires (ex : train_model.py).
- Utiliser des outils comme **Papermill** pour automatiser l'exécution des notebooks.

Bonnes pratiques pour le data scientist

Écriture de code pour le MLOps

Utilisation d'environnements virtuels

- Créer des environnements isolés pour éviter les conflits de dépendances :
- **Poetry**: Gère les dépendances et les versions de Python.
- **Docker** : Fournit des conteneurs légers pour packager les environnements de manière standardisée.

Exemple Dockerfile basique

dockerfile

```
FROM python:3.9
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
CMD ["python", "src/main.py"]
```

Bonnes pratiques pour le data scientist

Versionning

Versionning du code avec Git

- Suivre les bonnes pratiques Git
- Utiliser une convention de noms pour les branches (**main**, **dev**, **feature/**).
- Faire des commits fréquents et descriptifs.
- Intégrer des outils de revue de code (ex : **GitHub Actions** pour la CI/CD).

Bonnes pratiques pour le data scientist

Versionning

Versionning des données et des modèles

- Les données et les modèles évoluent au fil du temps, nécessitant un versionnage.

MLflow

- Gère le tracking des expériences (hyperparamètres, métriques, modèles).
- Exemple de workflow :

```python
import mlflow

with mlflow.start_run():
    mlflow.log_param("learning_rate", 0.01)
    mlflow.log_metric("accuracy", 0.95)
    mlflow.log_artifact("model.pkl")
```

DVC (Data Version Control)

- Suit les versions des fichiers volumineux (données, modèles).
- Intègre des pipelines reproductibles via des fichiers YAML.

Bonnes pratiques pour le data scientist

Testing dans un pipeline IA

Tests unitaires pour le preprocessing

- Vérifier que les transformations des données fonctionnent correctement :
- Test d’intégrité des données (valeurs manquantes, doublons).
- Test des pipelines (par ex : vérifier que l’encodage de variables catégoriques donne le résultat attendu).

Bonnes pratiques pour le data scientist

Testing dans un pipeline IA

Tests de performance et de qualité des modèles

- Définir des tests spécifiques pour évaluer les modèles
- Métriques de base : précision, rappel, F1-score, AUC.
- Tests de robustesse : vérifier la performance sur des jeux de données bruitées.
- Comparaison avec des benchmarks (modèles précédents).

Bonnes pratiques pour le data scientist

Testing dans un pipeline IA

Utilisation de CI/CD pour les pipelines MLOps

- Intégrer les tests dans un pipeline CI/CD pour automatiser les vérifications à chaque modification :

- Exemple avec **GitHub Actions** :

Ces pratiques peuvent paraîtres plus coûteuses en temps… MAIS elles évitent les régressions et les problèmes lors de développements futurs

yaml

```yaml
name: CI for ML
on: [push]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Set up Python
        uses: actions/setup-python@v2
        with:
          python-version: 3.9
      - name: Install dependencies
        run: pip install -r requirements.txt
      - name: Run tests
        run: pytest
```

Bonnes pratiques pour le data scientist

Réentraînement et monitoring

Détection des dérives de données

- Surveiller les distributions de données pour détecter des changements (concept drift) :
- Exemple : Histogramme des valeurs de features, comparaison avec les distributions historiques.
- Outils : **Evidently AI**, **WhyLabs**.

Bonnes pratiques pour le data scientist

Réentraînement et monitoring

Automatisation du réentraînement

- Réentraînement basé sur des triggers (ex : détection de dérives, nouveaux jeux de données) :
  - **Airflow** ou **Kubeflow** pour orchestrer les workflows.

- Exemple de DAG avec Airflow :

```python
from airflow import DAG
from airflow.operators.python_operator import PythonOperator

def retrain_model():
    # Code de réentraînement ici
    pass

dag = DAG('retrain_pipeline', schedule_interval='@weekly')
retrain_task = PythonOperator(task_id='retrain', python_callable=retrain_model, dag=dag)
```

Bonnes pratiques pour le data scientist

Stratégies de déploiement

Différentes approches pour minimiser les risques lors du déploiement de nouveaux modèles :

- **Shadow Deployment** : Le nouveau modèle fonctionne en parallèle sans affecter les utilisateurs.
- **A/B Testing** : Les utilisateurs sont divisés en groupes, chaque groupe utilisant un modèle différent.
- **Canary Release** : Déploiement progressif sur une fraction des utilisateurs avant une mise en production complète.

03

Retrieval Augmented Generative Concepts

Qu’est ce qu’un RAG ?

