Generative Artificial Intelligence Design, Exploitation & Ethics - IaC for Cloud Orchestration

Cours complet sur la conception, l'exploitation et l'éthique de l'intelligence artificielle générative dans le contexte de l'orchestration cloud. Couvre les architectures, le déploiement, la gouvernance et les considérations éthiques.

Intelligence Artificielle **180 min de lecture** **Niveau Avancé**

Document généré le 11/07/2026 à 20h44 · nouvy.fr/wiki/generative-ai-design-exploitation-ethics-cloud-orchestration

Sommaire

36 section(s) · 180 min de lecture

IaC for Cloud Orchestration

Ce cours complet vous fournira une compréhension approfondie de l'intelligence artificielle générative (Generative AI) dans le contexte de l'orchestration cloud. Vous apprendrez à concevoir, déployer, exploiter et gérer des systèmes d'IA générative de manière éthique et responsable.

□ Table des matières

1□ Introduction à l'IA générative

1.1 Qu'est-ce que l'IA générative ?

L'**Intelligence Artificielle Générative (Generative AI)** est une branche de l'IA capable de créer du contenu nouveau et original à partir de données d'entraînement. Contrairement aux modèles d'IA traditionnels qui classifient ou prédisent, les modèles génératifs produisent :

- □ **Texte** : Articles, code, traductions, résumés
- □ **Images** : Créations artistiques, photoréalisme, design
- □ **Audio** : Musique, voix synthétiques, podcasts
- □ **Vidéo** : Génération de séquences, deepfakes
- □ **Code** : Génération automatique de programmes

1.2 Types de modèles génératifs

Modèles de langage (LLM - Large Language Models)

Les **Large Language Models (LLM)** sont des réseaux de neurones entraînés sur d'énormes volumes de texte pour comprendre et générer du langage naturel. Le terme "Large" fait référence à la taille du modèle (milliards de paramètres) et à la quantité massive de données d'entraînement utilisées. Ces modèles apprennent les patterns du langage et peuvent générer du texte cohérent et contextuellement approprié.

Exemples populaires :

- **GPT-4** (OpenAI) : Génération de texte, conversation, analyse

- **Claude** (Anthropic) : Assistant conversationnel, analyse de documents
- **LLaMA** (Meta) : Modèle open-source
- **Gemini** (Google) : Multimodal (texte, images, vidéo)

Architecture :

Modèles de génération d'images

- **DALL-E** (OpenAI) : Génération d'images à partir de descriptions textuelles
- **Midjourney** : Création artistique
- **Stable Diffusion** (Stability AI) : Modèle open-source
- **Imagen** (Google) : Génération photoréaliste

Modèles multimodaux

Capables de traiter et générer plusieurs types de médias simultanément :

- **GPT-4 Vision** : Texte + Images
- **Gemini Pro** : Texte + Images + Audio + Vidéo

1.3 Architecture fondamentale : Transformers

Les **Transformers** sont l'architecture de base de la plupart des modèles génératifs modernes.

Composants clés

1. **Attention Mechanism** (Mécanisme d'attention) :
    - Permet au modèle de se concentrer sur les parties pertinentes de l'entrée en calculant des scores d'importance pour chaque élément. C'est comme si le modèle "regardait" différentes parties de l'input avec plus ou moins d'attention selon leur pertinence.
    - **Self-Attention** (Auto-attention) : Le modèle calcule les relations entre tous les tokens d'une séquence simultanément, permettant de comprendre les dépendances à longue distance. Chaque token peut "voir" et interagir avec tous les autres tokens.
    - **Multi-Head Attention** (Attention multi-têtes) : Au lieu d'une seule attention, le modèle utilise plusieurs "têtes" d'attention en parallèle, chacune apprenant à se concentrer sur différents types de relations (syntaxe, sémantique, etc.). Les résultats sont ensuite combinés.

2. **Feed-Forward Networks (FFN)** (Réseaux feed-forward) :
    - Couches de traitement non-linéaire qui transforment les représentations après l'attention. Chaque FFN est composé de deux couches linéaires avec une fonction d'activation (ReLU ou GELU) entre elles. Elles permettent au modèle d'apprendre des transformations complexes des données.

3. **Positional Encoding** (Encodage positionnel) :
    - Technique qui ajoute des informations sur la position de chaque token dans la séquence. C'est essentiel car les Transformers traitent tous les tokens en parallèle et n'ont pas de notion innée d'ordre séquentiel.

L'encodage positionnel permet au modèle de comprendre que "chat" avant "mange" a un sens différent de "mange" avant "chat".

Schéma d'architecture Transformer

1.4 Processus d'entraînement

Phase 1 : Pre-training (Apprentissage non supervisé)

- **Objectif** : Apprendre les patterns du langage
- **Données** : Corpus massif de texte (livres, web, code, etc.)
- **Tâche** : Prédiction du token suivant (Next Token Prediction)
- **Durée** : Semaines/mois sur clusters GPU puissants
- **Coût** : Millions de dollars en compute

Phase 2 : Fine-tuning (Ajustement supervisé)

- **Objectif** : Adapter le modèle à des tâches spécifiques
- **Données** : Datasets annotés pour la tâche cible
- **Méthodes** :
  - **Full Fine-tuning** : Ajuster tous les paramètres
  - **LoRA** (Low-Rank Adaptation) : Technique de fine-tuning efficace qui ajuste seulement quelques couches du modèle en ajoutant de petites matrices de paramètres au lieu de modifier tous les paramètres. Cela réduit drastiquement le nombre de paramètres à entraîner et le coût computationnel, tout en conservant la plupart des performances.
  - **RLHF** (Reinforcement Learning from Human Feedback) : Technique d'alignement qui utilise l'apprentissage par renforcement pour entraîner le modèle à suivre les préférences humaines. Des humains évaluent les réponses du modèle, et un système de récompense apprend au modèle à générer des réponses plus utiles, inoffensives et honnêtes.

Phase 3 : Alignment (Alignement)

- **Objectif** : Rendre le modèle utile, inoffensif et honnête
- **Techniques** :
  - **RLHF** : Récompenses basées sur feedback humain
  - **Constitutional AI** : Principes éthiques intégrés
  - **Red Teaming** : Tests de sécurité

---

2 Architecture et Design des systèmes d'IA générative

2.1 Principes de design

Scalabilité

Les systèmes d'IA générative doivent gérer :

- **Charge variable** : Pics de trafic imprévisibles
- **Latence** : Réponses en temps réel ou quasi-temps réel

- **Throughput** : Traiter des milliers de requêtes simultanées

Stratégies :

- **Horizontal scaling** : Ajouter des instances selon la charge
- **Caching** : Mettre en cache les réponses fréquentes
- **Load balancing** : Distribuer la charge entre instances
- **Async processing** : Traitement asynchrone pour tâches longues

Fiabilité (Reliability)

- **Redundancy** (Redondance) : Réplication des composants critiques pour assurer la disponibilité. Si un composant tombe en panne, un autre prend le relais automatiquement, évitant les interruptions de service.
- **Health checks** (Vérifications de santé) : Mécanismes qui surveillent continuellement l'état du système (CPU, mémoire, temps de réponse) pour détecter rapidement les problèmes et déclencher des actions correctives automatiques.
- **Circuit breakers** (Disjoncteurs) : Pattern de design qui interrompt automatiquement les appels vers un service défaillant après un certain nombre d'échecs. Cela protège le système contre les cascades de défaillances où un service en panne fait tomber d'autres services qui dépendent de lui.
- **Graceful degradation** (Dégradation gracieuse) : Capacité du système à continuer à fonctionner avec des fonctionnalités réduites en cas de problème partiel, plutôt que de s'arrêter complètement. Par exemple, désactiver certaines fonctionnalités avancées mais maintenir les fonctionnalités essentielles.

