

Programme

Matin

Fondamentaux du ML:

- Applications
- Histoire
- Modèles
- Entrainement
- Données

Durée : ~30 min

Fondamentaux du NLP:

- Token
- Modèles
- Fine Tuning

Durée : ~45 min

NLP avancé:

- Embeddings
- Rag
- Rag Agentic

Durée : ~45 min

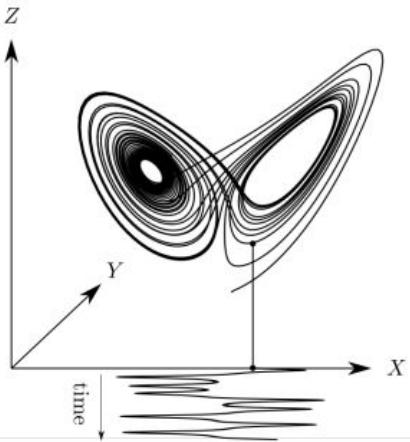
Cas pratique

- Token
- Embeddings
- Rag

Durée : ~30 min

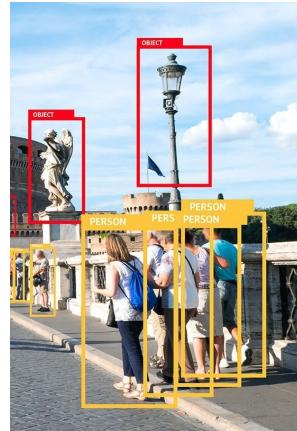
Fondamentaux du ML

IA : applications



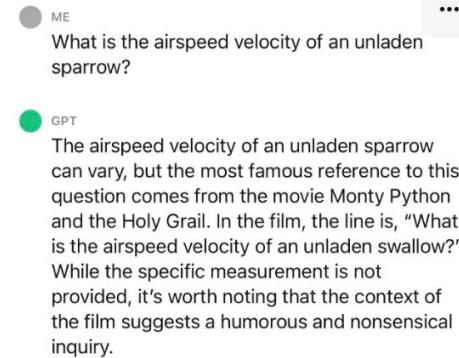
Predictive Analytics

Analyse les tendances dans les données pour prévoir les résultats, de la prédiction des tendances du marché à la recommandation de produits basée sur le comportement des utilisateurs.



Computer Vision

Analyse et interprète les données visuelles, de l'imagerie médicale qui détecte les maladies aux véhicules autonomes qui reconnaissent les panneaux de signalisation et les piétons.



Natural Language Processing

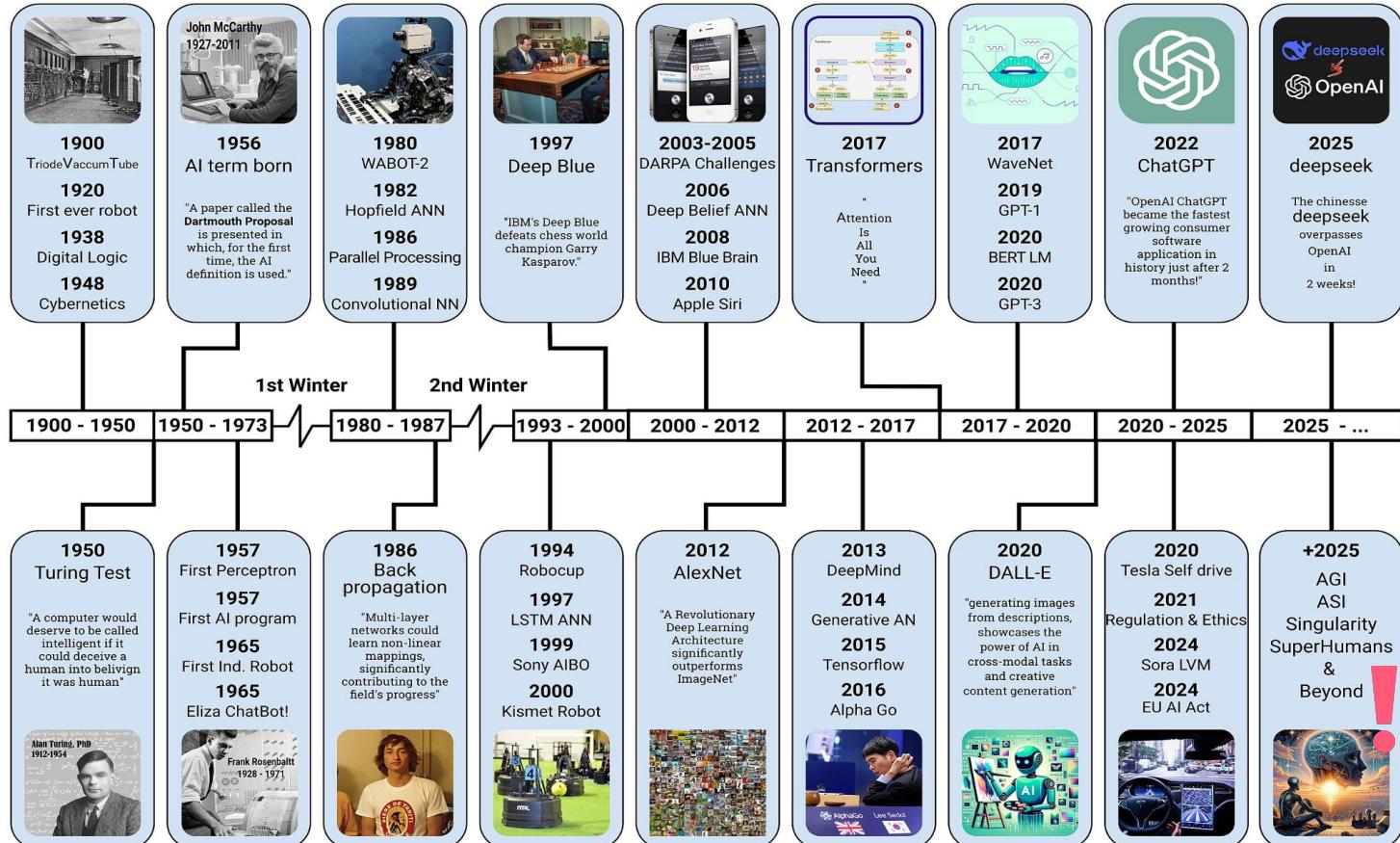
Traite le langage humain pour la traduction, la génération de contenu et la conversation. Les assistants virtuels et les chatbots comprennent et répondent au langage naturel.



Automation & Robotics

Automatisation intelligente dans la fabrication, la logistique et les industries de service, réduisant l'erreur humaine et augmentant l'efficacité.

IA : Histoire



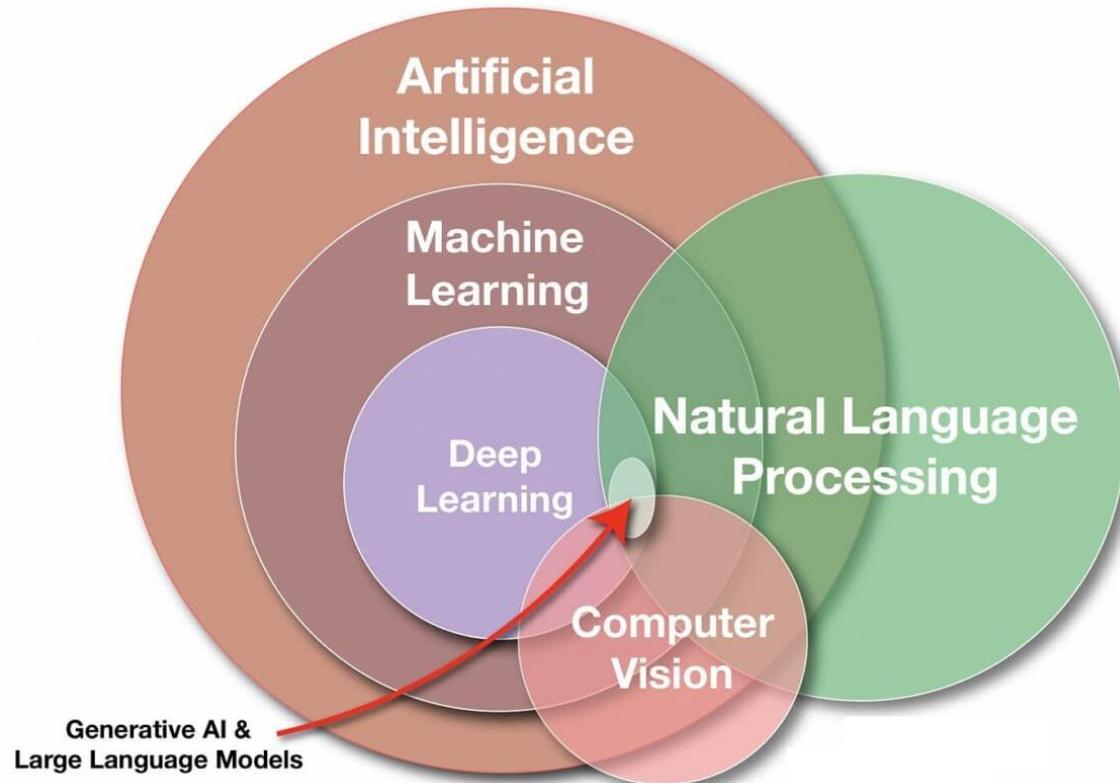
source: <https://commons.wikimedia.org/wiki/File:AI-History-Timeline-300dpi.jpg>

IA : Segmentation

IA: Automatisations faisant intervenir des algorithmes pour reproduire des capacités cognitives

ML: Algorithmes statistiques apprenant automatiquement sur des données

DL: Algorithmes statistiques avec un très grand nombre de paramètres

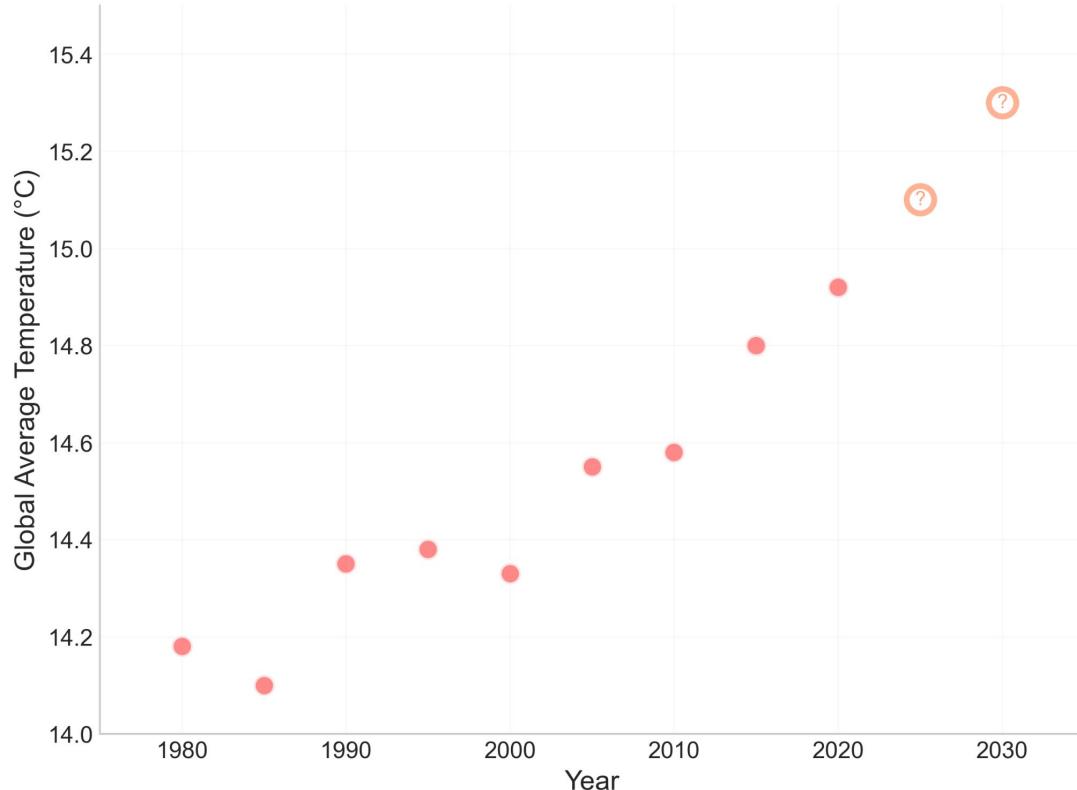


source: <https://medium.com/geekculture/ai-revolution-your-fast-paced-introduction-to-machine-learning-914ce9b6ddf>

