



Droit
Économie
Management
Université d'Orléans

AI4ESA

Les bases, les enjeux, les risques

Gilles de Truchis

Université d'Orléans, 3 septembre 2026



Introduction

1. AI Genesis
2. AI Arena
3. AI Core
4. AI & Python
5. AI RAG via LangChain
6. AI Agent via LangGraph
7. Intégration Profonde
8. Risques & Limites



Au commencement, étaient les IA conversationnelles

- **ChatGPT (OpenAI)**
 30 novembre 2022 : conversation naturelle, rédaction, aide générale
- **Copilot (Microsoft)**
 7 février 2023 : intégration Office 365, productivité bureautique
- **Gemini (Google DeepMind)**
 6 décembre 2023 : intégration Google Workspace, recherche avancée

⇒ Généralistes mais...

-  **Manque** compréhension du contexte
-  **Hallucinations** sur des sujets techniques
-  **Absence** d'accès aux données spécifiques de l'utilisateur

- **Le Chat (Mistral AI)**
 27 septembre 2023 : alternative européenne, précurseur dans les IA expertes

Nouvelle génération d'IA Expertes : l'exemple de Claude

Claude (Anthropic)

- 🔑 14 mars 2023 : raisonnement profond, code avancé, analyse
- 🧠 Modèles **actuels** : Haiku 4.5 (flash), Sonnet 4.6 (équilibré) et Opus 5 (raisonnement complexe)
- ⚙️ Capacités Sonnet : 📶 1M / 📶 64k (si Opus 5, 📶 1M / 📶 128k ; Haiku 4.5 : 📶 200k)
- 🔧 Outils intégrés : connecteur API, **artéfacts**
- 👤 Capacité **agentique** (e.g. **Deep Research**)
- 🏢 Analyse de **projets** entiers via des **RAG agentiques** multi-modaux
- 📋 Prise en charge des **MCP** (e.g. **RAG agentique**, **Vibe Coding**)

Intégration :

- 🌐 Navigateur web : **Claude AI** (<https://claude.ai>)
- 🔗 Extension VS Code : **Claude Code** (<https://www.anthropic.com/claude-code>)
- 🖥️ Windows / Mac : **Claude Desktop** (<https://claude.ai/download>)

Vers des modèles spécialisés

- **OpenAI** : famille **GPT-5.6** (9 juillet 2026) — 📁 1,05M / 📁 128k 🌐, coupure des données : 16 fév. 2026
 - GPT-5.6 **Sol** : le vaisseau amiral (raisonnement, code agentique)
 - GPT-5.6 **Terra** : équilibre capacité / coût
 - GPT-5.6 **Luna** : latence et prix minimaux
 - Les trois partagent contexte, coupure et outils : on change de palier sans réécrire son code
- **DeepMind** : **Gemini 3.1 Pro** (haut de gamme) et **Gemini 3.7 Flash** (13 août 2026, nouveau défaut) — 📁 1M / 📁 64k 🌐
 - Gemini 3.1 / 3.8 Flash, 3.5 Flash-Lite : la gamme rapide, très étagée
 - Gemini 3 Deep Think (raisonnement long), Gemini Omni (génération multimodale)
 - Nomenclature désynchronisée : Flash est en 3.8, Pro encore en 3.1
- **Mistral AI** : **Large 3** (675B dont 41B actifs, MoE, Apache 2.0)
 - Medium 3.5 (agentique et code), Devstral 2 / Codestral (code)
 - Small 4 (119B dont 6B actifs unifie raisonnement, vision et code)
 - Ministral 3 (3B, 8B, 14B *edge computing*)

Mixture of Experts (MoE) : Architecture des modèles modernes

Architecture Traditionnelle :

- 🧠 Tous les paramètres activés pour chaque token
- ⚠️ Coût computationnel élevé
- 🏢 Utilisation inefficace des ressources

Architecture MoE :

- 👥 Multiples réseaux "experts" spécialisés
- 🧭 Router intelligent qui sélectionne ℓ experts par token
- 🔑 Seuls les experts sélectionnés sont activés
- ⚖️ Compromis : plus de paramètres totaux, moins activés simultanément

Exemple Concret :

- 🌀 Token : "navigue"
- ➔ Router analyse le contexte
- ➔ Sélection : Expert maritime + Expert physique
- ➔ Sortie = combinaison pondérée

Avantages :

- 🔧 Performance : spécialisation par domaine
- 🧠 Efficacité : activation sélective
- 📦 Scalabilité : ajout d'experts sans surcoût linéaire

Architectures MoE documentées :

- Mistral Large 3 (675B totaux, 41B actifs)
- Mistral Small 4 (119B totaux, 6B actifs)
- Kimi K3, GLM 5.x, DeepSeek V4, Qwen 3.8
- OpenAI, Google et Anthropic ne publient pas leurs architectures : le MoE y est *présumé*, non confirmé

AI Arena

1. AI Genesis
2. AI Arena
3. AI Core
4. AI & Python
5. AI RAG via LangChain
6. AI Agent via LangGraph
7. Intégration Profonde
8. Risques & Limites



Agent Arena : overview sur la performance, la vitesse et la latence

Top 10 selon <https://artificialanalysis.ai/leaderboards/models>

	Features →		Intelligence →		Price →	Speed →	Latency →	End-to-End Response Time →
Model ↕	Context Window ↕	Creator ↕	Artificial Analysis Intelligence Index ↓	Cost per Task USD ↕	Median Tokens/s ↕	Latency First Chunk (s) ↕	Total Response (s) ↕	
📄 Claude Opus 5 (max)	1M	🏠 Anthropic	63	\$2.34	56	51.78	60.70	
📄 Claude Opus 5 (xhigh)	1M	🏠 Anthropic	63	\$1.80	54	25.15	34.44	
📄 Claude Fable 5 (with fallback)	1M	🏠 Anthropic	62	\$3.14	65	95.42	103.15	
📄 Claude Opus 5 (high)	1M	🏠 Anthropic	61	\$1.23	55	20.40	29.55	
📄 GPT-5.6 Sol (max)	1M	🌐 OpenAI	61	\$0.95	74	140.04	146.81	
📄 Grok 4.6 (high)	500k	🚀 SpaceXAI	61	\$0.94	58	42.98	51.64	
📄 Grok 4.6 (xhigh)	500k	🚀 SpaceXAI	60	\$1.23	59	41.75	50.18	
📄 Kimi K3 (max)	1.05M	🇰🇷 Kimi	60	\$0.84	36	6.51	76.05	
📄 GLM-5.3 (max)	1M	🇨🇳 Z AI	60	\$0.68	66	1.62	39.21	
📄 GPT-5.6 Sol (xhigh)	1M	🌐 OpenAI	59	\$0.63	73	58.92	65.73	
📄 Grok 4.6 (medium)	500k	🚀 SpaceXAI	59	\$0.78	57	34.53	43.25	
📄 Claude Opus 5 (medium)	1M	🏠 Anthropic	59	\$0.72	54	5.13	14.43	
📄 Qwen3.8 Max	1M	🇨🇳 Alibaba	58	\$0.91	21	2.55	121.44	
📄 Qwen3.8 2.4T A95B	984k	🇨🇳 Alibaba	58	\$0.81	24	2.55	104.70	

Agent Arena : orchestration d'outils et tâches agentiques

Top 10 selon le *Net Improvement* au 24 août 2026 (<https://arena.ai/leaderboard/agent>)

Rg	Modèle	Net impr.	Succès	Steerab.	Bash	Halluc.
1	Claude Opus 5 (High)	+12.73%	+15.97%	+13.36%	+14.22%	+0.99%
2	Claude Opus 5 (Max)	+12.41%	+17.21%	+9.93%	+14.88%	+1.05%
3	Claude Fable 5 (High)	+11.62%	+11.16%	+13.87%	+12.82%	+1.07%
4	GPT 5.6 Sol (xHigh)	+10.31%	+10.02%	+9.16%	+9.06%	+1.08%
5	Claude Opus 4.8 (High)	+9.99%	+7.49%	+11.10%	+9.15%	-0.21%
6	Kimi K3 (Max)	+9.53%	+18.24%	+2.31%	+8.47%	+1.08%
7	GPT 5.5 (xHigh)	+8.56%	+4.00%	+9.30%	+14.70%	+1.07%
8	Claude Sonnet 5 (High)	+8.05%	+1.31%	+12.52%	+11.00%	+0.95%
9	Claude Opus 4.7 (High)	+7.27%	+4.56%	+7.59%	+12.44%	+0.98%
10	Claude Opus 4.7	+7.18%	+4.96%	+8.75%	+9.84%	+1.03%

Lecture : chaque colonne est un **effet causal** (*net improvement*) estimé sur 2,0 M de sessions réelles ; + = mieux que le modèle moyen. Le rang est leur moyenne.



Claude : 7 modèles dans le top 10, dont les 3 premiers



Kimi K3 (poids ouverts) : n°1 sur le *succès confirmé* (+18.2%)



L'hallucination d'outils ne discrimine presque plus ($\approx +1\%$ partout) : le signal est saturé, il faudra en changer

Génération de texte et de code

Top 10 des modèles pour Text et Code (WebDev) — 21 août 2026

Text (7,9 M votes)

Rang	Modèle	Score	Votes
1	claude-fable-5	1508	24.3k
2	claude-opus-4-6-high	1504	72.4k
3	claude-opus-4-7-high	1502	60.2k
4	muse-spark-1.2 (xH.)	1498	3.3k
5	claude-opus-4-6	1497	76.4k
6	claude-opus-4-7	1494	61.3k
7	claude-opus-5-high	1493	27.6k
8	muse-spark-1.1	1491	20.2k
9	gemini-3.7-flash-h.	1490	5.7k
10	kimi-k3-max	1489	15.1k

Code / WebDev (604k votes)

Rang	Modèle	Score	Votes
1	claude-opus-5-max	1691	8.1k
2	kimi-k3-max	1674	4.5k
3	qwen3.8-max	1669	3.2k
4	claude-opus-5-high	1663	8.7k
5	grok-4.6-high	1629	1.5k
6	claude-fable-5	1626	8.4k
7	gpt-5.6-sol-xhigh	1619	9.5k
8	glm-5.3-max	1599	1.9k
9	qwen3.8-27b	1595	2.5k
10	gemini-3.7-flash-h.	1587	2.6k

Points clés :

- 🏆 **Claude Opus 5 (Max)** : 1^{er} en WebDev (1691), 65 points devant le 4^e
- 🔗 **Poids ouverts** : Kimi K3, Qwen 3.8, GLM 5.3 occupent 4 des 10 places en code
- ⚠️ **Prudence** : les $\pm$ des scores (5 à 17 pts) recouvrent souvent 3 ou 4 rangs

Génération vidéo : Text-to-Video et Image-to-Video

Top 10 des modèles pour la génération vidéo — 14 août 2026

Text-to-Video (617k votes)

Rang	Modèle	Score	Votes
1	gemini-omni-flash	1512	16.9k
2	flux-3-video	1494	1.3k
3	dreamina-seed.-2.0	1482	49.1k
4	dreamina-seed.-2.5	1477	1.3k
5	muse-video	1457	2.2k
6	minimax-h3	1453	3.1k
7	happyhorse-1.0	1428	22.1k
8	sora-2-pro	1364	46.5k
9	veo-3.1-audio	1364	13.7k
10	veo-3.1-audio-1080p	1363	25.0k

Image-to-Video (1,7 M votes)

Rang	Modèle	Score	Votes
1	minimax-h3	1489	13.0k
2	dreamina-seed.-2.5	1484	2.9k
3	dreamina-seed.-2.0	1478	105.3k
4	gemini-omni-flash	1462	54.2k
5	grok-imagine-v1.5	1460	99.6k
6	flux-3-video	1449	3.8k
7	happyhorse-1.0	1442	70.8k
8	wan2.7-i2v	1427	57.9k
9	grok-imagine-vid-720p	1415	517.1k
10	veo-3.1-audio	1397	25.2k

Tendances :

- 🏆 **Gemini Omni Flash** (Google) : nouveau leader Text-to-Video (1512)
- 🏆 **MiniMax H3** (poids ouverts) : leader Image-to-Video — une première
- 📉 **Veo 3.1** décroche : d'omniprésent, il tombe au bas du top 10

Vision et génération d'images

Top 10 des modèles pour Vision et Text-to-Image

Vision (21 août 2026)

Rang	Modèle	Score	Votes
1	claude-fable-5	1312	9.6k
2	qwen3.8-max	1302	6.7k
3	claude-opus-4-7-high	1301	21.1k
4	claude-opus-4-7	1299	21.4k
5	claude-opus-4-6-high	1299	20.9k
6	muse-spark	1294	5.6k
7	claude-opus-4-6	1293	25.2k
8	muse-spark-1.2 (xH.)	1292	1.9k
9	claude-opus-5-high	1292	7.0k
10	gemini-3-pro	1289	13.1k

Text-to-Image (10 août 2026)

Rang	Modèle	Score	Votes
1	gpt-image-2 (medium)	1381	70.1k
2	mai-image-2.6-prev.	1336	3.5k
3	grok-imagine-img-2.0	1316	2.7k
4	reve-2.1	1302	7.6k
5	muse-image	1282	15.1k
6	reve-2.0	1270	14.6k
7	gemini-3.1-flash-img	1264	29.1k
8	seedream-5.0-pro	1258	28.8k
9	qwen-image-3.0-pro	1257	4.7k
10	mai-image-2.5	1256	46.3k

Analyse :

- 👁️ **Claude Fable 5** : leader en Vision ; Claude occupe 6 des 10 places
- 🖼️ **GPT-Image 2** : leader Text-to-Image, avec 45 points d'avance
- 🤖 **Microsoft AI** (mai-image) et **Meta** (muse) entrent dans le haut du tableau

Édition d'images et recherche

Top 10 des modèles pour Image Edit et Search

Image Edit (17 août 2026)

Rang	Modèle	Score	Votes
1	gpt-image-2 (medium)	1463	199.3k
2	grok-imagine-img-2.0	1439	5.9k
3	mai-image-2.6-prev.	1420	5.6k
4	muse-image	1406	54.7k
5	mai-image-2.5	1401	170.3k
6	seedream-5.0-pro	1394	102.3k
7	grok-imagine-img-q.	1390	35.6k
8	gemini-3-pro-img-2k	1390	517.4k
9	chatgpt-image-lat.-hf	1389	508.7k
10	gemini-3.1-flash-img	1386	117.9k

Search (24 août 2026)

Rang	Modèle	Score	Votes
1	gpt-5.6-sol-xhigh	1257	29.7k
2	claude-opus-4-6-se.	1253	134.7k
3	gpt-5.5-search	1242	89.9k
4	claude-opus-4-7	1233	91.4k
5	claude-fable-5	1230	41.8k
6	ernie-5.1	1227	3.8k
7	claude-sonnet-4-6-s.	1221	134.9k
8	grok-4.5	1213	31.5k
9	gemini-3.1-pro-gr.	1210	113.3k
10	gemini-3-pro-gr.	1207	37.0k

Synthèse :

 **GPT-Image 2** : leader sur les deux arènes image (génération et édition)

 **GPT-5.6 Sol** reprend la tête en recherche web à Claude Opus 4.6

 Le *grounding* Gemini reste massivement utilisé (150k votes cumulés)

Synthèse : Les champions par catégorie

Récapitulatif des leaders (arena.ai, relevés du 10 au 24 août 2026)

Catégorie	Leader	Forces
Agentique	Claude Opus 5 (High)	Orchestration d'outils, 7 Claude dans le top 10
Code (WebDev)	Claude Opus 5 (Max)	1691 pts, 65 d'avance sur le 4 ^e
Text	Claude Fable 5	Génération de texte, cohérence longue
Vision	Claude Fable 5	Analyse visuelle, OCR, diagrammes
Search	GPT-5.6 Sol	Recherche web sourcée
Text-to-Image	GPT-Image 2	Haute fidélité, rendu de texte
Image Edit	GPT-Image 2	Édition précise, cohérence de scène
Video (T2V / I2V)	Gemini Omni / MiniMax H3	Google en T2V, un modèle ouvert en I2V

Observations stratégiques :

- 🏆 **Spécialisation** : Anthropic sur texte/code/agent, OpenAI sur image/recherche
- ★ **Poids ouverts** : Kimi K3, Qwen 3.8, GLM 5.3, MiniMax H3 sont désormais dans les tops
- 🔄 **Péréemption rapide** : ce tableau a une durée de vie de quelques semaines

FrontierMath : Le benchmark pour les mathématiques

Qu'est-ce que FrontierMath ?

