

# L'IAG pour chefs de projets, analystes, développeurs et architectes SI

Redha Moulla

16 - 17 octobre 2025

# Plan de la formation

- Qu'est-ce que l'intelligence artificielle ?
- Eléments de machine learning et de deep learning
- IA générative
- IAG dans le cycle de développement
- Faire évoluer l'héritage
- IAG pour la gestion collaborative des projets
- IAG pour les processus agiles
- Enjeux éthique, de sécurité et de conformité de l'IAG

# Qu'est-ce que l'intelligence artificielle ?

# Définition littérale de l'intelligence artificielle

## 1. Intelligence

Ensemble des fonctions mentales ayant pour objet la connaissance conceptuelle et relationnelle.

- Larousse

## 2. Artificielle

Qui est produit de l'activité humaine (opposé à la nature).

- Larousse

# Qu'est-ce que l'intelligence ?

La notion d'intelligence recouvre plusieurs facultés cognitives :

- ① **Raisonnement** : La capacité à résoudre des problèmes et à faire des déductions logiques.
- ② **Apprentissage** : L'aptitude à acquérir de nouvelles connaissances et à s'améliorer grâce à l'expérience.
- ③ **Perception** : La compétence pour reconnaître et interpréter les stimuli sensoriels.
- ④ **Compréhension** : L'habileté à saisir le sens et l'importance de divers concepts et situations.
- ⑤ **Mémorisation** : La faculté de stocker et de rappeler des informations.
- ⑥ **Créativité** : Le pouvoir d'inventer ou de produire de nouvelles idées, de l'originalité dans la pensée.

Mais est-ce que l'intelligence est réductible à des facultés mesurables ?

# Conférence de Dartmouth ?

Articles

AI Magazine Volume 27 Number 4 (2006) © AAAI

## A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence

August 31, 1955

John McCarthy, Marvin L. Minsky,  
Nathaniel Rochester,  
and Claude E. Shannon

The 1956 Dartmouth summer research project on artificial intelligence was organized by John McCarthy, Marvin Minsky, Nathaniel Rochester, and Claude Shannon. The report consists of 17 pages plus a title page. Copies of the report can be found at the Dartmouth College library and at Stanford University. The first 5 pages state the purpose of the study and give the names and interests of the four who proposed the study. In the interest of brevity, this article summarizes the last 12 pages, which contain bibliographical statements of the proposals.

We propose that a 2-month, 10-man study of artificial intelligence be carried out during the summer of 1956 at Dartmouth College in Hanover, New Hampshire. The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use lan-

guage, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves. We think that a significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer.

### 1. Automatic Computers

If a machine can do a job, then an automatic calculator can be programmed to simulate the machine. It may be argued that present computers may be insufficient to simulate many of the higher functions of the mind. This may be true, but it is not because of lack of machine capacity, but our inability to write programs taking full advantage of what we have.

### 2. How Can a Computer be Programmed to Use a Language

It may be speculated that a large part of human language consists of words according to rules of reasoning and rules of conjecture. From this point of view, learning a generalization consists of admitting a new

"We propose that a 2-month, 10-man study of artificial intelligence be carried out during the summer of 1956 at Dartmouth College in Hanover, New Hampshire. The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves. We think that a significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer."

# L'intelligence artificielle selon John McCarthy

*"It is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable."*

— John McCarthy

# Le Test de Turing

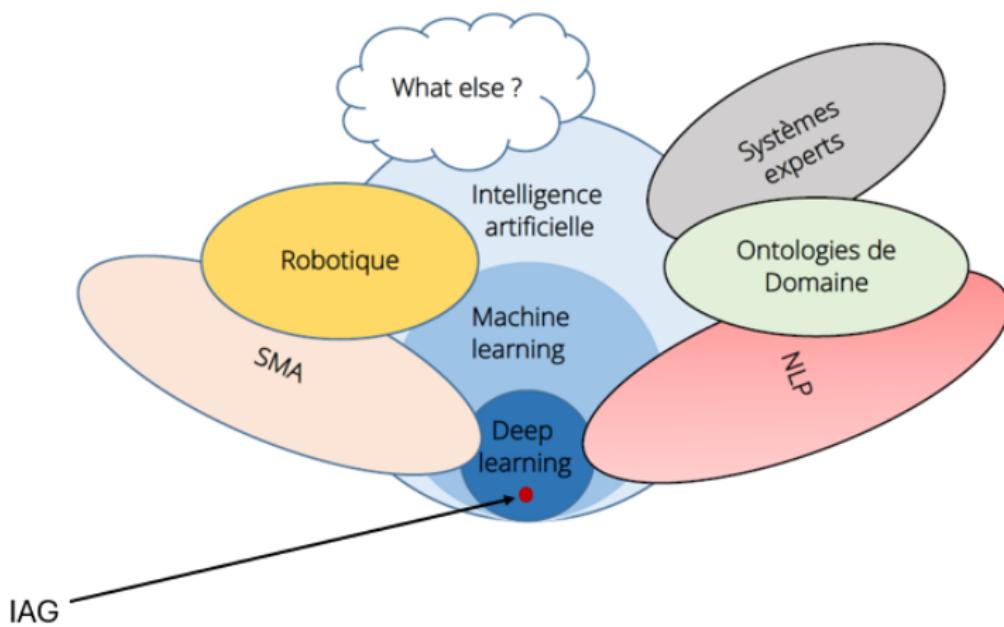
Le Test de Turing, développé par Alan Turing en 1950, est une tentative de mesurer l'intelligence d'une machine, plus précisément de la faculté d'une machine à penser. Cette dernière n'étant pas si évidente à mesurer, le test substitute finalement à la faculté de penser celle de traiter le langage naturel comme un humain.

Les points clés du Test de Turing sont :

- Un interrogateur humain engage une conversation avec un humain et une machine, chacun étant caché de la vue de l'interrogateur.
- Si l'interrogateur ne peut pas déterminer systématiquement quelle est la machine, celle-ci est considérée comme ayant passé le test.
- Le test ne mesure pas la connaissance ou la capacité à être vérifique, mais plutôt la capacité de reproduire le comportement humain.

# Définition pragmatique de l'intelligence artificielle

Il s'agit d'un ensemble de techniques qui permettent à la machine d'accomplir des tâches qui requièrent traditionnellement une intelligence humaine.



# IA forte vs IA faible

La distinction entre IA forte et IA faible se réfère à deux approches conceptuelles différentes dans le domaine de l'intelligence artificielle.

## IA faible :

- Aussi connue sous le nom d'IA "étroite", elle est conçue pour effectuer des tâches spécifiques et ne possède pas de conscience.
- Les systèmes d'IA faible agissent et réagissent uniquement en fonction des instructions programmées et des algorithmes spécifiques.
- Exemples : assistants virtuels, systèmes de recommandation, reconnaissance vocale.

## IA forte :

- Vise à créer des machines dotées de conscience, de compréhension et d'esprit, similaires à l'intelligence humaine.
- L'IA forte serait capable d'apprendre, de raisonner, de résoudre des problèmes et de prendre des décisions indépendamment.
- À ce jour, l'IA forte reste un objectif à atteindre, qui fait l'objet de recherches intensives.

# Eléments de machine learning et de deep learning

# Qu'est-ce que la machine learning ?

L'apprentissage automatique est une branche de l'intelligence artificielle qui consiste à doter les machines de la capacité d'apprendre à partir de données sans que celles-ci ne soient explicitement programmées pour exécuter des tâches spécifiques.

- **IA discriminative** : Modélise la frontière entre les classes.

*Exemples* : régression logistique, SVM, réseaux de neurones pour la classification.

- **IA générative** : modélise la distribution des données pour en générer de nouvelles.

*Exemples* : GAN, VAE, modèles de langage comme GPT.

- **Apprentissage par renforcement (Reinforcement Learning)** : apprend à prendre des décisions séquentielles dans un environnement.

L'agent n'imiter pas ou ne classe pas, il agit pour maximiser une récompense.

*Exemples* : Q-learning, PPO, RLHF pour le fine-tuning des LLMs.

# Machine learning discriminatif

L'apprentissage automatique est une branche de l'intelligence artificielle qui consiste à doter les machines de la capacité d'apprendre à partir de données sans que celles-ci ne soient explicitement programmées pour exécuter des tâches spécifiques.

Le machine learning englobe deux grandes familles d'apprentissage :

- **Supervisé** : Les algorithmes apprennent à partir de données étiquetées pour faire des prédictions ou classifications.
- **Non supervisé** : L'apprentissage est effectué sur des données non étiquetées pour trouver des structures cachées.

# L'apprentissage supervisé

**L'apprentissage supervisé** consiste à apprendre un modèle qui associe une étiquette (*label*) à un ensemble de caractéristiques (*features*).

- **Inputs** : un jeu de données *annotées* pour entraîner le modèle.
  - Exemple : des textes (tweets, etc.) avec les *sentiment* associés, positifs ou négatifs.
- **Output** : une étiquette pour un point de donnée inconnu par le modèle.

L'apprentissage supervisé se décline lui-même en deux grandes familles :

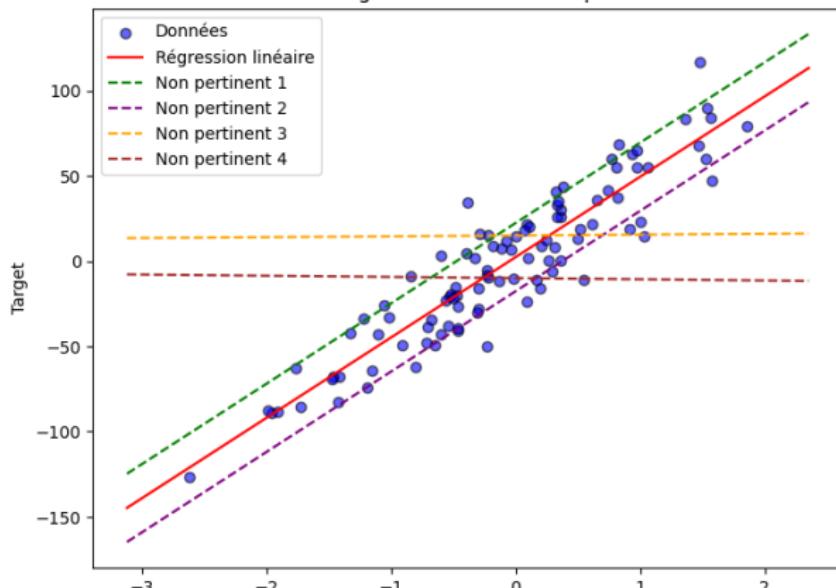
- **La classification** : prédire une catégorie ou une classe.
  - Exemple : prédire l'étiquette d'une image (chat, chien, etc.), le sentiment associé à un texte, le centre d'intérêt d'un client à partir de ses commentaires, etc.
- **La régression** : prédire une valeur continue (un nombre réel typiquement).
  - Exemple : prédire le prix d'un appartement, la lifetime value d'un client, etc.

# Régression linéaire simple

Soit un ensemble de  $n$  observations  $x_1, x_2, \dots, x_n$  avec les labels correspondants  $y_1, y_2, \dots, y_n$ , on cherche le modèle linéaire qui ajuste le mieux ces données.

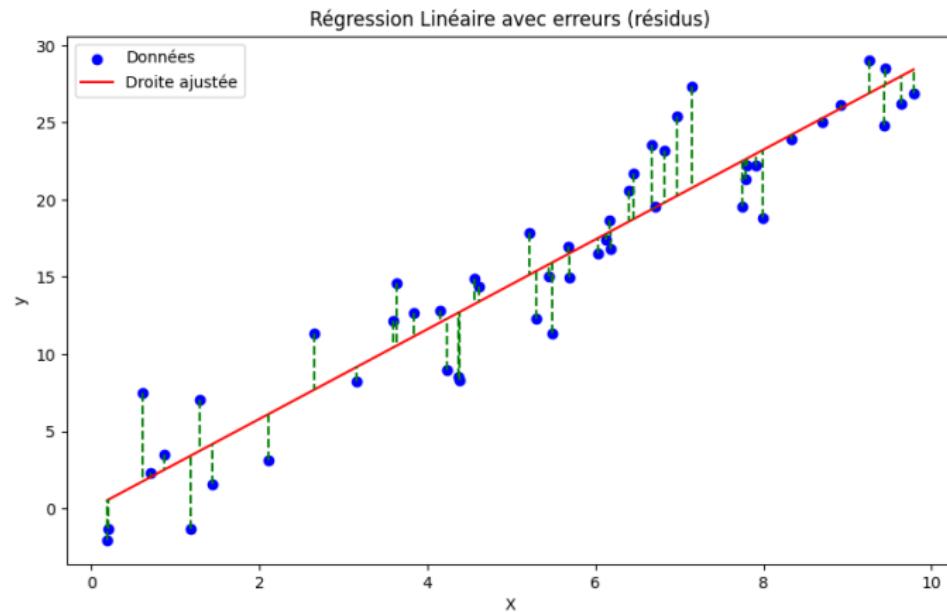
$$\hat{y} = \beta_0 + \beta_1 x$$

Illustration de la régression linéaire avec plusieurs modèles

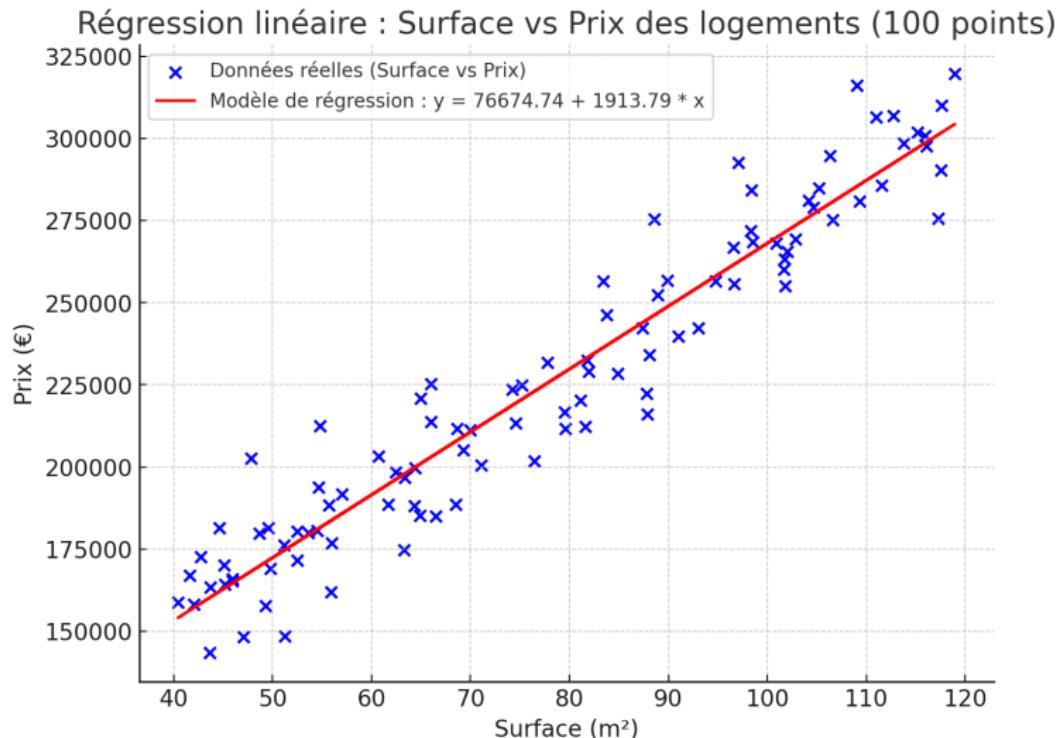


## Minimisation du risque empirique 1/2

L'erreur de prédiction pour la  $i$  ième observation est :  $e_i = y_i - \hat{y}_i$ . où  $\hat{y}_i = \beta_0 + \beta_1 x_i$ .