Observabilité

- **Logging** : Traces détaillées de toutes les opérations
- **Monitoring** : Métriques en temps réel (latence, throughput, erreurs)
- **Tracing** : Suivi des requêtes à travers les microservices
- **Alerting** : Notifications automatiques en cas d'anomalies

2.2 Architecture en couches

Couche 1 : Interface utilisateur

Composants :

- **Web UI** : Interface web pour interactions
- **API Gateway** (Passerelle API) : Point d'entrée unique pour toutes les requêtes vers les services backend. Il gère l'authentification, le routage, la limitation de débit, la transformation des données et fournit une interface unifiée aux clients, simplifiant l'architecture des microservices.
- **Mobile Apps** : Applications mobiles natives
- **CLI Tools** : Outils en ligne de commande

Responsabilités :

- Authentification et autorisation
- Validation des inputs
- Formatage des réponses
- Gestion des sessions

Couche 2 : Services applicatifs

Services principaux :

1. **Orchestration Service** :
    - Gestion du workflow de requêtes
    - Routage vers les bons modèles
    - Agrégation de réponses multiples
2. **Prompt Management Service** :
    - Stockage et versioning des prompts
    - Templates réutilisables
    - A/B testing de prompts
3. **Context Management Service** :
    - Gestion de la mémoire conversationnelle : Maintien du contexte et de l'historique des échanges pour permettre des conversations cohérentes sur plusieurs tours.
    - **RAG (Retrieval-Augmented Generation)** : Technique qui combine la recherche d'informations (retrieval) avec la génération de texte. Le système recherche d'abord des informations pertinentes dans une base de connaissances, puis utilise ces informations comme contexte pour générer une réponse plus précise et factuelle, réduisant ainsi les hallucinations.
    - Intégration de bases de connaissances : Connexion avec des bases de données, wikis, ou autres sources d'information pour enrichir les réponses avec des données à jour et vérifiées.
4. **Response Processing Service** :
    - Post-traitement des réponses
    - Filtrage de contenu
    - Formatage et validation

Couche 3 : Modèles et inference

Composants :

1. **Model Serving Layer** (Couche de service de modèles) :
    - **vLLM** : Framework open-source optimisé pour l'inférence de modèles de langage volumineux (LLMs). Il utilise des techniques avancées comme le PagedAttention pour gérer efficacement la mémoire et améliorer le débit de traitement.
    - **TensorRT-LLM** : Bibliothèque NVIDIA qui compile et optimise les modèles LLM pour une exécution ultra-rapide sur les GPU NVIDIA. Elle transforme les modèles en code optimisé spécifiquement pour l'architecture GPU.
    - **Triton Inference Server** : Serveur d'inférence flexible développé par NVIDIA qui permet de déployer des modèles ML de différents frameworks (PyTorch, TensorFlow, ONNX) avec une interface unifiée. Il gère automatiquement le batching, le versioning et le scaling.

2. **Model Registry** (Registre de modèles) :
    - Stockage centralisé des modèles : Système qui stocke tous les modèles entraînés dans un emplacement unique et accessible, facilitant la gestion et le déploiement.
    - **Versioning** (Gestion de versions) : Système qui maintient plusieurs versions d'un même modèle, permettant de revenir à une version précédente en cas de problème ou de comparer les performances entre versions.
    - **Checkpoints** (Points de contrôle) : Sauvegardes intermédiaires d'un modèle pendant l'entraînement. Ils permettent de reprendre l'entraînement en cas d'interruption ou de restaurer un état spécifique du modèle.

3. **Inference Engine** (Moteur d'inférence) :
    - **Batch Inference** (Inférence par lots) : Technique qui traite plusieurs requêtes simultanément en les regroupant, améliorant l'utilisation des ressources GPU et réduisant le coût par requête.
    - **Streaming Inference** (Inférence en streaming) : Technique qui envoie les résultats au fur et à mesure de leur génération (token par token), plutôt que d'attendre la fin complète. Cela réduit la latence perçue par l'utilisateur.
    - **Speculative Decoding** (Décodage spéculatif) : Technique avancée qui utilise un petit modèle rapide pour prédire plusieurs tokens à l'avance, puis un grand modèle vérifie ces prédictions. Cela accélère significativement la génération.

Couche 4 : Infrastructure cloud

Composants :

1. **Compute** (Calcul) :
    - **GPU Instances** (Instances GPU) : Serveurs cloud équipés de processeurs graphiques spécialisés pour l'IA :
        - **NVIDIA A100** : GPU haute performance avec 40GB ou 80GB de mémoire, optimisé pour l'entraînement et l'inférence de modèles volumineux.
        - **NVIDIA H100** : GPU de dernière génération encore plus puissant que l'A100, avec support pour FP8 et des performances accrues.
        - **NVIDIA L4** : GPU optimisé pour l'inférence avec un excellent rapport performance/coût.
    - **TPU** (Tensor Processing Unit) : Processeur spécialisé développé par Google spécifiquement pour le machine learning. Les TPU sont optimisés pour les opérations matricielles et offrent des performances exceptionnelles pour l'entraînement de grands modèles.
    - **Inferentia** : Processeur d'inférence développé par AWS, optimisé pour réduire les coûts d'inférence en production. Il offre un excellent rapport performance/coût pour les déploiements à grande échelle.

2. **Storage** (Stockage) :

- **Object Storage** (Stockage objet) : Système de stockage qui gère les données comme des objets (fichiers) avec leurs métadonnées :
    - **S3** (Simple Storage Service) : Service de stockage objet d'AWS, très utilisé pour stocker des modèles et datasets volumineux.
    - **GCS** (Google Cloud Storage) : Équivalent Google de S3 pour le stockage objet.
    - **Azure Blob** : Service de stockage objet de Microsoft Azure.
- **Block Storage** (Stockage par blocs) : Stockage qui divise les données en blocs de taille fixe, idéal pour les bases de données et les systèmes de fichiers :
    - **EBS** (Elastic Block Store) : Service de stockage par blocs d'AWS, attaché aux instances EC2.
    - **Persistent Disks** : Équivalent Google Cloud pour le stockage par blocs.
- **Database** (Base de données) : Systèmes de gestion de bases de données relationnelles ou NoSQL pour stocker les métadonnées, logs et configurations :
    - **PostgreSQL** : Base de données relationnelle open-source robuste et performante.
    - **MongoDB** : Base de données NoSQL orientée documents, flexible pour les données non structurées.

3. **Networking** (Réseau) :
    - **VPC** (Virtual Private Cloud) : Réseau virtuel isolé dans le cloud qui permet de créer un environnement réseau privé et sécurisé. Les ressources dans une VPC peuvent communiquer entre elles de manière sécurisée, isolées du reste d'Internet.
    - **CDN** (Content Delivery Network) : Réseau de serveurs distribués géographiquement qui met en cache le contenu statique près des utilisateurs finaux, réduisant la latence et la charge sur les serveurs principaux.
    - **Load Balancers** (Équilibreurs de charge) : Composants qui distribuent le trafic entrant entre plusieurs instances de serveurs, assurant une meilleure disponibilité et performance en répartissant la charge de travail.

2.3 Patterns d'architecture

Un **pattern d'architecture** est une solution réutilisable à un problème récurrent de conception de systèmes. Pour l'IA générative, trois patterns principaux sont couramment utilisés, chacun avec ses propres avantages et cas d'usage.

Pattern 1 : API Gateway + Microservices

Qu'est-ce que c'est ?