IA : Prédire le futur

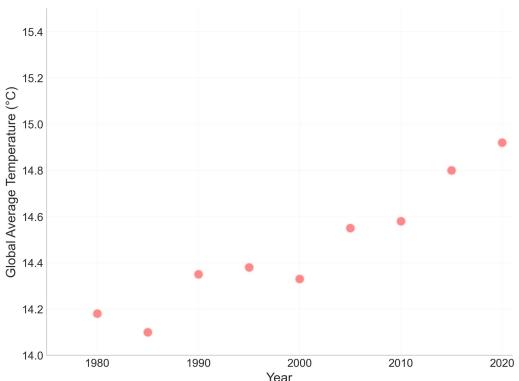
Exemple

Nous avons des relevés de températures historiques et nous aimerais prédire les températures futures

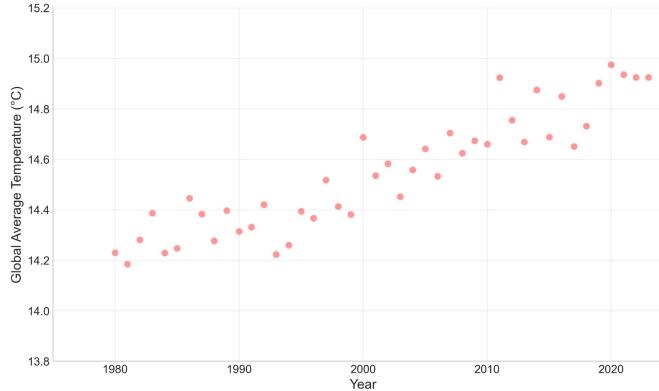


Caractéristique des données

Quantité/Taille d'échantillon :
nombre d'observations

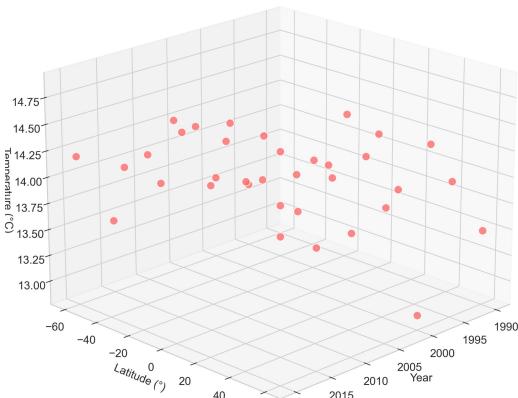


9 observations / 2 variables



44 observations / 2 variables

Dimension: nombre de variables par observation (features)



9 observations / 3 variables

Année	Latitude	Altitude	CO ₂	Humidité	°C
2000	45.0	200	370	65	13.8
2010	30.0	800	390	45	12.5
2020	60.0	50	415	80	14.2

3 observations / 6 variables

ML : Apprentissage Statistique

X : Variables explicatives (features/caractéristiques)

Y : Variable à expliquer (target/cible)

Phase d'entraînement

Nous disposons de **couples (X, Y)** observés/mesurés

Le modèle **apprend la relation entre X et Y**

Phase de prédiction

Nous recevons de **nouveaux X**

Prédire les Y correspondants

Concrètement

On aimerait trouver le **modèle** : la **fonction** $Y = f(X)$ pour **calculer Y à partir de X**

Exemple

X : (année, latitude, altitude) Y : Température

$$\text{Température} = 0.12 \times \text{année} - 17 \times \text{latitude} - 0.6 \times \text{altitude} + 14 - 0.11 \times \text{année} \times \text{altitude}$$

Caractéristique d'un modèle

Dimension entrée/sortie (Input/Output Dimension)

- **Dimension d'entrée** : dimension des variables explicatives (features)
- **Dimension de sortie** : nombre de variables à prédire (target/cible)

Exemple

$f(\text{année}, \text{latitude}, \text{altitude}) = (\text{Température}, \text{Co2})$ -> dimension entrée 3, sortie 2

Paramètres : valeurs ajustables du modèle

Exemple :

$\text{Température} = \text{paramètre}_1 \times \text{année}$ -> **1 paramètre**

$\text{Température} = \text{paramètre}_1 \times \text{année} + \text{paramètre}_2$ -> **2 paramètres**

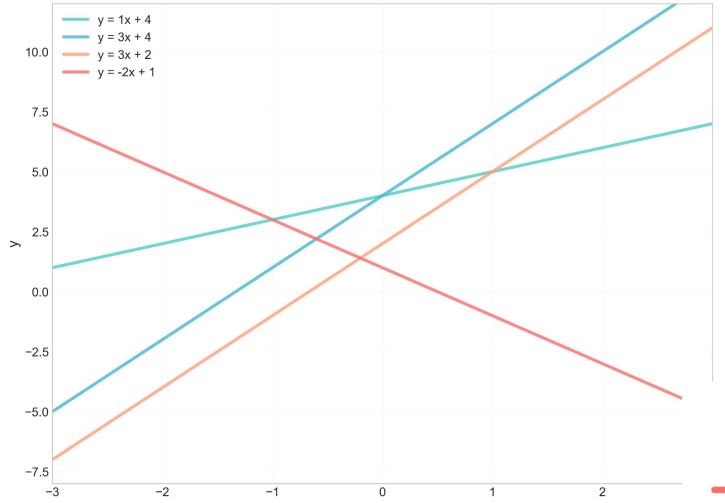
Opérations entre les paramètres et les variables

Exemple :

$\text{Température} = 0.12 \times \text{année} - 14$ -> **addition / multiplication**

$\text{Température} = \text{année}^2 + \sinus(\text{année})$ -> **opérations non linéaires**

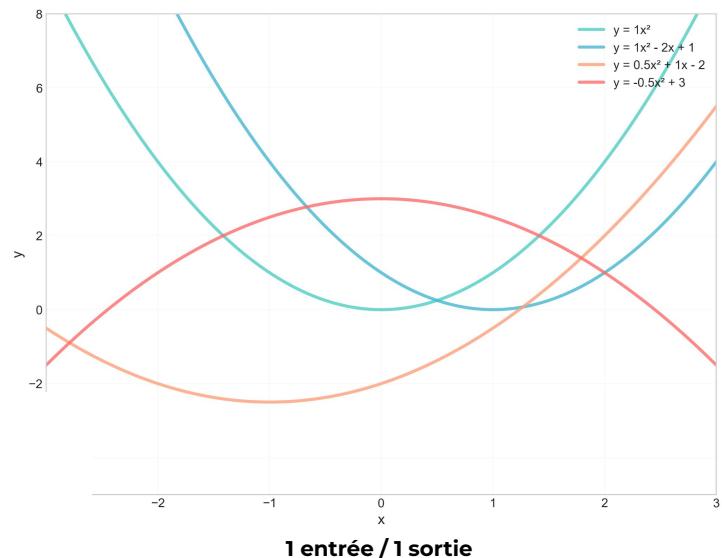
Linear Functions: $y = ax + b$
(2 parameters: slope a, intercept b)



1 entrée / 1 sortie

Exemples de modèle

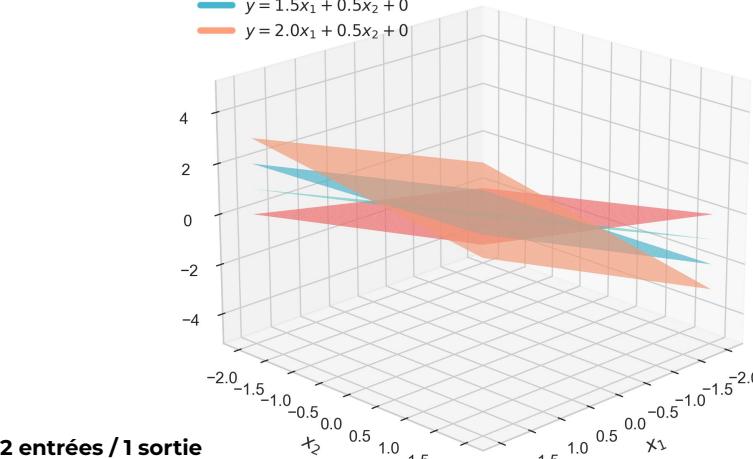
Polynomial Functions: $y = ax^2 + bx + c$
(3 parameters: a, b, c)



1 entrée / 1 sortie

Linear Functions: $y = ax_1 + bx_2 + c$
(3 parameters)

- $y = 0.5x_1 + 0.5x_2 + 0$
- $y = 1.0x_1 + 0.5x_2 + 0$
- $y = 1.5x_1 + 0.5x_2 + 0$
- $y = 2.0x_1 + 0.5x_2 + 0$



2 entrées / 1 sortie

ML: Processus d'entraînement d'un modèle

0. Initialisation

- Choisir un type de modèle (ex: régression linéaire)
- Initialiser les paramètres aléatoirement

1. Prédiction

- Prédire Y à partir de X avec le modèle actuel

2. Calcul de l'erreur

- Comparer $Y_{\text{prédit}}$ vs $Y_{\text{réel}}$

3. Ajustement

- Modifier les paramètres pour réduire l'erreur (descente de gradient)

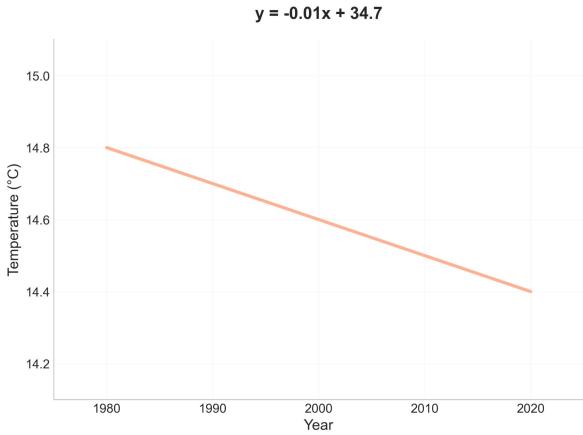
4. Itération

- Répéter depuis 1 jusqu'à ce que l'erreur ne réduise plus

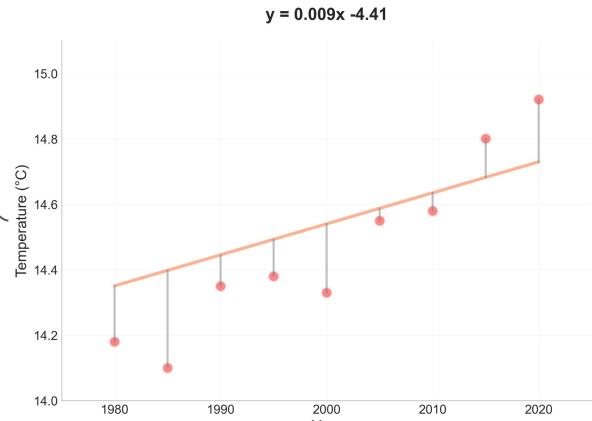
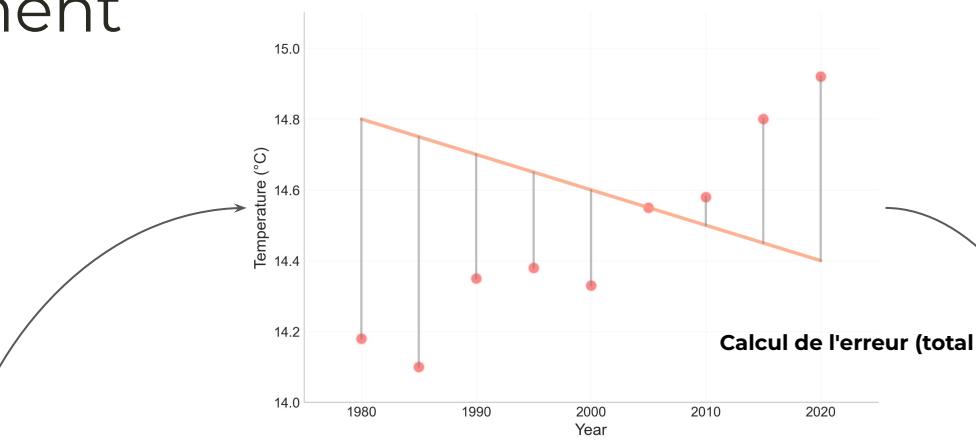
Objectif : Minimiser l'erreur de prédiction en optimisant les paramètres.