# Prédire le prix en fonction de la surface : inférence



# L'apprentissage non supervisé

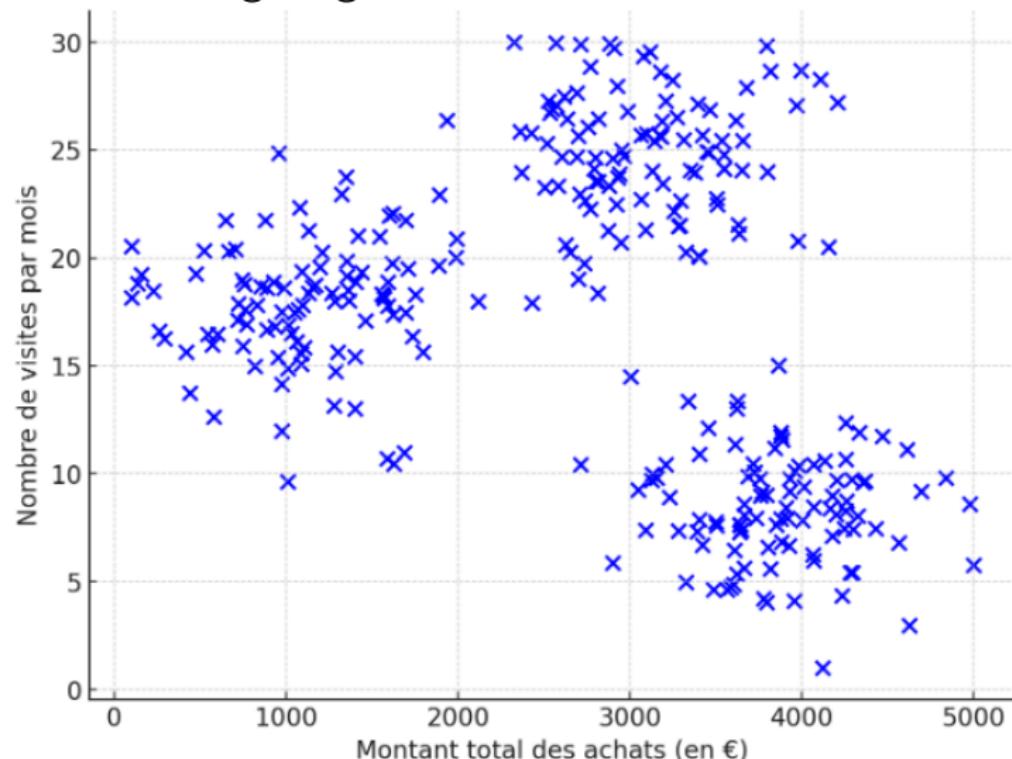
**L'apprentissage non supervisé** se réfère à l'utilisation de modèles d'apprentissage automatique pour identifier des patterns et des structures dans des données qui ne sont pas étiquetées.

Principales typologies de l'apprentissage non supervisé :

- **Clustering** : Regroupement de points de données similaires ensemble.  
Exemple : segmentation de marché, regroupement social.
- **Détection d'anomalies** : Déetecter des observations dont les caractéristiques sont inhabituelles par rapport à la majorité.

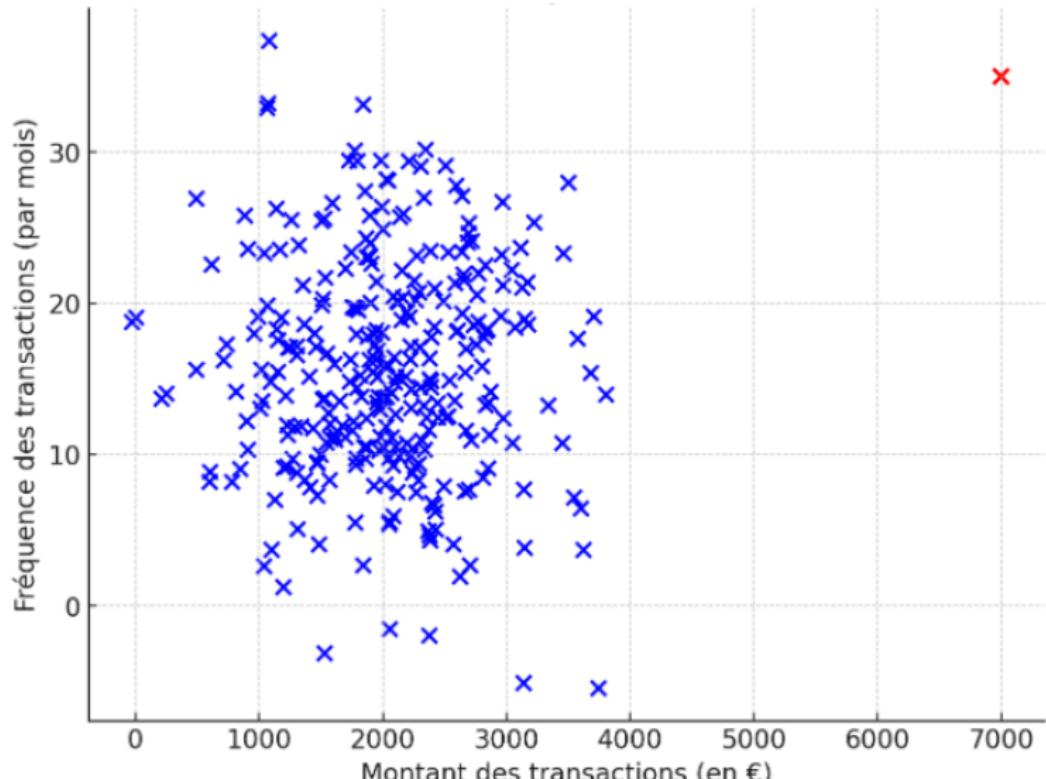
# Clustering

## Exemple de clustering : segmentation clients



# Détection d'anomalies

## Exemple de détection d'anomalies : fraude bancaire



# Eléments de deep learning

# Brève histoire du deep learning

## 1940-1980 : Fondations

- 1943 : Neurone artificiel (McCulloch & Pitts).
- 1986 : Rétropropagation (Rumelhart, Hinton, Williams).

## 1990-2000 : Hiver

- Manque de données et puissance de calcul.
- Progrès en théorie : SVM, modèles bayésiens.

## 2010-2020 : Renaissance

- 2012 : AlexNet révolutionne la vision par ordinateur.

## Depuis 2020 : Multimodalité et LLM

- 2017 : Transformer, base des LLM.
- 2018 - 2020 : GPT-3, BERT.
- 2022 : ChatGPT...

# Introduction aux réseaux de neurones

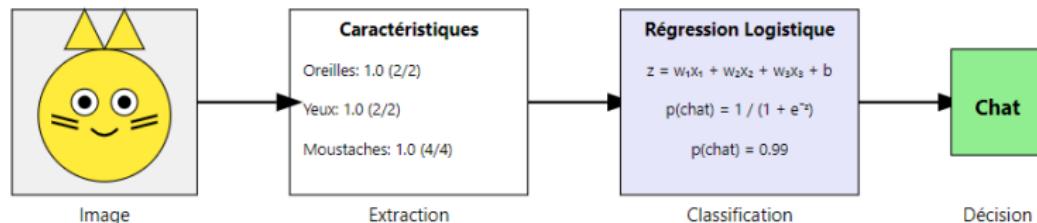
**Définition :** Les réseaux de neurones sont des modèles computationnels inspirés par le fonctionnement des neurones dans le cerveau humain. Ils sont capables d'apprendre des tâches complexes en modélisant des relations non linéaires entre les entrées et les sorties.

## Caractéristiques :

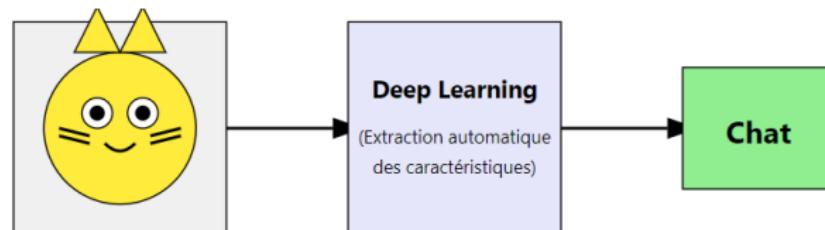
- **Extraction automatique des features** : Capacité d'adaptation et d'extraction des features à partir des données sans programmation explicite.
- **Modélisation non linéaire** : Aptitude à capturer des relations complexes dans les données.
- **Modélisation en grande dimension** : Les modèles de deep learning sont particulièrement adaptés pour les données en grande dimension (images, texte, etc.).
- **Flexibilité** : Applicables à un large éventail de tâches et de typologies de données (images, langage naturel, données graphiques, etc.).

# Extraction de features

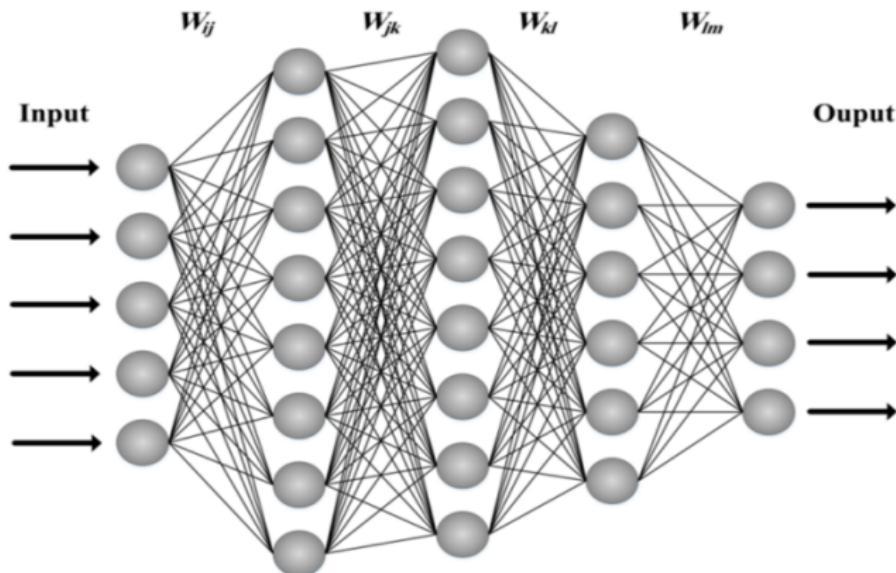
## Extraction de features en machine learning "classique"



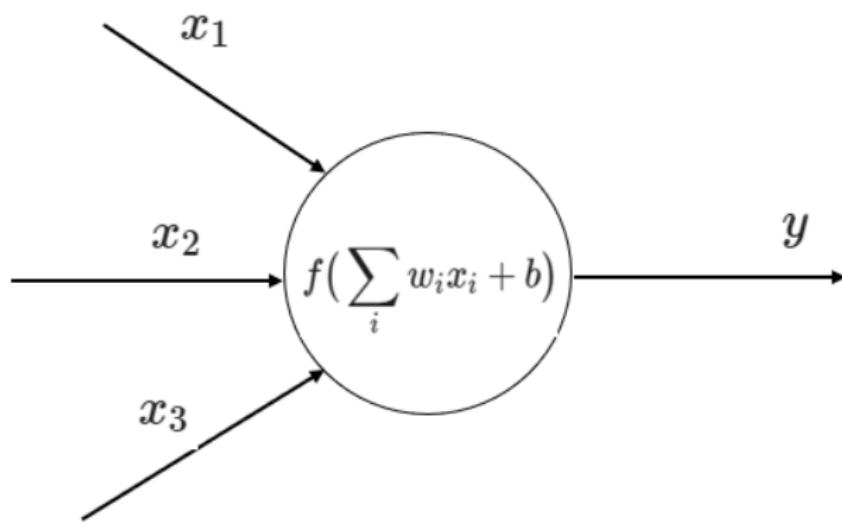
## Extraction de features en deep learning



# Illustration d'un réseau de neurones classique

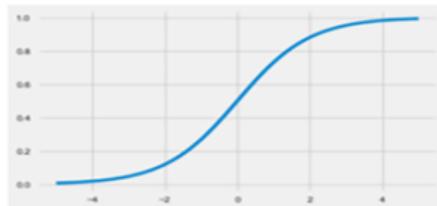


# Neurone aritificial

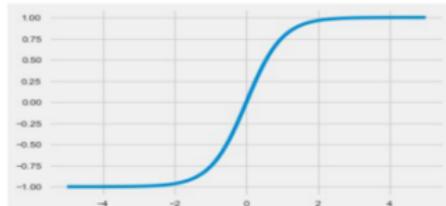


# Illustration des fonctions d'activation

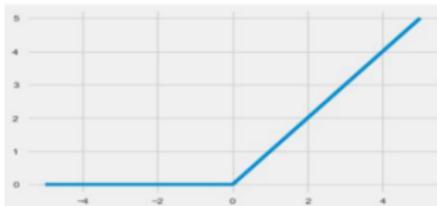
Sigmoïde



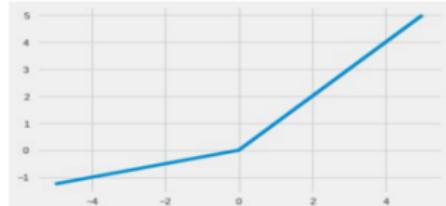
Tanh



ReLU



ReLU paramétrique



# La fonction SoftMax dans la classification dans la dernière couche

Le SoftMax transforme une liste de scores en **probabilités de classes**. C'est la dernière étape d'un modèle de classification multilclasses.

**Formule :**

$$P(y_i) = \frac{e^{s_i}}{\sum_{j=1}^K e^{s_j}}$$

où  $s_i$  représente le score associé à la classe  $i$ , et  $K$  le nombre total de classes.