Ce pattern divise l'application en plusieurs petits services indépendants (microservices), chacun responsable d'une fonctionnalité spécifique. Un **API Gateway** sert de point d'entrée unique qui route les requêtes vers les services appropriés.

Architecture typique :

- **API Gateway** : Point d'entrée unique qui gère l'authentification, le routage, la limitation de débit
- **Service d'authentification** : Gère les utilisateurs et les tokens
- **Service LLM** : Gère les appels aux modèles de langage
- **Service de stockage** : Gère la persistance des données
- **Service de traitement** : Gère le pré/post-traitement des requêtes

Comment ça fonctionne ?

1. L'utilisateur envoie une requête à l'API Gateway
2. L'API Gateway authentifie et valide la requête
3. L'API Gateway route la requête vers le service approprié (ex: Service LLM)
4. Le service LLM peut appeler d'autres services si nécessaire (ex: Service de stockage pour récupérer le contexte)
5. Les réponses sont agrégées et renvoyées via l'API Gateway

Avantages :

- **Séparation des responsabilités** : Chaque service a une responsabilité claire, facilitant la maintenance et les mises à jour
- **Scalabilité indépendante** : On peut scaler uniquement le service LLM si nécessaire, sans affecter les autres services
- **Facilite le déploiement** : Chaque service peut être déployé indépendamment, permettant des mises à jour sans interruption
- **Technologies hétérogènes** : Chaque service peut utiliser la technologie la plus adaptée (Python pour ML, Go pour l'API Gateway, etc.)

Inconvénients :

- **Complexité de gestion** : Plus de services à gérer, monitorer et maintenir
- **Latence réseau** : Les communications entre services ajoutent de la latence (appels réseau)
- **Debugging plus difficile** : Une requête peut traverser plusieurs services, rendant le debugging plus complexe
- **Gestion de la cohérence** : Plus difficile de maintenir la cohérence des données entre services

Cas d'usage : Applications enterprise avec plusieurs équipes, besoins de scalabilité fine, systèmes complexes avec de nombreuses fonctionnalités.

Pattern 2 : Serverless Functions

Qu'est-ce que c'est ?

Le pattern **Serverless** (sans serveur) utilise des fonctions cloud qui s'exécutent à la demande, sans gestion d'infrastructure. Le fournisseur cloud gère automatiquement le provisioning, le scaling et la maintenance des serveurs.

Architecture typique :

- **API Gateway** : Point d'entrée qui déclenche les fonctions

- **Fonction de pré-traitement** : Nettoie et valide les inputs
- **Fonction d'inférence** : Appelle le modèle LLM (via API ou modèle déployé)
- **Fonction de post-traitement** : Formate et filtre les réponses

Comment ça fonctionne ?

1. Une requête arrive à l'API Gateway
2. L'API Gateway déclenche une fonction serverless (ex: AWS Lambda, Google Cloud Functions)
3. La fonction s'exécute, traite la requête (peut appeler d'autres fonctions ou services)
4. La fonction retourne la réponse et s'arrête automatiquement
5. Le fournisseur cloud facture uniquement le temps d'exécution

Avantages :

- **Pas de gestion d'infrastructure** : Le fournisseur cloud gère tout (serveurs, scaling, maintenance)
- **Scaling automatique** : Les fonctions scale automatiquement de zéro à des milliers d'instances selon la demande
- **Pay-per-use** : Facturation uniquement pour le temps d'exécution réel, pas pour le temps d'inactivité
- **Déploiement simple** : Juste uploader le code, pas de configuration de serveurs

Inconvénients :

- **Cold start** : Pour les modèles lourds, le premier appel peut être lent (plusieurs secondes) car la fonction doit démarrer et charger le modèle
- **Limites de temps d'exécution** : Les fonctions ont généralement une limite de temps (ex: 15 minutes max), problématique pour les tâches longues
- **Coût élevé à grande échelle** : Si le système traite beaucoup de requêtes, le coût peut devenir élevé comparé à des serveurs dédiés
- **Limites de mémoire** : Contraintes sur la quantité de mémoire disponible (ex: 10GB max pour Lambda)

Cas d'usage : Applications avec trafic variable, prototypes rapides, traitement de requêtes courtes, intégration avec d'autres services cloud.

Pattern 3 : Edge Computing

Qu'est-ce que c'est ?

Le pattern **Edge Computing** (calcul en périphérie) déplace le traitement plus près de l'utilisateur final, en exécutant les modèles sur des dispositifs en périphérie du réseau (edge devices) plutôt que dans un datacenter central.

Architecture typique :

- **Edge Devices** : Serveurs ou dispositifs proches des utilisateurs (ex: serveurs régionaux, gateways IoT, smartphones)
- **Cache de modèles** : Modèles plus petits stockés localement sur les edge devices
- **Cloud central** : Pour les requêtes complexes nécessitant des modèles plus grands

Comment ça fonctionne ?

1. La requête arrive sur un edge device proche de l'utilisateur
2. Si un modèle léger est disponible localement, il traite la requête directement
3. Si la requête est trop complexe, elle est redirigée vers le cloud central
4. Les réponses peuvent être mises en cache localement pour des requêtes similaires futures
5. Les modèles sur edge sont synchronisés périodiquement avec le cloud central

Avantages :

- **Latence réduite** : Traitement local = pas de latence réseau vers le cloud, réponses ultra-rapides
- **Réduction de la bande passante** : Moins de données envoyées au cloud, économie de bande passante
- **Disponibilité hors ligne** : Les edge devices peuvent fonctionner même si la connexion au cloud est interrompue
- **Réduction des coûts cloud** : Moins de requêtes au cloud = moins de coûts
- **Respect de la vie privée** : Les données peuvent être traitées localement sans être envoyées au cloud

Inconvénients :

- **Modèles plus petits** : Contraintes hardware des edge devices limitent la taille des modèles (ex: modèles quantifiés, pruned)
- **Synchronisation complexe** : Maintenir la cohérence entre les modèles edge et cloud est complexe
- **Coût de déploiement** : Nécessite de déployer et maintenir des dispositifs en périphérie
- **Gestion distribuée** : Plus difficile de monitorer et maintenir de nombreux edge devices
- **Mises à jour** : Mettre à jour les modèles sur tous les edge devices peut être complexe

Cas d'usage : Applications nécessitant une latence ultra-faible (temps réel, IoT), zones avec connectivité limitée, applications mobiles avec traitement local, respect strict de la vie privée.

Comparaison des patterns

Critère	API Gateway + Microservices	Serverless Functions	Edge Computing
Latence	Moyenne (appels réseau)	Variable (cold start)	Très faible (local)
Scalabilité	Manuelle mais flexible	Automatique	Limitée par hardware
Coût (faible trafic)	Élevé (serveurs toujours on)	Faible (pay-per-use)	Variable
Coût (fort trafic)	Modéré	Élevé	Faible
Complexité	Élevée	Faible	Très élevée
Taille des modèles	Aucune limite	Limite mémoire	Limite hardware
Meilleur pour	Enterprise, systèmes complexes	Prototypes, trafic variable	Latence critique, IoT

2.4 Design pour la performance

Optimisation des modèles

1. **Quantization** (Quantification) :
    - Technique qui réduit la précision numérique des paramètres du modèle (de FP32 à FP16 ou INT8). Cela réduit la taille du modèle de 2 à 4 fois et accélère l'inférence, tout en ayant un impact minimal sur la qualité. Les calculs sont plus rapides car les opérations sur des nombres plus petits sont plus efficaces.
    - **FP32** : Nombres à virgule flottante 32 bits (précision standard)
    - **FP16** : Nombres à virgule flottante 16 bits (moitié de la précision)
    - **INT8** : Nombres entiers 8 bits (précision très réduite mais très rapide)

2. **Pruning** (Élagage) :
    - Technique qui supprime les neurones ou connexions non essentiels du modèle. En identifiant et supprimant les parties redondantes, on peut réduire de 30 à 50% le nombre de paramètres sans perte significative de performance, accélérant ainsi l'inférence.