ML : Erreur et Ajustement



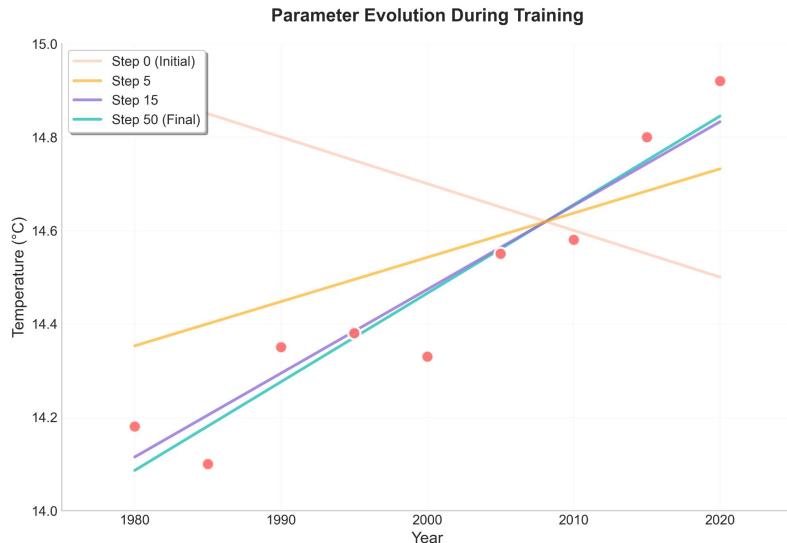
Initialisation



Ajustement des paramètres

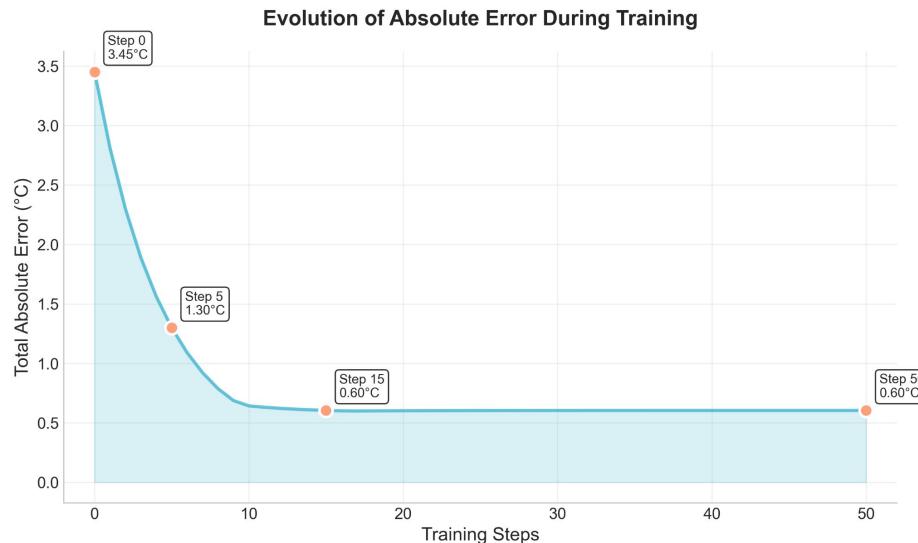
Année	Y_réel	Y_prédit	Erreur
1980	14.18	14.80	0.62
1985	14.10	14.75	0.65
1990	14.35	14.70	0.35
1995	14.38	14.65	0.27
2000	14.33	14.60	0.27
2005	14.55	14.55	0.00
2010	14.58	14.50	0.08
2015	14.80	14.45	0.35
2020	14.92	14.40	0.52

ML : Exemple d'entraînement



Model Initial
 $a=-0.01, b=34.70$

Model entraîné
 $a=0.019, b=-23.47$



Prédiction initial du modèle en 2030
 $-0.01 * 2030 + 34.70 = 14.4$

Prédiction du modèle entraîné pour 2030
 $0.019 * 2030 - 23.47 = 15.1$

Diversité des modèles

Types de modèles :

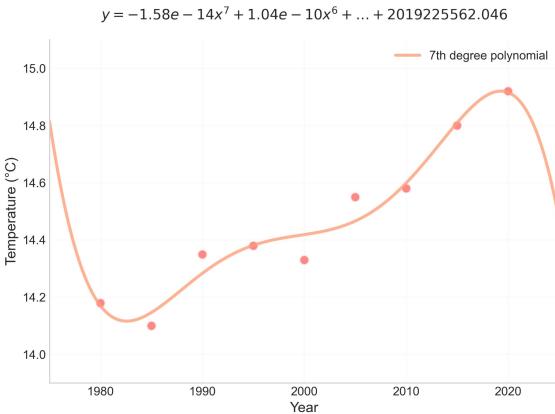
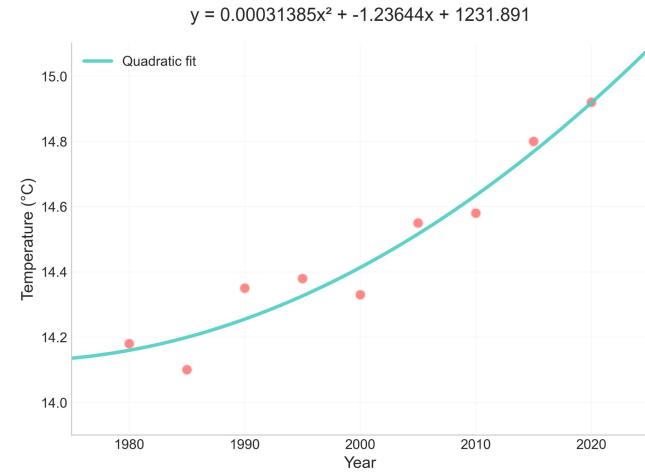
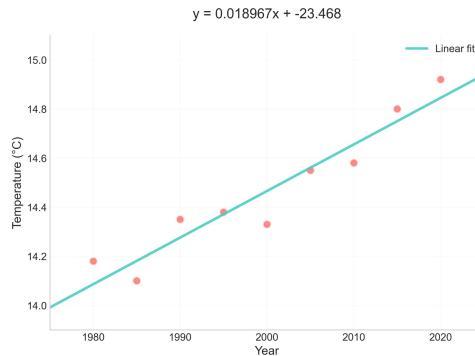
- Régression linéaire/polynomiale
- Forêt aléatoire (arbres de décision)
- Boosting
- Support Vector Machine (SVM)
- Réseaux de neurones
-

Même objectif : Ajuster les paramètres pour **minimiser l'erreur** entre :

- Les **prédictions** du modèle : $f(X)$
- Les **données réelles** : $Y_{réel}$

La science des données

- **Préparer les données**
- **Sélectionner** le type de modèle adapté au problème
- **Entraîner** le modèle de manière optimale



Plus de paramètres ≠ Meilleur modèle

Réseaux de neurones

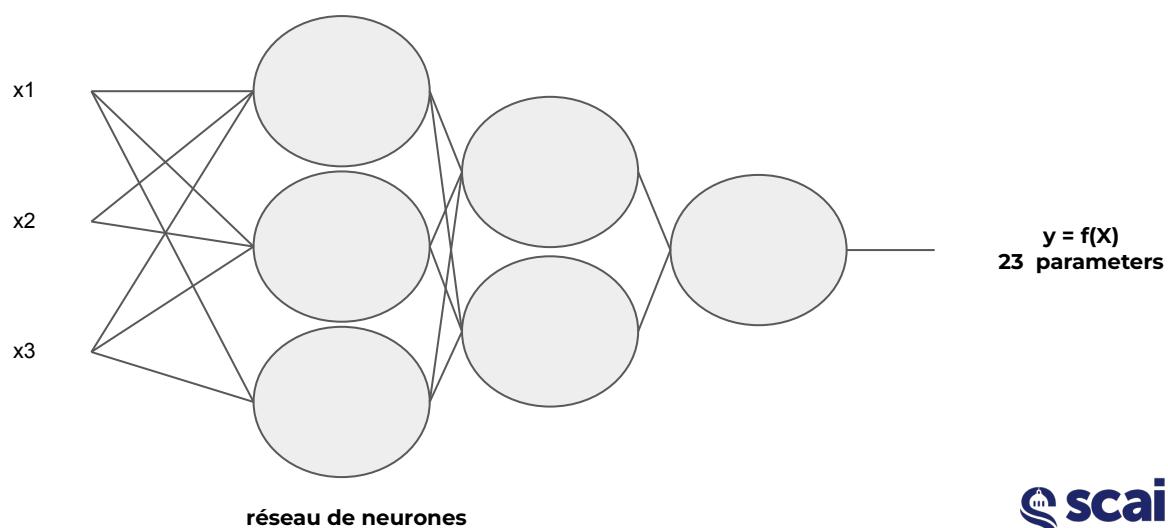
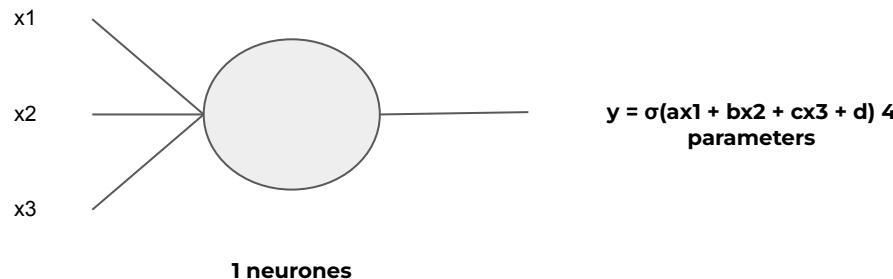
Un modèle comme un autre

Avantages clés :

- **Modularité** : architectures très variées et adaptables
- **Parallélisation** : entraînement rapide et efficace
- **Performance** : modélisation d'informations complexes

Types principaux :

- Perceptron Multicouche (MLP)
- Convolutionnels (CNN)
- Récurrents (RNN)



Représentation des données

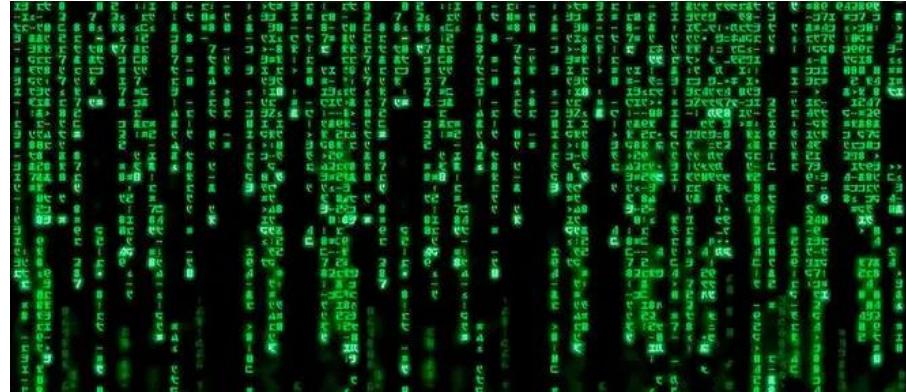
Quel que soit le type de données, les modèles ne traitent que des **valeurs numériques**:

Texte : "Hello World" → [156, 892, 445]

Images : Photo de chat → [0.8, 0.2, 0.9, 0.1, ...]