**Exemple :**

$$\text{Image} \rightarrow \begin{cases} \text{chat : 0.85} \\ \text{chien : 0.10} \\ \text{oiseau : 0.05} \end{cases}$$

→ Le modèle prédit la classe la plus probable (\*chat\*), tout en conservant une estimation pour les autres.

# Intelligence artificielle générative (IAG)

# Qu'est-ce que l'IA générative ?

**Définition :** L'intelligence artificielle générative (IAG) désigne les modèles capables de produire de nouvelles données (texte, image, son, code) qui imitent ou enrichissent des données existantes.

**Idée clé :** alors que le machine learning classique prédit, l'IAG crée.

**Caractéristiques :**

- Modèles entraînés sur des volumes massifs de données hétérogènes.
- Génération conditionnée par un prompt (instruction utilisateur).
- Capacité à généraliser au-delà des données vues pendant l'entraînement.

# Deux grandes familles d'IA générative

## IA générative pour le langage :

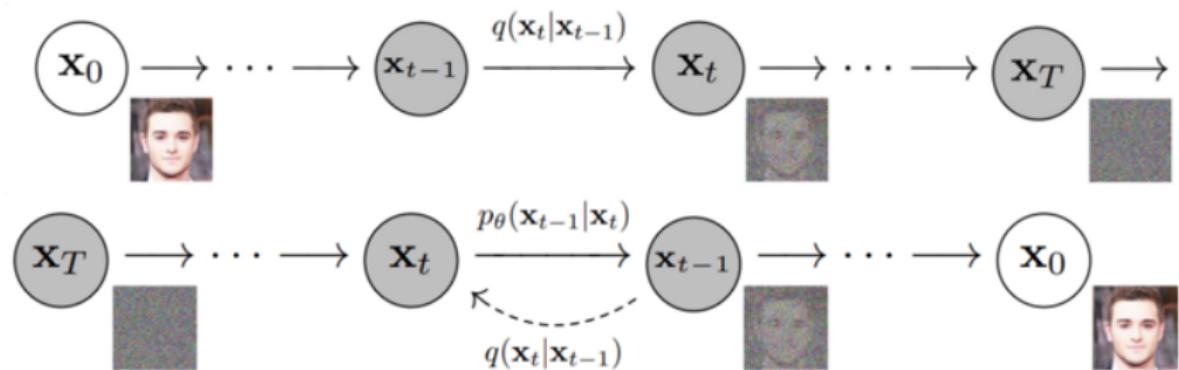
- Modèles de langage de grande taille (LLMs).
- Génération de texte cohérent et contextuel.
- Applications : chatbots, rédaction assistée, analyse de documents, génération de code.
- Exemples : GPT-4, Claude, LLaMA, Mistral.

## IA générative pour les images :

- Modèles de diffusion et GANs.
- Création d'images réalistes ou stylisées à partir de descriptions textuelles.
- Applications : design, marketing, prototypage, art numérique.
- Exemples : DALL·E, Stable Diffusion, MidJourney.

# Génération d'images

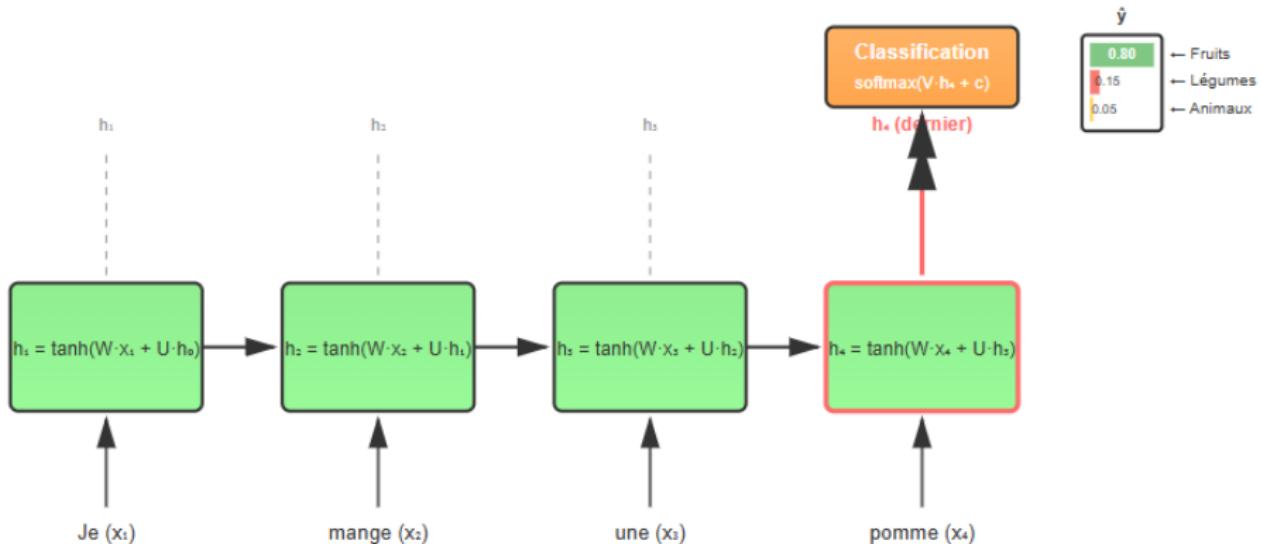
La génération d'images utilise généralement des techniques dites de diffusion.



# Large Language Models (LLMs)

## Anciens modèles pour le traitement du langage : RNN

- **Mémoire à court terme** : Les réseaux de neurones récurrents (RNN) ont la capacité de se "souvenir" d'informations passées grâce à leurs connexions récurrentes.
  - **Traitements de séquences** : Ils sont particulièrement adaptés pour des tâches où les données sont séquentielles et où le contexte est important.



## Attention Is All You Need

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

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

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

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

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

**Aidan N. Gomez\*** †  
University of Toronto  
aidan@cs.toronto.edu

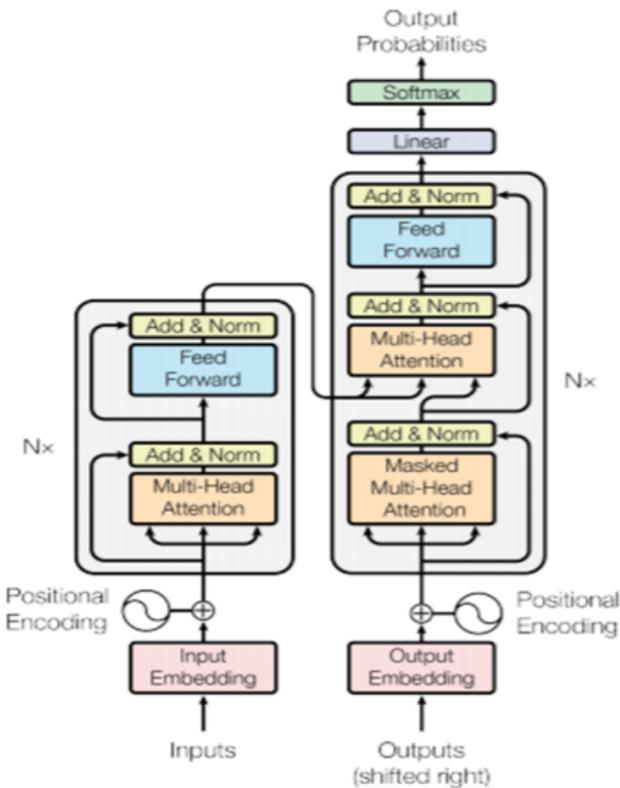
**Lukasz Kaiser\***  
Google Brain  
lukasz.kaiser@google.com