Définition et principes

Retrieval-Augmented Generation (RAG) est une méthode d’intelligence artificielle combinant deux étapes principales :

1. Retrieval : Recherche d’informations pertinentes à partir d’une base de connaissances ou d’une source externe.
2. Generation : Utilisation d’un modèle génératif pour produire une réponse enrichie à partir des informations récupérées.

- Objectif : améliorer la précision et la pertinence des réponses générées par des modèles tout en limitant les hallucinations (générations incorrectes ou hors-sujet).

Principe fondamental

- Contrairement à un modèle génératif qui s’appuie uniquement sur ses connaissances internes (apprises durant l’entraînement), un système RAG interroge une **source de données externe** pour **enrichir sa réponse en temps réel**.

Différence avec les approches classiques de NLP

Aspect	Approche Classique (Generative)	RAG
Dépendance aux données	Modèle autonome, dépend uniquement de l'entraînement initial	Récupère des données externes à la demande pour augmenter ses réponses
Flexibilité	Limité aux données vues pendant l'entraînement	Peut s'adapter en temps réel à des données externes mises à jour
Performance	Risque d'hallucinations si les données ne sont pas dans le modèle	Réponses plus précises grâce à l'enrichissement externe
Exemples d'utilisation	Chatbots classiques, traduction automatique	Chatbots augmentés, assistants juridiques, recherche scientifique

Fonctionnement d’un système RAG

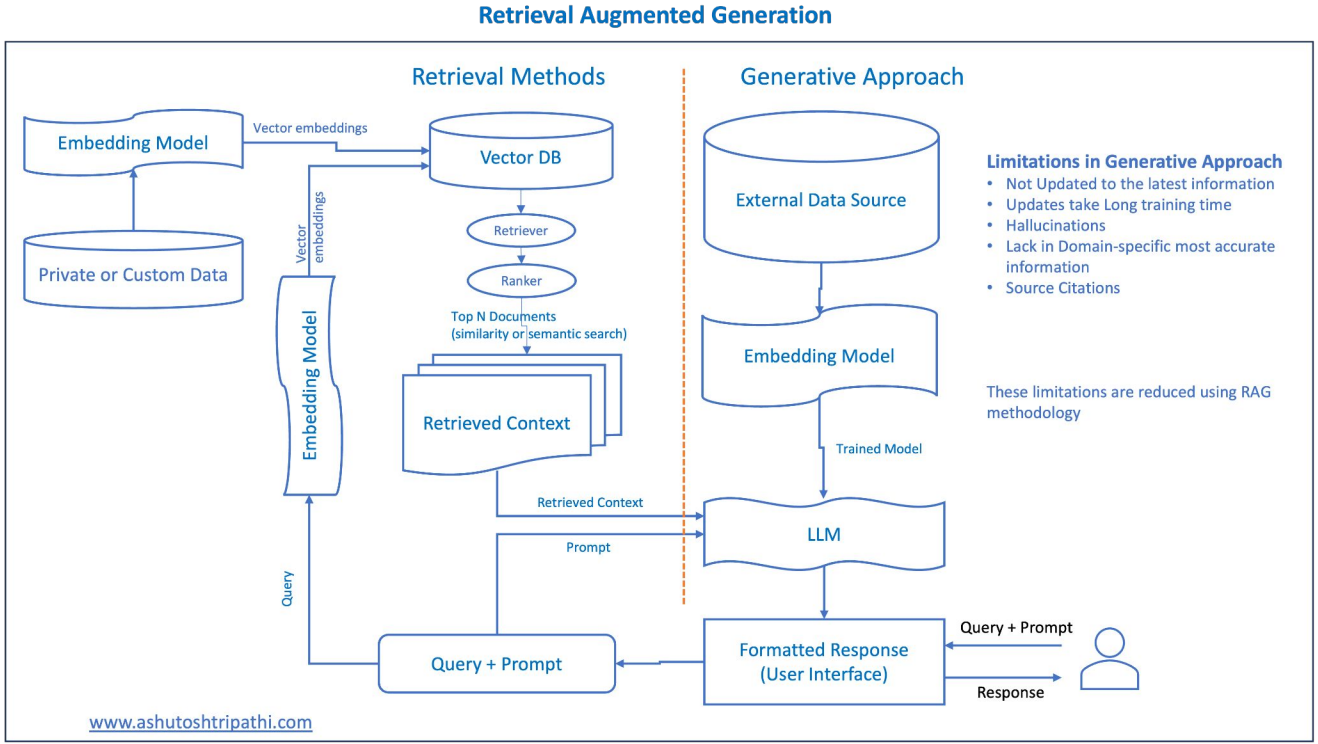
Étape 1 : Recherche d’information (retrieval)

- **Objectif** : Identifier des documents ou informations pertinentes à partir d’une base de connaissances.
- **Approches courantes**
  - Dense Retrieval :
    - Utilise des vecteurs pour représenter les documents et les requêtes.
    - Outils : FAISS (Facebook AI Similarity Search) pour l’indexation et la recherche rapide.
  - Sparse Retrieval :
    - Basé sur des techniques traditionnelles de recherche (ex : BM25).
    - Outils : Elasticsearch, Lucene.