Vidéos : Séquence d'images → [0.3, 0.7, 0.4, ...]

Données mixtes : Combinaison → [892, 0.8, 445, 0.2, ...]



Regression/classification

Régression

Prédire des valeurs continues

Année	Latitude	Altitude	°C	CO ₂
x_1	x_2	x_3	y_1	y_2
2000	45.0	200	13.8	370
2010	30.0	800	12.5	390
2020	60.0	50	14.2	415

$$f(\text{année}, \text{latitude}, \text{altitude}) = \text{°C, CO}_2$$

Classification

Prédire des catégories (encodées en nombres)

pixel_1	pixel_2	...	pixel_1000	chien	chat	poulet
x_1	x_2	...	x_1000	y_1	y_2	y_3
14	42	...	65	1	0	0
250	128	...	1	0	1	0
29	18	...	214	1	0	0

$$f(\text{image}) = [1,0,0] \text{ ou } [0,1,0] \text{ ou } [0,0,1]$$

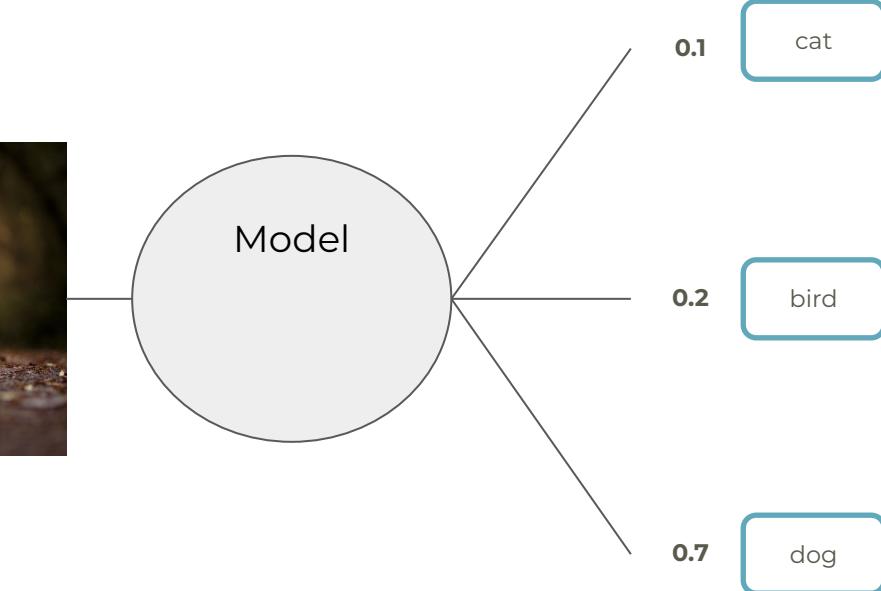
Classification : Prédiction de probabilités

Le modèle ne prédit pas directement la classe, mais des probabilités

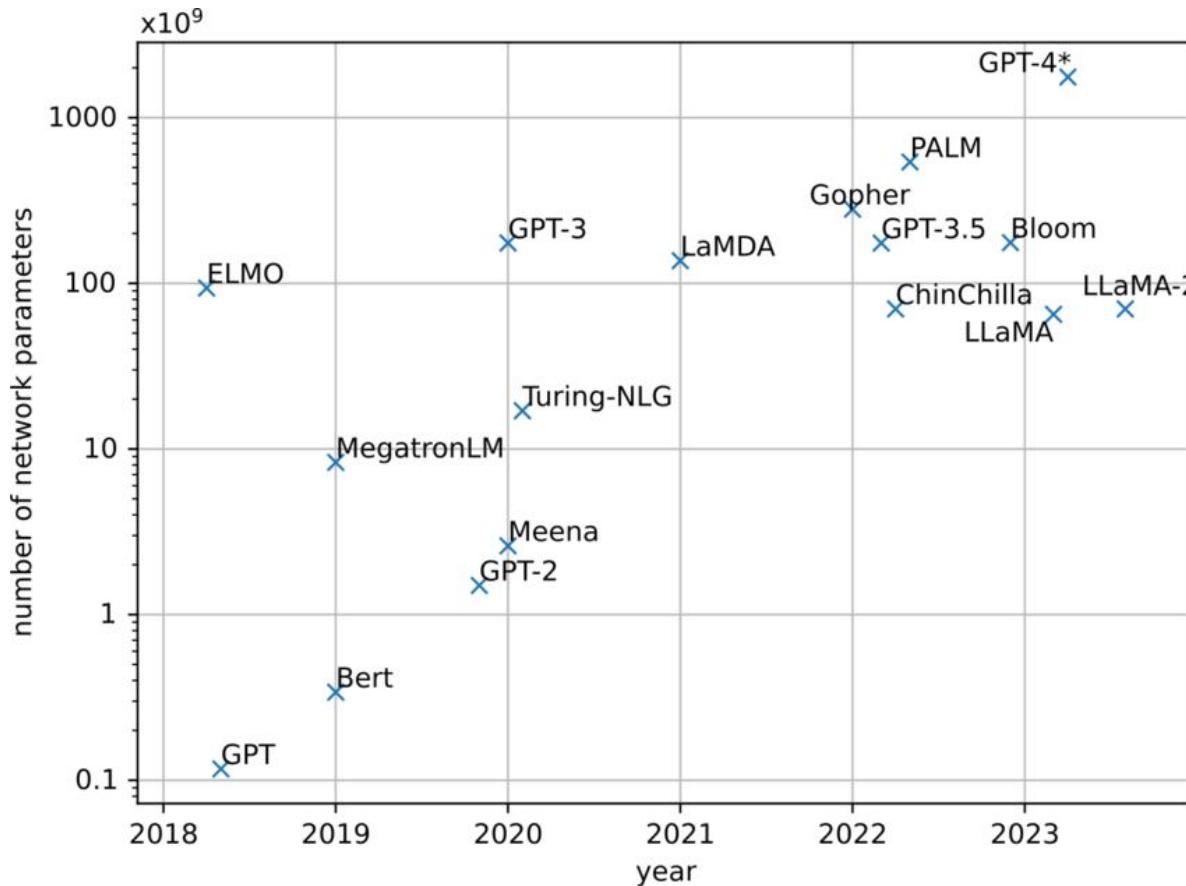
degré de certitude

- 0.99 → très confiant
- 0.51 → peu confiant

Classe prédictive = celle avec la probabilité maximale



AI : models / parameters



AI : success skills

1. Données de qualité

- Suffisantes, propres et représentatives

2. Puissance de calcul

- Hardware performant (GPU/TPU)

3. Expertise technique

- Savoir entraîner et optimiser les modèles



Les Echos

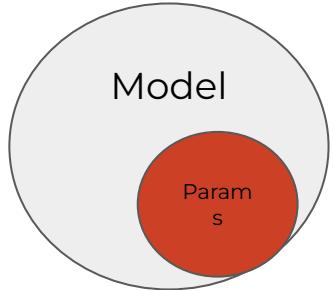
<https://www.lesechos.fr> > ... > Intelligence artificielle

⋮

Nvidia devient la première capitalisation boursière mondiale

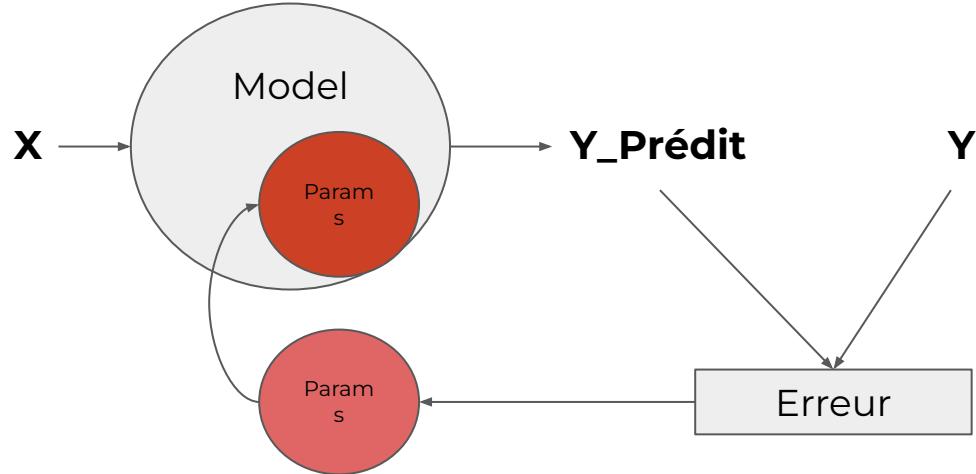
18 juin 2024 — Les actions du fabricant de puces ont grimpé en séance de 3,5 % à 135,58 dollars, portant sa capitalisation boursière à **3335 milliards** de ...

Take Home message



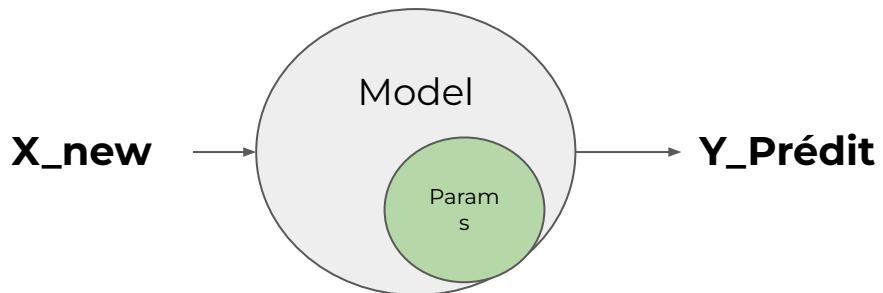
1. Choisir le modèle

Adapter aux dimensions d'entrée/sortie



2. Entrainer le modèle

Ajuster les paramètres pour minimiser l'erreur : $|f(X) - Y_{\text{réell}}|$



3. Prédire

Utiliser le modèle entraîné sur de nouvelles données X

Librairies



Manipulation et analyse de données
tabulaires



Machine Learning classique et
préprocessing



Visualisation de données et
graphiques



Deep Learning et réseaux de
neurones

NLP/LLM

NLP : applications

Le NLP (Traitement du langage naturel) combine linguistique, informatique et IA pour permettre aux ordinateurs d'interpréter et générer le langage humain.

Traduction

Conversion automatique de textes d'une langue à une autre.

Les systèmes modernes comprennent le contexte et les nuances culturelles pour produire des traductions naturelles et précises.

Chatbot

Assistant virtuel conversationnel qui comprend les questions en langage naturel et fournit des réponses pertinentes.

Summary

Extraction automatique des informations essentielles d'un document long pour créer un résumé concis.

Analyse de sentiment

Classification automatique des émotions et opinions exprimées dans un texte (positif, négatif, neutre).