**Illia Polosukhin\*** ‡  
illia.polosukhin@gmail.com

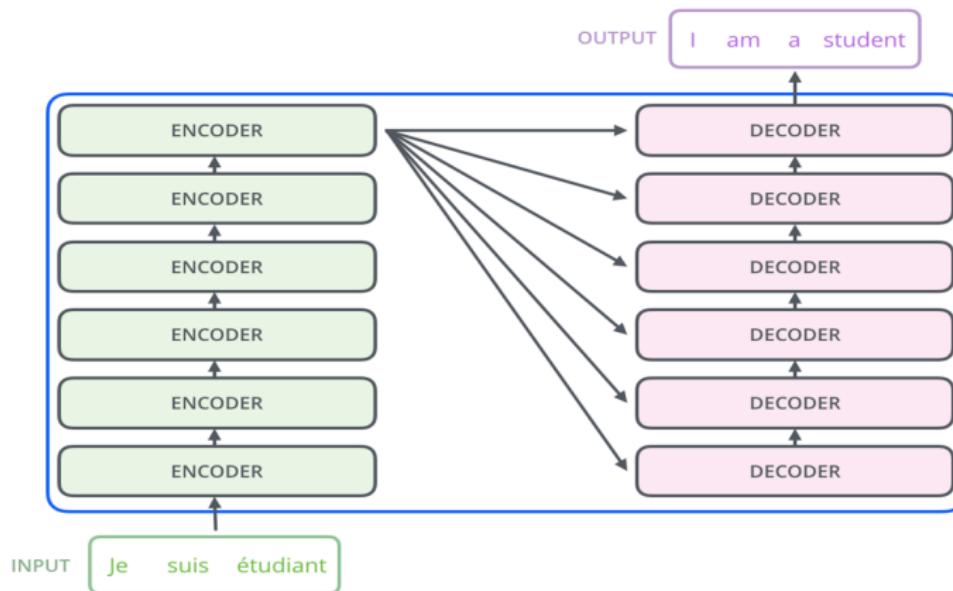
### Abstract

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

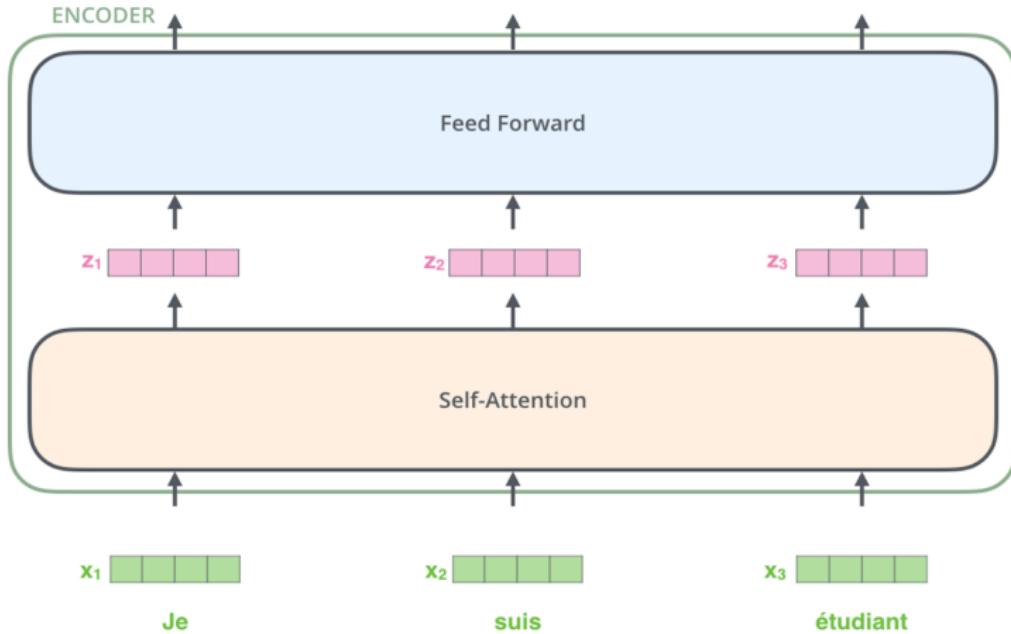
# Transformer originel



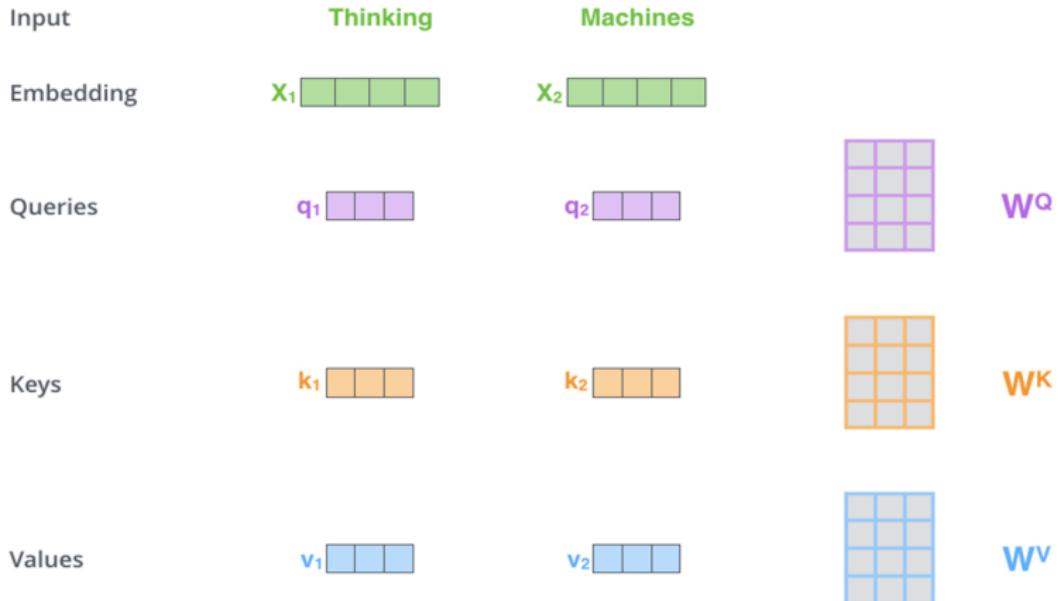
## Mécanisme de cross-attention



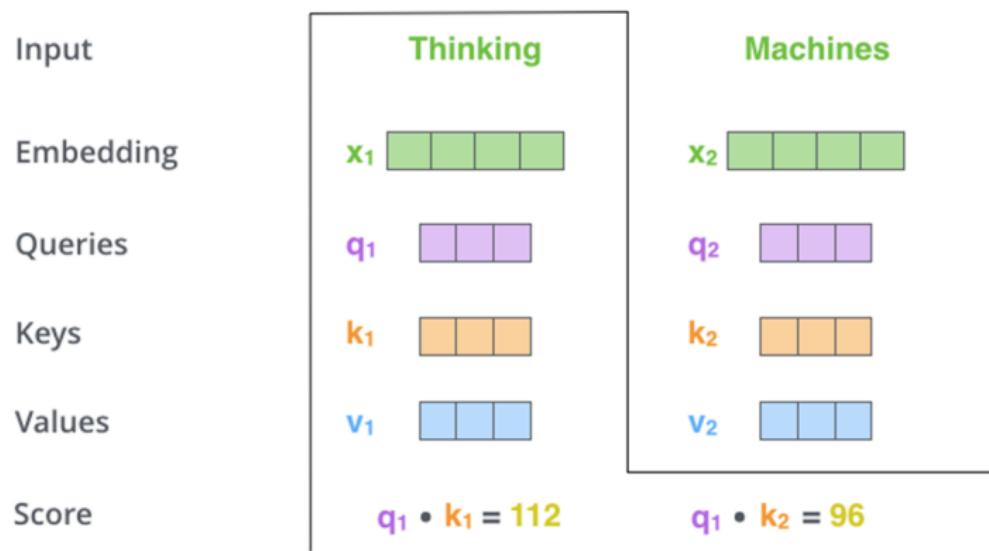
## Mécanisme de self-attention



# Fonctionnement du mécanisme de self-attention



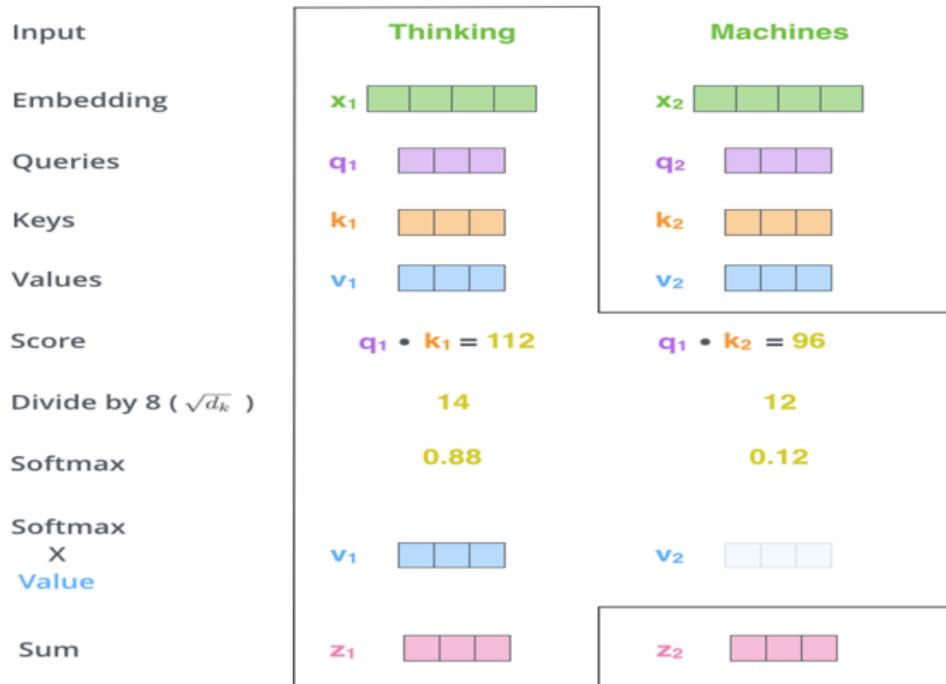
# Calcul des scores de self-attention



# Calcul des scores de self-attention

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

# Calcul de la nouvelle représentation



# Performances du tranformer originel

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

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

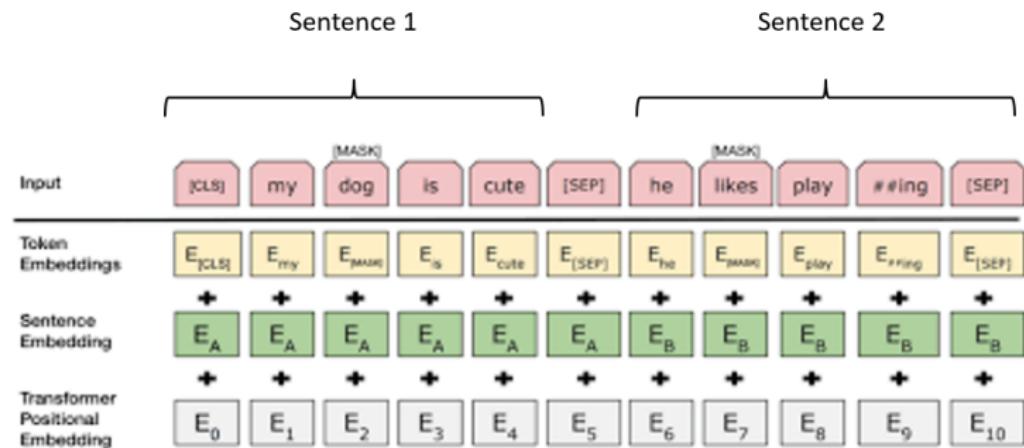
Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

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

# BERT

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

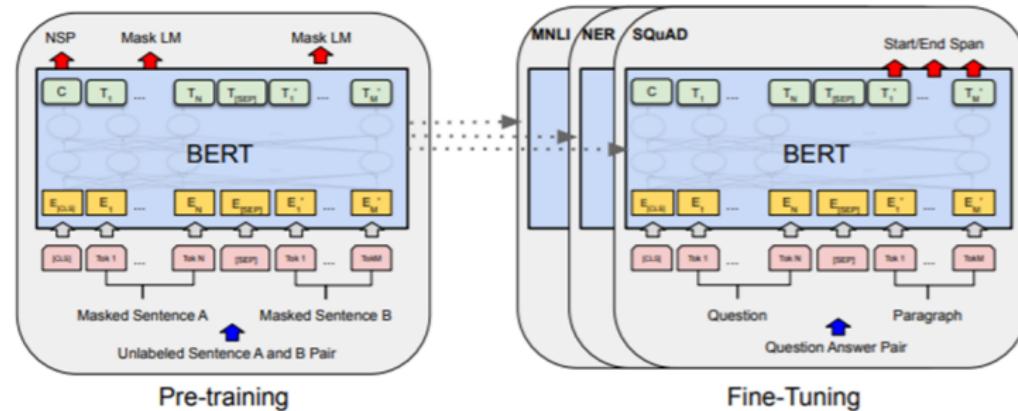
## Pré-entraînement



## Fine-tuning de BERT pour des tâches spécifiques

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

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



# Performances de BERT

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

Deux modèles :

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

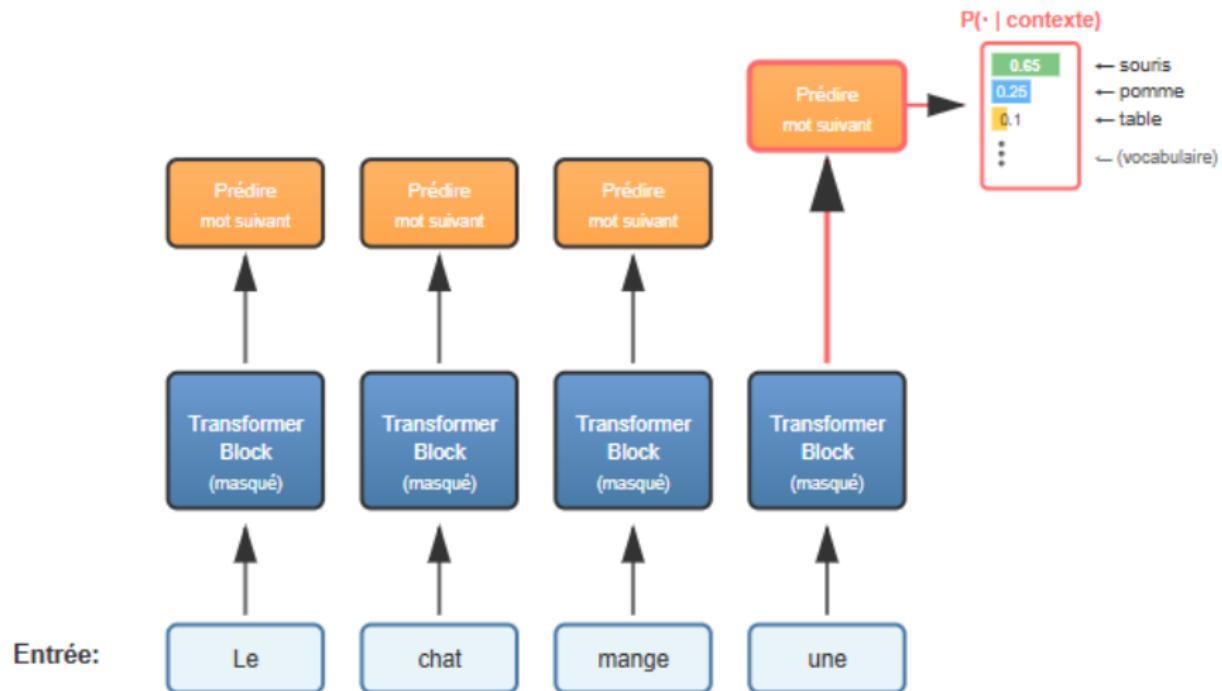
System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Average
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
BERT 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

# Modèles génératifs (GPT)

*« Ce que je ne peux pas créer,  
je ne le comprends pas. »*

– Richard Feynman

## Fonctionnement des modèles génératifs



# Le paramètre de température

**Définition** La température ( $T$ ) contrôle le degré d'aléa dans les réponses générées. Elle agit sur la distribution de probabilité des mots avant l'échantillonnage.

**Formule mathématique :**

$$P(x_i) = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}}$$

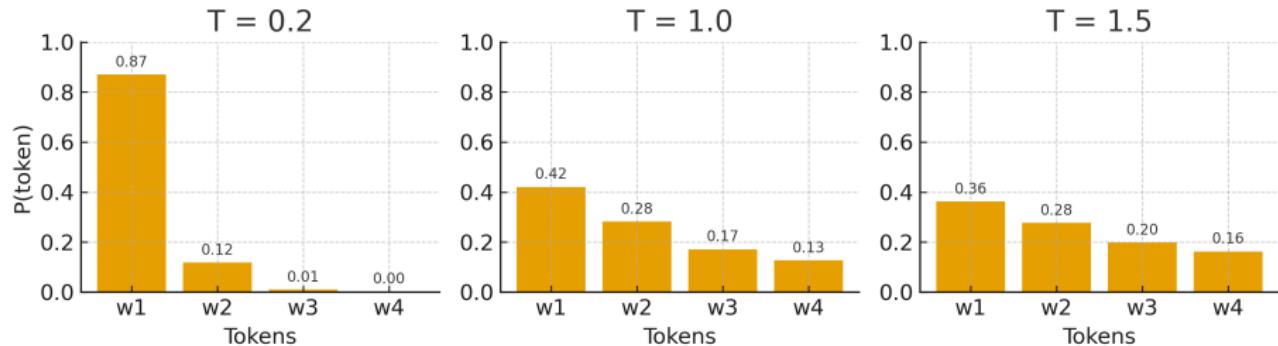
- $T < 1 \rightarrow$  distribution plus concentrée, réponses plus prévisibles
- $T > 1 \rightarrow$  distribution plus plate, réponses plus variées et créatives

**Autrement dit :**

- Basse température  $\rightarrow$  précision et cohérence.
- Haute température  $\rightarrow$  imagination et diversité.

# Illustration graphique de l'effet de la température

Effet de la température sur la distribution des probabilités



# La tokénisation

**Problème de départ :** Les modèles de langage ne peuvent pas traiter directement du texte brut : les mots, les accents, la ponctuation et les variations sont trop nombreux (plusieurs millions de combinaisons possibles).

**La tokénisation :** Elle consiste à découper le texte en **unités réutilisables et limitées en nombre**, appelées *tokens*. Chaque token est ensuite représenté par un identifiant numérique que le modèle peut apprendre à prédire.

## Intérêt de la tokénisation :

- Réduire la complexité du vocabulaire : on passe de millions de mots à quelques dizaines de milliers de tokens.
- Gérer les mots inconnus ou rares en les décomposant en sous-unités connues.
- Offrir une représentation numérique uniforme du langage, compatible avec les réseaux de neurones.

# La composition des données d'entraînement des modèles

**Principe général** Les modèles de langage sont entraînés sur d'immenses corpus de texte — souvent plusieurs milliers de milliards de tokens — issus de sources très diverses pour couvrir la langue, les faits et les styles.

## Sources typiques de données :

- **Web public** : pages web, articles, forums, Wikipédia (ex. Common Crawl).
- **Livres et documents** : œuvres littéraires, articles scientifiques, rapports techniques.
- **Données de code** : dépôts GitHub, notebooks, documentation technique.
- **Conversations et dialogues** : jeux de données de chat, discussions de forums, données annotées par des humains.
- **Données synthétiques** : textes générés par d'autres modèles pour enrichir certaines thématiques.

## Équilibre recherché :

- Diversité linguistique et thématique (langues, registres, domaines).
- Qualité et cohérence (filtrage du spam, des doublons, des contenus toxiques).
- Représentation équilibrée entre langage courant, technique et conversationnel.

# GPT-3

- Développé par OpenAI, publié en 2020.
- Modèle auto-régressif basé sur l'architecture Transformer.

Caractéristique	Valeur / Détail
Date	2020
Paramètres	175 milliards
Corpus d'entraînement	Environ 300 milliards de tokens
Contexte maximum	2048 tokens
Ressources GPU	Plusieurs centaines de GPU (type V100)
Coût d'entraînement (estim.)	4,6 M\$ à 12 M\$ (selon les sources)

- **Applications** : génération de texte, Q&R, résumé, traduction, assistance à la programmation.

# Le few-shot learning

- Les grands modèles (comme GPT-3) peuvent réaliser des tâches **sans être explicitement réentraînés** dessus.
- Le principe repose sur **l'utilisation de quelques exemples** (exemples étiquetés) directement dans le prompt ou l'entrée.
- **Zero-shot** : aucune démonstration fournie, le modèle doit comprendre la tâche à partir de sa connaissance générale.
- **One-shot / Few-shot** : on inclut un nombre très réduit d'exemples (un ou quelques-uns) pour guider le modèle dans la résolution de la tâche.
- Exemple :

English : "cat" → French : "chat"

English : "house" → French : "maison"

English : "car" → French : "voiture"

English : "tree" → French :

# ChatGPT

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

- ① Pré-entraînement sur un corpus volumineux :** Comme GPT-3, ChatGPT est d'abord pré-entraîné sur un vaste ensemble de données textuelles, englobant un large éventail de la littérature disponible sur Internet, pour apprendre une compréhension générale du langage.
- ② Fine-tuning supervisé :** Ensuite, ChatGPT est affiné sur des dialogues spécifiques pour améliorer ses compétences conversationnelles. Cette étape utilise des paires question-réponse et des conversations pour enseigner au modèle des structures de dialogue et des réponses contextuellement appropriées.
- ③ Reinforcement Learning from Human Feedback (RLHF) :** Utilisation de techniques de renforcement pour ajuster les réponses du modèle basées sur les préférences et les corrections fournies par des évaluateurs humains, raffinant davantage la pertinence et la naturalité des réponses.