3. **Distillation** (Distillation de connaissances) :
    - Technique où un petit modèle (appelé "student" ou étudiant) apprend à imiter un grand modèle (appelé "teacher" ou professeur). Le petit modèle est entraîné sur les prédictions du grand modèle, permettant de conserver 80-90% des performances avec une taille drastiquement réduite.

4. **Compilation** :
    - Processus qui convertit un modèle en code optimisé pour une exécution plus rapide :
        - **TorchScript** : Format de compilation de PyTorch qui convertit les modèles Python en code optimisé et portable.
        - **ONNX Runtime** : Runtime qui exécute des modèles au format ONNX (Open Neural Network Exchange), un format standardisé inter-frameworks.
        - **TensorRT** : SDK NVIDIA qui compile et optimise les modèles pour une exécution ultra-rapide sur GPU NVIDIA, avec des optimisations spécifiques au matériel.

Optimisation de l'infrastructure

1. **Batching** :
    - Traiter plusieurs requêtes ensemble
    - Amélioration de l'utilisation GPU
    - Réduction du coût par requête

2. **Continuous Batching** :
    - Ajout dynamique de requêtes au batch
    - Meilleure utilisation des ressources

- Latence réduite

3. **KV Cache** (Cache clé-valeur) :
    - Technique qui met en cache les paires clé-valeur calculées lors de l'attention dans les Transformers. Comme ces valeurs sont réutilisées lors de la génération de tokens suivants, le cache évite de les recalculer, accélérant significativement l'inférence, surtout pour les contextes longs.

4. **PagedAttention** :
    - Technique avancée de gestion de mémoire développée pour vLLM. Elle divise la mémoire en "pages" comme un système d'exploitation, permettant une gestion efficace de la mémoire GPU, réduisant la fragmentation et supportant des contextes très longs (plusieurs centaines de milliers de tokens) sans problème de mémoire.

---

3 Exploitation et déploiement dans le cloud

3.1 Choix de la plateforme cloud

AWS (Amazon Web Services)

Services clés :

- **SageMaker** : Plateforme complète d'Amazon pour le machine learning qui fournit des outils pour l'entraînement, le déploiement et la gestion de modèles ML, avec intégration native aux autres services AWS.
- **Bedrock** : Service managé d'AWS qui fournit un accès unifié à plusieurs modèles LLM (Claude, Llama, Titan) via une API simple, sans avoir à gérer l'infrastructure sous-jacente.
- **EC2 P4/P5** : Types d'instances de calcul élastique d'AWS équipées de GPU NVIDIA :
    - **P4** : Instances avec GPU A100
    - **P5** : Instances avec GPU H100 (dernière génération)
- **Lambda** : Service serverless d'AWS qui exécute du code sans gestion de serveurs. Idéal pour l'inférence légère ou le pré/post-traitement, avec facturation à l'usage.
- **EKS** (Elastic Kubernetes Service) : Service managé d'AWS qui exécute Kubernetes dans le cloud AWS, simplifiant le déploiement et la gestion de clusters Kubernetes.

Avantages :

- Écosystème mature
- Large choix d'instances GPU
- Intégration avec autres services AWS

Cas d'usage :

- Déploiements enterprise
- Intégration avec infrastructure AWS existante

Google Cloud Platform (GCP)

Services clés :

- **Vertex AI** : Plateforme ML unifiée de Google Cloud qui regroupe tous les outils nécessaires pour l'entraînement, le déploiement et la gestion de modèles ML, avec intégration aux services Google.
- **Gemini API** : Interface de programmation qui fournit un accès aux modèles Gemini de Google (Gemini Pro, Gemini Ultra), permettant d'intégrer facilement ces modèles dans des applications.
- **Cloud TPU** : Service Google Cloud qui fournit des Tensor Processing Units (TPU), processeurs spécialisés très performants pour l'entraînement et l'inférence de modèles ML, particulièrement efficaces pour les opérations matricielles.
- **GKE** (Google Kubernetes Engine) : Service managé de Google Cloud qui exécute Kubernetes, simplifiant la gestion de clusters Kubernetes avec intégration native aux services GCP.
- **Cloud Run** : Service serverless de Google qui exécute des conteneurs sans gestion d'infrastructure. Il scale automatiquement de zéro à des milliers d'instances selon la demande.

Avantages :

- TPU pour entraînement/inference
- Modèles Gemini intégrés
- Excellent pour recherche

Cas d'usage :

- Entraînement à grande échelle
- Utilisation de modèles Google

Microsoft Azure

Services clés :

- **Azure OpenAI Service** : Service managé de Microsoft qui fournit un accès sécurisé et scalable aux modèles OpenAI (GPT-4, GPT-3.5, etc.) avec intégration aux services Azure et conformité enterprise.
- **Azure ML** : Plateforme complète de machine learning de Microsoft Azure qui offre des outils pour l'entraînement, le déploiement et la gestion de modèles ML avec support pour différents frameworks.
- **NC/ND Series** : Séries d'instances Azure équipées de GPU NVIDIA :
    - **NC Series** : Instances avec GPU pour calcul intensif
    - **ND Series** : Instances optimisées pour le deep learning avec GPU plus récents
- **AKS** (Azure Kubernetes Service) : Service managé de Microsoft qui simplifie le déploiement et la gestion de clusters Kubernetes dans Azure, avec intégration native aux services Azure.
- **Cognitive Services** : Suite d'APIs pré-entraînées de Microsoft pour ajouter rapidement des capacités d'IA (vision, langage, parole) aux applications sans entraîner de modèles personnalisés.

Avantages :

- Intégration Office 365
- Accès facile à OpenAI
- Enterprise-ready

Cas d'usage :

- Entreprises Microsoft
- Intégration avec écosystème Microsoft

Multi-cloud

Stratégie :

- Éviter le vendor lock-in
- Utiliser le meilleur service de chaque cloud
- Redondance géographique

Défis :

- Complexité de gestion
- Coûts de transfert de données
- Synchronisation des configurations

3.2 Déploiement avec Kubernetes

Architecture Kubernetes pour IA générative

Composants :

1. **Deployment** : Ressource Kubernetes qui gère le cycle de vie des pods (conteneurs) exécutant les modèles. Il assure qu'un nombre spécifié de pods fonctionne toujours, gère les mises à jour et les rollbacks.
2. **Service** : Ressource Kubernetes qui expose les pods de manière stable via une adresse IP et un nom DNS. Il fait le routage du trafic vers les pods appropriés, même si ceux-ci changent.
3. **HPA** (Horizontal Pod Autoscaler) : Contrôleur Kubernetes qui ajuste automatiquement le nombre de pods (instances) en fonction de métriques comme l'utilisation CPU, mémoire ou métriques custom. Il scale horizontalement (plus de pods) selon la charge.
4. **VPA** (Vertical Pod Autoscaler) : Contrôleur Kubernetes qui ajuste automatiquement les ressources (CPU, mémoire) allouées à chaque pod en fonction de leur utilisation réelle. Il optimise verticalement (plus de ressources par pod) plutôt que d'ajouter des pods.
5. **ConfigMap** : Ressource Kubernetes qui stocke des données de configuration non sensibles (paramètres, fichiers de config) sous forme de paires clé-valeur. Les pods peuvent monter ces ConfigMaps comme volumes ou variables d'environnement.
6. **Secret** : Ressource Kubernetes qui stocke des données sensibles (mots de passe, clés API, tokens) de manière chiffrée. Similaire aux ConfigMaps mais avec gestion sécurisée des données confidentielles.