Caractéristiques du langage naturel

- **Séquentialité** : L'ordre des mots change le sens
"Le chien mord le chat" ≠ "Le chat mord le chien"
- **Dépendances à long terme** dans les séquences
"Il" page 127 fait référence au personnage principal mentionné plusieurs pages auparavant
- **Contexte crucial** pour lever les ambiguïtés
"la souris est dans le placard" = animal ou périphérique
- **Volume massif** : Milliards de mots à traiter
- **Diversité linguistique** : Syntaxes, grammaires et structures différentes selon les langues



Token

Comment représenter le langage naturel?

La tokenisation : Processus qui découpe le texte en unités (tokens) via un **vocabulaire/dictionnaire**.

Tokenisation :

1. **Texte d'entrée** → 2. **Découpage en tokens** (via vocabulaire) → 3. **Séquence numérique**

Reconstruction:

2. **Séquence numérique** → 2. **tokens associées** (via vocabulaire) → 3. **Texte de sortie**

Token	ID
"Th"	1
"at"	2
"run"	3
"c"	4
"e"	5

Exemple : "The cat" → [Th, e, c, at] → [1, 5, 4, 2]

Vocabulaire

Tokenisation : Approche Caractère par Caractère

Vocabulaire : Alphabet + caractères spéciaux (~30 tokens)

Exemple :

Token	ID	Token	ID
"a"	1	"z"	26
"b"	2	" "	27
"c"	3	".."	28

Vocabulaire

"The cats are running" → [T,h,e, ,c,a,t,s, ,a,r,e, ,r,u,n,n,i,n,g] → [20,8,5,27,3,1,20,19,27,1,18,5,27,18,21,14,14,9,14,7]

✗ Problèmes : Séquences très longues + perte de la structure des mots

Tokenisation : Approche Mot par Mot

Vocabulaire : Tous les mots du corpus (exemple wikipédia anglais ~13M tokens)

Exemple :

Token	ID
"The"	1
"cats"	856,432
"running"	2,341,567

Vocabulaire

"The cats are running" → [The, cats, are, running] → [1, 856432, 15, 2341567]

✗ Problèmes : Vocabulaire gigantesque + mots rares sous-représentés + "mange" ≠ "manges"

Subword Tokenisation

Vocabulaire optimal : Séquences les plus fréquentes dans le corpus

Processus de construction :

1. Démarrer avec l'alphabet + caractères spéciaux (~30 tokens)
2. Identifier la paire de tokens adjacents la plus fréquente
3. L'ajouter au vocabulaire et réitérer
4. S'arrêter à ~50k-100k tokens

Exemple :

Token	ID	Fréquence
"a"	1	
"s"	19	
"are"	38	100M
"ing"	40	80M
"The"	37	50M
"cat"	901	2M
"runn"	1517	500k

Vocabulaire

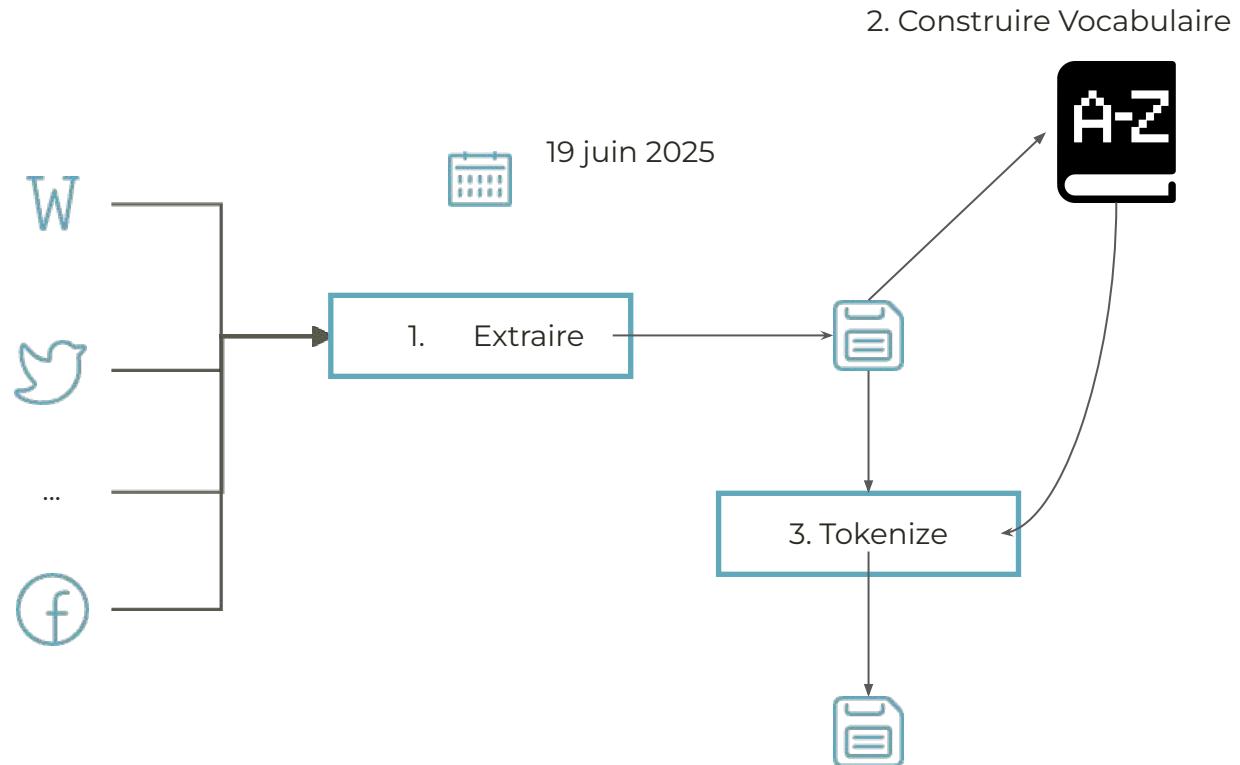
"The cats are running" → [The, cat, s, are, runn, ing] → [37, 901, 19, 38, 1517, 40]

 **Avantages :** Vocabulaire gérable + structure préservée + proximité sémantique

LLM

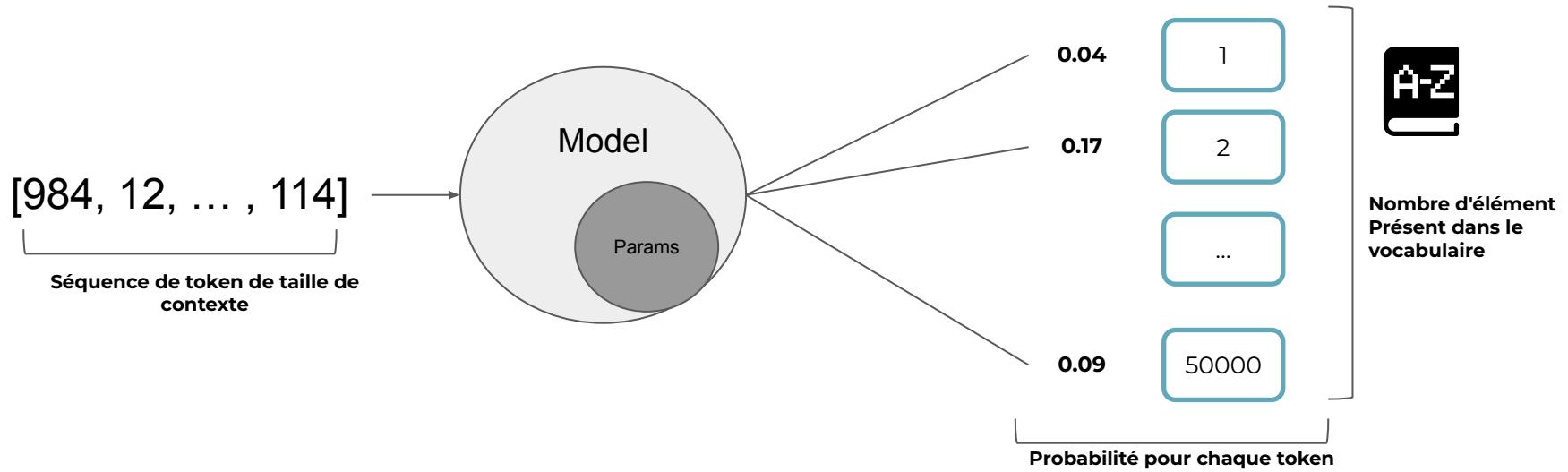
LLM : Corpus / Data

	Training Set (Words)	Training Set (Tokens)
Recent LLMs		
Llama 3	11 trillion	15T
GTP-4	5 trillion	6.5T
Humans		
Human, age 5	30 million	
Human, age 20	150 million	



source:
<https://www.educatingsilicon.com/2024/05/09/how-much-lm-training-data-is-there-in-the-limit/>

NLP : modèle caractéristiques



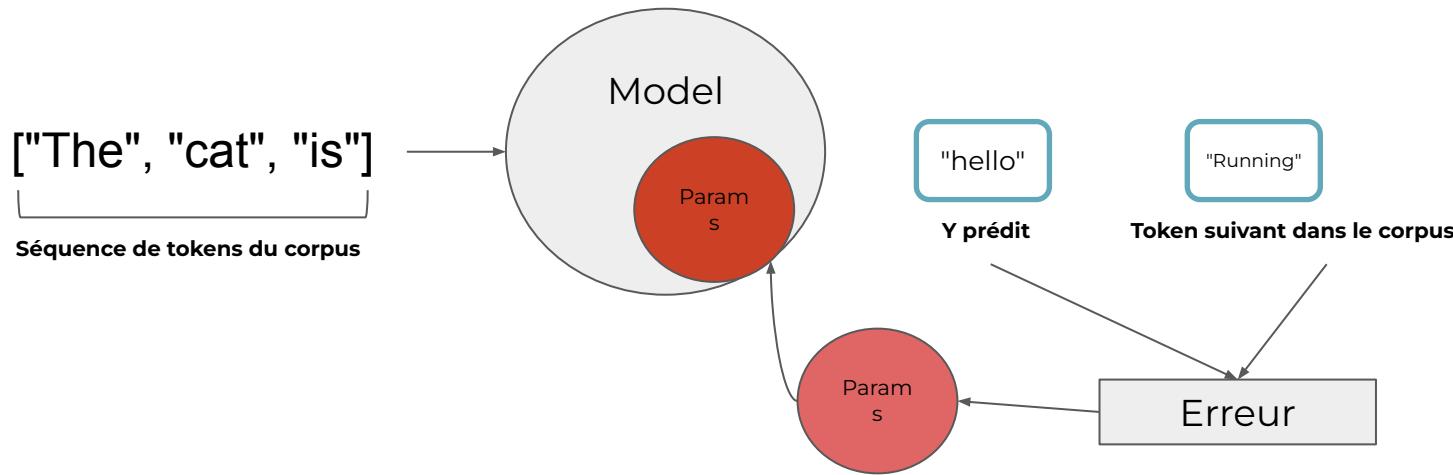
Dimension d'entrée : Taille de la fenêtre de contexte (**context window**), nombre de tokens en sortie.