# LLMs propriétaires vs Open Source

- **Modèles propriétaires**

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

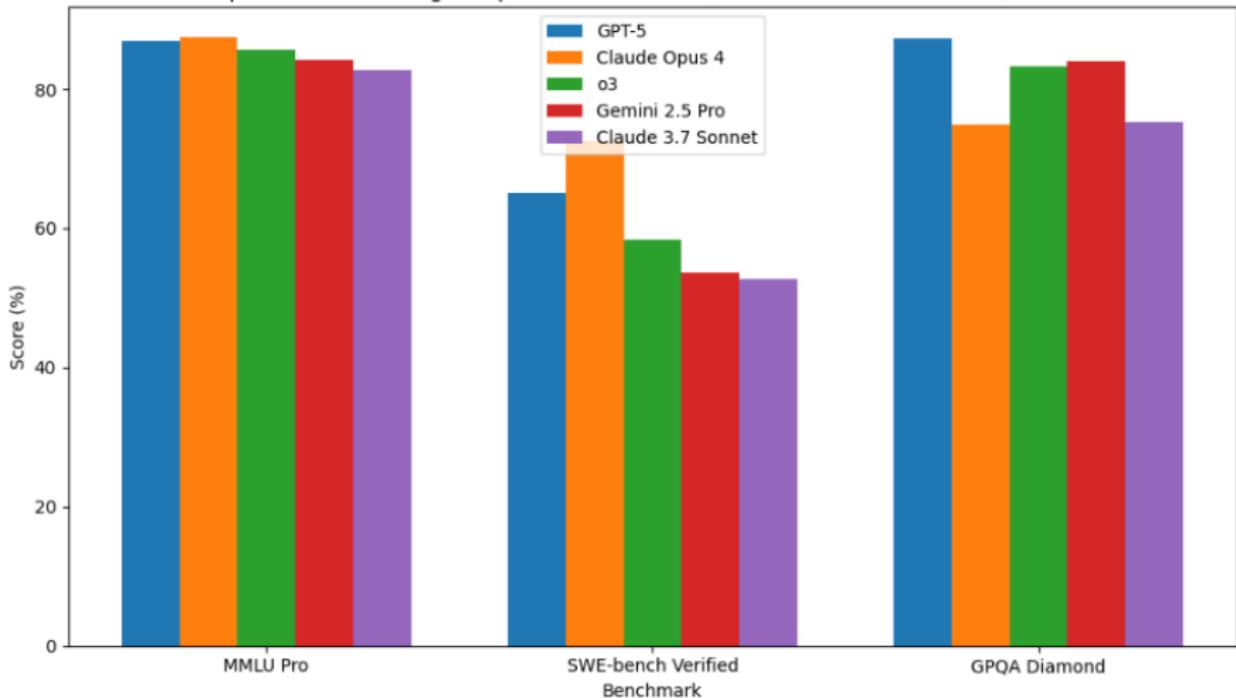
- **Modèles open source**

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

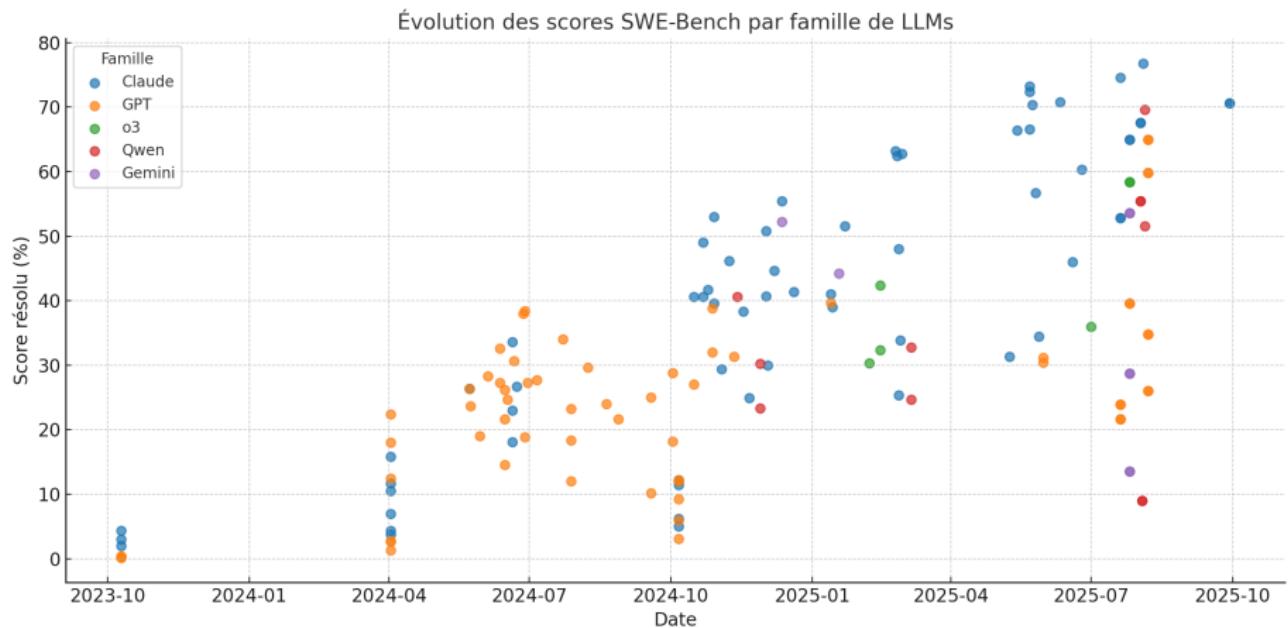
**Conclusion :** Choisir entre contrôle, flexibilité et performances optimales.

# Comparaison des performances des LLMs : benchmarks classiques

Comparaison des LLMs grand public sur MMLU Pro, SWE-bench Verified et GPQA Diamond



# Evolution des performances des LLMs sur SWE-Bench



# Introduction au prompting

**Définition :** Le prompting est l'art de formuler des instructions à un modèle de langage afin d'obtenir une réponse pertinente et de qualité.

**Idée clé :** La qualité de la sortie dépend fortement de la manière dont l'entrée est rédigée.

**Enjeux :**

- Obtenir des résultats précis et cohérents.
- Réduire les risques d'ambiguités et d'hallucinations.
- Adapter le style et le niveau de détail de la réponse.

# Techniques de base du prompting

- **Prompt clair et explicite** : éviter les formulations vagues.
- **Contexte** : fournir des informations supplémentaires pour orienter le modèle.
- **Format attendu** : indiquer la structure de sortie souhaitée (liste, tableau, texte court).
- **Exemples** : montrer un ou plusieurs cas pour guider la génération.

Exemple : Au lieu de "C'est quoi le machine learning?", écrire "Donne une explication détaillée du fonctionnement du machine learning, illustrée avec un exemple".

# 1. Le ton change la structure du raisonnement

## Principe

Le ton du prompt influence non seulement le style, mais aussi la structure logique de la réponse.

## Effets observés

- Ton calme et analytique : produit une écriture déductive et ordonnée, où le modèle explicite les causes et développe les idées avant de conclure.
- Ton urgent ou tendu : génère une écriture télégraphique, orientée vers l'action, avec phrases courtes et listes de décisions.

## Exemples

« Explique posément, en suivant un raisonnement logique. » « Vite : résume-moi ce que je dois dire au client dans deux minutes. »

## À retenir

Le ton est un levier cognitif : il modifie la manière dont le modèle raisonne avant même d'écrire

## 2. Les métaphores encadrent la pensée

### Principe

La métaphore impose un cadre de raisonnement : elle force le modèle à réencoder le concept dans une structure familière.

### Effets observés

- Elle permet d'aborder des notions abstraites par transfert analogique.
- Elle structure naturellement la réponse selon les dimensions de la métaphore (par exemple : ingrédients, recettes, cuisson).

### Exemples

« Explique le machine learning comme si c'était une cuisine. » « Explique l'optimisation comme une randonnée avec des étapes et des sommets. »

### À retenir

La métaphore agit comme un cadre cognitif qui oriente la logique et le vocabulaire du texte.

### 3. Les rôles activent des corpus latents

#### Principe

Attribuer un rôle au modèle sélectionne une manière de parler et de raisonner issue de son corpus.

#### Effets observés

- Historien : met l'accent sur la chronologie et les causes profondes.
- Ingénieur sceptique : insiste sur les hypothèses, les tests et les limites.
- Poète critique : privilégie les images, les tensions et le rythme.

#### Exemples

« Agis comme un historien en expliquant les causes profondes. » « Réponds comme un ingénieur qui doute de ses propres hypothèses. »

#### À retenir

Le rôle agit comme un filtre stylistique et culturel qui oriente la réponse sans en changer le fond.

## 4. La précision du contexte vaut plus que la longueur

### Principe

Un contexte précis et incarné produit de meilleures réponses qu'une longue liste d'instructions abstraites.

### Effets observés

- Le modèle comprend mieux la situation lorsqu'elle est ancrée dans un cadre concret.
- Les réponses deviennent plus pertinentes et plus naturelles.

### Exemples

« Tu écris à ton supérieur, ton ferme mais respectueux, objectif : obtenir une validation avant 18 h. » « En 200 mots, identifie trois risques et formule une demande de décision. »

### À retenir

Un cadre incarné aide le modèle à raisonner comme dans une situation réelle.

## 5. Les verbes d'action changent la dynamique

### Principe

Le verbe choisi détermine le type de raisonnement attendu.

### Effets observés

- Explique : logique de cause à effet.
- Montre : logique d'exemple et d'illustration.
- Révèle : logique d'enquête ou de découverte.

### Exemples

« Explique en trois causes principales. » « Montre deux exemples concrets et un contre-exemple. » « Révèle l'élément caché qui change la conclusion. »

### À retenir

Le verbe oriente la structure du texte et la nature du raisonnement.

## 6. Le silence comme outil

### Principe

Ne pas tout préciser laisse au modèle une marge d'interprétation, propice à la créativité.

### Effets observés

- Le modèle complète avec des régularités contextuelles, souvent pertinentes.
- Trop de contraintes bloque la pensée et conduit à des réponses formatées.

### Exemples

« Donne trois options, dont une plus audacieuse, sans les justifier. » « Rédige un brouillon brut, sans titres, avec les idées clés uniquement. »

### À retenir

Le vide agit comme un espace de respiration qui stimule la créativité.

## 7. Les émotions modulent l'énergie du contenu

### Principe

Le ton émotionnel du prompt influence le rythme et la profondeur de la réponse.

### Effets observés

- Colère : texte direct, tranchant, orienté vers la critique.
- Tristesse : rythme lent, ton grave, introspection.
- Joie : rythme rapide, style fluide et enthousiaste.
- Admiratio : langage figuré et ample.

### Exemples

« Sans détour, dis-moi ce qui ne va pas dans cette stratégie. » << Parle avec gravité des risques systémiques en trois points. »

### À retenir

L'émotion structure le raisonnement en modulant l'énergie et la focalisation.

## 8. Le paradoxe pousse la nuance

### Principe

Introduire une contradiction dans le prompt oblige le modèle à produire une réponse nuancée.

### Effets observés

- Le modèle cherche un équilibre entre deux pôles logiques.
- Il produit une pensée dialectique plutôt qu'un simple commentaire.

### Exemples

« Explique pourquoi l'IA est à la fois une menace et une promesse. » <> « Montre en quoi le progrès peut être à la fois un risque et une chance. »

### À retenir

Le paradoxe favorise la nuance et la réflexion véritable.

# 1. Le prompt comme dispositif de situation

- Les modèles réagissent mieux à une situation incarnée qu'à une instruction abstraite. → Exemple : « Vous êtes face à un comité sceptique, vous devez défendre votre position. »
- L'efficacité vient d'un processus itératif : cadrer, ajuster, affiner le ton et la forme.
- Les consignes les plus performantes précisent l'objectif plutôt que les moyens. → Exemple : « Rédige un texte qui donne envie de s'inscrire à la formation. »

*Un bon prompt crée un contexte de pensée, pas une suite d'ordres.*

## 2. Créer un cadre ouvert mais dirigé

- Laisser une marge d'interprétation encourage la créativité sans perdre le contrôle. → Exemple : « Propose trois options, dont une plus audacieuse. »
- La reformulation est un outil d'ancrage : elle stabilise le style et le ton. → Exemple : « Conserve ce ton, mais rends le propos plus concret. »
- Les prompts dialectiques favorisent la nuance et la profondeur. → Exemple : « Expose les arguments pour et contre l'usage de l'IA générative. »

*L'ouverture maîtrisée est plus féconde que la prescription exhaustive.*

### 3. Soigner l'entrée du modèle

- La première phrase fixe le registre cognitif du texte. → « Agis comme... » crée un ton professionnel. → « Imagine que... » stimule la créativité. → « Explique-moi... » installe un registre analytique.
- Certains mots volontairement flous activent des régularités de style. → Exemple : « Écris un texte juste » conduit souvent à une écriture sobre et sincère.

*Les premiers mots d'un prompt déterminent souvent la posture du modèle.*

## 4. Guider la génération au fil du dialogue

- Les relances directionnelles (« Va plus loin », « Simplifie sans appauvrir ») orientent la production sans l'alourdir.
- Demander au modèle de se corriger déclenche une forme de métacognition.  
→ Exemple : « Relis ta réponse comme un relecteur exigeant. Que modifierais-tu ? »
- Le feedback positif est plus structurant que la contrainte négative. → Il permet au modèle d'ajuster progressivement sa cohérence stylistique.

*Le dialogue, bien conduit, devient un véritable apprentissage contextuel.*

## 5. Dialoguer avec une voix, pas avec une machine

- Les modèles ne « obéissent » pas : ils estiment la suite la plus probable. → Il s'agit donc d'orienter plutôt que d'imposer.
- « Tu peux être incisif » produit souvent de meilleurs résultats que « Sois incisif ».
- Initier un ton puis le prolonger (« Continue dans le même style ») garantit la cohérence d'ensemble.
- Les LLMs sont des condensés de mémoire culturelle : plus le langage est humain, plus la réponse le sera.

*Un prompt efficace s'adresse à une voix, pas à un programme.*

## 6. Structurer le prompt : du zero-shot au few-shot

- **Zero-shot** : le modèle exécute la tâche uniquement à partir d'une consigne.  
→ Adapté aux tâches simples ou familières. → Exemple : « Rédige un e-mail de relance professionnel pour un client inactif. »
- **Few-shot** : on fournit au modèle quelques exemples du comportement attendu avant la demande réelle. → Le modèle en déduit la structure, le ton et le registre à reproduire.
- **Chain-of-thought** : on invite le modèle à expliciter son raisonnement avant la conclusion. → Exemple : « Décris ton raisonnement étape par étape avant de proposer la recommandation finale. »

*La structuration du prompt agit comme un guide cognitif : elle oriente la logique, la précision et la cohérence du résultat.*

# Modèles de raisonnement

**Objectif** : améliorer la capacité des LLMs à résoudre des problèmes complexes en rendant explicites les étapes de raisonnement.

## Principe des Chain-of-Thoughts (CoT) :

- Décomposer une question en **étapes intermédiaires logiques**.
- Forcer le modèle à “**penser à voix haute**” avant de donner la réponse finale.
- Exemple : résoudre une équation, planifier une suite d'actions, raisonner sur des données tabulaires.