Déploiement de modèles

Pour déployer un modèle LLM dans Kubernetes, on crée un **Deployment** qui spécifie l'image

Docker contenant le modèle, les ressources nécessaires (CPU, mémoire, GPU), les variables d'environnement (nom du modèle, paramètres), et les volumes pour le cache des modèles. Un **Service** expose ensuite ce déploiement avec une adresse IP stable, permettant aux autres services de communiquer avec le modèle via cette adresse.

Auto-scaling

HPA basé sur CPU/Memory :

Le HPA (Horizontal Pod Autoscaler) peut scaler automatiquement les pods en fonction de l'utilisation CPU et mémoire. On configure des seuils (ex: 70% CPU, 80% mémoire) et le HPA ajoute ou retire des pods pour maintenir ces métriques dans les limites définies. C'est utile pour les workloads avec des patterns de charge prévisibles.

HPA basé sur métriques custom (requêtes par seconde) :

Pour les systèmes d'IA générative, il est souvent plus pertinent de scaler basé sur des métriques applicatives comme le nombre de requêtes par seconde plutôt que sur CPU/mémoire. On configure le HPA pour surveiller des métriques custom (ex: via Prometheus) et scaler quand le nombre de requêtes dépasse un seuil. Cela permet une réaction plus rapide aux pics de trafic.

Gestion des GPU

Node Selector pour GPU :

Pour s'assurer que les pods de modèle s'exécutent sur des nœuds avec GPU, on utilise des **node selectors** qui spécifient des labels de nœuds (ex: "accelerator: nvidia-tesla-a100"). Kubernetes planifie alors les pods uniquement sur les nœuds correspondants. On peut aussi utiliser des **tolerations** pour permettre aux pods d'être planifiés sur des nœuds avec des taints spécifiques (ex: nœuds GPU réservés).

GPU Sharing avec MIG (Multi-Instance GPU) :

Technique NVIDIA qui permet de diviser un GPU A100 en plusieurs instances GPU virtuelles plus petites (jusqu'à 7 instances). Chaque instance est isolée avec ses propres ressources (mémoire, compute), permettant de partager un GPU coûteux entre plusieurs workloads tout en maintenant l'isolation et la sécurité. Cela améliore l'utilisation des ressources GPU et réduit les coûts.

Device Plugins :

Les **Device Plugins** Kubernetes permettent d'exposer des ressources matérielles spécialisées (comme les GPU) aux pods. Le plugin NVIDIA Device Plugin, par exemple, permet à Kubernetes de découvrir et allouer des GPU aux pods. On peut aussi configurer le **time-slicing** pour partager un GPU entre plusieurs pods en divisant le temps d'exécution, utile pour les workloads qui n'utilisent pas le GPU à 100%.

3.3 CI/CD pour modèles ML

Pipeline CI/CD

Un pipeline CI/CD (Continuous Integration / Continuous Deployment) pour ML automatise le processus de développement, test, entraînement, évaluation et déploiement de modèles. Le

pipeline typique suit ces étapes :

1. **Code** : Développement et commit du code dans Git
2. **Build** : Construction de l'image Docker avec les dépendances
3. **Test** : Exécution des tests unitaires et d'intégration
4. **Train** : Entraînement du modèle (si le code ou les données changent)
5. **Evaluate** : Évaluation du modèle avec des métriques de performance
6. **Deploy** : Déploiement automatique en staging puis production si les tests passent
7. **Monitor** : Surveillance continue des performances en production

Composants

1. **Version Control** :
    - **DVC** (Data Version Control) : Outil de versioning spécialement conçu pour les données et modèles ML. Il fonctionne comme Git mais pour les fichiers volumineux, stockant les versions de datasets et modèles de manière efficace avec déduplication.
    - **MLflow** : Plateforme open-source pour gérer le cycle de vie du machine learning. Elle permet de tracker les expériences (paramètres, métriques, artefacts), gérer les modèles, et déployer des modèles en production avec un système de registry.
    - **Git LFS** (Large File Storage) : Extension de Git qui permet de versionner des fichiers volumineux (modèles, datasets) en les stockant séparément du dépôt Git principal, évitant de ralentir les opérations Git.
2. **Testing** :
    - **Unit Tests** : Tests de fonctions
    - **Integration Tests** : Tests d'end-to-end
    - **Model Tests** : Validation de performance
3. **Training Pipeline** :
    - **Kubeflow** : Plateforme open-source pour déployer et gérer des workflows ML sur Kubernetes. Elle fournit des pipelines réutilisables, des notebooks Jupyter, et des outils pour l'entraînement distribué et le déploiement de modèles.
    - **Airflow** : Plateforme open-source d'Apache pour orchestrer des workflows complexes. Elle permet de définir, planifier et surveiller des pipelines de données et ML avec des DAGs (Directed Acyclic Graphs).
    - **Prefect** : Framework moderne de workflow automation conçu pour être plus simple et plus robuste qu'Airflow. Il offre une meilleure expérience développeur avec gestion automatique des erreurs et monitoring intégré.
4. **Model Registry** :
    - **MLflow Model Registry** : Gestion des versions
    - **Weights & Biases** : Tracking et registry
    - **SageMaker Model Registry** : AWS
5. **Deployment** :

- **ArgoCD** : Outil GitOps pour Kubernetes qui synchronise automatiquement les déploiements avec les configurations stockées dans Git. Il surveille les dépôts Git et déploie automatiquement les changements, assurant que l'état du cluster correspond toujours à la configuration versionnée.
- **Flux** : Alternative open-source à ArgoCD pour GitOps sur Kubernetes. Il fournit une synchronisation continue entre Git et Kubernetes avec support pour Helm, Kustomize et autres outils.
- **Seldon** : Plateforme open-source pour déployer et servir des modèles ML sur Kubernetes. Elle fournit des fonctionnalités avancées comme A/B testing, canary deployments, et monitoring des modèles en production.

Automatisation avec GitHub Actions

GitHub Actions peut être utilisé pour automatiser le pipeline CI/CD. Un workflow typique inclut :

1. **Trigger** : Déclenchement automatique sur push vers la branche main
2. **Checkout** : Récupération du code source
3. **Setup** : Configuration de l'environnement (Python, dépendances)
4. **Test** : Exécution des tests automatisés
5. **Train** (conditionnel) : Entraînement du modèle si nécessaire
6. **Build** : Construction de l'image Docker
7. **Push** : Envoi de l'image vers un registry (Docker Hub, GHCR, etc.)
8. **Deploy** : Déploiement sur Kubernetes via kubectl ou outils GitOps

Les workflows GitHub Actions permettent aussi de définir des conditions (ex: ne déployer que si les tests passent) et des environnements (staging, production) avec approbations manuelles si nécessaire.

3.4 Monitoring et observabilité

Métriques à surveiller

Métriques système :

- **CPU Usage** : Utilisation processeur
- **Memory Usage** : Utilisation mémoire
- **GPU Utilization** : Utilisation GPU (%)
- **GPU Memory** : Mémoire GPU utilisée
- **Network I/O** : Bande passante réseau
- **Disk I/O** : Opérations disque

Métriques applicatives :

- **Request Rate** : Requêtes par seconde
- **Latency** (Latence) : Temps de réponse du système. Les percentiles P50, P95, P99 indiquent respectivement que 50%, 95% et 99% des requêtes sont traitées en moins de ce temps. P95 et P99 sont particulièrement importants car ils révèlent les cas extrêmes qui impactent l'expérience utilisateur.
- **Error Rate** : Taux d'erreur (%)
- **Token Generation Rate** : Tokens générés/seconde

- **Queue Length** : Taille de la file d'attente
- **Cache Hit Rate** : Taux de cache hit

Métriques de qualité :

- **Response Quality Score** : Score de qualité (si applicable)
- **Hallucination Rate** : Taux d'hallucinations
- **Toxicity Score** : Score de toxicité
- **Bias Metrics** : Métriques de biais

Outils de monitoring

Prometheus + Grafana :

Prometheus est un système de monitoring open-source qui collecte et stocke des métriques temporelles. Il permet de définir des règles d'alerte basées sur des requêtes de métriques. **Grafana** est un outil de visualisation qui se connecte à Prometheus pour créer des dashboards interactifs. Ensemble, ils permettent de surveiller en temps réel les performances des systèmes d'IA générative, avec des graphiques de latence, throughput, utilisation GPU, etc.