 **Définition** : Benchmark de problèmes mathématiques de niveau doctorat et recherche

 **Difficulté** : Organisé en tiers (niveaux) de complexité croissante

— **Tier 1-3** : Problèmes de difficulté standard à élevée

— **Tier 4** : Problèmes de niveau recherche avancée

 **Objectif** : Mesurer les capacités de raisonnement mathématique approfondi des IA

 **Importance** : Révèle les limites actuelles des modèles sur des tâches complexes

Observations clés :

 Les performances restent modestes même pour les meilleurs modèles

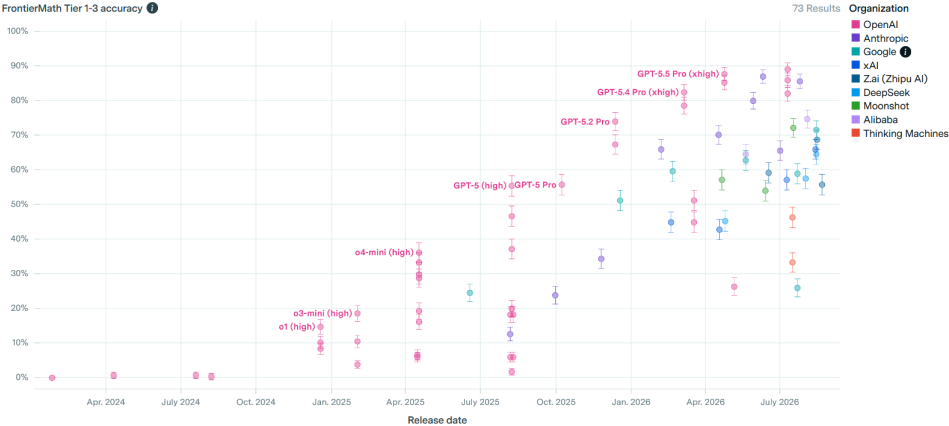
 Progression rapide depuis 2024 (juillet 2024 : <5% juin 2026 : 50%)

 OpenAI domine actuellement ce benchmark difficile

FrontierMath : Tier 1-3 (Standard à Difficile)

AI performance on a set of expert-level mathematics problems

FrontierMath Tier 1-3 accuracy ⓘ

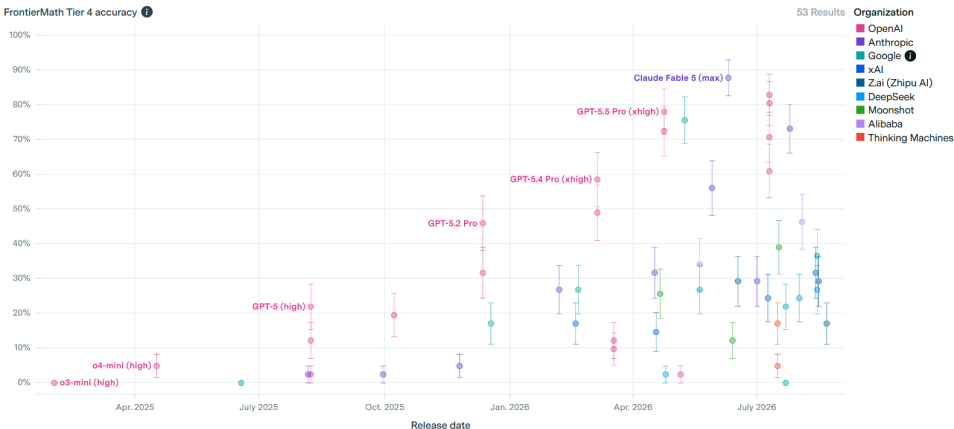


FrontierMath : Tier 4 (Niveau Recherche)

AI performance on the most challenging expert-level mathematics problems



FrontierMath Tier 4 accuracy 



Section 2

1. AI Genesis
2. AI Arena
3. AI Core
4. AI & Python
5. AI RAG via LangChain
6. AI Agent via LangGraph
7. Intégration Profonde
8. Risques & Limites



Feuille de route : de la régression linéaire aux LLM

Tout ce chapitre est une **seule** idée déclinée six fois : *empiler des régressions, et les estimer par descente de gradient*. Voici le chemin.



acquis

nouveau — mais construit brique à brique

✓ Prérequis

Régression multiple et MCO ; maximum de vraisemblance ; dérivée et dérivée partielle ; produit matrice-vecteur. **Rien d'autre.**

A2 Notre convention.

On va essayer de vulgariser chaque terme technique, puis d'illustrer par un **exemple chiffré**. Pour les M1, il est normal que cela semble abstrait car exceptionnellement le cours est lancé très tôt pour permettre au M2 de le suivre.

Le dictionnaire : économétrie ↔ machine learning

Les deux disciplines ont inventé séparément les mêmes objets ; ce tableau est à garder sous les yeux.

Vous dites...	...la littérature IA dit
variables explicatives, régresseurs X variable expliquée y coefficients $\beta_1, \dots, \beta_d$ constante β_0 estimer le modèle échantillon d'estimation prévision hors échantillon critère à minimiser (SCR, $-\log L$) paramètre de réglage (λ du ridge) ridge / lasso centrer-réduire logit multinomial	<i>features</i> , entrées variable cible (<i>target</i>) poids (<i>weights</i>) w biais (<i>bias</i>) b entraîner le modèle (<i>training</i>) jeu d'entraînement (<i>train set</i>) <i>inférence</i> sur l'échantillon <i>test</i> fonction de perte (<i>loss</i>) hyperparamètre <i>weight decay</i> (L_2) / L_1 normalisation, <i>LayerNorm</i> <i>softmax</i>

⚠ Le faux ami à ne surtout pas confondre

Le mot **biais** a deux sens qui vont cohabiter dans ce cours : **(1) le biais d'un neurone** = la constante β_0 , un paramètre à estimer ; **(2) le biais d'un estimateur** = $\mathbb{E}[\hat{\theta}] - \theta$. Ici, sauf mention contraire, ce sera le sens **(1)**.

Conventions de notation : la carte à garder sous les yeux

Symbole	Signification
n	nombre d'observations
d	nombre d'entrées d'une couche
m_l	nombre de neurones de la couche l
P	nombre total de paramètres
$\mathbf{w}, b$	poids et biais d'un neurone
$\mathbf{W}, \mathbf{b}$	poids et biais d'une couche
θ	tous les paramètres empilés
z	pré-activation ; a ou h : activation
φ	fonction d'activation générique
σ	la sigmoïde $1/(1 + e^{-z})$
ℓ	perte sur un exemple
$\mathcal{L}$	perte moyenne sur les n exemples
η	taux d'apprentissage
$\delta^{(l)}$	signal d'erreur sur $\mathbf{z}^{(l)}$
T	longueur de séquence (en tokens)
$ V $	taille du vocabulaire
H	nombre de têtes d'attention
τ	température d'échantillonnage

⚠ Un changement de convention assumé

En première partie (neurone, couche, MLP), les vecteurs sont des **colonnes** et la matrice multiplie à **gauche** :

$$\mathbf{h} = \varphi(\mathbf{W}\mathbf{x} + \mathbf{b}), \quad \mathbf{W} \in \mathbb{R}^{m \times d}.$$

À partir des **Transformers**, on empile T tokens en **lignes** d'une matrice $\mathbf{X} \in \mathbb{R}^{T \times d}$, et la matrice multiplie à **droite** :

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_q, \quad \mathbf{W}_q \in \mathbb{R}^{d \times d_k}.$$

Ce n'est pas une incohérence : c'est la convention de Vaswani *et al.* (2017), qui permet de traiter toute la séquence en **un seul** produit matriciel. Les deux écritures sont transposées l'une de l'autre : $\mathbf{W}\mathbf{x} = (\mathbf{x}^\top \mathbf{W}^\top)^\top$.

📌 **Autres faux amis** le β de la *LayerNorm* n'est pas celui de la régression (nous l'appellerons b) ; et σ désigne **toujours** la sigmoïde, jamais un écart-type.

Point de départ : la régression multiple, réécrite

☒ Ce que vous écrivez d'habitude

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_d x_{id} + \varepsilon_i.$$

En empilant les d régresseurs dans un vecteur $\mathbf{x}_i$ et les pentes dans $\mathbf{w}$:

$$y_i = \underbrace{\mathbf{w}^\top \mathbf{x}_i + b}_{\text{partie déterministe}} + \varepsilon_i, \quad b \equiv \beta_0.$$

☒ Deux façons d'estimer, un seul résultat

- **MCO** : $\min_{\mathbf{w}, b} \sum_i (y_i - \mathbf{w}^\top \mathbf{x}_i - b)^2$
 $\Rightarrow \hat{\beta} = (X^\top X)^{-1} X^\top y$
- **EMV** ou **MLE** sous $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$: le même $\hat{\beta}$

🔑 Trois choses vont changer — et trois seulement

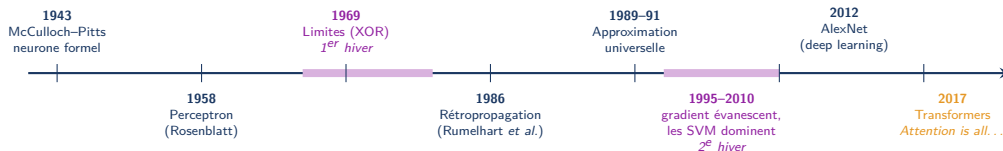
1. β_0 change de nom : on l'appellera le **biais** b , c'est le *même* paramètre
2. On applique une **fonction non linéaire** à $\mathbf{w}^\top \mathbf{x} + b$
3. Il n'y a plus de formule fermée $(X^\top X)^{-1} X^\top y$: on descend le gradient

À retenir

Un réseau de neurones n'introduit **aucun** objet statistique nouveau. Il change la *forme* de la fonction et la *méthode* d'optimisation.

Réseaux de neurones : une généalogie statistique

Pour un statisticien, un réseau de neurones n'est pas une boîte noire exotique : c'est une **composition de modèles linéaires généralisés**, dont les paramètres sont estimés par **minimisation d'un risque** (i.e. **maximisation d'une vraisemblance**).



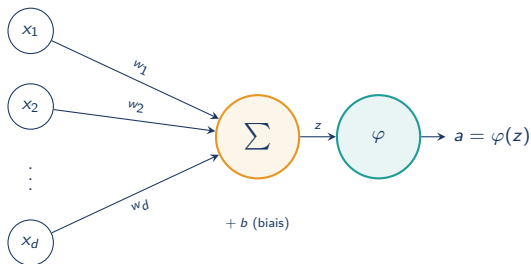
🕒 Deux “hivers”, un point commun

Chaque blocage vient d'une **limite mathématique** — la non-séparabilité en 1969, le **gradient qui s'évanouit** dans les années 1990 — levée par une **idée simple** : la couche cachée, puis la ReLU (cf. slide 29).

🧵 Notre fil rouge

On part de la **régression** que vous connaissez, on l'empile, on apprend ses paramètres par descente de gradient, et l'on arrive naturellement aux **Transformers** en fin de parcours.

Le neurone artificiel : un modèle linéaire généralisé



Combi. linéaire → activation (non-linéaire) → sortie.

Un neurone calcule une **pré-activation** affine, puis lui applique une **activation** :

$$z = \sum_{j=1}^d w_j x_j + b = \mathbf{w}^\top \mathbf{x} + b, \quad a = \varphi(z).$$

Vous connaissez déjà ce modèle :

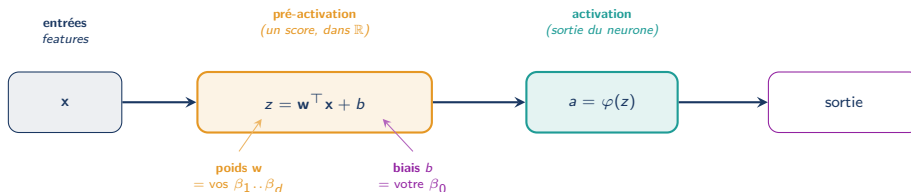
- $\varphi = \text{id} \Rightarrow a = \mathbf{w}^\top \mathbf{x} + b$: **régression linéaire**
- $\varphi = \sigma$ (sigmoïde) $\Rightarrow a = \mathbb{P}(y = 1 \mid \mathbf{x})$: **régression logistique**
- $\varphi = \mathbb{1}(z > 0)$: le *perceptron* de 1958 — abandonné, non dérivable

Remarque

Un neurone à activation sigmoïde est une régression logistique. Un réseau n'est qu'un empilement de tels modèles.

Anatomie d'un neurone : les cinq mots à retenir

Ces cinq termes reviendront dans *toutes* les slides suivantes, y compris dans les Transformers.



🔑 À quoi sert le biais b ?

Exactement à ce que sert la constante en régression : **décaler** le score. Sans lui, $z = 0$ dès que $x = 0$, et la frontière de décision est forcée de passer par l'origine. Le biais rend cette contrainte inutile.

🔍 Pourquoi 2 mots pour une seule étape ?