- **Processus typique**

1. Une requête utilisateur est transformée en un vecteur ou en une structure compréhensible pour le système.
2. Le système interroge une base de connaissances pour récupérer les informations les plus pertinentes.
3. Les informations retournées sont utilisées comme contexte pour la génération.

Architecture d'un RAG



Etape 1 - Exemple

python

```python
import faiss
import numpy as np

# Création d'un index pour une base de données
d = 128  # Dimension des vecteurs
index = faiss.IndexFlatL2(d)
vectors = np.random.random((1000, d)).astype('float32')
index.add(vectors)

# Recherche
query_vector = np.random.random((1, d)).astype('float32')
D, I = index.search(query_vector, k=5)  # Trouver les 5 plus proches
print("Indices des documents pertinents :", I)
```

Fonctionnement d’un système RAG

Étape 2 : Génération de réponses augmentées

- **Objectif** : Créer une réponse en combinant les informations récupérées et la capacité de génération du modèle

- **Fonctionnement**

1. Les informations récupérées sont utilisées comme contexte pour le modèle génératif.
2. Un modèle (par ex. GPT) est utilisé pour générer une réponse complète en langage naturel.

Etape 2 : Exemple avec Hugging Face Transformers

python

```python
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM

tokenizer = AutoTokenizer.from_pretrained("t5-base")
model = AutoModelForSeq2SeqLM.from_pretrained("t5-base")

# Contexte récupéré par le système de retrieval
retrieved_context = "Python est un langage de programmation populaire."

# Génération augmentée
input_text = f"Contexte: {retrieved_context} Question: Qu'est-ce que Python?"
inputs = tokenizer(input_text, return_tensors="pt")
outputs = model.generate(**inputs)
print(tokenizer.decode(outputs[0], skip_special_tokens=True))
```

Intégration dans une chaîne MLOps

Collecte et versionning des données de recherche

- **Importance** : Les bases de connaissances évoluent dans le temps. Les systèmes RAG doivent intégrer des données récentes et versionnées.

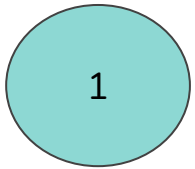
- **Pratiques recommandées** :
  - Stocker les documents dans un système versionné (ex : DVC, MLflow Artifacts).
  - Indexer les données avec des outils comme FAISS ou Elasticsearch.
  - Automatiser les mises à jour de l'index avec des workflows (ex : Airflow, Dagster).

Intégration dans une chaîne MLOps

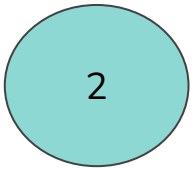
Entraînement et fine-tuning des modèles génératifs

- Les modèles génératifs doivent être adaptés au domaine spécifique (fine-tuning) :
  - Exemple : Fine-tuning d'un modèle GPT sur des questions-réponses médicales.
  - Utilisation d'outils comme Hugging Face Trainer pour gérer le fine-tuning.

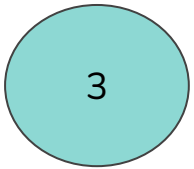
- **Pipeline typique**



Préparation des données d'entraînement : Contexte + Question → Réponse



Fine-tuning du modèle génératif



Évaluation sur un jeu de validation (précision des réponses, cohérence).

Intégration dans une chaîne MLOps

Déploiement et monitoring dans un contexte Big Data

- **Déploiement**
  - Les systèmes RAG peuvent être conteneurisés avec Docker et orchestrés via Kubernetes.
  - Utilisation d'APIs (FastAPI) pour permettre un accès en temps réel.

- **Monitoring**
  - Surveillance des performances du retrieval (précision des résultats).
  - IMesure de la qualité des réponses générées (feedback utilisateurs).
  - Outils : Prometheus, Grafana, ou outils spécifiques comme Evidently AI.

En résumé

1. RAG combine retrieval et génération pour fournir des réponses plus précises et adaptées aux contextes dynamiques.

2. Le retrieval et la génération sont deux étapes complémentaires : une pour chercher, l'autre pour répondre.

3. L'intégration dans une chaîne MLOps nécessite des workflows robustes pour gérer l'évolution des données et la performance des modèles.

04

Retrieval Augmented Generative Practical

Task 1