## Méthodes d'entraînement :

- **Fine-tuning supervisé** sur des datasets annotés avec étapes de raisonnement.
- **Self-consistency** : générer plusieurs chaînes et agréger la réponse la plus fréquente/cohérente.
- **Distillation** : transférer les capacités de raisonnement d'un grand modèle vers un plus petit.

# Agents basés sur les LLMs

**Définition** : un agent est un système qui exploite un LLM comme moteur de raisonnement et de décision, capable d'interagir avec des outils et un environnement externe.

**Caractéristiques principales** :

- **Boucle perception-action** : observe, raisonne, agit, et réévalue.
- **Accès à des outils** : API, bases de données, navigateurs, scripts.
- **Mémoire** : conserve l'historique et les connaissances pertinentes.
- **Planification** : décompose un objectif en sous-tâches exécutables.

# Limites actuelles des LLMs : deux axes majeurs

## 1. Hallucinations

- Génération de contenus inexacts ou inventés, parfois avec une forte confiance.
- Difficulté à distinguer le vrai du plausible, surtout hors du domaine d'entraînement.
- Conséquences : risque de désinformation, perte de fiabilité, nécessité de vérification humaine.

## 2. Non-déterminisme

- Un même prompt peut produire plusieurs réponses différentes.
- Résultats sensibles à la température, au contexte ou à de légères variations de formulation.
- Enjeu pour la reproductibilité et le contrôle qualité des systèmes fondés sur les LLMs.

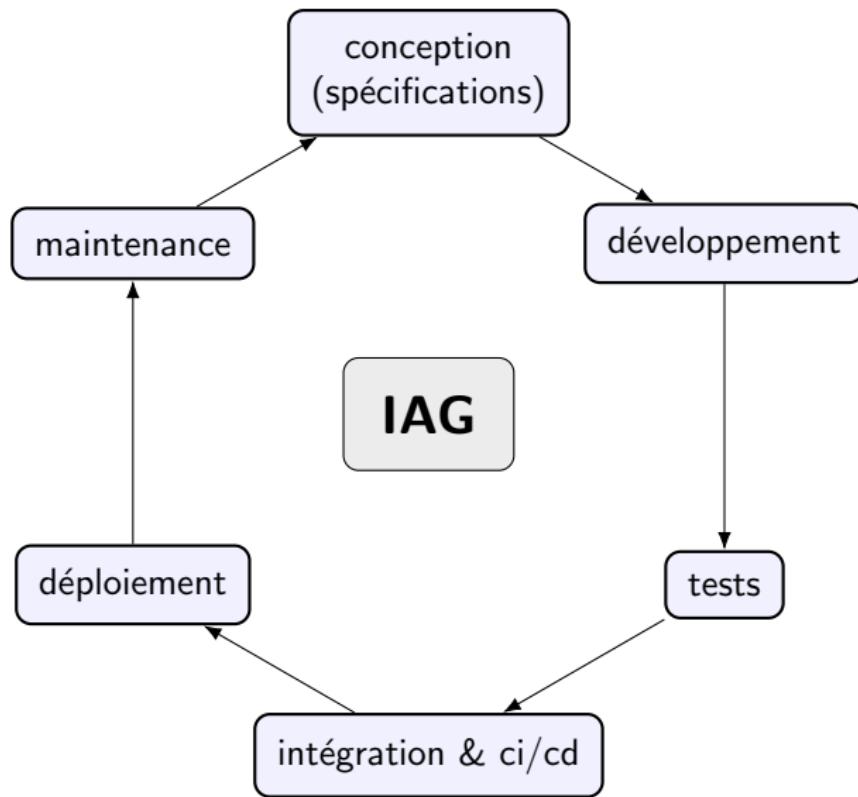
# IAG dans le cycle de développement

# L'IAG dans le cycle de vie logiciel

L'intelligence artificielle générative ne se limite pas à l'autocompléction de code : elle peut s'intégrer à chaque étape du cycle de vie logiciel pour assister les équipes.

- **Conception** : clarification des besoins, spécifications, ébauches d'architecture.
- **Développement** : génération et refactor de code, assistance aux patterns.
- **Tests** : génération de tests unitaires et fonctionnels, détection de cas limites.
- **Intégration & CI/CD** : review automatique de PR, génération de changelog et documentation.
- **Déploiement** : notes de version, assistance à l'infrastructure-as-code.
- **Maintenance** : classification des tickets, suggestions de correctifs, documentation à partir du code.

# IAG dans le cycle de vie logiciel



# Conception et spécifications

**Objectif** : réduire l'ambiguïté et accélérer le passage du besoin métier à la solution technique.

- Transformer une *user story* en endpoints d'API (routes, verbes, schémas JSON).
- Générer une première ébauche d'architecture (services, flux, dépendances).
- Produire un glossaire ou des critères d'acceptation testables.

**Exemple de prompt** : « *À partir de cette user story, propose les endpoints REST, les codes d'erreur et un schéma JSON pour POST /orders.* »

# Développement

**Objectif** : accélérer l'implémentation tout en améliorant la lisibilité et la maintenabilité.

- Génération de fonctions à partir d'une description en langage naturel.
- Refactor : extraction de sous-fonctions, renommage de variables, simplification de structures.
- Suggestions de design patterns adaptés au contexte.

**Exemple** : Avant : fonction monolithique de 80 lignes. Après : découpage en validate(), transform(), persist() avec docstrings et exceptions typées.

# Tests

**Objectif** : améliorer la couverture et détecter les cas limites dès la phase de développement.

- Génération de tests unitaires directement à partir du code.
- Proposition de cas extrêmes : valeurs limites, erreurs réseau, formats inattendus.
- Création de scénarios fonctionnels dérivés des user stories.

**Exemple de prompt** : « Voici la fonction `price(order)`. Génère 8 tests unitaires pytest couvrant remise, TVA, devise inconnue, quantité négative, seuils, valeur nulle. »

# Intégration et CI/CD

**Objectif** : enrichir les pipelines d'intégration continue avec des capacités d'analyse et de génération.

- Review automatique de pull requests : détection de failles de sécurité, de duplications, de style incohérent.
- Génération automatique de tests complémentaires pour valider les commits.
- Production de documentation et de changelogs à partir des messages de commits.

**Exemple** : Un workflow GitHub Actions envoie le *diff* à un LLM, qui commente directement la PR avec ses suggestions.

# Déploiement

**Objectif** : automatiser et simplifier les tâches de mise en production.

- Assistance à la rédaction de scripts *infrastructure-as-code* (Terraform, Ansible).
- Génération automatique des notes de version lisibles à partir des commits.
- Support aux équipes DevOps pour documenter et communiquer les changements.

**Exemple** : À partir de commits techniques, l'IAG rédige un changelog clair pour les utilisateurs finaux.

# Maintenance et support

**Objectif** : accélérer la résolution de problèmes et réduire la dette technique.

- Classification automatique des tickets d'incidents par priorité et catégorie.
- Proposition de correctifs initiaux pour certains bugs.
- Génération de documentation à partir de code existant ou de logs.

**Exemple** : Un log d'erreur est soumis à l'IAG, qui propose un diagnostic et une piste de correction.

# Limites et précautions

## Points de vigilance lors de l'intégration de l'IAG :

- **Qualité variable** : sorties parfois incomplètes ou erronées, besoin de supervision humaine.
- **Sécurité et confidentialité** : attention à l'envoi de code ou de données sensibles.
- **Coût et performance** : appels API facturés au token, temps de latence selon les modèles.
- **Dépendance technologique** : choix entre solutions cloud, open source locales ou hybrides.

**Message clé** : l'IAG est un assistant puissant, mais elle ne remplace pas le rôle critique du développeur et de l'architecte.

# Conclusion

## Synthèse :

- L'IAG peut intervenir à chaque étape du cycle de vie logiciel.
- Elle augmente la productivité, améliore la documentation et renforce les tests.
- Elle exige des précautions (qualité, sécurité, gouvernance).

## Ouverture :

- Quelle phase du cycle de vie tirerait le plus de bénéfice de l'IAG dans vos projets ?
- Quels freins organisationnels ou techniques voyez-vous à son adoption ?

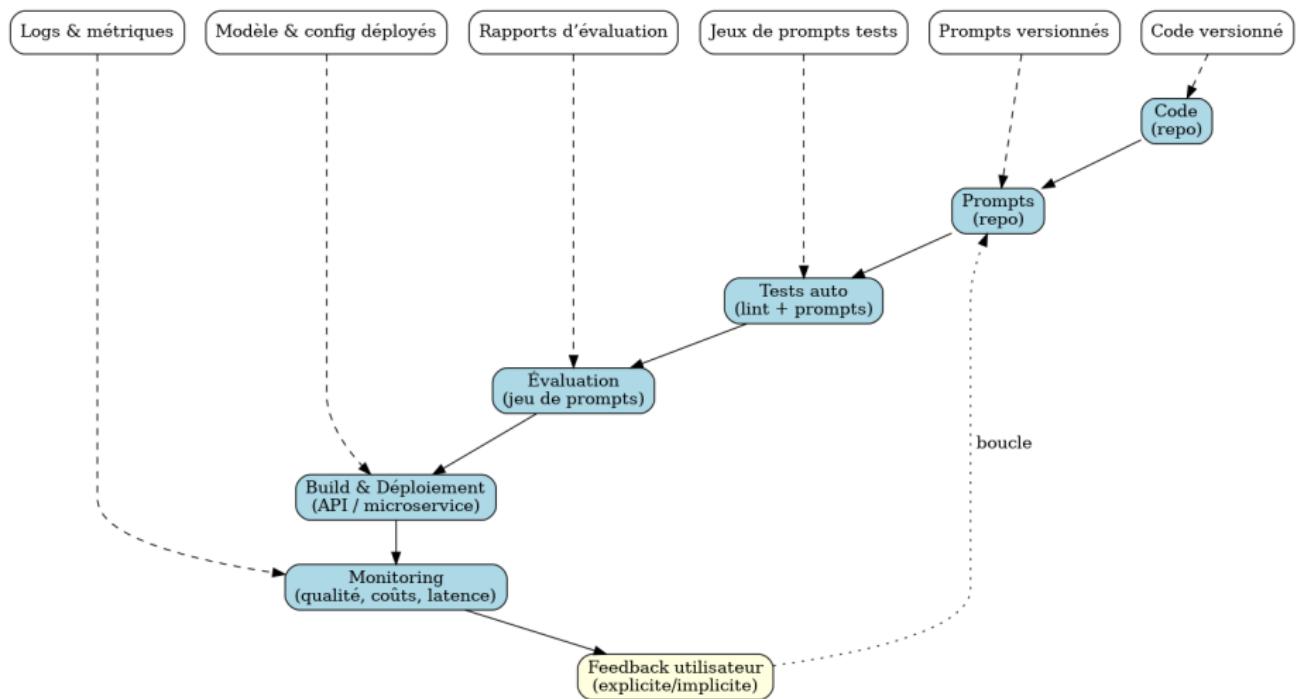
# Introduction : pourquoi un MLOps pour l'IAG ?

L'intégration de l'IAG dans les projets ne peut pas se limiter à un POC : il faut assurer robustesse, traçabilité et scalabilité.

- **Définition de MLOps :**

- Différences avec le MLOps classique :
  - Les prompts deviennent des artefacts centraux, au même titre que le code.
  - Les modèles évoluent très vite (mises à jour fréquentes, nouvelles versions).
  - Les métriques ne sont pas seulement de la précision, mais aussi qualité, coûts, latency, hallucinations.
- Objectif : gérer l'IAG comme un système logiciel vivant, soumis à un cycle de développement et d'exploitation continu.

## Pipeline CI/CD pour l'IAG



# Cycle MLOps adapté à l'IAG

Un cycle spécifique se met en place autour de l'IAG.

- ① Conception des prompts et jeux d'évaluation.
- ② Fine-tuning éventuel de modèles (spécifiques au métier).
- ③ Déploiement sous forme d'API ou de microservice.
- ④ Supervision : suivi de la qualité, des coûts, de la conformité.
- ⑤ Boucle de feedback : amélioration continue des prompts et des pipelines.

**Message clé** : le prompt devient un artefact versionné, testé et monitoré.

# Versioning des artefacts

L'IAG introduit de nouveaux éléments à gérer dans Git ou DVC.

- **Prompts** : stockés comme du code, avec historique et commentaires.
- **Modèles** : versions différentes d'LLMs (cloud, open source, fine-tunés).
- **Embeddings** : nécessaires pour la recherche sémantique et le RAG.
- **Datasets d'évaluation** : corpus de prompts/questions pour tester la cohérence.

**Bonne pratique** : utiliser Git + DVC pour tracer simultanément code, prompts et jeux d'évaluation.

# Tests automatisés pour l'IAG

Les tests ne portent pas seulement sur le code, mais aussi sur la qualité des réponses générées.

- **Tests fonctionnels** : vérifier que les réponses respectent un format attendu (JSON, regex).
- **Tests de cohérence** : comparer la sortie avec une référence (gold standard).
- **Tests de robustesse** : évaluer la stabilité des réponses face à des variations de prompt.
- **Non-régression** : détecter les changements introduits par une mise à jour de modèle.

**Outil** : intégration de jeux de prompts dans les suites de tests CI.

# CI/CD pour l'IAG

Le pipeline CI/CD s'enrichit avec de nouveaux artefacts.

- **CI :**

- Linting et validation des prompts.
- Génération de tests automatiques.
- Évaluation sur un dataset de référence.

- **CD :**

- Déploiement automatisé de modèles et prompts validés.
- Publication de l'API IAG ou du microservice associé.
- Documentation et changelog générés automatiquement.