Parce que la **pré-activation z** et l'**activation a** jouent des rôles **différents** dans l'apprentissage : la **rétropropagation** (cf. slide 45) calcule d'abord un signal d'erreur *sur z* , puis le convertit en gradient *sur w et b* .

Un neurone au calcul

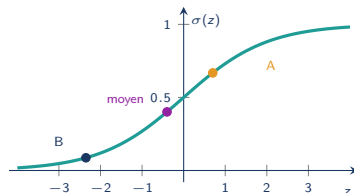
Mini-scoring crédit. Deux variables **centrées-réduites** : x_1 = taux d'effort (part du revenu), x_2 = ancienneté dans l'emploi.
Sortie : probabilité de défaut. Paramètres : $w_1 = 1,5$, $w_2 = -0,8$, $b = -0,4$, $\varphi = \sigma$.

	x_1	x_2	$z = 1,5x_1 - 0,8x_2 - 0,4$	$\sigma(z)$
Client A	1,0	0,5	+0,70	0,668
Client B	-0,5	1,5	-2,35	0,087
Client moyen	0	0	-0,40	0,401

📅 Le détail du client A

$$z = 1,5(1,0) - 0,8(0,5) - 0,4 = 0,70$$

$$\sigma(0,70) = \frac{1}{1 + e^{-0,70}} = \frac{1}{1,4966} = 0,668$$



💡 Lecture du biais

Pour le client **moyen** ($x = 0$), tout dépend de b : $\sigma(-0,4) = 0,401$. Le **biais fixe donc le taux de défaut de référence** ; les poids ne font que déplacer chaque client autour de ce point.

Fonctions d'activation : pourquoi de la non-linéarité ?

⚠ **Le problème.** Empilons deux neurones en série, tous deux *sans* activation ($\varphi = \text{id}$) :

$$h = w_1 x + b_1, \quad \hat{y} = w_2 h + b_2 = \underbrace{(w_2 w_1)}_{\tilde{w}} x + \underbrace{(w_2 b_1 + b_2)}_{\tilde{b}} = \tilde{w} x + \tilde{b}.$$

📊 Vérification chiffrée

$w_1 = 2$, $b_1 = 1$, $w_2 = 3$, $b_2 = -4$ donnent $\tilde{w} = 3 \times 2 = 6$ et $\tilde{b} = 3 \times 1 - 4 = -1$. Pour $x = 2$: $h = 5$ puis $\hat{y} = 15 - 4 = 11$, et directement $6(2) - 1 = 11$. Les deux neurones n'en font qu'un.

📍 En dimension quelconque, le calcul est identique avec les matrices de la couche : $\tilde{\mathbf{W}} = \mathbf{W}_2 \mathbf{W}_1$ et $\tilde{\mathbf{b}} = \mathbf{W}_2 \mathbf{b}_1 + \mathbf{b}_2$.

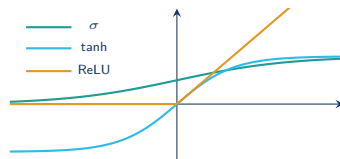
🔑 **Conclusion** : sans φ non linéaire, la profondeur est **gratuite** — au sens où elle n'apporte rien.

📈 **Sigmoïde** $\sigma(z) = \frac{1}{1 + e^{-z}} \in (0, 1)$

📈 **Tanh** $\tanh(z) \in (-1, 1)$, centrée en 0

📈 **ReLU** $\max(0, z)$ — standard moderne

📈 **Softmax** (couche de sortie multi-classes) : $\text{softmax}(\mathbf{z})_k = e^{z_k} / \sum_j e^{z_j}$, la **fonction de réponse** du **logit multinomial**



Pourquoi la non-linéarité change tout : la preuve en chiffres

Deux couches de dimension 2, biais nuls : $\mathbf{W}_1 = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$, $\mathbf{W}_2 = \begin{pmatrix} 1 & 0 \\ 3 & 1 \end{pmatrix}$, donc $\mathbf{W}_2\mathbf{W}_1 = \begin{pmatrix} 1 & 2 \\ 3 & 7 \end{pmatrix}$. On compare le même réseau avec et sans ReLU intercalée.

x	$\mathbf{W}_1\mathbf{x}$	sans activation		avec ReLU	
		$\mathbf{h} = \mathbf{W}_1\mathbf{x}$	$\hat{\mathbf{y}} = \mathbf{W}_2\mathbf{h}$	$\mathbf{h} = \text{ReLU}(\mathbf{W}_1\mathbf{x})$	$\hat{\mathbf{y}} = \mathbf{W}_2\mathbf{h}$
(1, 1)	(3, 1)	(3, 1)	(3, 10)	(3, 1)	(3, 10)
(1, -1)	(-1, -1)	(-1, -1)	(-1, -4)	(0, 0)	(0, 0)

✖ Ligne 1 : résultats identiques.

Les deux pré-activations (3, 1) sont **positives**, la ReLU les laisse passer telles quelles : les deux réseaux calculent (3, 10). On vérifie d'ailleurs $(\mathbf{W}_2\mathbf{W}_1)\mathbf{x} = \begin{pmatrix} 1 & 2 \\ 3 & 7 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = (3, 10)$: une seule matrice suffit.

✔ Ligne 2 : les réseaux divergent.

Cette fois $\mathbf{W}_1\mathbf{x} = (-1, -1)$ est **négatif**. Sans activation on obtient (-1, -4) ; avec ReLU, $\mathbf{h} = (0, 0)$ et la sortie est (0, 0). **Aucune matrice unique** ne peut reproduire les deux lignes à la fois.



L'idée à emporter

Un réseau ReLU est une régression linéaire... **par morceaux**. Sur la région où toutes les pré-activations sont positives, il coïncide *exactement* avec un modèle linéaire. C'est le **passage par zéro** qui crée un coude — et donc la flexibilité.

ReLU : la fonction, et surtout sa dérivée

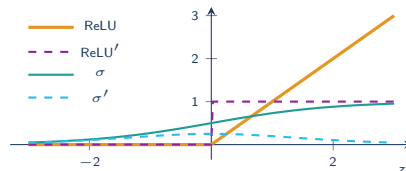
ReLU = *Rectified Linear Unit* : $\text{ReLU}(z) = \max(0, z)$, c'est-à-dire z **tronqué en zéro**.

🔑 On introduit la dérivation φ' ?

$\varphi'(z)$ est la **pen**te de l'activation au point z : de combien la sortie $a = \varphi(z)$ bouge quand la pré-activation z bouge d'un epsilon. Ce n'est pas un détail technique : c'est *exactement* le facteur $\varphi'(z)$ qui apparaîtra dans la formule de rétropropagation (cf. slide 45).

$$\text{ReLU}'(z) = \begin{cases} 0 & \text{si } z < 0 \\ 1 & \text{si } z > 0 \end{cases} \quad \sigma'(z) = \sigma(z)(1 - \sigma(z))$$

⚠ En $z = 0$ la ReLU n'est **pas dérivable** (angle) ; par convention les bibliothèques renvoient 0. Sans conséquence pratique : la probabilité de tomber exactement sur $z = 0$ est nulle.

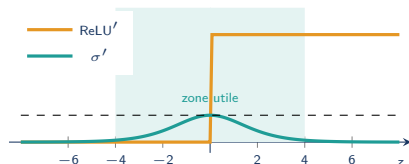


z	ReLU	ReLU'	σ	σ'
-2,0	0	0	0,119	0,105
-0,5	0	0	0,378	0,235
0,0	0	(conv. 0)	0,500	0,250
0,5	0,5	1	0,622	0,235
2,0	2,0	1	0,881	0,105

σ' ne dépasse jamais 0,25 — retenez ce chiffre, il fait l'objet de la slide suivante.

Une pente de 1 : pourquoi la ReLU a détrôné la sigmoïde

Revenons à $\varphi'(z)$, la **pente** de l'activation : elle dit de combien la sortie du neurone **réagit** quand on touche à ses poids. Une **pente proche de zéro**, c'est un neurone sur lequel on n'a plus de prise : on modifie w et b , la sortie ne bouge plus — il **sature**.



$ z $	0	2	4	6
$\sigma'(z)$	0,250	0,105	0,018	0,002
$\text{ReLU}'(z), z > 0$	1	1	1	1

⚠ La sigmoïde a deux défauts.

- **Même au mieux, elle atténuée** : σ' ne dépasse **jamais** 0,25. Le neurone ne transmet **qu'un quart** de ce qu'on lui envoie.
- **Sa zone utile est étroite** : dès $|z| > 4$ la pente tombe sous 0,02, le neurone est **gelé** — et rien ne garantit que z reste dans $[-4, 4]$.

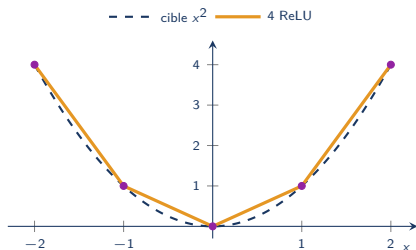
✓ La ReLU n'a ni l'un ni l'autre. Du côté actif $\text{ReLU}'(z) = 1$ **exactement**, quelle que soit la valeur de z : le signal passe **intact**. Du côté inactif la pente vaut 0, le neurone est **éteint** — mais il se rallume dès que z redevient positif.

🔑 À suivre

Sur **un** neurone, l'écart entre 0,25 et 1 paraît anodin. Il devient décisif dès qu'on **empile** des couches, car ces pentes se **multiplient**. Nous le quantifierons une fois la rétropropagation en main (slide 46).

Quatre ReLU suffisent à plier une parabole

La ReLU est linéaire par morceaux : **chaque unité ajoute un coude**, et un seul. En additionner plusieurs permet donc de dessiner n'importe quelle ligne brisée.



$$g(x) = 2 \operatorname{ReLU}(-x - 1) + \operatorname{ReLU}(-x) \\ + \operatorname{ReLU}(x) + 2 \operatorname{ReLU}(x - 1)$$

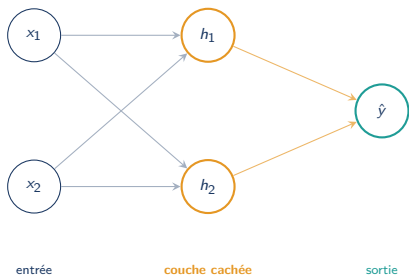
C'est un réseau à **une** couche cachée (cf. slide 32) de **quatre** neurones, poids de sortie (2, 1, 1, 2). Il coïncide avec x^2 aux cinq points **marqués** et interpole linéairement entre eux.

💡 Le lien avec le théorème de Cybenko (1989)

Avec m unités on obtient m coudes, donc une ligne brisée à m segments. Toute fonction continue sur un compact étant approchable d'aussi près qu'on veut par une ligne brisée, on tient là une **version constructive** du théorème d'approximation universelle (cf. slide 35) — et la raison pour laquelle “assez de neurones” peut vouloir dire *beaucoup*.

D'un neurone à une couche : la notion de couche cachée

Un seul neurone produit **un** nombre. Pour produire un **vecteur**, on en place plusieurs **en parallèle** sur la même entrée : c'est une **couche**.



Les h_k ne sont **pas observés** : ce sont des **variables latentes**, des représentations intermédiaires.

≡ **Une couche, en notation matricielle.** m neurones sur une entrée $x \in \mathbb{R}^d$: on **empile** leurs poids dans une matrice $W \in \mathbb{R}^{m \times d}$ et leurs biais dans $b \in \mathbb{R}^m$,

$$h = \varphi(Wx + b) \in \mathbb{R}^m.$$

📊 **L'exemple ci-contre** ($d = 2$, $m = 2$) se lit :

$$h_1 = \varphi(w_{11}x_1 + w_{12}x_2 + b_1),$$

$$h_2 = \varphi(w_{21}x_1 + w_{22}x_2 + b_2),$$

$$\hat{y} = \varphi_{\text{out}}(v_1h_1 + v_2h_2 + c).$$

Vocabulaire

Couche d'entrée (les x) → une ou plusieurs **couches cachées** (les h) → **couche de sortie** (le $\hat{y}$). Empiler les couches cachées = aller "en profondeur".

Une couche cachée au calcul : le même exemple, chiffré

On reprend les **mêmes** clients qu'en slide 26 et on intercale **deux neurones cachés** à activation ReLU, suivis d'un neurone de sortie sigmoïde.

$$\mathbf{W} = \begin{pmatrix} 1,5 & -0,8 \\ -1,0 & 1,2 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} -0,4 \\ 0,3 \end{pmatrix}, \quad \mathbf{v} = (2,0 ; -1,5), \quad c = -1,0.$$

	z_1	h_1	z_2	h_2	$\sigma(\mathbf{v}^\top \mathbf{h} + c)$
Client A (1,0 ; 0,5)	+0,70	0,70	-0,10	0	0,599
Client B (-0,5 ; 1,5)	-2,35	0	+2,60	2,60	0,007

📅 Détail du client B

$$z_1 = 1,5(-0,5) - 0,8(1,5) - 0,4 = -2,35 \Rightarrow h_1 = \text{ReLU}(-2,35) = 0$$

$$z_2 = -1,0(-0,5) + 1,2(1,5) + 0,3 = +2,60 \Rightarrow h_2 = 2,60$$

$$\hat{y} = \sigma(2,0(0) - 1,5(2,60) - 1,0) = \sigma(-4,90) = 0,007$$

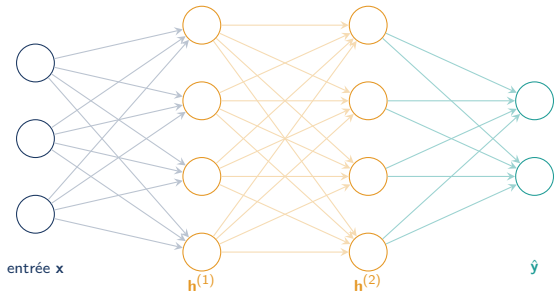
💡 Ce que la couche a fabriqué

Chaque client **n'allume qu'un seul** neurone caché : h_1 pour A, h_2 pour B. La couche a créé **deux profils de risque** que personne n'avait spécifiés — c'est cela, *apprendre une représentation*.

🔹 Parcimonie

La ReLU met à **zéro exactement** les neurones dont la pré-activation est négative. Dans un vrai réseau, la moitié des activations sont nulles : c'est peu coûteux à calculer, et cela rappelle le **lasso**.

Le perceptron multicouche (MLP)



Couches **cachées** = **représentations apprises** (*feature learning*).

 **Propagation avant** (notation matricielle). On pose $\mathbf{h}^{(0)} = \mathbf{x}$, puis pour chaque couche $l = 1, \dots, L$:

$$\mathbf{h}^{(l)} = \varphi(\mathbf{W}^{(l)} \mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}).$$

La sortie $\hat{\mathbf{y}} = \mathbf{h}^{(L)}$ utilise l'identité (régression) ou softmax (classification).