Dimension de sortie : Égale aux nombre de token du vocabulaire

Architecture : Basée sur des **transformers** utilisant des mécanismes d'**attention**

Paramètres : Nombre de paramètres très élevé (millions à trillions)

NLP : modèle entraînement (type GPT)



1. Prédiction - Le modèle prédit le token suivant à partir de la séquence X

2. Calcul de l'erreur - Comparer la prédiction vs le vrai token suivant

3. Ajustement - Modifier les paramètres du modèle pour réduire l'erreur

4. Itération - Répéter avec la séquence suivante du corpus

LLM : Modèle avec beaucoup de paramètres

Modèle	Date de sortie	Paramètres	Fenêtre de contexte	Taille vocabulaire
GPT-4	mars 2023	~1.76T (estimé)	8,192 tokens	~100,000 tokens
GPT-4 Turbo	nov. 2023	~1.8T (estimé)	128,000 tokens	~100,000 tokens
Claude 3 Opus	mars 2024	~175B (estimé)	200,000 tokens	~100,000+ tokens
Claude Sonnet 4	février 2025	Non divulgué	200,000 tokens	~100,000+ tokens
Gemini 1.0 Ultra	déc. 2023	~1T (estimé)	32,000 tokens	Non divulgué
Gemini 2.0	avril 2024	Non divulgué	1,000,000 tokens	Non divulgué
LLaMA 3	avril 2024	8B-70B	8,192 tokens	~32,000 tokens
LLaMA 4	avril 2025	8B-400B	10,000,000 tokens	~32,000 tokens
DeepSeek-R1	janvier 2025	671B	500,000 tokens	Non divulgué
GPT-2	2019	1.5B	1,024 tokens	~50,000 tokens
BERT	2018	110M-340M	512 tokens	~30,000 tokens

LLM : Ressources d'Entraînement et Coûts

Échelle du modèle	Ressources nécessaires	Durée d'entraînement	Coût estimé
1B paramètres	~10-50 GPUs	Jours à semaines	\$10K - \$100K
10B paramètres	~100-500 GPUs	Semaines à mois	\$100K - \$1M
100B+ paramètres	1,000+ GPUs	Plusieurs mois	\$1M - \$100M+

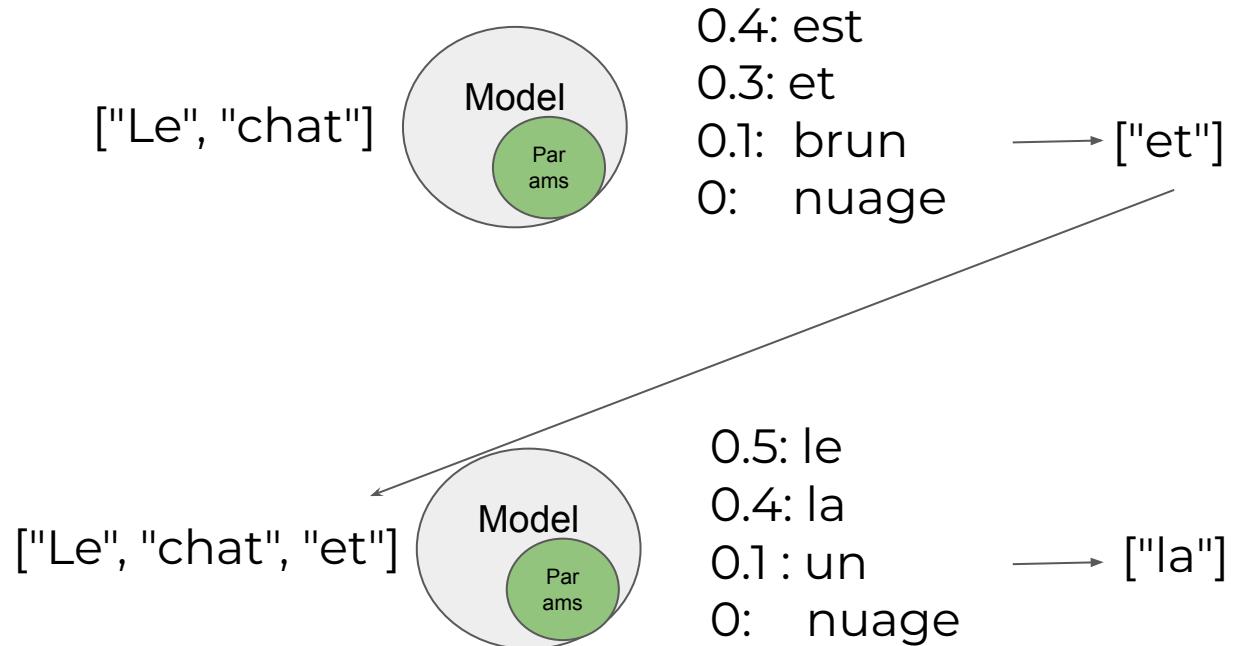
LLM : prédition/génération

Génération auto-régressive :

Le processus de génération est **itératif** : chaque token prédit devient partie de l'entrée pour prédire le token suivant.

Température :

- **Température élevée (1.0+)** : Prédictions créatives et variées
- **Température = 0** : Toujours choisir le token le plus probable



Prompting

Le LLM, **prédit la suite probable** des mots après votre prompt:

- **Détermine la qualité** de la réponse
- **Guide la prédition** vers le résultat souhaité
- **Contrôle le comportement**

1. Soyez spécifique et clair

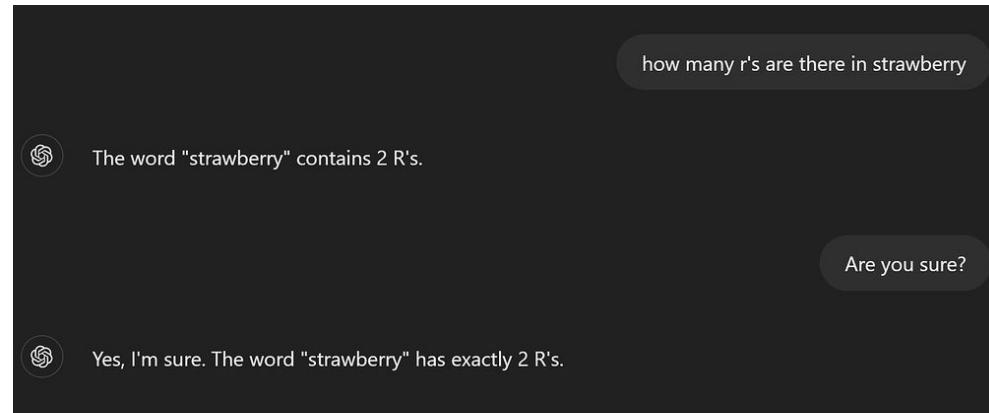
2. Structurez votre prompt

Contexte : Tu es un expert en [domaine]

Tâche : [Action précise à effectuer]

Format : [Structure de réponse souhaitée]

Contraintes : [Limitations/style/longueur]



3. Utilisez des exemples (Few-shot)

"Voici 2 exemples de ce que j'attends : Exemple 1: [...] Exemple 2: [...] Maintenant, fais la même chose pour : [votre cas]"

4. Demandez du raisonnement

"Explique ton raisonnement étape par étape"

🔑 **Règle d'or** : Plus votre prompt est précis, meilleure sera la réponse !

LLM : chatbot

Des LLM adaptés à la conversation

- Entraînement spécialisé :

- Données de type instruction/conversation
- Optimisation pour le format question-réponse plutôt que continuation de texte

Différence clé

LLM de base : "Le chat mange..." → "de la nourriture dans sa gamelle"

LLM Chatbot : "Que mange un chat ?" → "Les chats sont carnivores et mangent principalement..."

 **Résultat** : Interactions naturelles et réponses structurées adaptées au dialogue

Quiz/Take Home message



Langage naturel

- Quelles sont les caractéristiques du langage naturel ?
- Comment représenter le texte numériquement ?



Entraînement des LLM

- Quel volume de données pour entraîner un LLM ?
- À quelle tâche un modèle GPT est-il entraîné ?



Architecture

- Qu'est-ce que la fenêtre de contexte ?
- À quoi correspond la taille de sortie ?

Fine-tuning

Fine-tuning

⌚ Définition

Adaptation d'un modèle pré-entraîné à une tâche ou domaine spécifique

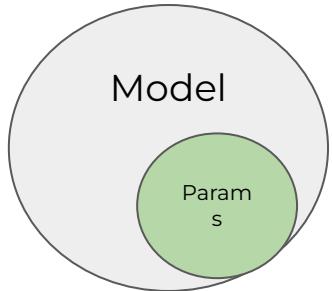
🤔 Pourquoi faire du fine-tuning ?

- **Spécialisation** : Adapter le modèle à un domaine précis (médical, juridique...)
- **Performance** : Améliorer les résultats sur des tâches spécifiques
- **Style** : Modifier le ton ou format de réponse
- **Efficacité** : Obtenir de meilleurs résultats qu'avec du prompting seul

✓ Avantages

- **Conserve** les connaissances générales du modèle de base
- **Nécessite moins** de données que l'entraînement from scratch
- **Coût réduit** par rapport à l'entraînement complet

Processus : Fine-tuning



Modèle de base

Modèle avec des milliards de paramètres entraînés pendant des milliers d'heures sur des très gros volume de données



Modèles open source/propriétaires

Modèles Open source

Avantages :

- **Contrôle total** (fine tuning possible)
- **Confidentialité**
- **Coût**
- **Transparence**

Qu'est-ce que Hugging Face ?

- **Plateforme collaborative** pour partager modèles d'IA et datasets
- **GitHub de l'IA** : +1,000,000 modèles publics disponibles



En 2 ligne de code, importer et utiliser un modèles

Modèles propriétaires

Deux modes d'accès :

- **Interface web** : Utilisation directe via navigateur
- **API** : Intégration dans vos applications

Avantages :

- **Performance maximale** : Modèles de pointe constamment mis à jour
- **Simplicité** : Pas de gestion d'infrastructure

Les prix OpenAI ont baissé de 83% depuis le lancement de GPT-4

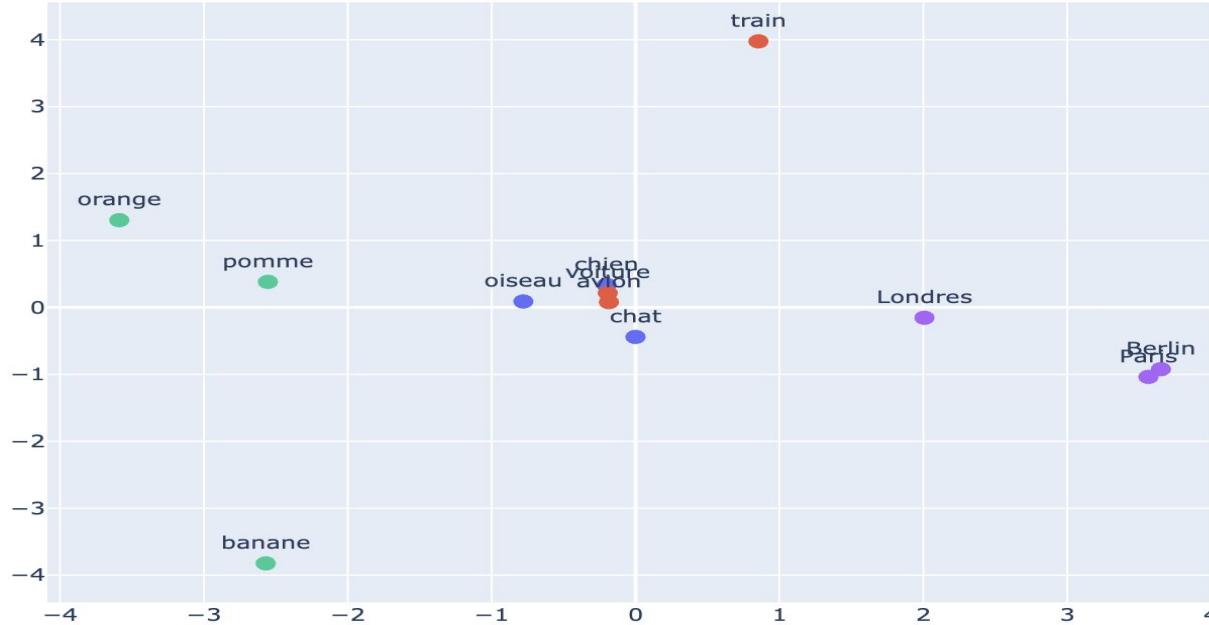
Plateforme	Modèle phare	Prix/1M tokens (Entrée/Sortie)
OpenAI	GPT-4.1	\$3/\$10
Anthropic	Claude Sonnet 4	\$3/\$15
Google	Gemini 1.5 Pro	\$1.25/\$5
Mistral AI	Mistral Large 2	\$0.4/\$2
DeepSeek	DeepSeek-V3	\$0.27/\$1.10
xAI	Grok 3	\$3/\$15