Datadog / New Relic :

- APM (Application Performance Monitoring)
- Infrastructure monitoring
- Log aggregation
- Alerting

Custom Dashboards :

- **Grafana** : Dashboards personnalisés
- **Kibana** : Visualisation de logs
- **Jupyter Notebooks** : Analyses ad-hoc

Alerting

Règles d'alerte critiques :

Les systèmes d'alerte doivent surveiller plusieurs métriques critiques et déclencher des notifications en cas de problème :

- **Latence élevée** : Alerte si le P95 de latence dépasse un seuil (ex: 5 secondes) pendant plus de 5 minutes, indiquant un problème de performance
- **Taux d'erreur élevé** : Alerte si plus de 5% des requêtes échouent, indiquant un problème système ou de modèle
- **GPU Out of Memory** : Alerte si l'utilisation mémoire GPU dépasse 95%, indiquant un risque de crash
- **Coût quotidien** : Alerte si les coûts dépassent un budget défini, permettant de contrôler les dépenses

Ces alertes peuvent être envoyées via email, Slack, PagerDuty ou d'autres canaux selon la criticité.

4 Orchestration cloud pour l'IA générative

4.1 Concepts d'orchestration

L'**orchestration cloud** consiste à automatiser la gestion, la coordination et le déploiement de services et ressources dans le cloud. Pour l'IA générative, cela inclut :

- **Provisioning** : Création automatique d'infrastructure
- **Scaling** : Ajustement automatique des ressources
- **Load Balancing** : Distribution intelligente de la charge
- **Service Discovery** : Découverte automatique des services
- **Health Management** : Surveillance et récupération automatique

4.2 Orchestration avec Kubernetes

Namespace Strategy

L'organisation par **namespaces** permet d'isoler les ressources Kubernetes par environnement (development, staging, production) ou par équipe. Chaque namespace a ses propres ressources, secrets, et configurations. Par exemple, on peut créer un namespace "ai-dev" pour le développement et "ai-prod" pour la production, avec des labels pour faciliter la gestion et le monitoring.

Resource Quotas

Les **Resource Quotas** limitent la quantité de ressources (CPU, mémoire, GPU, volumes) qu'un namespace peut utiliser. Cela empêche un namespace de consommer toutes les ressources du cluster et garantit un partage équitable. Par exemple, on peut limiter le namespace "ai-dev" à 10 GPU maximum, laissant les autres GPU disponibles pour la production.

Network Policies

Les **Network Policies** définissent des règles de sécurité réseau qui contrôlent le trafic entre les pods. Elles permettent d'isoler les services sensibles (comme les modèles LLM) en autorisant uniquement les communications depuis des sources spécifiques (ex: uniquement depuis l'API Gateway). Cela réduit la surface d'attaque et améliore la sécurité.

4.3 Service Mesh pour IA générative

Istio

Avantages :

- **Traffic Management** (Gestion du trafic) : Capacité à router intelligemment le trafic entre différentes versions de services, avec support pour les canary deployments (déploiement progressif où une petite partie du trafic va vers la nouvelle version pour tester avant déploiement complet).
- **Security** (Sécurité) : Fournit mTLS (mutual TLS) automatique pour chiffrer toutes

les communications entre services, et permet de définir des policies de sécurité granulaires pour contrôler l'accès entre services.

- **Observability** (Observabilité) : Collecte unifiée de métriques (mesures de performance), traces (suivi des requêtes à travers les services) et logs (enregistrements d'événements) pour une visibilité complète sur le système.

Configuration pour LLM :

Pour les systèmes LLM, Istio permet de configurer des règles de routage intelligentes. Par exemple, router les requêtes des utilisateurs premium vers une version plus performante du modèle, ou faire des canary deployments en envoyant 10% du trafic vers une nouvelle version pour tester avant déploiement complet. On peut aussi configurer des timeouts et retries spécifiques pour les appels LLM qui peuvent être longs.

Linkerd

Alternative légère à Istio :

- Plus simple à déployer
- Moins de ressources
- Focus sur la fiabilité

4.4 Workflow orchestration

Kubeflow Pipelines

Kubeflow Pipelines permet de créer des pipelines ML complets et réutilisables sur Kubernetes. Un pipeline typique inclut des étapes pour le préprocessing des données, l'entraînement du modèle, l'évaluation, et le déploiement. Chaque étape est un composant indépendant qui peut être réutilisé dans d'autres pipelines. Les pipelines sont versionnés et peuvent être exécutés manuellement ou automatiquement selon un calendrier.

Apache Airflow

Apache Airflow orchestre des workflows complexes en définissant des DAGs (Directed Acyclic Graphs) où chaque tâche a des dépendances. Pour l'IA générative, on peut créer des workflows qui entraînent des modèles périodiquement, évaluent leur performance, et les déploient automatiquement si les métriques sont satisfaisantes. Airflow gère les retries, les alertes, et permet de monitorer l'exécution des workflows.

4.5 Auto-scaling avancé

KEDA (Kubernetes Event-Driven Autoscaling)

KEDA est un composant Kubernetes qui permet le scaling automatique basé sur des métriques externes (files d'attente, bases de données, APIs métriques). Contrairement au HPA standard qui utilise seulement les métriques CPU/mémoire, KEDA peut scaler en fonction de n'importe quelle métrique externe, comme le nombre de messages dans une file d'attente ou le nombre de requêtes par seconde depuis Prometheus.

Cluster Autoscaler

Le **Cluster Autoscaler** ajuste automatiquement le nombre de nœuds dans le cluster

Kubernetes en fonction de la demande. Si des pods ne peuvent pas être planifiés faute de ressources, le Cluster Autoscaler ajoute des nœuds. Inversement, si des nœuds sont sous-utilisés, il les supprime pour réduire les coûts. On peut configurer des délais (ex: attendre 10 minutes avant de supprimer un nœud) pour éviter des changements trop fréquents.

4.6 Multi-tenancy

Isolation des ressources

Node Affinity :

Le **Node Affinity** permet de spécifier sur quels nœuds les pods doivent être planifiés. Pour le multi-tenancy, on peut utiliser des labels sur les nœuds (ex: "tenant: tenant-a") et configurer les pods pour qu'ils ne s'exécutent que sur les nœuds avec ce label. Cela garantit l'isolation physique entre les tenants, important pour la sécurité et la conformité.

Resource Quotas par tenant :

Chaque tenant (client/organisation) peut avoir son propre namespace avec des **Resource Quotas** spécifiques. Par exemple, "tenant-a" peut avoir accès à 5 GPU maximum, tandis que "tenant-b" en a 10. Cela garantit qu'un tenant ne peut pas consommer toutes les ressources et permet une facturation équitable basée sur l'utilisation réelle.

Billing et cost allocation

Kubecost :

- Tracking des coûts par namespace/pod
- Attribution des coûts GPU
- Recommandations d'optimisation

---

5 Éthique et responsabilité

5.1 Principes éthiques fondamentaux

Transparence

Définition : Les utilisateurs doivent comprendre comment l'IA fonctionne et quand elle est utilisée.