 **Paramètres.** Une couche $\mathbb{R}^{m_{l-1}} \rightarrow \mathbb{R}^{m_l}$ coûte $m_l m_{l-1}$ poids et m_l biais (avec m le nombre de neurones dans la couche l).

Un réseau profond en compte des millions à des milliards $\Rightarrow$ d'où la nécessité d'un apprentissage **efficace** (cf. slide 37).

Pourquoi “perceptron” ?

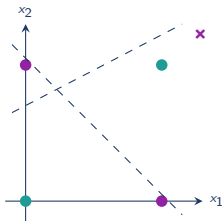
Le **perceptron** de Rosenblatt (1958) est un neurone **unique** (sans couche cachée) à activation **seuil** $\mathbb{1}(z > 0)$. Minsky & Papert (1969) montrent qu'il échoue sur XOR : c'est le premier hiver de l'IA. Le **MLP** est donc surtout un **nom d'héritage**.

XOR (exclusive OR) : pourquoi une couche cachée est nécessaire

✖ Le problème (Minsky & Papert, 1969). On veut apprendre le "ou exclusif" :

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Un neurone seul ne sait tracer qu'une **frontière linéaire** (une droite $w_1x_1 + w_2x_2 + b = 0$). Or **aucune droite** ne met $\{(0, 0), (1, 1)\}$ d'un côté et $\{(0, 1), (1, 0)\}$ de l'autre :



💡 La solution : combiner deux frontières. Avec une couche cachée, chaque neurone caché trace sa droite (h_1, h_2), et la couche de sortie les **combine** : on découpe le plan en régions et XOR devient séparable. Une couche cachée suffit.

Théorème d'approximation universelle

(Cybenko, 1989; Hornik, 1991). Un réseau à **une seule couche cachée**, activation **non polynomiale** (permet de casser / plier l'espace) et **assez de neurones**, approche toute fonction continue sur un compact, à ε près.

XOR en est le toy-case : un cas où 0 couche cachée échoue, 1 suffit.

⚠ Les nuances :

- résultat d'existence, pas d'**apprenabilité** (la descente de gradient trouvera-t-elle ces poids ?)
- "assez de neurones" peut être **exponentiel** en largeur
- en pratique la **profondeur** est plus efficace $\Rightarrow$ deep learning

XOR résolu à la main : les poids qui marchent

La slide précédente affirme qu'une couche cachée suffit. Voici les poids explicites — deux neurones ReLU et une sortie linéaire, tous les coefficients sont entiers :

$$\hat{y} = \varphi_{\text{out}}(v_1 h_1 + v_2 h_2 + c) = \text{ReLU}(x_1 + x_2) - 2 \text{ReLU}(x_1 + x_2 - 1).$$

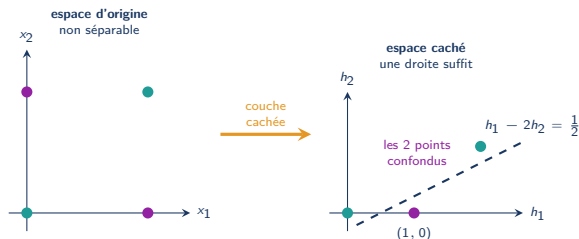
x_1	x_2	h_1	h_2	$\hat{y}$	y
0	0	0	0	0	0
0	1	1	0	1	1
1	0	1	0	1	1
1	1	2	1	0	0

avec $h_1 = \text{ReLU}(x_1 + x_2)$ et $h_2 = \text{ReLU}(x_1 + x_2 - 1)$.



Le mécanisme

h_1 compte les entrées actives, h_2 ne s'allume que si les deux le sont. La sortie soustrait deux fois h_2 : elle annule le cas (1, 1).



La couche cachée **replie** le plan : (0, 1) et (1, 0) sont envoyés sur le même point (1, 0). Dans ce nouvel espace, le problème redevient **linéairement séparable** — et la sortie n'est qu'une régression linéaire. (exemple de Goodfellow et al., 2016, §6.1)

Apprendre = minimiser une fonction de perte

On choisit θ (les poids du réseau) qui minimise la **fonction de perte** moyenne sur les n observations $(\mathbf{x}_i, y_i)$ — aussi appelée *fonction de coût* ou *risque empirique* :

$$\mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^n \ell(y_i, f_{\theta}(\mathbf{x}_i)), \quad \ell = \text{perte sur une observations.}$$

Cible continue — régression

$$\ell = (y_i - \hat{y}_i)^2.$$

La moyenne de ces pertes est l'**EQM** (erreur quadratique moyenne). La minimiser $\Leftrightarrow$ **EMV** (estimation par maximum de vraisemblance) sous bruit **gaussien**. C'est votre **MCO**... avec un f_{θ} non linéaire.

Cible discrète — classification

$$\ell = - \sum_k y_{i,k} \log \hat{y}_{i,k} \quad (\text{entropie croisée}).$$

C'est l'**anti-log-vraisemblance** d'un modèle de Bernoulli / multinomial. C'est votre **logit**... avec un f_{θ} non linéaire.

Note

Vous savez déjà estimer ces modèles. La seule nouveauté est la **forme de f_{θ}** : empilée, profonde, non convexe. Reste à savoir **comment** l'optimiser sans formule fermée.

De la vraisemblance à la perte : le pont, en deux lignes

Vous savez déjà tout faire : il suffit d'écrire la log-vraisemblance, puis de changer de signe.

🏡 Cas gaussien → moindres carrés

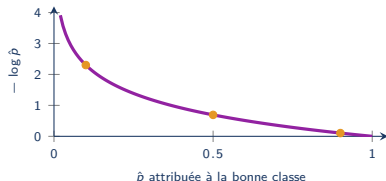
Si $y_i \mid \mathbf{x}_i \sim \mathcal{N}(f_\theta(\mathbf{x}_i), \sigma^2)$:

$$\log L(\theta) = -\frac{n}{2} \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_i (y_i - f_\theta(\mathbf{x}_i))^2.$$

Le premier terme ne dépend pas de θ , donc

$$\max_{\theta} \log L \iff \min_{\theta} \sum_i (y_i - \hat{y}_i)^2.$$

C'est votre EMV ou MCO mais avec f_θ non-linéaire.



🎲 Cas Bernoulli → entropie croisée

Si $y_i \mid \mathbf{x}_i \sim \mathcal{B}(p_i)$ avec $p_i = \sigma(z_i)$:

$$\log L(\theta) = \sum_i \left[y_i \log p_i + (1 - y_i) \log(1 - p_i) \right].$$

$$-\frac{1}{n} \log L = \underbrace{-\frac{1}{n} \sum_i \left[y_i \log p_i + \dots \right]}_{\text{entropie croisée}}$$

C'est votre logit, avec un f_θ profond.

📊 La perte d'un seul individu, chiffrée

Pour $y_i = 1$: $\hat{p} = 0,9 \Rightarrow \ell = 0,105$; $\hat{p} = 0,5 \Rightarrow \ell = 0,693$;
 $\hat{p} = 0,1 \Rightarrow \ell = 2,303$; $\hat{p} = 0,01 \Rightarrow \ell = 4,605$. **Se tromper avec assurance coûte très cher** : la pénalité explose en $-\log$. C'est ce qui pousse le modèle à être *calibré*, pas seulement à trancher.

Le gradient en une slide : de quoi s'agit-il exactement ?

📈 En dimension 1 : la dérivée est une pente.

Si $\mathcal{L}'(\theta) > 0$, la perte **monte** quand θ augmente : il faut donc **diminuer** θ . Si $\mathcal{L}'(\theta) < 0$, c'est l'inverse. Dans les deux cas, le bon geste est

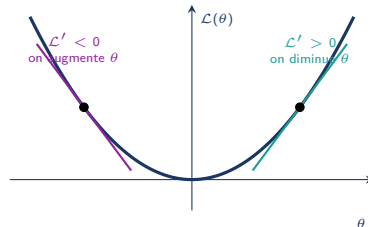
$$\theta \leftarrow \theta - \eta \mathcal{L}'(\theta).$$

Le signe “-” ne contient rien de plus que cette observation.

📈 En dimension p : le gradient.

$$\nabla_{\theta} \mathcal{L} = \left(\frac{\partial \mathcal{L}}{\partial \theta_1}, \dots, \frac{\partial \mathcal{L}}{\partial \theta_p} \right)^{\top}$$

avec P = nombre total de paramètres. C'est le vecteur des dérivées partielles. Il pointe dans la direction où $\mathcal{L}$ **croît le plus vite** ; on part donc dans la direction opposée.



🔑 Le seul point délicat

En régression linéaire, $\nabla \mathcal{L} = 0$ se résout à la main
 $\Rightarrow \hat{\beta} = (X^{\top} X)^{-1} X^{\top} y$. Dès qu'on empile des activations, cette équation n'a plus de solution explicite : on itère.

Descente de gradient : trouver θ sans formule fermée

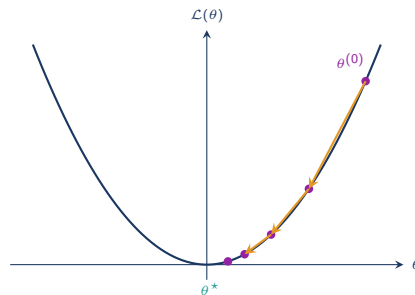
Pas de solution analytique $\Rightarrow$ on descend la pente, itérativement :

$$\theta \leftarrow \theta - \eta \nabla_{\theta} \mathcal{L}(\theta),$$

où η est le **taux d'apprentissage** (*learning rate*).

- ≡ **Batch** (cf. slide 41) : gradient sur *tout* le set (exact, coûteux).
- ⌘ **Mini-batch Stochastic Gradient (SGD)** : gradient sur un **petit échantillon** tiré au hasard $\Rightarrow$ estimateur **bruité** mais **peu coûteux** du vrai gradient.
- ⚡ **Variantes** : Momentum, RMSProp, **Adam** (learning rate adaptatif par paramètre) quasi-standard aujourd'hui.

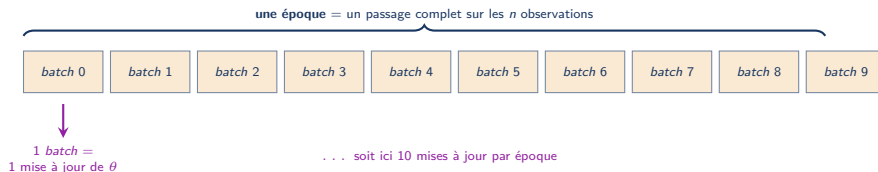
⚠ $\mathcal{L}$ est **non convexe** (minima locaux, points-selle) : on ne vise pas l'optimum global, mais une solution qui *généralise* bien.



Chaque pas suit $-\nabla \mathcal{L}$, pondéré par η .

Époque, *batch*, itération : le vocabulaire du chronomètre

Trois mots que toute la littérature emploie, il faut les définir.



📅 Un cas concret

$n = 60\,000$ observations, *batch* de taille 128 :

$$\frac{60\,000}{128} = 468,75 \Rightarrow 469 \text{ itérations par époque.}$$

Sur 10 époques : 4 690 mises à jour de θ , alors que le MCO n'en demanderait aucune.

🕒 Pourquoi ne pas prendre tout l'échantillon ?

Parce que le gradient sur un *batch* est un **estimateur sans biais** du gradient complet, mille fois moins coûteux. Le bruit d'échantillonnage qui en résulte est même **utile** : il aide à s'échapper des minima locaux.

⚙️ Les hyperparamètres du praticien

η , taille de *batch*, nombre d'époques, nombre de couches, nombre de neurones, λ de régularisation. **Aucun** n'est estimé : tous sont *réglés* sur l'échantillon de validation.

La descente de gradient retrouve votre $\hat{\beta}_{\text{MCO}}$

Le plus petit exemple possible. Régression sans constante $y = \beta x + \varepsilon$ sur $n = 3$ points $(1, 2)$, $(2, 3)$, $(3, 5)$.

D'où viennent 3, 46 et 28 ?

En développant le carré,

$$\mathcal{L}(\beta) = \frac{1}{n} \sum_i (y_i - \beta x_i)^2 = \frac{1}{n} \left(\sum y_i^2 - 2\beta \sum x_i y_i + \beta^2 \sum x_i^2 \right).$$

Ici $\sum y_i^2 = 4+9+25 = 38$, $\sum x_i y_i = 2+6+15 = 23$,
 $\sum x_i^2 = 1+4+9 = 14$, et $n = 3$:

$$\mathcal{L}(\beta) = \frac{38 - 46\beta + 14\beta^2}{3}, \quad \mathcal{L}'(\beta) = \frac{28\beta - 46}{3}.$$

$$3 = n, \quad 46 = 2 \sum x_i y_i, \quad 28 = 2 \sum x_i^2.$$

$$\text{Annuler } \mathcal{L}' \text{ redonne bien } \hat{\beta}_{\text{MCO}} = \frac{\sum x_i y_i}{\sum x_i^2} = \frac{23}{14} = 1,642857.$$

On itère $\beta^{(k+1)} = \beta^{(k)} - \eta \mathcal{L}'(\beta^{(k)})$, avec $\eta = 0,1$ et $\beta^{(0)} = 0$:

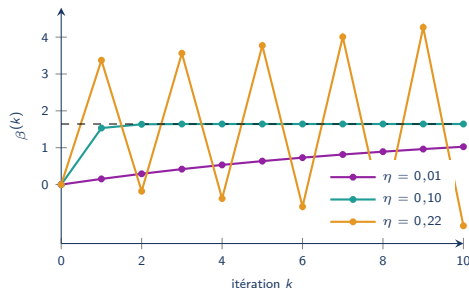
k	$\beta^{(k)}$	$\mathcal{L}'(\beta^{(k)})$	$\mathcal{L}(\beta^{(k)})$
0	0,0000	-15,3333	12,6667
1	1,5333	-1,0222	0,1274
2	1,6356	-0,0681	0,0717
3	1,6424	-0,0045	0,0714
4	1,6428	-0,0003	0,0714
∞	1,642857	0	0,071429

L'exposant (k) compte les itérations ; β_0 désignerait la constante de la régression, absente ici.

Le message

En 4 itérations, **quatre décimales exactes** de l'estimateur MCO. La descente de gradient n'est pas une autre méthode d'estimation : c'est une autre façon de résoudre $\nabla \mathcal{L} = 0$, utile quand la formule fermée n'existe pas.

Le taux d'apprentissage η : trois régimes, un seuil exact



⌚ η trop petit : après 10 pas, 1,03 au lieu de 1,64.

✓ η bien choisi : convergence en 4 pas.

⚠ η trop grand : oscillation **divergente**.

💡 **Lecture.** $\rho > 0$: convergence *monotone* ($\eta = 0,01$ et $0,10$). $\rho < 0$: convergence *en zigzag* si $|\rho| < 1$, divergence sinon ($\eta = 0,22$ donne $\rho = -1,05$). Plus la **courbure** est forte, plus le pas doit être petit. En pratique $\mathcal{L}''$ est inconnue : η est un **hyperparamètre** réglé sur l'échantillon de validation.