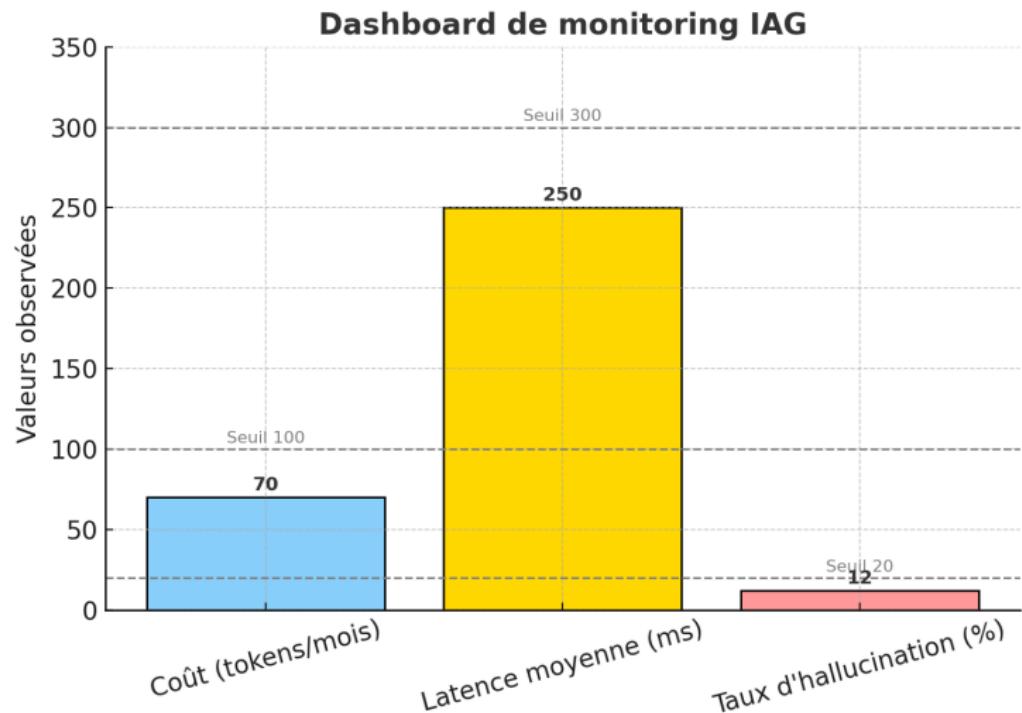
# Monitoring en production

Une IAG en production doit être surveillée en continu.

- **Qualité des réponses** : taux d'erreurs, hallucinations, satisfaction utilisateur.
- **Performance** : latence des réponses, disponibilité du service.
- **Coûts** : suivi du nombre de tokens, alertes en cas de dépassement.
- **Dérive** : détecter une baisse de qualité au fil du temps (concept drift).

**Bonne pratique** : associer logs classiques et résumés IAG pour rendre les anomalies lisibles.

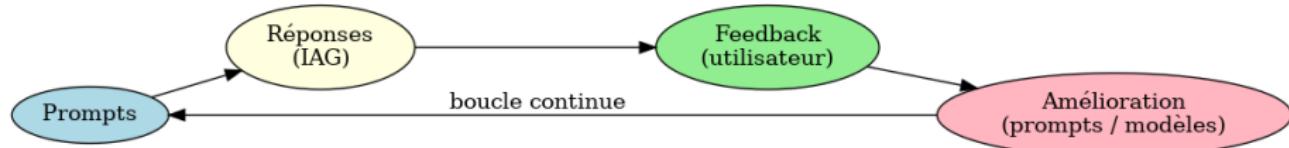
# Dashboard de monitoring IAG



# Observabilité et feedback utilisateur

L'amélioration continue passe par le retour d'expérience.

- **Feedback explicite** : notation des réponses par les utilisateurs.
- **Feedback implicite** : temps passé, clics, rejets ou corrections.
- **Boucle de réentraînement** : réutiliser les feedbacks pour améliorer prompts et modèles.
- **Dashboards partagés** : rendre visible la performance de l'IAG auprès des équipes.



**Message clé** : le feedback devient une donnée d'entraînement précieuse.

# Gestion des coûts et scalabilité

L'industrialisation impose de maîtriser les ressources.

- **Choix du modèle :**
  - léger (ex. GPT-3.5, Mistral) pour les tâches simples,
  - avancé (ex. GPT-4, Claude) pour les cas complexes.
- **Optimisation des prompts** : réduction du contexte inutile, segmentation intelligente.
- **Scalabilité** : load balancing entre modèles, mise en cache des réponses fréquentes.
- **Alertes coût** : seuils et budgets par équipe/projet.

# Sécurité et conformité

Les pipelines IAG doivent respecter les contraintes légales et de gouvernance.

- **Validation des prompts** : interdiction d'envoyer certains contenus (PII, secrets).
- **Auditabilité** : traçabilité des prompts et réponses utilisées en production.
- **Conformité RGPD** : localisation des données, droit à l'oubli.
- **Responsabilité** : définir les rôles et la supervision humaine obligatoire.

**Bonne pratique** : proxy IAG interne contrôlant toutes les interactions avec les modèles externes.

# Conclusion : MLOps pour l'IAG

## Synthèse :

- Les principes MLOps s'appliquent à l'IAG, mais avec de nouveaux artefacts : prompts, embeddings, jeux d'évaluation.
- La supervision humaine reste indispensable (qualité, éthique, conformité).
- Le succès repose sur un triptyque : **versioning, tests, monitoring**.

**Message clé** : l'IAG n'est pas seulement un outil d'expérimentation ; c'est un composant logiciel critique qui doit être géré avec rigueur.

# Introduction au fine-tuning des LLMs

## Pourquoi fine-tuner un LLM ?

- Adapter un modèle générique (GPT, LLaMA, Mistral) à un domaine spécifique (médical, juridique, entreprise).
- Réduire la dépendance au prompt engineering.
- Améliorer la pertinence, la cohérence et le style des réponses.

## Types d'adaptation :

- **Fine-tuning complet** : mise à jour de tous les poids.
- **Instruction tuning** : alignement sur des données question-réponse.
- **PEFT (LoRA, Prefix Tuning...)** : entraînement partiel et efficace.

# Fine-tuning complet

**Principe** : réentraîner l'ensemble des paramètres du modèle.

- **Avantages** : adaptation maximale au domaine, contrôle complet.
- **Inconvénients** :
  - Nécessite une infrastructure GPU massive.
  - Très coûteux en temps et en énergie.
  - Rarement accessible aux entreprises classiques.

**Message clé** : réservé aux acteurs disposant de moyens importants (Big Tech, laboratoires).

# Approches efficaces : LoRA et PEFT

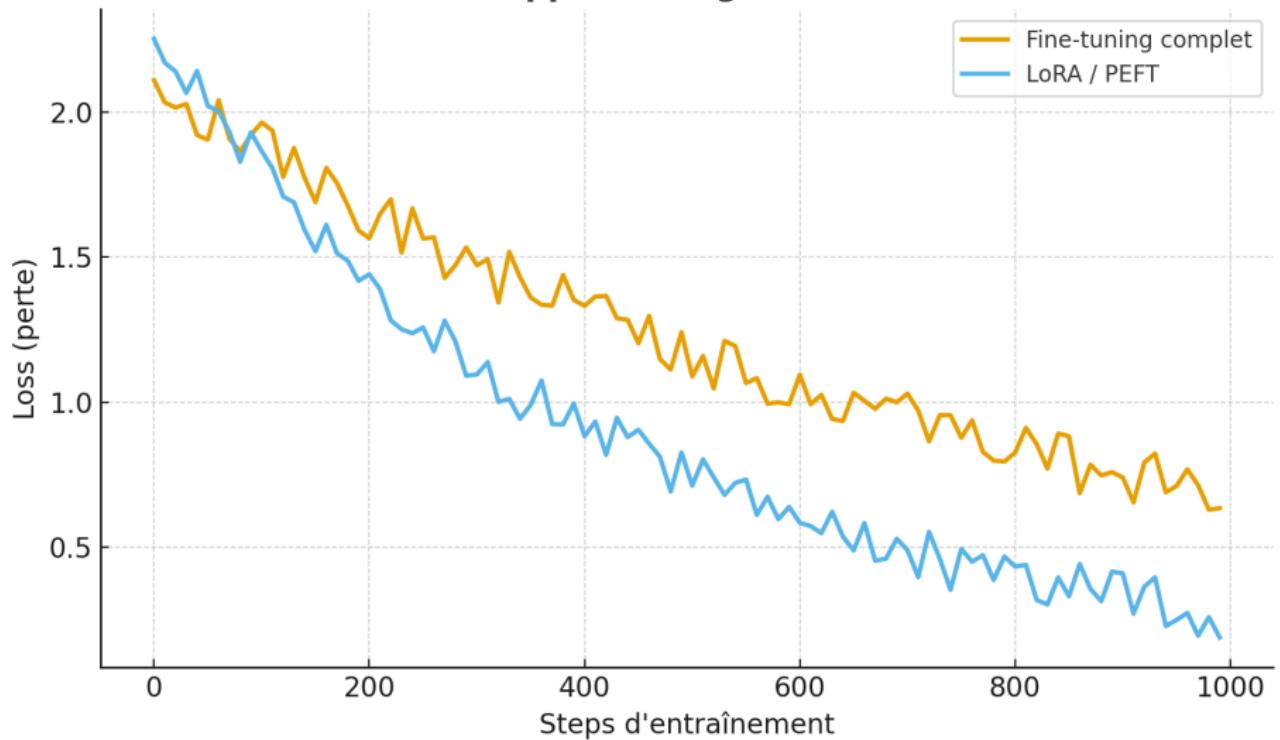
**Low-Rank Adaptation (LoRA)** : on n'entraîne que de petites matrices « d'adaptation ».

- **PEFT** (Parameter-Efficient Fine-Tuning) regroupe LoRA, Prefix Tuning, Adapters.
- Les poids d'origine sont gelés ; seuls les modules légers sont optimisés.
- **Avantages :**
  - Beaucoup moins coûteux.
  - Stockage des « adapters » léger (quelques Mo).
  - Facile à combiner avec différents modèles de base.



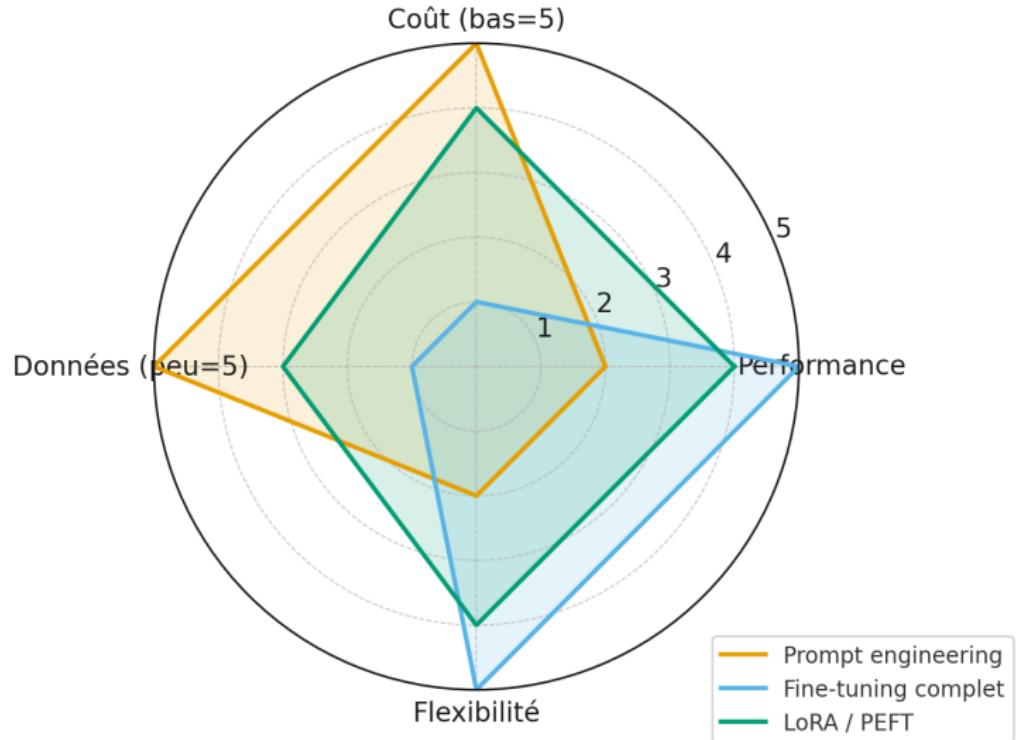
## Full fine tuning vs LoRA

## Courbes d'apprentissage : full FT vs LoRA



# Comparatif des approches

## Comparatif d'approches pour adapter un LLM



# Outils utilisés dans la pratique

## Écosystème Hugging Face et PyTorch :

- **PyTorch** : framework de référence pour l'entraînement des LLMs.
- **Transformers** : chargement et utilisation des modèles pré-entraînés.
- **PEFT** : implémentation de LoRA et autres techniques paramètre-efficaces.
- **Trainer** : API haut-niveau pour gérer l'entraînement.
- **Accelerate** : distribution sur GPU/TPU, multi-device, mixed precision.

Ces briques se combinent pour simplifier et accélérer le fine-tuning.

# Le rôle de Trainer

**Trainer** est une abstraction pour simplifier l'entraînement.

- Gère la boucle d'entraînement : epochs, batchs, évaluation, checkpoints.
- Supporte nativement les modèles Transformers et PEFT.
- Intègre le logging (TensorBoard, Weights & Biases).

**Exemple :**

- Fournir le modèle LoRA + datasets.
- Définir `TrainingArguments`.
- Appeler `trainer.train()`.

# Le rôle d'Accelerate

**Accelerate** optimise et distribue l'entraînement.

- Masque la complexité du multi-GPU et du multi-node.
- Permet l'entraînement en précision réduite (FP16, bfloat16).
- S'intègre au Trainer ou à une boucle PyTorch personnalisée.

**Message clé** : tirer le maximum de l'infrastructure matérielle sans changer le code métier.

# Pipeline typique avec LoRA + Trainer + Accelerate

## Étapes principales :

- ➊ Charger un modèle pré-entraîné (Transformers).
- ➋ Appliquer LoRA (PEFT).
- ➌ Préparer dataset (instructions, Q/A).
- ➍ Définir les arguments d'entraînement (batch size, epochs).
- ➎ Lancer avec Trainer, optimisé par Accelerate.
- ➏ Sauvegarder uniquement les adapters LoRA.

**Résultat** : un modèle spécialisé, léger et facile à déployer.

# Quand utiliser LoRA ?

## Cas pertinents :

- Domaines spécialisés (finance, médical, juridique).
- Données propriétaires ou confidentielles.
- Adaptation de style (ton, vocabulaire interne).

## Limites :

- Dépendance à la qualité du dataset.
- Maintenance des adapters si le modèle de base évolue.

# Conclusion

## Résumé :

- Le fine-tuning complet est rare et coûteux.
- LoRA/PEFT rend le fine-tuning accessible aux entreprises.
- Trainer et Accelerate facilitent et optimisent le processus.

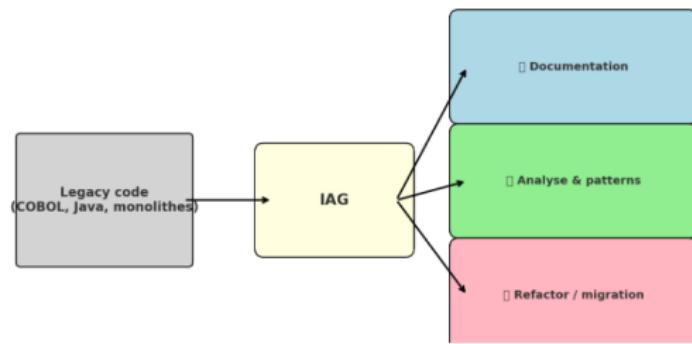
**Message clé :** avec LoRA + Trainer + Accelerate, il est possible d'adapter un LLM aux besoins métiers sans infrastructure démesurée.