Embedding / Similarities

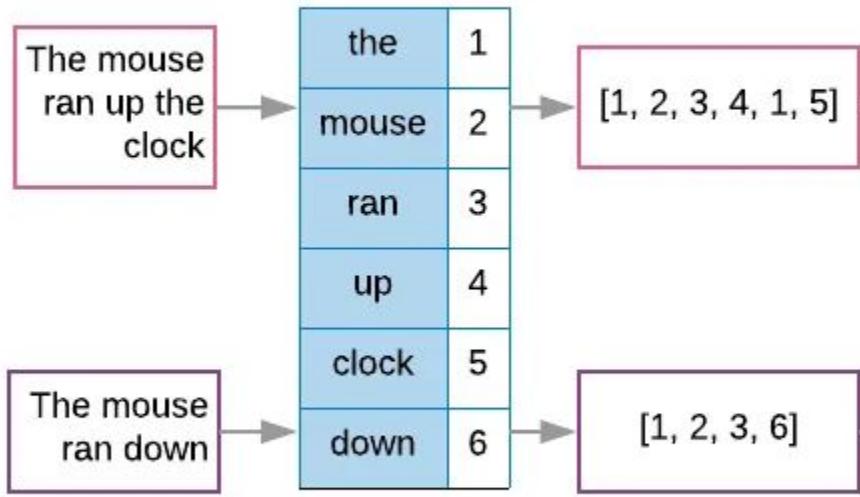
Embeddings

Définition : Représentation numérique dense d'un mot/texte dans un espace vectoriel multidimensionnel

- Transformer du texte en vecteur de nombres réels
- **Chaque dimension** représente une caractéristique abstraite des données
- **Mots/textes similaires → vecteurs proches dans l'espace**



Embeddings vs Token

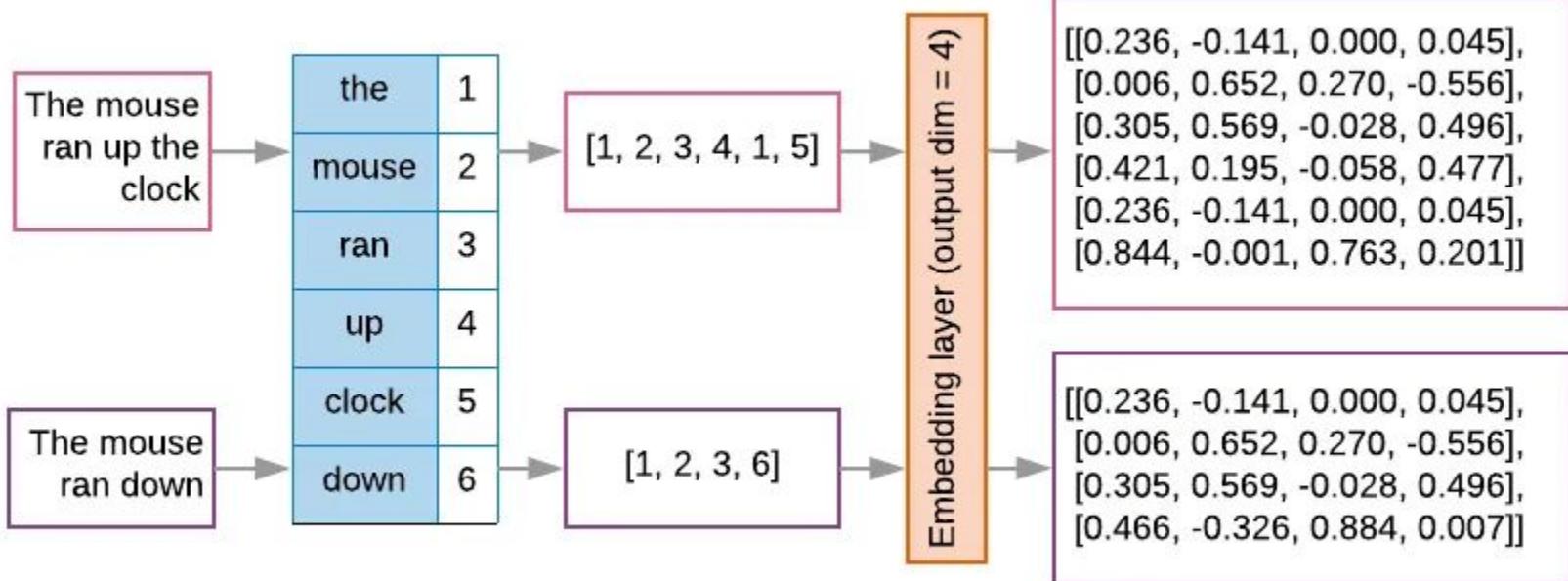


Tokens = découpage du texte

Embeddings = capture du sens:

- Token : "chat" → ID 1547 (juste un identifiant)
- Embedding : "chat" → [0.2, -0.1, 0.8, ...] (vecteur porteur de sens)

Embeddings



En pratique : on embeded des tokens

Construire des Embeddings avec les LLM

Comment un LLM produit des embeddings ?

- Un LLM traite le texte couche par couche
- Chaque couche enrichit la compréhension du sens
- Les **représentations intermédiaires** deviennent nos embeddings

Exemple :

Texte d'entrée : "Le chat dort paisiblement"



Tokenisation : [Le, chat, dort, paisiblement]



Couche 1 : [vecteur_basique_1, vecteur_basique_2, ...]

Couche 2 : [vecteur_enrichi_1, vecteur_enrichi_2, ...]

...

Couche N-1 : [embedding_final_1, embedding_final_2, ...] ← ****On récupère ça !****



Couche finale : Prédiction du token suivant

Embeddings de phrases/documents

- Un seul vecteur pour tout le texte
- Techniques : moyenner les tokens

Exemple : Embedding de phrase

Phrase d'entrée : "Le chat dort"



Tokenisation : [Le] [chat] [dort]



LLM (couches intermédiaires) produit des embeddings individuels :



"Le" → [0.2, -0.1, 0.8, 0.3]

"chat" → [0.4, 0.3, 0.1, -0.2]

"dort" → [0.1, 0.5, -0.2, 0.7]



Moyennage (addition ÷ 3) :



Embedding final : [0.23, 0.23, 0.23, 0.27]

Similarité : Embeddings

Tailles d'embeddings classiques:

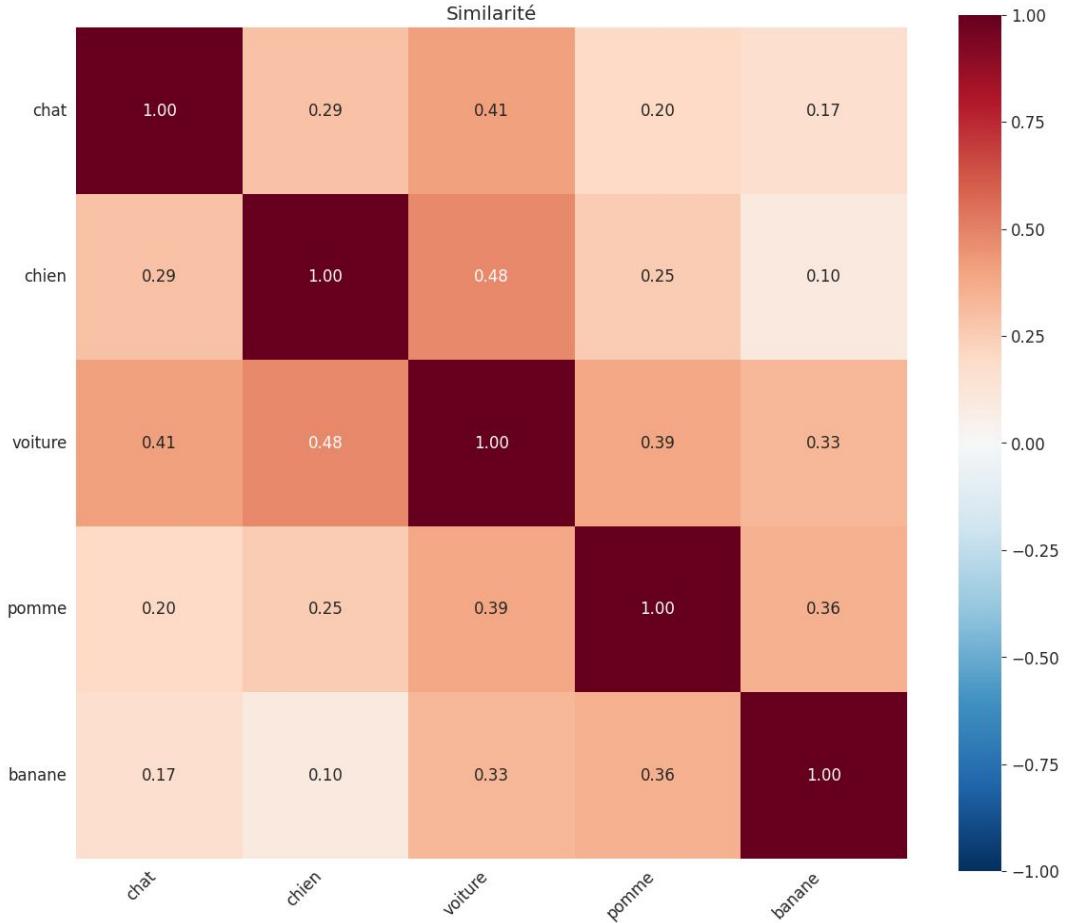
Entre **256** et **2048**

 **Impact** : Plus la dimension est élevée, plus l'embedding capture de nuances, mais plus il consomme de mémoire et de calcul.

Calcul de similarité : Distance Cosinus

Principe : Mesure l'angle entre deux vecteurs (de 0 à 1)

- **1.0** = Identiques (même direction)
- **0.0** = Orthogonaux (aucune relation)
- **Proche de 1** = Très similaires



Quiz/Take Home message

Quels sont les limites des LLM ?

Comment utiliser les embeddings pour résoudre ces problèmes ?