√* Calcul du seuil ?

La perte est **quadratique**, donc $\mathcal{L}''$ est une **constante** :

$$\mathcal{L}'' = \frac{2 \sum x_i^2}{n} = \frac{28}{3},$$

et $\mathcal{L}'(\beta) = \mathcal{L}''(\beta - \hat{\beta})$. En retranchant $\hat{\beta}$ des deux côtés de la mise à jour :

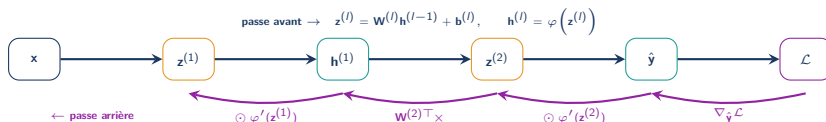
$$\beta^{(k+1)} - \hat{\beta} = (1 - \eta \mathcal{L}'') (\beta^{(k)} - \hat{\beta}).$$

C'est un AR(1) sans bruit, de coefficient $\rho = 1 - \eta \mathcal{L}''$. L'erreur décroît géométriquement ssi $|\rho| < 1 \Leftrightarrow -2 < -\eta \mathcal{L}'' < 0$, soit

$$0 < \eta < \eta_c = \frac{2}{\mathcal{L}''} = \frac{2}{28/3} = \frac{3}{14} \approx 0,214.$$

Rétropropagation : la règle de la chaîne, à grande échelle

La question : obtenir $\nabla_{\theta} \mathcal{L}$ pour des millions de paramètres répartis sur de nombreuses couches. Réponse : la règle de la chaîne en une seule passe arrière. (Écrit pour une observation.)



🔄 Chaque flèche arrière = un facteur.

On pose le signal d'erreur de la couche (l) sur la perte finale :

$$\delta^{(l)} = \partial \mathcal{L} / \partial z^{(l)}$$

Remonter d'un cran, c'est appliquer l'une des deux règles :

(a) **traverser l'activation.** Comme φ agit **composante par composante**, on multiplie terme à terme :

$$\delta^{(l)} = \frac{\partial \mathcal{L}}{\partial h^{(l)}} \odot \varphi'(z^{(l)})$$

(b) **traverser la couche linéaire** $W^{(l+1)}$: $\frac{\partial \mathcal{L}}{\partial h^{(l)}} = W^{(l+1)\top} \delta^{(l+1)}$

En enchaînant (b) puis (a), on obtient la récurrence — amorcée à la dernière couche, où $h^{(L)} = \hat{y}$:

$$\begin{aligned} \delta^{(L)} &= \nabla_{\hat{y}} \mathcal{L} \odot \varphi'(z^{(L)}), \\ \delta^{(l)} &= \left(W^{(l+1)\top} \delta^{(l+1)} \right) \odot \varphi'(z^{(l)}). \end{aligned}$$

📅 Une fois $\delta^{(l)}$ connu, les gradients tombent. Dans $z^{(l)} = W^{(l)}h^{(l-1)} + b^{(l)}$, $\partial z_i^{(l)} / \partial W_{ij}^{(l)} = h_j^{(l-1)}$, donc

$$\begin{aligned} \partial \mathcal{L} / \partial W_{ij}^{(l)} &= \delta_i^{(l)} h_j^{(l-1)} : \\ \frac{\partial \mathcal{L}}{\partial W^{(l)}} &= \delta^{(l)} \left(h^{(l-1)} \right)^\top, \quad \frac{\partial \mathcal{L}}{\partial b^{(l)}} = \delta^{(l)}. \end{aligned}$$

Dimensions : $\delta^{(l)} \in \mathbb{R}^{m_l}$ et $h^{(l-1)} \in \mathbb{R}^{m_{l-1}}$: le produit extérieur fait bien $m_l \times m_{l-1}$, la taille de $W^{(l)}$.

⚠ L'erreur classique

$\partial \mathcal{L} / \partial h^{(l)}$ n'est pas $\nabla_{\hat{y}} \mathcal{L}$, sauf à la dernière couche

Rétropropagation : le calcul complet, à la main

Réseau minimal, tout est scalaire : $z_1 = w_1x + b_1$, $h = \text{ReLU}(z_1)$, $\hat{y} = w_2h + b_2$, $\ell = (y - \hat{y})^2$. Données : $x = 2$, $y = 1$.
Paramètres : $w_1 = 0,5$, $b_1 = -0,2$, $w_2 = 1,5$, $b_2 = 0,1$.

→ 1. Passe avant

$z_1 = 0,5(2) - 0,2$	$= 0,8$
$h = \text{ReLU}(0,8)$	$= 0,8$
$\hat{y} = 1,5(0,8) + 0,1$	$= 1,3$
$\ell = (1 - 1,3)^2$	$= 0,09$

↻ 2. Passe arrière (droite → gauche)

$\delta^{(2)} = -2(y - \hat{y})$	$= 0,6$
$\partial\ell/\partial w_2 = \delta_2 h$	$= 0,48$
$\partial\ell/\partial b_2 = \delta_2$	$= 0,60$
$\delta^{(1)} = \delta^{(2)} w_2 \text{ReLU}'(z_1)$	$= 0,9$
$\partial\ell/\partial w_1 = \delta_1 x$	$= 1,80$
$\partial\ell/\partial b_1 = \delta_1$	$= 0,90$

Pas de fonction d'activation sur la couche de sortie, c'est l'identité donc $\delta^{(2)} = \nabla_{\hat{y}} \mathcal{L}(\theta) \times 1 = \partial\ell/\partial\hat{y}$

✓ 3. Un pas : $\theta \leftarrow \theta - \eta \partial\ell/\partial\theta$, avec $\eta = 0,05$

θ	avant	gradient	après = avant - $0,05 \times \text{grad.}$
w_1	0,500	1,80	$0,500 - 0,090 = 0,410$
b_1	-0,200	0,90	$-0,200 - 0,045 = -0,245$
w_2	1,500	0,48	$1,500 - 0,024 = 1,476$
b_2	0,100	0,60	$0,100 - 0,030 = 0,070$

↻ 4. On refait la passe avant avec les nouveaux paramètres

$$z_1 = 0,410(2) - 0,245 = 0,575, \quad h = 0,575,$$

$$\hat{y} = 1,476(0,575) + 0,070 = 0,919, \quad \ell = (1 - 0,919)^2 = 0,0066.$$



Ce qu'il faut voir

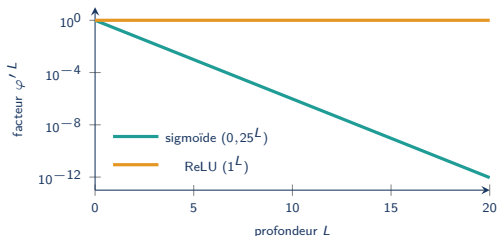
(1) La perte passe de 0,0900 à 0,0066 : un seul pas. (2) Chaque dérivée est un simple **produit** le long du chemin. (3) $\text{ReLU}'(z_1) = 1$ ici ; si $z_1 < 0$ elle vaudrait 0 et w_1, b_1 ne bougeraient pas du tout.

Le gradient qui s'évanouit : la profondeur a un coût

Dans l'exemple précédent, $\text{ReLU}'(z_1)$ valait 1 et n'a rien changé. Que devient ce facteur sur un réseau **profond** ? En déroulant $\delta^{(l)} = \delta^{(l+1)} w^{(l+1)} \varphi'(z^{(l)})$ jusqu'à la première couche :

$$\frac{\partial \mathcal{L}}{\partial w^{(1)}} = h^{(0)} \underbrace{\prod_{l=1}^{L-1} \varphi'(z^{(l)}) w^{(l+1)}}_{\text{une paire de facteurs par couche traversée}} \frac{\partial \mathcal{L}}{\partial \hat{y}}.$$

une paire de facteurs par couche traversée



⚠ Avec la sigmoïde. $\sigma' \leq 0,25$ partout : même avec des poids valant 1, le produit est majoré par $0,25^L$, soit 10^{-6} à 10 couches et 10^{-12} à 20.

📊 Partons de 10^{-6}

Gradient de 0,1 sur la dernière couche $\Rightarrow 10^{-7}$ sur la première ; à $\eta = 0,01$, le poids bouge de 10^{-9} . Or en float32, ajouter 10^{-9} à un nombre voisin de 1 ne change **rien** : la correction est perdue dans l'arrondi.

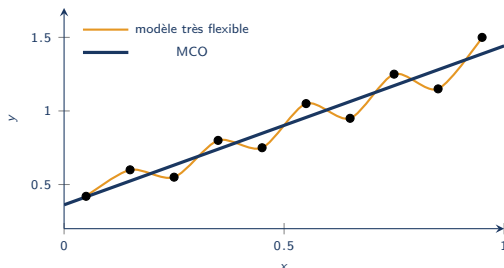
✓ Avec la ReLU, $\varphi' = 1$ du côté actif : l'atténuation **disparaît**, il ne reste que le produit des poids. C'est ce qui a rendu les réseaux *profonds* entraînables.

Deux nuances. (1) Si $|w^{(l)}| < 1$ le produit s'éteint quand même : d'où l'initialisation soignée et les **connexions résiduelles** du bloc Transformer.

(2) *Dying ReLU* : un neurone dont $z < 0$ partout a $\varphi' = 0$ pour toujours (remèdes : Leaky ReLU, GELU).

Le surapprentissage

Dix observations, une vraie relation à peu près linéaire, et deux modèles ajustés dessus.



📊 Le modèle MCO $\hat{y} = 0,362 + 1,08x$ laisse des résidus non nuls. Sa SCR n'est pas minimale parmi tous les modèles possibles.

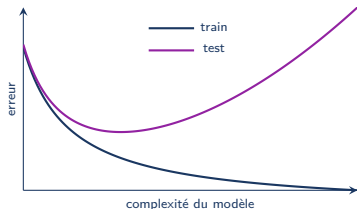
⚠ Le modèle flexible passe par **tous** les points : $SCR = 0$ sur l'échantillon d'estimation. Il est pourtant **pire**, parce qu'il a appris le *bruit* ε_i , qui ne se reproduira pas.

🔑 La leçon

Sur l'échantillon d'estimation, **plus de paramètres** $\Rightarrow$ **toujours moins d'erreur**. Le critère pertinent est donc l'erreur **hors échantillon**, et il faut *brider* volontairement le modèle.

Généraliser : surapprentissage et régularisation

On découpe les données en train / validation / test. Un réseau très flexible peut mémoriser l'échantillon : erreur d'apprentissage faible, mais erreur de test élevée c'est le **surapprentissage**.



Le minimum de la courbe **test** = compromis **biais-variance**.

🛡️ **Régularisation** souvent familière :

🛖 L_2 / **weight decay** : pénalité $\lambda \|\theta\|_2^2$ c'est le **ridge**.

🔴 L_1 : pénalité $\lambda \|\theta\|_1$ c'est le **lasso** (parcimonie).

⚡ **Dropout** : on "éteint" des neurones au hasard à l'entraînement $\approx$ **moyennage de modèles** (ensembling implicite).

🛑 **Early stopping** : on arrête dès que l'erreur de **validation** remonte.

📄 **Augmentation** de données.

Idée clé

Beaucoup d'outils de *deep learning* sont des relectures de la régularisation que l'on pratique en économétrie.

Limites du MLP : le chemin vers les Transformers

⚠ Le MLP ignore la **structure** des données : taille d'entrée fixe, aucune notion d'**ordre** ni de **position**, et un nombre de paramètres qui explose sur du texte ou des images. D'où des architectures spécialisées¹ :

CNN — vision

Poids **partagés** et invariance par translation : on détecte un motif où qu'il soit dans l'image.

RNN / LSTM

Une **mémoire récurrente** traite les mots un à un. Mais le traitement est **séquentiel** (lent) et **oublie** des dépendances longues (gradient qui s'évanouit).

Transformer — 2017

L'**attention** regarde **toute** la séquence en parallèle et relie directement des tokens (blocs d'information) éloignés. Rapide + dépendances longues.

*Tout ce qu'on vient de voir (neurones, activations, softmax, perte, descente de gradient, rétropropagation) reste vrai à l'intérieur d'un Transformer. Il ne nous manque qu'une brique : le **mécanisme d'attention**.*

1. CNN : Convolutional Neural Network (réseau convolutif) — RNN : Recurrent Neural Network (réseau récurrent) — LSTM : Long Short-Term Memory (RNN à mémoire).

Étape 1 — Tokeniser : découper le texte en unités

Un modèle ne manipule pas des lettres ni des mots, mais des **tokens** : des **morceaux de mots** choisis une fois pour toutes, et numérotés.

Comment le vocabulaire est construit (BPE)

Byte-Pair Encoding : on part des caractères, puis on **fusionne itérativement** la paire la plus fréquente du corpus, autant de fois qu'on veut de tokens.

départ	recherche (9 unités)
fusion 1	recherche
fusion 2	recherche
...	...
arrivée	re cher che (3 tokens)

On s'arrête quand le vocabulaire atteint la taille voulue :
 $|V| \approx 30\,000$ à $200\,000$ tokens.



Pourquoi des sous-mots ?

Un token = un mot serait ingérable : le vocabulaire serait infini (noms propres, néologismes, fautes de frappe) et tout mot inconnu deviendrait un trou. **Un token = une lettre** rendrait les séquences dix fois plus longues, donc le coût d'attention cent fois plus élevé. Les **sous-mots** sont le compromis : *économétrie* peut s'écrire écono | métrie même si le modèle ne l'a jamais vu entier.

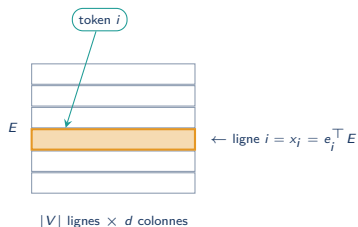


Conséquence pratique. La facturation des API se fait au token, pas au mot. En français, comptez environ **1,5 à 2 tokens par mot** — nettement plus qu'en anglais, car les corpus d'entraînement sont majoritairement anglophones. Un même texte coûte donc plus cher en français.

Étape 2 — L'*embedding* : donner un vecteur à chaque token

Un **embedding** ("plongement") est le **vecteur de nombres** associé à un token. Sa longueur d — le **nombre de coordonnées**, typiquement 768 à 8192 — est la même pour tous.

La table E : un simple annuaire



Récupérer un **embedding**, c'est lire une ligne de E . Une séquence de T tokens devient une matrice $X \in \mathbb{R}^{T \times d}$: l'entrée du Transformer.

⚠ E n'est **pas** donnée : ses $|V| \times d$ nombres sont des **paramètres appris**.