# Faire évoluer l'héritage

# Faire évoluer l'héritage : introduction

La majorité des systèmes d'information reposent encore sur du **legacy code** : COBOL, Java anciens, monolithes, applications peu documentées.

- **Défis principaux :**
  - comprendre un code massif et mal documenté,
  - refactorer progressivement sans tout réécrire,
  - migrer vers des architectures modernes tout en limitant les risques.
- **Rôle de l'IAG** : agir comme un accélérateur de modernisation, en assistant les développeurs sur la **compréhension**, la **documentation**, la **refactorisation** et la **migration**.



# Documentation automatique du legacy

**Problème** : beaucoup d'applications critiques manquent de documentation claire.

- Génération de documentation à partir de code source non commenté.
- Résumés automatiques de modules, classes, fonctions avec description du rôle et des dépendances.
- Production de diagrammes ou d'explications en langage naturel pour faciliter l'onboarding.

**Exemple** : un fichier COBOL de 2000 lignes → résumé clair en français, avec liste des variables principales et description des calculs.

# Analyse et détection de patterns

**Objectif** : identifier les points faibles et les dépendances cachées.

- Détection automatique de duplications et de mauvaises pratiques (*code smells*).
- Identification des dépendances critiques entre modules.
- Proposition de refactor prioritaire : classes ou fonctions à isoler en priorité.

**Cas d'usage** : préparer une migration vers microservices en repérant les modules fortement couplés.

# Refactor assisté par l'IAG

**Problématique** : le code spaghetti rend difficile la maintenance et l'évolution.

- Proposition de réécriture par petits morceaux, avec extraction de fonctions claires.
- Simplification des conditions complexes ou boucles imbriquées.
- Harmonisation des conventions (noms de variables, structure).

**Exemple** : une fonction monolithique de 300 lignes → 5 fonctions spécialisées avec tests unitaires générés automatiquement.

# Migration vers architectures modernes

**Enjeu** : passer d'un monolithe ancien à une architecture modulaire et scalable.

- Génération de squelettes de microservices à partir de modules identifiés.
- Conversion d'un langage ancien vers un langage moderne (Java → Python, COBOL → Java).
- Proposition de schémas d'API REST à partir du code legacy.

**Attention** : les sorties doivent toujours être validées par un expert ; l'IAG propose des bases, pas un résultat prêt pour la production.

# Tests et sécurisation du legacy

**Problème** : modifier du legacy sans tests est risqué.

- Génération de tests unitaires pour entourer le code legacy.
- Création de jeux de tests fonctionnels à partir d'exemples historiques.
- Vérification automatique de la non-régression après refactor.

**Bénéfice** : sécuriser les évolutions du code legacy, limiter les régressions, faciliter la migration progressive.

# Limites et précautions

## Points de vigilance lors de l'utilisation de l'IAG sur du legacy :

- **Qualité variable** : l'IA peut mal interpréter des parties spécifiques ou très anciennes.
- **Hallucinations** : risque de générer du code inventé non exécutable.
- **Confidentialité** : attention à l'envoi de code sensible vers des API externes.
- **Supervision humaine indispensable** : un expert doit toujours valider la proposition avant mise en production.

# Conclusion : moderniser l'héritage

## Synthèse :

- L'IAG accélère la compréhension, la documentation et la refactorisation du legacy.
- Elle facilite la migration vers des architectures modernes et sécurise les évolutions grâce aux tests générés.
- Elle ne remplace pas l'expertise humaine mais la démultiplie.

**Message clé :** moderniser l'héritage devient plus accessible grâce à l'IAG, à condition de mettre en place une gouvernance et des garde-fous.

# IAG pour la gestion collaborative des projets

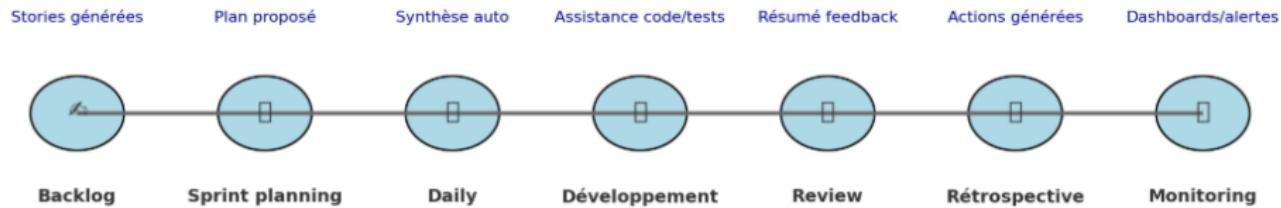
# Gestion collaborative et IAG : introduction

La gestion de projets informatiques implique une coordination fine entre les équipes, une communication constante et une surveillance continue des tâches et des performances.

**Problématique** : beaucoup de temps est perdu dans des tâches répétitives (comptes rendus, relances, mise à jour de dashboards).

## Apport de l'IAG :

- Automatiser les tâches récurrentes.
- Structurer et organiser les besoins.
- Améliorer la collaboration entre équipes avec des assistants intelligents.
- Générer des visualisations et alertes en temps réel.



# Coordination d'équipe et automatisation

**Objectif** : déléguer à l'IAG les tâches de coordination et de suivi.

- **Suivi des actions** : extraction automatique des *to-do* à partir des réunions.
- **Relances automatiques** : rappels envoyés sur Slack/Teams ou par mail.
- **Résumé de réunions** : transcription et synthèse via des outils comme Fireflies.ai.
- **Chatbot interne** : répondre aux questions d'équipe sur l'avancement du sprint ou du backlog.

**Exemple** : un compte rendu de daily → un plan d'actions distribué automatiquement.

# Structuration des besoins

**Objectif** : clarifier et standardiser la formulation des besoins.

- **Mindmaps automatiques** : génération avec des outils comme Chatmind à partir de notes brutes.
- **Transformation des besoins** : conversion de spécifications métier en user stories prêtes pour le backlog.
- **Uniformisation** : mise en forme systématique (roadmap, backlog, priorisation).

**Bénéfice** : moins d'ambiguités, plus de clarté dans le backlog produit.

# Outils collaboratifs augmentés par l'IAG

**Notion, Fireflies.ai, MetaGPT** : trois exemples emblématiques.

- **Notion** : génération automatique de pages de projet, FAQ internes, synthèse de documents.
- **Fireflies.ai** : transcription et résumé de réunions, attribution d'actions aux bonnes personnes.
- **MetaGPT** : orchestration automatique de rôles (Product Owner, développeur, QA) pour structurer un projet logiciel.

L'IAG peut devenir un véritable co-pilote de la gestion de projet.

# Visualisation et monitoring

**Objectif** : offrir aux équipes une visibilité instantanée sur l'état du projet et du système.

- Résumer les logs applicatifs pour identifier anomalies et erreurs.
- Générer automatiquement des dashboards (exemple : avancement des sprints, vitesse).
- Surveiller la performance des systèmes (temps de réponse, taux d'erreur).
- Compléter les solutions de monitoring existantes (Grafana, Prometheus, Evidently) par des résumés intelligents.

**Exemple** : un tableau de bord généré automatiquement après chaque sprint.

# Analyse multifactorielle et gestion des alertes

**Apport** : l'IAG croise plusieurs sources pour enrichir le suivi.

- **Analyse multifactorielle** : corrélation entre logs, tickets de support, métriques de performance.
- **Détection proactive** : identification d'incidents récurrents et proposition d'actions correctives.
- **Gestion intelligente des alertes** : tri et routage vers les bons interlocuteurs (ex : dev, ops, PO).

**Bénéfice** : réduire le bruit des alertes et prioriser les incidents critiques.

# Conclusion : collaboration augmentée

## Synthèse :

- L'IAG automatise les tâches répétitives et libère du temps à valeur ajoutée.
- Elle aide à organiser les besoins et à uniformiser les livrables.
- Elle améliore la coordination d'équipe grâce à des outils intégrés (Notion, Fireflies.ai, MetaGPT).
- Elle enrichit la visualisation et le monitoring avec des dashboards et alertes intelligentes.

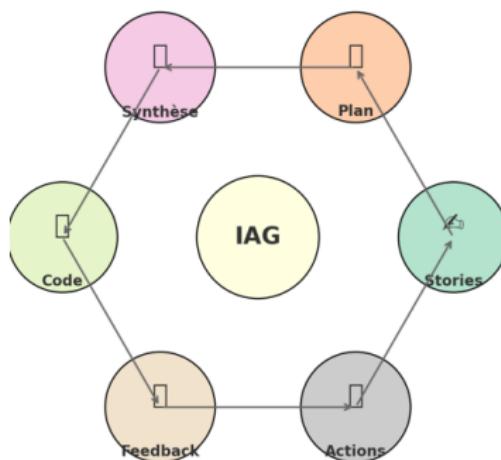
**Message clé :** l'IAG devient un co-pilote de la gestion de projet, à condition d'une supervision et d'un cadrage humain.

# LAG pour les processus agiles

# IAG et agilité : pourquoi

L'IAG peut accélérer le flux agile sans remplacer les rôles (PO, SM, devs).

- Réduire la charge cognitive : moins de tâches répétitives.
- Fluidifier la communication métier technique.
- Standardiser les artefacts agiles (stories, comptes rendus, doc).



# Backlog et user stories

**Objectif** : produire des stories claires et actionnables.

- Génération ou reformulation des user stories (format *en tant que... je veux... afin de...*).
- Extraction d'acceptance criteria à partir de notes ou tickets.
- Suggestion de priorisation (valeur, risque, dépendances) à challenger par l'équipe.

**Exemple** : notes de réunion → 3 stories + 6 critères d'acceptation.

# Planification de sprint

**Objectif** : accélérer la préparation sans figer la décision humaine.

- Proposition de plan de sprint à partir du backlog et de la vélocité historique.
- Décomposition automatique d'une story en sous-tâches techniques.
- Estimation initiale (story points) à discuter en poker planning.

**Remarque** : la planification finale reste une décision d'équipe.

# Daily et rétrospectives

**Objectif** : rendre visibles décisions et blocages.

- Synthèse des points d'hier/aujourd'hui, risques et dépendances.
- Génération de compte rendu de daily et suivi des actions.
- Rétro assistée : agrégation des faits, top/flop, actions d'amélioration proposées.

**Exemple** : transcript de rétro → plan d'actions priorisé.

# Documentation et communication

**Objectif** : documenter au fil de l'eau sans friction.

- Mise à jour de la documentation à partir des merges et releases.
- Rédaction de *definition of done*, checklists de qualité, guides internes.
- Adaptation multi-publics : résumé pour direction, détail pour dev/ops, pas-à-pas utilisateur.

**Exemple** : chatbot interne entraîné sur la doc projet et les règles agiles.

# Bonnes pratiques et limites

## Principes :

- L'IAG assiste, elle ne décide pas ; validation humaine systématique.
- Confidentialité : ne pas envoyer d'infos sensibles ; utiliser un proxy ou un modèle local si besoin.
- Transparence : tracer prompts, versions et décisions générées.

## Petits pas :

- Commencer par la reformulation de stories et les comptes rendus.
- Étendre à la planification et à la doc si l'équipe est convaincue.

# Enjeux éthiques, sécurité et conformité de l'IAG

# Pourquoi parler d'éthique et de sécurité ?

L'intelligence artificielle générative ne pose pas seulement des défis techniques : elle soulève des enjeux critiques pour l'entreprise.

- **Sécurité** : risques de fuite de secrets ou d'attaques via les prompts.
- **Confidentialité** : conformité au RGPD, respect des données sensibles.
- **Propriété intellectuelle** : propriété des données d'entraînement et du contenu généré par l'IA.
- **Éthique** : biais, équité, responsabilité dans l'usage.

**Message clé** : l'adoption de l'IAG doit être accompagnée d'une gouvernance adaptée.

# Risques de sécurité

## Exemples de menaces :

- **Fuite de secrets** : code ou identifiants envoyés à une API externe.
- **Prompt injection** : manipulation du modèle par un utilisateur malveillant.
- **Dépendance externe** : indisponibilité ou vulnérabilité du fournisseur de LLM.

**Bonne pratique** : utiliser des proxys, filtrer les prompts et éviter d'envoyer des données sensibles.

# Enjeux de confidentialité

**Problématique centrale :** l'IAG traite souvent des données sensibles ou personnelles.

- **RGPD** : droit à l'oubli, consentement, minimisation des données.
- **Localisation des données** : datacenters UE vs hors UE.
- **Solutions :**
  - anonymisation avant envoi,
  - hébergement de modèles open source en interne,
  - passage par un proxy d'entreprise.

## Un enjeu majeur souvent sous-estimé dans l'usage de l'IAG.

- **Origine des données d'entraînement** : risque que le modèle régénère des contenus protégés (texte, code, images).
- **Droits sur les contenus générés** :
  - Dans l'UE, pas de droit d'auteur automatique sur une production 100% IA.
  - L'entreprise doit définir une politique claire (contrats, licences internes).
- **Licences et open source** : vérifier la licence des modèles (Apache 2.0, MIT, restrictions) et des datasets utilisés.
- **Bonnes pratiques** :
  - tracer les sources de données et les outputs,
  - sensibiliser les équipes aux risques de réutilisation non conforme,
  - éviter d'intégrer sans vérification un contenu généré dans des livrables contractuels.

# Biais et équité

**Constat** : les modèles reflètent les biais de leurs données d'entraînement.

- Risque de stéréotypes et de discriminations dans les réponses.
- Impact fort dans les domaines sensibles (recrutement, médical, juridique).
- Besoin d'auditer les modèles dans leur contexte métier.

**Message clé** : l'évaluation humaine reste indispensable pour corriger les biais.

# Responsabilité et gouvernance

## Questions clés :

- Qui est responsable si une IA génère du code erroné ou trompeur ?
- Comment tracer et auditer les décisions de l'IA ?

## Bonnes pratiques :

- Supervision humaine obligatoire pour les tâches critiques.
- Mise en place d'une charte d'usage de l'IAG.
- Définition de rôles et responsabilités clairs.

# Durabilité et coûts cachés

## Impact énergétique :

- Fine-tuning complet = forte consommation GPU et empreinte carbone.
- LoRA/PEFT = alternatives plus sobres, adaptées aux entreprises.

## Coûts cachés :

- Facturation à l'usage (tokens, appels API).
- Dépendance à un fournisseur unique.

**Message clé** : arbitrer entre innovation, budget et responsabilité environnementale.

# Conclusion et bonnes pratiques

## Synthèse :

- Les enjeux de sécurité, confidentialité et éthique sont incontournables.
- La gouvernance doit accompagner toute intégration de l'IAG.
- Commencer petit, superviser, documenter et auditer.

**Bonne pratique** : intégrer les aspects éthiques dès la conception des projets IAG.

# Merci de votre attention

redha.moulla@axia-conseil.com