RAG

RAG

RAG (Retrieval-Augmented Generation) : Combinaison de recherche d'information et génération de texte

Problème résolu :

- LLM limités par leurs données d'entraînement (knowledge cutoff)
- Impossible de connaître toutes les informations spécifiques/récentes
- Hallucinations sur des faits non connus

Principe :

1. **Rechercher** des informations pertinentes dans une base de connaissances
2. **Augmenter** le prompt avec ces informations comme contexte
3. **Générer** une réponse basée sur les informations récupérées

Avantage clé : Réponses factuelles et à jour sans re-entraîner le modèle

RAG : Ingestion des Données

📁 Étape 1 : Extraction et Préparation

- **PDFs complexes** : Tableaux, images, mise en page
- **Formats propriétaires** : Extraction du texte "propre"
- **Métadonnées** : Préserver titre, date, auteur, source
- **Encodage** : Gestion des caractères spéciaux et langues

✂️ Étape 2 : Chunking

- **Limite des embeddings** : Performance optimale sur 200-1000 mots
- **Recherche précise** : Retrouver le passage exact, pas tout le document
- **Contexte LLM** : Injecter uniquement l'information pertinente

📊 Étape 3 : Génération des Embeddings

Choix du modèle d'embedding

- **text-embedding-3-small** (OpenAI) : 1536 dimensions
- **all-MiniLM-L6-v2** : 384 dimensions (rapide, gratuit)
- **multilingual-e5-large** : Support multilingue

🗄️ Étape 4 : Stockage Vectoriel

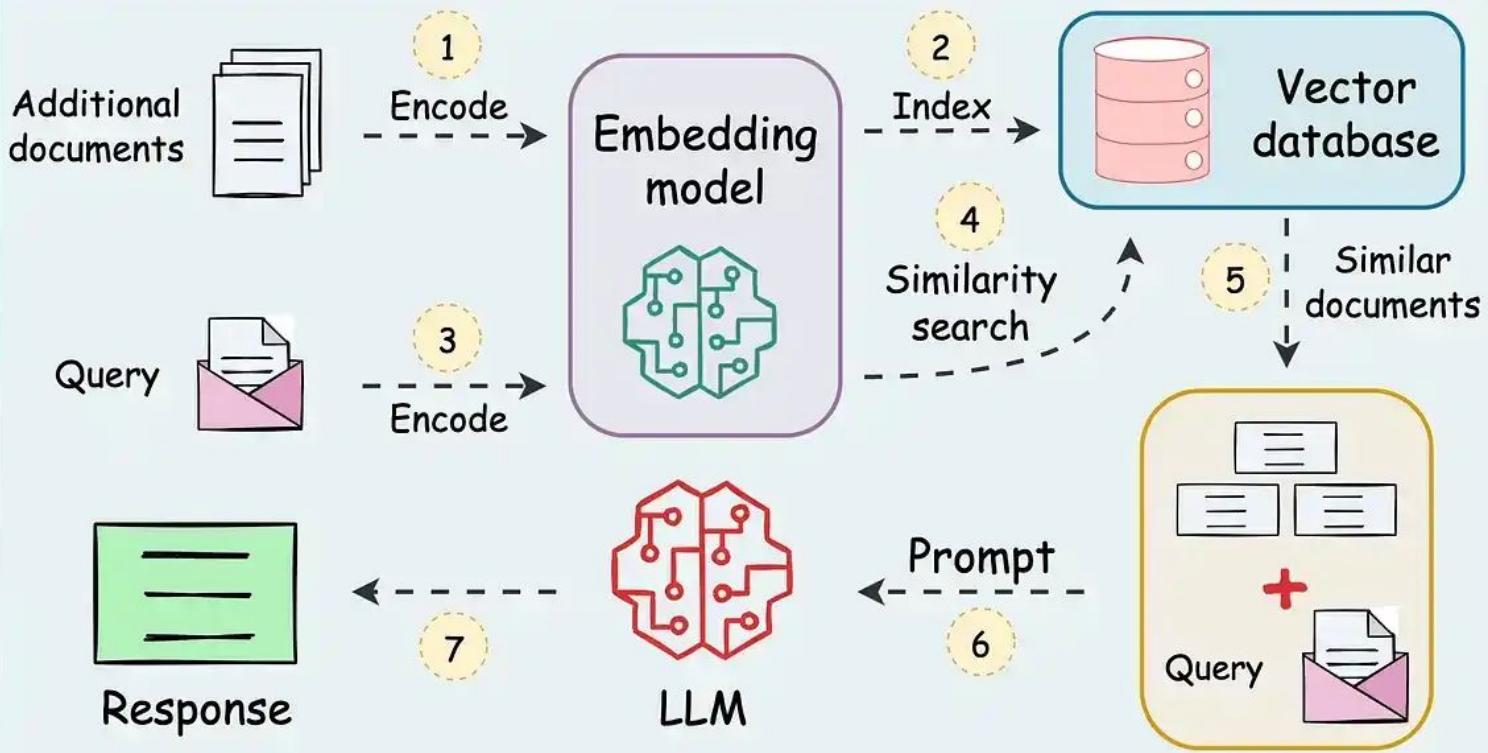
Bases de données vectorielles

- **Pinecone** : Cloud, simple d'usage
- **Chroma** : Open source, local
- **Weaviate** : Recherche hybride (texte + vecteur)
- **FAISS** : Bibliothèque Meta, très rapide

Indexation et performance

- **Index HNSW** : Recherche approximative rapide
- **Métadonnées** : Filtrage par date, source, catégorie
- **Mise à jour** : Ajout/suppression incrémentale

RAG



RAG: Préparation - Indexation des Documents

Documents de l'entreprise à indexer :

Document A : "Les congés payés sont de 25 jours par an pour tous les employés à temps plein..."

Document B : "Le télétravail est autorisé jusqu'à 3 jours par semaine après validation du manager..."

Document C : "La mutuelle d'entreprise rembourse 80% des frais dentaires et 100% des consultations..."

Processus d'indexation :

Document A → Modèle d'embedding → [0.2, -0.1, 0.8, ...] → Base vectorielle

Document B → Modèle d'embedding → [0.5, 0.3, -0.2, ...] → Base vectorielle

Document C → Modèle d'embedding → [-0.1, 0.7, 0.4, ...] → Base vectorielle

Résultat : Base de données vectorielle prête pour la recherche sémantique

RAG: Recherche - Query Utilisateur et Similarité

Question employé :

"Combien de jours de vacances ai-je droit ?"

Processus de recherche :

Query : "Combien de jours de vacances ai-je droit ?"



Modèle d'embedding → [0.3, -0.2, 0.7, ...]



Calcul similarité avec tous les documents :



Document A : Score 0.89 ★ (le plus proche)

Document B : Score 0.23

Document C : Score 0.15

Documents récupérés (top 2) :

1. **Document A** (score: 0.89) - Politique des congés
2. **Document B** (score: 0.23) - Télétravail (*moins pertinent mais récupéré*)

RAG: Génération - Construction du Prompt Enrichi

Construction du prompt final :

CONTEXTE RÉCUPÉRÉ :

—
Document A : "Les congés payés sont de 25 jours par an pour tous les employés à temps plein..."

Document B : "Le télétravail est autorisé jusqu'à 3 jours par semaine après validation du manager..."

INSTRUCTION AU LLM :

En te basant sur le contexte ci-dessus, réponds à la question suivante de manière précise et utile :

QUESTION : "Combien de jours de vacances ai-je droit ?"

RÉPONSE :

Le LLM reçoit : Question + Documents pertinents + Instructions

RAG: Réponse Finale - Génération Informée

Réponse du LLM enrichi par RAG :

"Selon la politique de l'entreprise, vous avez droit à **25 jours de congés payés par an** si vous êtes employé à temps plein. Cette information provient de notre document de politique RH officiel."

Avantages observés :

Sans RAG : "Je ne connais pas la politique spécifique de votre entreprise..."

Avec RAG :

- Réponse précise et factuelle
- Basée sur les documents officiels
- Pas d'hallucination
- Information à jour

Le cycle RAG complet :

Indexation → Recherche → Augmentation → Génération

Agentic

RAG Agentique - Définition et Enjeux

🤔 Limites du RAG traditionnel :

RAG classique : "Quelle est notre politique de congés ?"

- Une recherche → Un document → Une réponse

Question nécessitant calculs : "Combien coûte une formation Excel pour notre équipe de 5 personnes ?"

- RAG traditionnel : Trouve le prix unitaire mais ne fait pas le calcul
- Pas de raisonnement sur les remises éventuelles
- Une seule recherche insuffisante

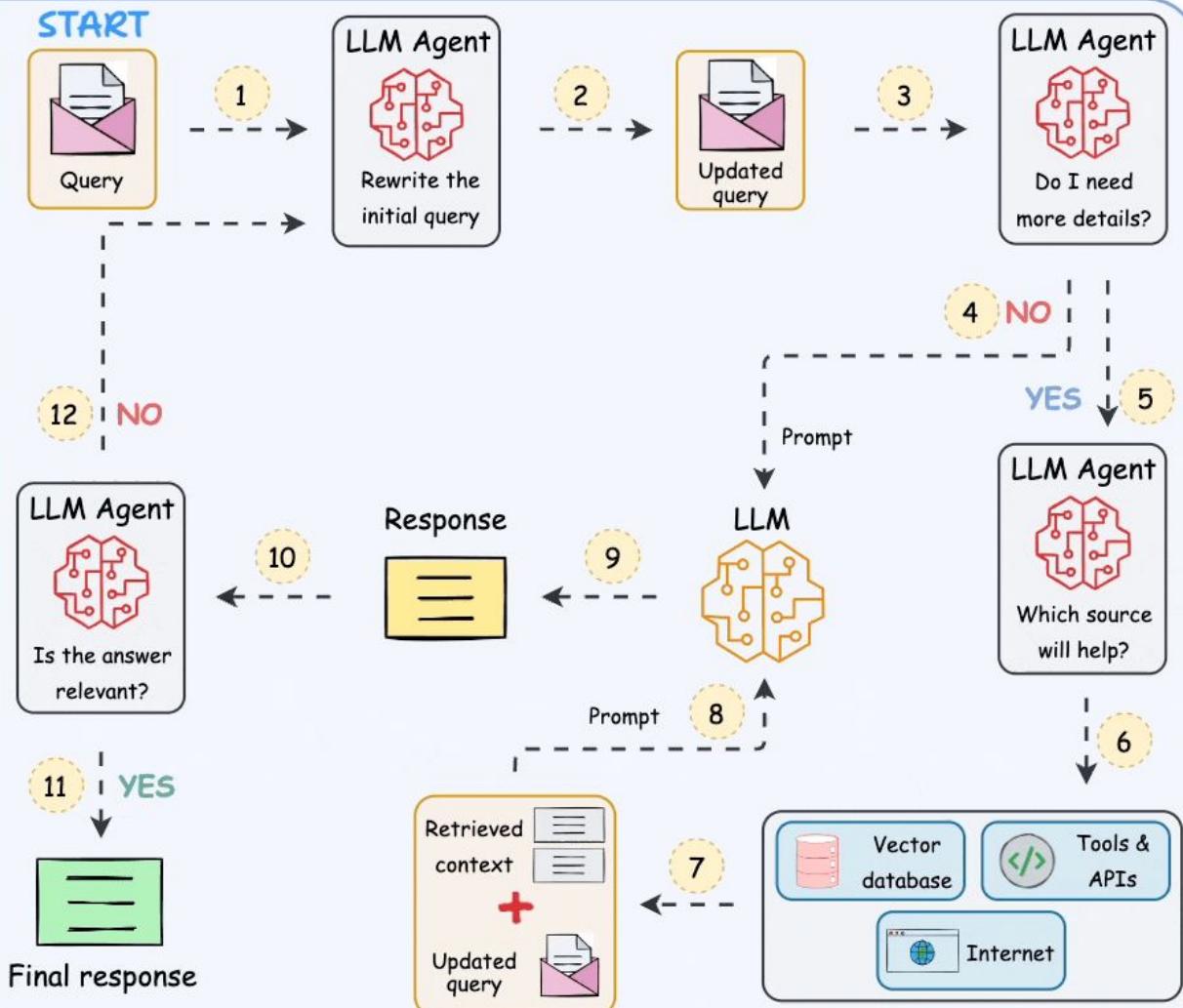
🧠 RAG Agentique = Agents + Outils

Définition : Système où des **agents intelligents** utilisent des **outils spécialisés** pour résoudre des problèmes complexes

Distinction clé :

- **Agents** = LLM avec prompts spécifiques qui prennent des décisions
- **Outils** = Fonctions que les agents peuvent utiliser (calculatrice, recherche, APIs...)

Agentic RAG



Librairies



LangChain

Framework Python pour construire des applications LLM avec des chaînes de traitement modulaires

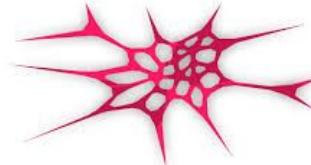


LangGraph

Extension de LangChain pour créer des workflows multi-agents avec des graphes de décision

FAISS

Scalable Search With Facebook AI



Bibliothèque Meta ultra-rapide pour la recherche de similarité dans de grandes collections de vecteurs



Chroma

Base de données vectorielle open source simple à utiliser, idéale pour débuter en RAG local