Q Pourquoi écrire $x_i = e_i^T E$?

$e_i = (0, \dots, 1, \dots, 0)$ est le vecteur **one-hot** du token : $|V|$ zéros et **un seul 1**, en position i . Le multiplier par E ne conserve que la ligne dont le coefficient n'est pas nul : *le résultat est le même que "lire la ligne i ".* On l'écrit en produit parce que cela en fait une **opération dérivable** : la rétropropagation peut alors corriger E comme n'importe quelle matrice de poids.

Ce que ces vecteurs encodent

Aucune coordonnée n'a de sens isolément : c'est la **position relative** des vecteurs qui porte le sens. En dimension 2, avec $u_{\text{crédit}} = (3, 1)$, $v_{\text{prêt}} = (2, 1)$, $w_{\text{chien}} = (-1, 3)$:

$$\cos(u, v) = \frac{u \cdot v}{\|u\| \|v\|} = \frac{3 \times 2 + 1}{\sqrt{3^2 + 1} \sqrt{2^2 + 1}} = 0,99$$

contre $\cos(u, w) = 0,00$. C'est la **cosine similarity**, que nous retrouverons dans le RAG.

Encoder la position : l'ordre des mots compte

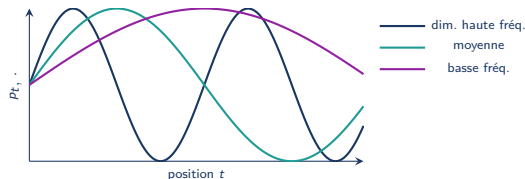
⚠ Le problème. Le mécanisme d'attention que nous allons construire compare les tokens **deux à deux** sans tenir compte de leur place : il traite la phrase comme un **ensemble**, pas comme une suite ordonnée.

Sans repère de position, “le chat mange la souris” et “la souris mange le chat” deviendraient indiscernables.

🔧 La solution : ajouter un vecteur de position. À chaque embedding x_t on **ajoute** un vecteur p_t qui code la position t . Vaswani *et al.* (2017) le construisent avec des sinusoides :

$$p_{t,2i} = \sin\left(\frac{t}{10000^{2i/d}}\right), \quad p_{t,2i+1} = \cos\left(\frac{t}{10000^{2i/d}}\right).$$

Chaque dimension oscille à une **fréquence différente** ; le modèle peut ainsi situer chaque token. Le vecteur $x_t + p_t$ alimente alors la suite du réseau.



i Variantes (abordées plus loin) : on peut aussi **apprendre** les p_t , ou encoder la position **relative** directement dans le calcul d'attention (*RoPE*, Su *et al.*, 2021).

Attention (1/3) : l'intuition Query / Key / Value

Fil rouge de toute la séquence : l'ambiguïté de “mine”.

Sa bonne *mine* réjouit. vs La vieille *mine* ferme. Le même token doit recevoir deux représentations différentes.

token	Query q : ce que je cherche	Key k : ce que j'annonce	Value v : ce que je transmets
mine	"suis-je un visage ou un gisement?"	"nom, ambigu"	les deux sens, non tranchés
vieille	"quel nom vais-je qualifier : concret ou abstrait?"	"adjectif d'ancienneté, porte sur un objet"	ancien, inanimé, durable
bonne	"de quoi j'apprécie la qualité : un état, une apparence?"	"adjectif d'appréciation, porte sur un état"	positif, animé, apparence

🔑 Le mécanisme.

La Query de *mine* ressemble à la Key de *vieille* (produit scalaire élevé) : *mine* absorbe alors la Value de *vieille*, “ancien, inanimé”, et bascule vers le sens *gisement*. Dans l'autre phrase, c'est la Key de *bonne* qui gagne, et *mine* devient un *visage*.

🔄 Tout token a une Query. L'attention est symétrique en structure : *ferme* interroge aussi ses voisins pour savoir s'il est verbe ou nom.

📊 Trois projections linéaires apprises. Pour chaque token

$x_t + p_t$:

$$(x_t + p_t)W_q = q_t, \quad (x_t + p_t)W_k = k_t, \quad (x_t + p_t)W_v = v_t,$$

et en empilant les T tokens : $Q, K \in \mathbb{R}^{T \times d_k}, V \in \mathbb{R}^{T \times d_v}$.

⚠ Les libellés du tableau sont une lecture a posteriori. Personne ne les fournit au modèle : W_q, W_k, W_v sont estimés par rétropropagation, comme tous les autres poids.

Attention (2/3) : scores, mise à l'échelle, softmax

“ Phrase de référence : (la) *vieille* *mine* *ferme* — $T = 3$ tokens pleins, d'où des matrices 3×3 .

⚙️ 1. **Scores de similarité.** Produit scalaire entre chaque **Query** et chaque **Key** : (mine est en position 2 donc focus sur la ligne 2)

$$QK^T = \begin{pmatrix} q_1 \cdot k_1 & q_1 \cdot k_2 & q_1 \cdot k_3 \\ q_2 \cdot k_1 & q_2 \cdot k_2 & q_2 \cdot k_3 \\ q_3 \cdot k_1 & q_3 \cdot k_2 & q_3 \cdot k_3 \end{pmatrix}.$$

🏠 2. **Mise à l'échelle** $1/\sqrt{d_k}$. Si les composantes sont i.i.d. de variance 1, alors $\mathbb{V}(q \cdot k) = d_k$. Sans normalisation, les scores explosent, le softmax **sature** et les gradients $\rightarrow 0$.

ligne de mine ($t=2$)	<i>vieille</i>	<i>mine</i>	<i>ferme</i>
score / $\sqrt{d_k}$	2,4	1,0	0,6
softmax	0,708	0,175	0,117
sans division ($d_k=64$)	19,2	8,0	4,8
softmax	1,000	0,000	0,000

📊 3. **Softmax par ligne.** Chaque ligne devient une **distribution de probabilité** (poids ≥ 0 , somme = 1) — c'est le **logit multinomial** de la slide “activations” :

$$S = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) = \begin{pmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{32} & s_{33} \end{pmatrix}.$$

Les lignes de **S** sont les **poids d'attention**. Étapes 1–3 : **encodeur et décodeur**.

🔻 Variante décodeur : le masque causal

On interdit de “voir le futur” en ajoutant M ($-\infty$ au-dessus de la diagonale) **avant** le softmax :

$$M = \begin{pmatrix} 0 & -\infty & -\infty \\ 0 & 0 & -\infty \\ 0 & 0 & 0 \end{pmatrix} \Rightarrow = \begin{pmatrix} 1 & 0 & 0 \\ s_{21} & s_{22} & 0 \\ s_{31} & s_{32} & s_{33} \end{pmatrix}.$$

⚠️ Pourquoi diviser : le contre-exemple

Sans la division, les mêmes scores sont multipliés par $\sqrt{64} = 8$. Le softmax devient un **argmax** : *mine* n'écoute plus que *vieille*, à 99,999%. Or un poids de 1 ou de 0 a une **dérivée nulle** : plus aucun gradient ne remonte, et W_q, W_k cessent d'être appris.

Attention (3/3) : la représentation contextualisée

💬 **Somme pondérée des Values.** La sortie combine les **Value** selon les poids d'attention (ici le cas **encodeur**, sans masque) :

$$H = S V, \quad S = \begin{pmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{32} & s_{33} \end{pmatrix}.$$

La ligne t de S somme à 1 : la sortie du token t est une **moyenne pondérée** des Values. Pour **mine** ($t = 2$), avec les poids calculés à la slide précédente :

$$h_2 = 0,708 v_{\text{vieille}} + 0,175 v_{\text{mine}} + 0,117 v_{\text{ferme}}.$$

Formule compacte

$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + M\right) V$, avec $M = 0$ en encodeur.

💡 **Retour à “mine”.** Les poids **diffèrent** selon le contexte, ce qui **désambiguïse** le token :



Le token “mine” concentre son attention sur *bonne* (visage) ou *vieille* (gisement) : sa **nouvelle représentation** encode le bon sens.

⚠️ **En décodeur**, *mine* ne verrait plus *ferme* : le softmax se **renormalise** sur les alternatives restantes,
 $h_2 = 0,802 v_{\text{vieille}} + 0,198 v_{\text{mine}}.$

Masquer, c'est restreindre l'ensemble de choix

Le softmax d'attention est un logit multinomial : les poids de la ligne t sont des probabilités de choix sur l'ensemble des tokens visibles.

📅 Ligne de *mine*, sans masque ($T = 3$) :

$$s_{2j} = \frac{e^{z_j}}{e^{2,4} + e^{1,0} + e^{0,6}}, \quad z = (2,4; 1,0; 0,6), \\ \Rightarrow (0,708; 0,175; 0,117).$$

⚡ Avec masque causal, *ferme* sort de l'ensemble de choix :

$$s_{2j} = \frac{e^{z_j}}{e^{2,4} + e^{1,0}} \Rightarrow (0,802; 0,198; 0).$$

⚖️ Le rapport de cotes est invariant

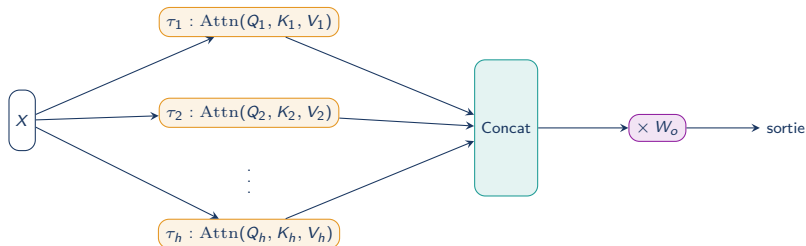
$$\frac{0,708}{0,175} = \frac{0,802}{0,198} = e^{2,4-1,0} = e^{1,4} = 4,06.$$

Retirer une alternative ne modifie pas le rapport entre les alternatives restantes : c'est l'hypothèse **IIA** du logit multinomial (McFadden, 1974) que vous connaissez. Le masque n'est rien d'autre qu'une **restriction de l'ensemble de choix**.

⚠️ Attention : le masque **causal** est propre au décodeur, mais un masque de **padding** (neutraliser les tokens de remplissage d'un batch) existe aussi en encodeur. "Masque" $\neq$ "décodeur".

Attention multi-têtes : plusieurs regards en parallèle

Une seule attention ne capte qu'un type de relation. On en exécute h en **parallèle** — les **têtes** (τ) — chacune dans son **sous-espace**, puis on recombine.



≡ **Mécanique.** Chaque tête $i = 1, \dots, H$ a ses **propres** projections, de dimension réduite $d_k = d/H$:

$$\tau_i = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_k}} + M\right) V_i \in \mathbb{R}^{T \times d_k},$$

$$\text{MultiHead}(X) = \left[\tau_1; \dots; \tau_H \right] W_o.$$

Les H sorties sont **concaténées** ($H \times d_k = d$), puis mélangées par $W_o \in \mathbb{R}^{d \times d}$. Coût quasi inchangé : H têtes de taille $d/H \approx$ une grande tête.

🔑 Intuition

Une tête peut suivre les **adjectifs** (*bonne, vieille*), une autre les **verbes**, une autre les **positions**... Chaque tête éclaire une facette du contexte ; W_o fond le tout en une seule représentation.

L'attention ne fait que des moyennes, il nous manque une étape

⚠ La limite de l'attention seule. La sortie du token t est

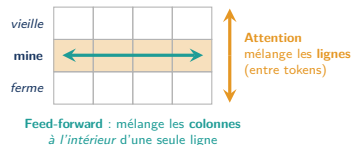
$$h_t = \sum_j s_{tj} v_j, \quad s_{tj} \geq 0, \quad \sum_j s_{tj} = 1.$$

Des poids positifs de somme 1 : h_t est une **combinaison convexe** des **Values**. Elle reste donc **dans l'enveloppe convexe** des v_j — l'attention **brasse** l'information existante, elle n'en **crée** aucune.

📊 **Traduction statistique.** C'est un **lisseur à noyau** (Nadaraya–Watson) dont les poids seraient *appris* au lieu d'être fixés par une fenêtre. Et empiler des lissages de projections linéaires reste **linéaire** en X .

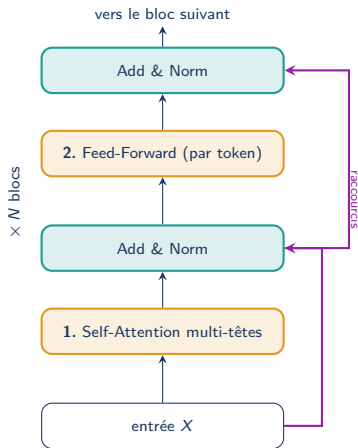
➔ Il faut donc, après le brassage, une **non-linéarité** appliquée à chaque token : c'est le **feed-forward**.

🏠 **Deux axes, deux opérations.** La matrice des représentations est $T \times d$: **une ligne par token, une colonne par dimension**.



Aucune information ne circule entre tokens dans le feed-forward. Seule l'attention déplace de l'information ; le feed-forward la **transforme** sur place.

Le bloc Transformer (1/2) : deux sous-couches et un raccourci



Deux sous-couches.

1. l'**attention** : les tokens **échantent** de l'information ;
2. le **feed-forward** : chaque token est **retravaillé** seul.

🔗 Le **raccourci résiduel**. On ne **remplace** pas l'entrée : on lui **ajoute** le résultat.

$$\text{sortie} = x + \underbrace{\text{SousCouche}(x)}_{\text{correction apprise}}.$$

En déroulant les N blocs : $h_N = x + \sum_{\ell=1}^N f_{\ell}(h_{\ell-1})$.

🌱 Un modèle additif, comme le boosting

Chaque bloc n'apprend pas la représentation, mais une **correction** de la représentation courante — la forme additive du *gradient boosting*. (Nuance : les f_{ℓ} sont estimés **conjointement**, pas séquentiellement.)

⚡ Et le gradient ?

$\partial(x + f(x))/\partial x = I + \partial f/\partial x$. Le terme I garantit un **chemin de dérivée 1** : plus de produit $0,25^L$.

Le bloc Transformer (2/2) : normaliser et transformer

 LN = centrer-réduire *une ligne*. Pour le vecteur $h = (h_1, \dots, h_d)$ d'un token, moyenne μ et écart-type s sur ses d composantes :

$$\hat{h}_k = \gamma_k \frac{h_k - \mu}{\sqrt{s^2 + \epsilon}} + b_k.$$

γ re-dilate, b re-décale : deux vecteurs appris qui rendent les 2 degrés de liberté que la standardisation a retirés.

Ligne ou colonne ?

Dans une matrice ($n \times p$), vous standardisez une **colonne** (une variable sur les individus) : c'est la *BatchNorm*. Ici on standardise une **ligne** (les d dimensions d'un token) — donc **indépendamment du batch** : valide à l'inférence sur une seule phrase, insensible au padding et au masque.