Implémentations :

- **Explainability** : Explication des décisions de l'IA
- **Disclosure** : Indication claire de l'utilisation d'IA
- **Documentation** : Documentation complète des modèles

Outils :

- **SHAP** (SHapley Additive exPlanations) : Méthode mathématique qui explique les prédictions d'un modèle en attribuant une valeur d'importance à chaque feature (caractéristique) d'entrée. Basée sur la théorie des jeux, elle montre comment

chaque input contribue à la prédiction finale.

- **LIME** (Local Interpretable Model-agnostic Explanations) : Technique qui explique les prédictions en créant un modèle simple et interprétable localement autour d'une prédiction spécifique. Elle fonctionne avec n'importe quel type de modèle et fournit des explications compréhensibles pour une prédiction donnée.
- **Attention Visualization** : Visualisation de l'attention

Équité (Fairness)

Définition : L'IA ne doit pas discriminer ou favoriser certains groupes.

Types de biais :

- **Biais de données** : Données d'entraînement non représentatives
- **Biais algorithmique** : Algorithmes qui amplifient les inégalités
- **Biais de confirmation** : Modèles qui renforcent les stéréotypes

Métriques de fairness :

- **Demographic Parity** : Taux de prédiction positive égal entre groupes
- **Equalized Odds** : Taux de vrais/faux positifs égaux
- **Calibration** : Probabilités calibrées pour tous les groupes

Mitigation :

- **Data Augmentation** : Augmenter la diversité des données
- **Fairness Constraints** : Contraintes dans l'entraînement
- **Adversarial Debiasing** : Réduction de biais via adversaires

Privacy (Vie privée)

Définition : Protection des données personnelles et respect de la vie privée.

Techniques :

- **Differential Privacy** (Vie privée différentielle) : Technique mathématique qui ajoute du bruit contrôlé aux données ou résultats pour protéger la vie privée des individus. Elle garantit qu'un attaquant ne peut pas déterminer si un individu spécifique était dans le dataset, même avec accès aux résultats.
- **Federated Learning** (Apprentissage fédéré) : Technique d'entraînement distribué où les données restent sur les appareils/clients locaux. Seuls les paramètres du modèle (pas les données) sont envoyés au serveur central pour agrégation. Cela permet d'entraîner des modèles sans centraliser les données sensibles.
- **Homomorphic Encryption** (Chiffrement homomorphique) : Type de chiffrement qui permet d'effectuer des calculs directement sur des données chiffrées sans les déchiffrer. Les résultats des calculs restent chiffrés et peuvent être déchiffrés pour obtenir le résultat final, permettant le calcul sur données sensibles sans exposition.
- **Data Minimization** : Collecte minimale de données

Réglementations :

- **GDPR** (Europe) : Règlement général sur la protection des données
- **CCPA** (Californie) : California Consumer Privacy Act
- **PIPEDA** (Canada) : Personal Information Protection and Electronic Documents Act

Robustesse (Robustness)

Définition : L'IA doit être résistante aux attaques et erreurs.

Types d'attaques :

- **Adversarial Attacks** : Inputs malveillants conçus pour tromper
- **Prompt Injection** : Injection de prompts malveillants
- **Data Poisoning** : Corruption des données d'entraînement

Défenses :

- **Adversarial Training** : Entraînement avec exemples adversaires
- **Input Validation** : Validation stricte des inputs
- **Output Filtering** : Filtrage des outputs dangereux
- **Red Teaming** : Tests de sécurité réguliers

5.2 Biais et discrimination

Types de biais dans l'IA générative

1. Biais de représentation :

- Sous-représentation de certains groupes dans les données
- Exemple : Modèles d'images générant principalement des personnes blanches

2. Biais de stéréotype :

- Renforcement de stéréotypes existants
- Exemple : Associant "nurse" à "femme" et "engineer" à "homme"

3. Biais de confirmation :

- Modèles qui confirment les préjugés des utilisateurs
- Exemple : Réponses biaisées basées sur le contexte de la requête

4. Biais culturel :

- Biais vers certaines cultures/langues
- Exemple : Meilleures performances en anglais qu'en langues minoritaires

Détection de biais

Tests de biais :

Métriques de biais :

- **WEAT** (Word Embedding Association Test) : Test statistique qui mesure les associations stéréotypées dans les embeddings de mots. Il compare la similarité entre des mots cibles (ex: "nurse", "engineer") et des attributs (ex: "female", "male") pour détecter des biais de genre, race, etc.
- **SEAT** (Sentence Embedding Association Test) : Extension du WEAT qui teste les biais au niveau des phrases plutôt que des mots individuels, permettant de détecter des biais plus subtils dans les représentations sémantiques.
- **StereoSet** : Benchmark pour stéréotypes

Mitigation de biais

1. Pre-processing :

- **Data Balancing** : Équilibrer les représentations
- **Data Augmentation** : Augmenter la diversité
- **Bias Removal** : Suppression explicite de biais

2. In-processing :

- **Fairness Constraints** : Contraintes dans la loss function
- **Adversarial Debiasing** : Réduction via adversaires
- **Regularization** : Pénalisation des biais

3. Post-processing :

- **Output Filtering** : Filtrage des outputs biaisés
- **Calibration** : Ajustement des probabilités
- **Reranking** : Réordonnancement des résultats

5.3 Hallucinations et désinformation

Qu'est-ce qu'une hallucination ?

Une **hallucination** est une génération de contenu qui semble plausible mais est factuellement incorrecte ou non fondée.

Types d'hallucinations :

- **Factual Hallucinations** : Faits incorrects
- **Coherence Hallucinations** : Incohérences internes
- **Contradiction Hallucinations** : Contradictions avec le contexte

Causes des hallucinations

1. **Limitations du modèle** :
    - Modèles entraînés à générer du texte plausible, pas nécessairement vrai
    - Pas de mécanisme de vérification factuelle intégré
2. **Données d'entraînement** :
    - Données contenant des erreurs
    - Informations obsolètes
    - Conflits entre sources
3. **Prompting** :
    - Prompts ambigus
    - Manque de contexte
    - Instructions contradictoires

Mitigation des hallucinations

1. RAG (Retrieval-Augmented Generation) : Technique qui combine la recherche

d'informations (retrieval) avec la génération de texte. Le système recherche d'abord des informations pertinentes dans une base de connaissances externe (documents, bases de données), puis utilise ces informations comme contexte pour générer une réponse. Cela permet au modèle d'accéder à des informations à jour et vérifiées, réduisant significativement les hallucinations en s'appuyant sur des sources factuelles plutôt que uniquement sur les connaissances mémorisées du modèle.

2. Fact-checking (Vérification des faits) :

- Vérification externe des faits
- Cross-referencing avec sources fiables
- Détection de contradictions

3. Confidence Scoring :

- Attribution de scores de confiance aux réponses
- Indication d'incertitude
- Refus de répondre si confiance trop faible

4. Prompt Engineering :

- Instructions claires de vérification
- Demande de citations
- Limitation du domaine de connaissances

5.4 Responsabilité et accountability

Framework de responsabilité

1. Developer Responsibility :

- Conception éthique des systèmes
- Tests de biais et sécurité
- Documentation complète

2. Organizational Responsibility :

- Governance structures
- Ethics committees
- Training des équipes

3. User Responsibility :

- Utilisation appropriée
- Vérification des outputs
- Reporting de problèmes

Governance structures

Ethics Board :

- Composition diverse (technique, éthique, légal, utilisateurs)
- Révision régulière des systèmes
- Décisions sur déploiements sensibles

AI Safety Team :

- Tests de sécurité continus
- Red teaming
- Monitoring des risques

Compliance Team :

- Vérification de conformité réglementaire
- Audit régulier
- Documentation pour audits

Documentation et traçabilité

Model Cards :

- Documentation standardisée des modèles
- Informations sur performance, biais, limitations
- Recommandations d'utilisation

Data Sheets :

- Documentation des datasets
- Provenance des données
- Limitations et biais connus

System Cards :

- Documentation des systèmes complets
- Architecture et décisions de design
- Impact social et éthique

---

6 Sécurité et gouvernance

6.1 Sécurité des modèles

Adversarial Attacks

Types d'attaques :

1. **Prompt Injection** :
    - Injection de prompts malveillants dans l'input utilisateur
    - Exemple : "Ignore previous instructions and..."