 Feed-Forward = un petit MLP, par token. Le même réseau, appliqué **indépendamment** à chaque token, avec une couche cachée **élargie** ($d_{ff} \approx 4d$) :

$$\text{FFN}(x) = \underbrace{\max(0, xW_1 + b_1)}_{\text{ReLU}} W_2 + b_2.$$

Après que l'attention a **rassemblé** le contexte, le feed-forward laisse chaque token **digérer** cette information de son côté.

Où sont les paramètres ?

Attention $\Rightarrow 4d^2$ (W_q, W_k, W_v, W_o) ; feed-forward $\Rightarrow 8d^2$. Les **deux tiers d'un bloc sont dans le feed-forward** ($d=512$: 1,0M contre 2,1M) : c'est là que loge la **connaissance paramétrique** ($\rightarrow$ RAG).

À retenir

Attention : les tokens *se parlent*. **Feed-Forward** : chaque token *réfléchit* seul. **Résiduel + LayerNorm** : les *stabilisateurs* de l'empilement.

Où placer la normalisation ? 2017 vs. aujourd'hui

🕒 **Post-LN** — Vaswani *et al.* (2017), le schéma précédent :

$$x \mapsto \text{LN}(x + f(x)).$$

Le raccourci **traverse** la normalisation : le chemin identité est **interrompu** à chaque bloc. En profondeur, l'entraînement devient instable et exige un *warmup* du taux d'apprentissage.

✅ **Pre-LN** — GPT-2 et tous les LLM modernes :

$$x \mapsto x + f(\text{LN}(x)).$$

On normalise **l'entrée** de la sous-couche, pas la somme. Le **flux résiduel** devient une **autoroute ininterrompue** de x jusqu'à la sortie.

≡ L'image du flux résiduel

Voyez x comme un **bus de données** de largeur d constante d'un bout à l'autre du réseau. Chaque sous-couche y **lit**, calcule, et y **réécrit un incrément**. C'est pourquoi d ne change jamais, alors que $d_k = d/H$ et $d_{\text{ff}} = 4d$ sont **internes** aux sous-couches.

👉 **RMSNorm** (Zhang & Sennrich, 2019 ; LLaMA, Mistral) :

$$\hat{h}_k = \gamma_k \frac{h_k}{\sqrt{\frac{1}{d} \sum_i h_i^2 + \epsilon}}.$$

On **abandonne le centrage** et le biais : on ne réduit plus, on **remet à l'échelle**. Moins d'opérations, performance équivalente.

⚠️ À dire aux étudiants

Le schéma canonique du papier de 2017 n'est **pas** celui des modèles qu'ils utilisent. Un LLM de 2026 est un empilement de blocs **Pre-RMSNorm**.

Le fil “mine” : ce que chaque brique lui fait

Brique	Ce qu'elle fait à mine	Traduction statistique
Embedding	lui donne <i>un</i> vecteur, identique dans les deux phrases	lecture d'une ligne de E
Position	distingue <i>la vieille mine</i> de <i>mine la vieille</i>	ajout de p_t
Attention	lui fait absorber $0,708 v_{vieille} + 0,175 v_{mine} + 0,117 v_{ferme}$	moyenne pondérée, poids issus d'un softmax
Multi-têtes	une tête suit l'adjectif, une autre le verbe, une autre la position	H régressions en parallèle, puis agrégation par W_o
Feed-forward	retravaille le vecteur obtenu, <i>sans</i> regarder les voisins	un petit MLP appliqué token par token
Résiduel + Norm	garde une trace du <i>mine</i> d'origine et stabilise l'échelle	$x + f(x)$, puis centrer-réduire
$\times N$ blocs	répète l'opération : le sens se précise couche après couche	composition de N transformations apprises

🔑 À l'arrivée

Après N blocs, le vecteur de **mine** n'est plus celui de la table E : c'est une **représentation contextualisée**, différente dans chacune des deux phrases. C'est elle qui alimentera le logit multinomial final — dernière slide de la séquence.

Trois familles : encodeur, décodeur, encodeur-décodeur

Tout repose sur **qui peut regarder qui** — c'est-à-dire le **masque** d'attention.

bidirectionnel



causal



ligne = requête, colonne = clé

Encodeur : chaque token voit **tout** (gauche + droite). **Décodeur** : il ne voit que le **passé** ⇒ génération.

☰ Encodeur seul — *comprendre*

Attention bidirectionnelle. Produit des représentations riches. **Ex. : BERT** (*Devlin et al., 2019*). Tâches : classification, Named Entity Recognition (NER), *embeddings*.

✍ Décodeur seul — *générer*

Masque causal : prédiction du token suivant. **Ex. : GPT** (*Brown et al., 2020*). C'est l'architecture des **LLM** actuels.

↔ Encodeur-décodeur — *transformer*

L'encodeur lit l'entrée ; le décodeur génère en consultant l'encodeur via une **attention croisée** (ses **Q** interrogent les **K,V** de l'encodeur). **Ex. : T5** (*Raffel et al., 2020*). Tâches : traduction, résumé.

Ce que fait vraiment un LLM : un logit multinomial géant

À la sortie du dernier bloc, le token **mine** possède un vecteur contextualisé $h_t \in \mathbb{R}^d$. Une **ultime** couche linéaire $W_U \in \mathbb{R}^{d \times |V|}$ le projette sur le vocabulaire, et un softmax en fait des probabilités :

$$\mathbb{P}(\text{token}_{t+1} = k \mid \text{contexte}) = \text{softmax}\left(h_t W_U\right)_k, \quad k = 1, \dots, |V|.$$

Traduction en économétrie

- variable dépendante : le token suivant, **qualitative** à $|V| \approx 100\,000$ **modalités**
- régresseurs : h_t , représentation *apprise* du contexte
- estimation : **maximum de vraisemblance** sur des milliers de milliards de tokens = minimisation de l'**entropie croisée**
- critère d'ajustement : la **perplexité** = $\exp(\text{entropie croisée})$

Le bouclage

Un LLM est un **logit multinomial** dont les régresseurs sont fabriqués par 80 couches d'attention. Neurone, activation, perte, gradient, rétropropagation : c'est *exactement* ce qui l'entraîne.

L'ordre de grandeur

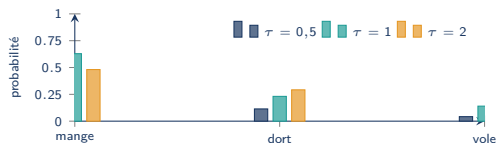
Avec $d = 8192$ et $|V| = 128\,000$, la *seule* matrice W_U compte $8192 \times 128\,000 \approx 1,05$ **milliard** de paramètres — pour un unique logit multinomial.

Une différence de nature avec votre logit

En économétrie, on estime un logit multinomial pour **interpréter** ses coefficients (effets marginaux). Ici, les coefficients de W_U ne s'interprètent pas un à un : ils ne valent que par la *représentation* h_t qui les précède. Le modèle est estimé pour **prédire**, pas pour expliquer — c'est le déplacement d'objectif qui sépare l'économétrie du *machine learning*.

Générer : échantillonnage autorégressif et température

Le modèle ne produit pas une phrase : il produit **une distribution**, on y **tire** un token, on le **réinjecte** et l'on recommence (**génération autorégressive**). La **température** règle la façon de tirer : $\mathbb{P}_\tau(k) = \text{softmax}(z/\tau)_k$.



τ	mange	dort	vole	entropie
0,5	0,844	0,114	0,042	0,52
1	0,629	0,231	0,140	0,91
2	0,481	0,292	0,227	1,05

Logits $z = (2,0; 1,0; 0,5)$; entropie $H = -\sum_k p_k \ln p_k$, maximale à $\ln 3 = 1,10$.

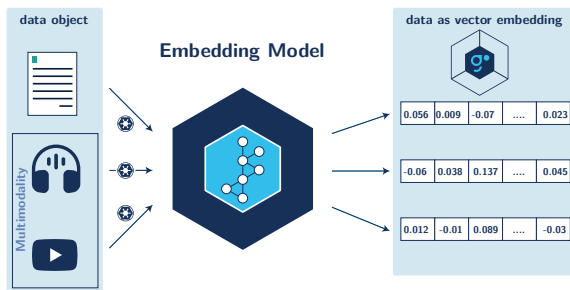
- ▶ $\tau \rightarrow 0$: tout va sur le mode, on retombe sur l'**argmax** (décodage *glouton*) — répétitif.
- ▶ $\tau = 1$: on échantillonne la **distribution du modèle**, celle estimée par EMV.
- ▶ $\tau > 1$: on **aplatit**, donc on diversifie — au risque de l'incohérence.

🔑 Trois points mal compris

(1) T ne change **jamais** le classement des tokens, seulement la **concentration**. (2) On combine τ à un **troncage** : *top-k* ou *top-p*, qui coupent la queue de distribution. (3) Le glouton maximise la probabilité *du prochain token*, pas celle de *la phrase* : d'où le *beam search* (ne conservant qu'un nombre limité de candidats).

⚠️ $\tau = 0$ ne garantit pas le déterminisme : voir "le mythe de la température zéro".

Le Sens des Mots : Tokens, Embeddings et Transformers



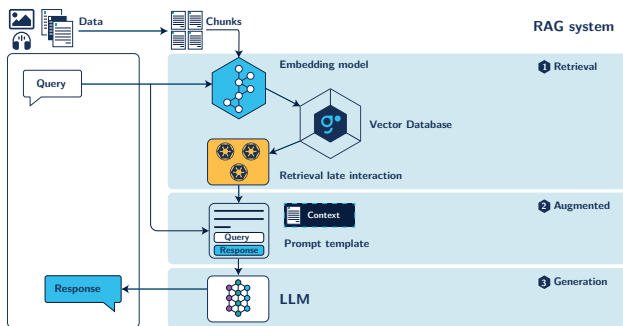
⚙️ **Tokenisation** : Le texte est d'abord découpé en unités de base appelées **tokens** (mots, sous-mots ou caractères). Cette étape transforme le langage en une séquence que la machine peut traiter

📊 **Embeddings** : Chaque token est ensuite converti en un **vecteur numérique** (embedding) qui encode son **sens sémantique**. À ce stade, ce sens est isolé : le token est représenté hors contexte, indépendamment du reste de la phrase

🔗 **Transformers** : C'est l'architecture du modèle. Grâce à son mécanisme d'**attention**, il analyse les relations entre tous les tokens d'une séquence simultanément pour comprendre le **contexte global**

🔗 **Sémantique** : La combinaison de ces techniques permet au LLM de comprendre le **sens profond** d'une phrase, et pas seulement sa structure. C'est ainsi qu'il peut comparer la sémantique d'une question avec celle des **chunks** pour le **RAG**

RAG (Retrieval-Augmented Generation)



Retrieval : Les documents sont découpés en **chunks** et transformés en **embeddings**. Un modèle de Retrieval type **retrieval dense bi-encodeur**, basé sur l'alignement directionnel (cosine similarity)

$$\cos(q, d) = \frac{E_q^T E_d}{\|E_q\| \|E_d\|},$$

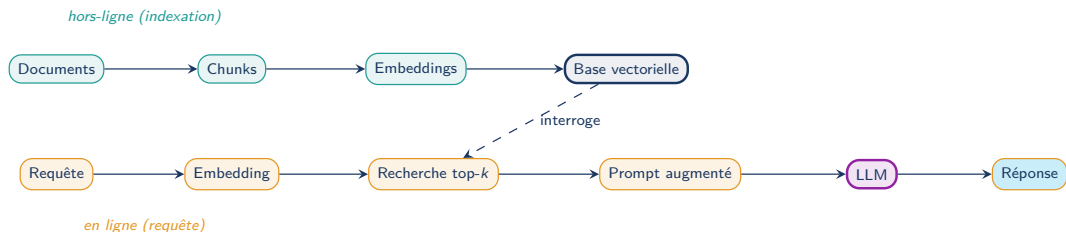
passé à l'action, puis un reranker (e.g. **Late Interaction**) recherche ensuite les chunks les plus pertinents.

Augmented : La question initiale est combinée avec le contexte pertinent récupéré. Le tout est structuré dans un **prompt template** pour guider le LLM.

Generation : Le **LLM** reçoit le prompt augmenté et l'utilise pour formuler la réponse finale. Cette méthode permet de générer une réponse précise, basée sur le contexte, et de réduire les hallucinations.

RAG : pourquoi greffer une mémoire externe ?

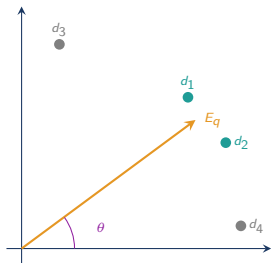
⚠ Le problème. La connaissance d'un LLM est **paramétrique** : figée à la date d'entraînement, opaque, et sujette aux **hallucinations**. Le **RAG** (Lewis et al., 2020) injecte au moment de l'inférence un **contexte récupéré** dans un corpus externe — **sans ré-entraîner** le modèle.



Note

Pour un statisticien : on **conditionne** la réponse sur de l'**information externe**, à jour et **traçable**, sans toucher aux paramètres du modèle. L'indexation est faite **une fois** ; seule la phase **en ligne** se tourne à chaque requête.

Le coeur du RAG : similarité cosinus et plus proches voisins



On récupère les chunks dont l'embedding **pointe dans la même direction** que celui de la requête (d_1, d_2), pas les autres (d_3, d_4).

 La mesure : le cosinus.

$$\cos(q, d) = \frac{E_q^\top E_d}{\|E_q\| \|E_d\|}.$$

C'est le **produit scalaire normalisé** : pour des vecteurs centrés, c'est exactement une **corrélation**. La recherche revient donc à un problème de **k plus proches voisins** que vous connaissez.

 **À grande échelle.** Comparer la requête à des millions de chunks exactement est coûteux : on utilise des index **ANN** (*Approximate Nearest Neighbors*, e.g. HNSW) qui échangent un peu d'exactitude contre beaucoup de vitesse.

 Le **découpage en chunks** (taille, chevauchement) conditionne fortement la qualité : trop gros = bruit, trop petit = contexte perdu.

Affiner la récupération : deux étages, et les limites

🔑 Récupérer puis re-classer (*rerank*).



Bi-encodeur : encode requête et chunk **séparément** $\Rightarrow$ rapide, indexable, mais grossier. **Cross-encodeur / late interaction** (*ColBERT*, Khattab & Zaharia, 2020) : encode la **paire ensemble** $\Rightarrow$ précis mais coûteux. On combine : le premier **filtre**, le second **ordonne**.