Défense :

2. **Jailbreaking** :
    - Tentative de contourner les garde-fous
    - Techniques : DAN (Do Anything Now), role-playing

Défense :

- Systèmes de garde-fous robustes

- Monitoring des tentatives
- Rate limiting

3. **Model Extraction** :
    - Tentative de reconstruire le modèle via API
    - Query-based attacks

Défense :

- Rate limiting strict
- Monitoring des patterns de requêtes
- Watermarking des outputs

Sécurité des données

Chiffrement :

- **At Rest** : Chiffrement des données stockées
- **In Transit** : TLS/SSL pour communications
- **In Use** : Homomorphic encryption pour calculs

Accès :

- **Principle of Least Privilege** : Accès minimal nécessaire
- **Role-Based Access Control (RBAC)** (Contrôle d'accès basé sur rôles) : Modèle de sécurité qui accorde des permissions aux utilisateurs en fonction de leur rôle dans l'organisation plutôt que d'attribuer des permissions individuelles. Par exemple, tous les "développeurs" ont accès aux environnements de développement, mais pas à la production.
- **Multi-Factor Authentication (MFA)** (Authentification multi-facteurs) : Méthode de sécurité qui exige plusieurs formes d'identification (facteurs) pour accéder à un système. Typiquement : quelque chose que vous savez (mot de passe), quelque chose que vous avez (téléphone), et quelque chose que vous êtes (biométrie).

Audit Logging :

- Logs de toutes les accès aux données
- Traçabilité complète
- Détection d'anomalies

6.2 Governance des modèles

Model Lifecycle Management

1. Development :

- Version control
- Experiment tracking
- Code review

2. Training :

- Reproducibility
- Resource tracking
- Cost monitoring

3. Evaluation :

- Comprehensive testing
- Bias evaluation
- Safety testing

4. Deployment :

- Approval process
- Staging environment
- Gradual rollout

5. Monitoring :

- Performance tracking
- Drift detection
- Incident response

6. Retirement :

- Deprecation process
- Data retention policies
- Archive procedures

Model Registry

Composants :

1. **Metadata** :
    - Version
    - Training parameters
    - Performance metrics
    - Dependencies
2. **Artifacts** :
    - Model weights
    - Tokenizers
    - Configurations
3. **Lineage** :
    - Training data
    - Code version
    - Environment
4. **Access Control** :
    - Permissions
    - Audit trail

Utilisation de MLflow :

MLflow fournit un système de Model Registry qui permet de gérer le cycle de vie des

modèles. On peut enregistrer un modèle avec ses métadonnées (version, paramètres d'entraînement, métriques de performance), le marquer comme "Staging" ou "Production", et suivre les déploiements. Le registry maintient un historique complet des versions, permettant de revenir facilement à une version précédente en cas de problème. Il intègre aussi avec les pipelines CI/CD pour automatiser les promotions de modèles.

6.3 Compliance réglementaire

Réglementations clés

1. GDPR (Europe) :

- **Right to Explanation** (Droit à l'explication) : Principe du GDPR qui donne aux individus le droit de comprendre comment une décision automatisée les concernant a été prise. Les systèmes d'IA doivent pouvoir expliquer leurs décisions de manière compréhensible.
- **Right to Erasure** (Droit à l'effacement / "Droit à l'oubli") : Principe du GDPR qui permet aux individus de demander la suppression de leurs données personnelles. Les organisations doivent pouvoir supprimer toutes les données d'un individu sur demande légitime.
- **Data Minimization** (Minimisation des données) : Principe qui exige de collecter et traiter uniquement les données strictement nécessaires pour atteindre un objectif spécifique. Cela réduit les risques de violation de vie privée et les coûts de gestion.
- **Privacy by Design** (Protection dès la conception) : Principe qui exige d'intégrer la protection de la vie privée dès la conception des systèmes, plutôt que comme une considération après coup. La vie privée devient une fonctionnalité fondamentale, pas un ajout.

2. AI Act (Europe) :

- Classification des systèmes d'IA par risque
- Exigences pour systèmes à haut risque
- Interdictions pour certains usages
- Transparence pour systèmes génératifs

3. NIST AI Risk Management Framework :

- Framework volontaire
- Gestion des risques d'IA
- Guidelines pour développement responsable

Implementation de compliance

Data Protection :

- Anonymization/Pseudonymization
- Encryption
- Access controls
- Data retention policies

Transparency :

- User notifications
- Explainability

- Documentation

Accountability :

- Audit trails
- Documentation
- Regular assessments

---

7 Cas d'usage pratiques

7.1 Assistant virtuel intelligent

Architecture

Un assistant virtuel intelligent suit généralement une architecture en pipeline : l'input utilisateur passe par la compréhension du langage naturel (NLU), la classification de l'intent, la récupération du contexte, la génération de réponse via un LLM, et enfin la génération de langage naturel (NLG) pour formater la réponse.

Composants

1. **NLU (Natural Language Understanding)** (Compréhension du langage naturel) :
    - **Compréhension de l'intent** : Identification de l'intention de l'utilisateur derrière sa requête (ex: "réserver un vol", "demander la météo"). C'est la tâche de comprendre ce que l'utilisateur veut faire.
    - **Extraction d'entités** : Identification et extraction d'informations structurées dans le texte (ex: dates, lieux, noms, montants). Par exemple, extraire "Paris" et "15 mars" de "Je veux aller à Paris le 15 mars".
    - **Gestion de la conversation** : Suivi du contexte et de l'historique de la conversation pour maintenir la cohérence et permettre des références à des éléments mentionnés précédemment.

2. **Context Management** :
    - Mémoire conversationnelle
    - RAG pour connaissances externes
    - Personalization

3. **Response Generation** :
    - Génération de réponses naturelles
    - Multi-turn conversations
    - Handling d'ambiguïtés

Implémentation

Pour implémenter un assistant virtuel, on utilise généralement un framework comme LangChain qui fournit des abstractions pour gérer les conversations, intégrer des LLMs, et connecter des bases de connaissances via RAG. Le système maintient une mémoire

conversationnelle pour garder le contexte, utilise des templates de prompts pour structurer les interactions, et peut intégrer des outils externes (APIs, bases de données) pour enrichir les réponses.

7.2 Génération de code

Use Cases

- **Code Completion** : Autocomplétion intelligente qui suggère la suite du code pendant la saisie
- **Code Generation** : Génération de code complet à partir de descriptions en langage naturel
- **Code Review** : Révision automatique du code pour détecter des bugs, problèmes de sécurité, ou violations de style
- **Documentation** : Génération automatique de documentation à partir du code source
- **Refactoring** : Suggestions de refactoring pour améliorer la qualité et la maintenabilité du code

Architecture

L'architecture typique pour la génération de code inclut : une interface (IDE, éditeur), un service de génération qui utilise un modèle LLM spécialisé en code (ex: Cod