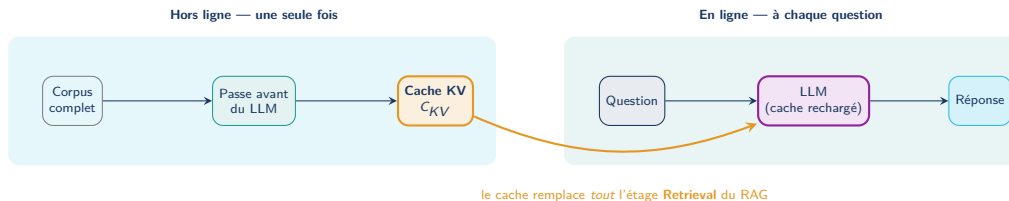
⚠ Ce que le RAG ne fait pas

- *Garbage in, garbage out* : la réponse ne vaut que ce que vaut le corpus.
- Il **réduit** les hallucinations, il ne les élimine pas.
- Si la récupération rate, le LLM **comble** (parfois faux).

” Pour la recherche, l’atout décisif est la **traçabilité** : chaque affirmation peut être **rattachée à une source**.

CAG : et si l'on mettait tout le corpus dans le contexte ?

Cache-Augmented Generation (Chan et al., 2025). Puisque les fenêtres atteignent 1M de tokens, on **précharge le corpus entier** et l'on **précalcule le Key-Value Cache**. À l'exécution, plus aucune récupération.



✓ Ce qu'on gagne

Zéro erreur de récupération : rien ne peut être "manqué", puisque rien n'est sélectionné. Ni base vectorielle, ni *chunking*, ni *reranker* à régler. Latence réduite. Et le modèle peut **croiser** deux documents éloignés, ce que le RAG top-*k* empêche.

⚠ Ce qu'on perd

Le corpus doit **tenir dans la fenêtre**. Le cache KV est volumineux : 80 couches, 8 têtes KV, $d_h = 128$ en 16 bits $\Rightarrow \approx 320$ ko *par token*, soit ≈ 33 Go pour 100k tokens. Toute mise à jour oblige à **tout recalculer**, et la **traçabilité** disparaît.

RAG ou CAG ? Le tableau d'arbitrage

	RAG	CAG
Taille de corpus	illimitée (millions de documents)	bornée par la fenêtre ($\lesssim$ 1M tokens)
Corpus qui change	réindexation incrémentale	recalcul complet du cache
Coût par question	faible (5 chunks dans le prompt)	élevé : le contexte entier est facturé
Risque principal	la récupération rate la bonne pièce	<i>lost in the middle</i> : l'information au milieu d'un long contexte est moins bien exploitée
Traçabilité	native (on sait quel chunk a servi)	faible : à reconstruire par <i>prompting</i>
Complexité d'ingénierie	élevée (chunking, index, reranker)	faible

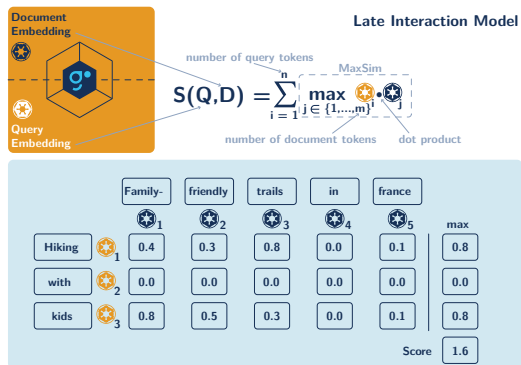
🔑 Règle de décision

Corpus **petit, stable et homogène** (un manuel, un jeu de normes, la documentation d'un produit) $\Rightarrow$ **CAG**. Corpus **vaste, mouvant, ou devant être cité** $\Rightarrow$ **RAG**. Entre les deux, l'**hybride** : CAG sur un noyau stable, RAG pour le reste.

🎓 Pour ELA (notre tuteur d'économétrie)

Le cours tient largement sous 1M de tokens et ne change qu'une fois par an : le CAG serait techniquement viable. Mais l'exigence pédagogique de **citer la section du cours** fait pencher pour le RAG. *C'est un bon sujet de discussion en séance : la contrainte décisive n'est pas technique, elle est pédagogique.*

Retrieval Late Interaction



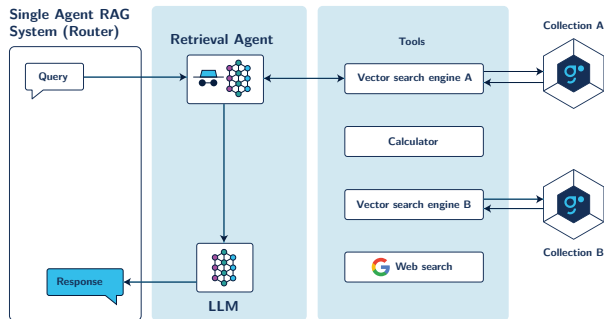
Principe : Au lieu de résumer la requête et le document en un seul vecteur chacun, la **Late Interaction** compare chaque token de la requête avec chaque token du document de manière granulaire.

Calcul de Similarité : Une matrice de scores est calculée. Chaque cellule représente la similarité (ici, un **produit scalaire**) entre un token de la requête (ex : *hiking*) et un token du document (ex : *trails*).

MaxSim : Pour chaque token de la requête, on ne conserve que le score le plus élevé de sa ligne. Cela identifie le meilleur **écho sémantique** pour ce mot dans le document (ex : pour *kids*, le meilleur score est 0.8 avec *Family-*).

Score Final : Le score de pertinence final est simplement la **somme** de ces scores maximums. Cette méthode récompense les documents qui offrent des correspondances fortes pour plusieurs concepts clés de la requête.

RAG Agentique



 **Router** : Un **agent IA** qui analyse la requête de l'utilisateur. Au lieu de consulter une seule source, il agit comme un chef d'orchestre et choisit l'outil le plus pertinent pour trouver la réponse.

 **Outils** : L'agent dispose d'un ensemble de **ressources externes** pour accomplir sa tâche, comme des moteurs de recherche vectorielle pour des bases de connaissances distinctes, une calculatrice pour les opérations mathématiques, ou une recherche web pour des données à jour.

 **Génération** : Une fois que les outils ont collecté les informations nécessaires, le **LLM** reçoit la requête originale et le contexte enrichi pour formuler une réponse finale complète et précise.

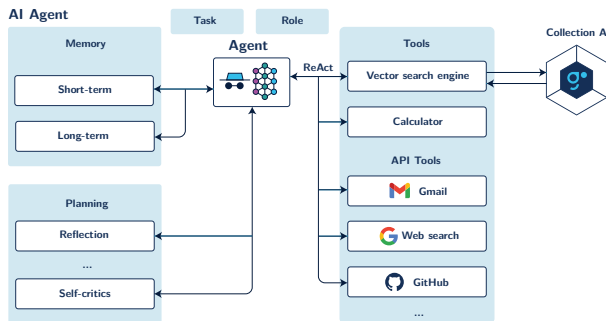
L'Agent IA

 **Agent Principal** : Agit comme un **chef d'orchestre** intelligent qui va au-delà du RAG. Il peut planifier, raisonner, et utiliser une mémoire pour accomplir une tâche.

 **Mémoire** : L'agent utilise une mémoire à **court terme** pour le contexte de la conversation actuelle et une mémoire à **long terme** pour stocker des informations et des leçons apprises sur la durée.

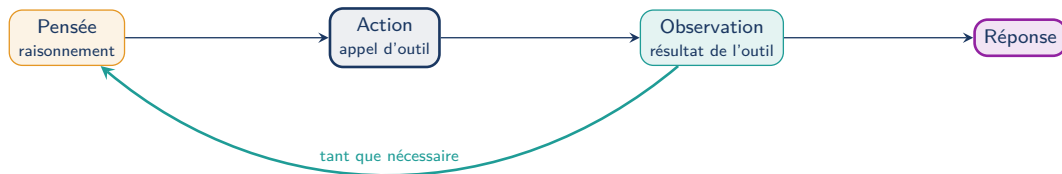
 **Planification & Réflexion** : L'agent est capable d'améliorer ses performances. Il analyse ses actions passées et **évalue ses plans** avant de les exécuter pour se corriger.

 **Outils & Actions** : Pour étendre ses capacités, l'agent utilise des outils externes et peut exécuter des actions au sein d'autres systèmes via des **APIs** (e.g. envoyer un email, poster un message sur Slack) ou des **MCP**.



La boucle agentique : ReAct (raisonner + agir)

Un LLM “nu” répond en **un coup**. Un **agent** fonctionne en **boucle** : il **raisonne**, **agit** (via un outil), **observe** le résultat, et recommence jusqu'à pouvoir conclure — c'est le schéma **ReAct** (Yao et al., 2022).



Note

C'est une **boucle de rétroaction** : un **raffinement itératif** (proche, dans l'esprit, de l'optimisation pas-à-pas). L'agent ne “sait” pas tout d'avance ; il **interroge le monde** et corrige sa trajectoire.

💡 **Différence avec le RAG.** Le RAG récupère **une fois** ; l'agent peut **enchaîner** plusieurs récupérations, calculs et actions, en **décidant** à chaque étape de la suite à donner.

Les piliers de l'agent : mémoire, planification, outils

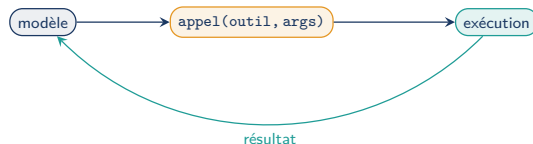
Mémoire

Court terme : la fenêtre de contexte (le fil de la conversation en cours). **Long terme** : un magasin vectoriel (un RAG) où l'agent range faits et ré leçons apprises à qui survivent à la session.

Planification

Décomposer une tâche complexe en sous-étapes (*Plan-and-Solve*, Wang et al., 2023), puis **s'auto-évaluer** et se corriger avant/après l'action (*Self-Refine*, Madaan et al., 2023 — ré réflexion, ré auto-critique).

 **Outils (*function calling*)**. Ce qui donne des ré mains à à l'agent : le modèle émet un **appel structuré** (nom d'outil + arguments), que le système **exécute** avant de lui renvoyer le résultat.

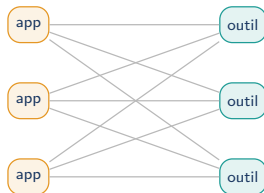


Recherche vectorielle, calculatrice, e-mail, web, dépôt de code... et, de façon standardisée, les **MCP**

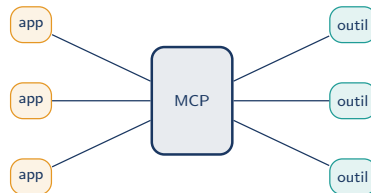
MCP (Model Context Protocol) by Anthropic

⚠ **Le problème d'intégration.** Brancher M applications sur N outils “à la main” demande jusqu’à $M \times N$ connecteurs sur mesure (explosion quadratique). Le **MCP** (Anthropic, nov. 2024) impose **un** protocole : chaque application implémente **une fois** le client, chaque outil **une fois** le serveur $\Rightarrow M + N$.

Sans protocole : $M \times N$ connecteurs



Avec MCP : $M + N$



🔌 L'architecture (le “USB-C de l'IA”)

Host : héberge l'agent et le **client**. **Serveur** : expose des **capacités** — pas seulement des API, mais aussi des logiciels locaux (LaTeX, Stata).

🛡 Sécurité

L'host impose une **approbation utilisateur** avant toute action ; le serveur **isole** l'accès aux ressources locales.

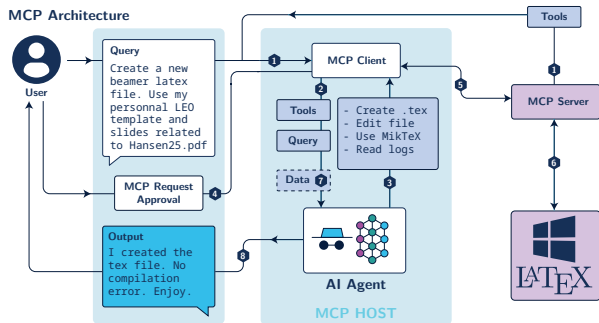
MCP (Model Context Protocol) by Anthropic

 **Host** : L'interface utilisateur héberge l'**Agent IA** et le **Client**. En analysant la requête, l'agent raisonne et choisit l'outil le plus pertinent dans la liste des capacités déclarées.

 **Serveur d'Outils** : Le serveur est une passerelle des **capacités externes**. Il n'est pas limité aux APIs et peut encapsuler des logiciels locaux (e.g. LaTeX, Stata), interagir avec le système de fichiers ou exécuter des scripts.

 **Processus** : L'Agent demande une action. Le client transmet cette demande au serveur, qui la traduit en **commandes concrètes** et renvoie le résultat (succès, log d'erreurs).

 **Sécurité** : L'Host impose une approbation utilisateur avant toute action de l'Agent, et le serveur isole l'accès aux ressources locales.

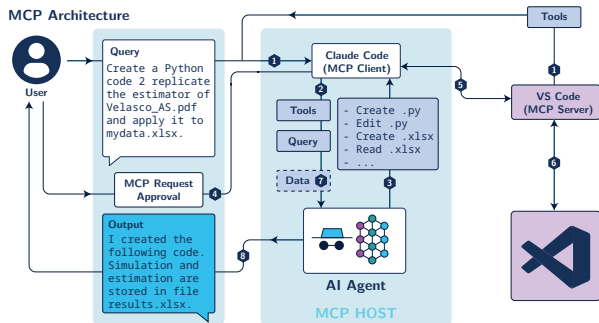


MCP & Vibe Coding

🔗 **AI-assisted Coding** : Application concrète du MCP où un agent (e.g. **Claude**) assiste un développeur sur des tâches complexes, directement dans son environnement de travail (e.g. **VS Code** + **Windows**) via un module intégré (e.g. **Claude Code**).

🧑 **L'Agent Planificateur** : Claude est au centre du système. Sa force est de comprendre une requête de haut niveau et de la décomposer en une séquence d'actions logiques.

📁 **Le Serveur de VS Code** : Il sert de pont sécurisé entre l'Agent et les outils locaux du développeur. Il expose les fonctionnalités de VS Code (lecteurs de fichiers, éditeur, terminal) et exécute les actions demandées.



⚠ **Mise en garde** : Cette architecture ouvre la porte au **Vibe Coding**, c'est à dire la délégation d'une programmation experte à l'IA qui expose les néophytes à une perte de contrôle du projet.

AI & Python

1. AI Genesis
2. AI Arena
3. AI Core
4. AI & Python
5. AI RAG via LangChain
6. AI Agent via LangGraph
7. Intégration Profonde
8. Risques & Limites



Pourquoi Python pour l'IA générative ?

Python est la **langue commune** de l'IA : c'est là que vivent les modèles, les bibliothèques et les exemples. Vous le connaissez déjà pour les **statistiques**, on va simplement **monter d'un cran**.

Le socle indispensable

-  Calcul & données : numpy, pandas
-  Modèles classiques : scikit-learn (*certification exigée*)
-  Réseaux de neurones : pytorch, transformers
-  Applications LLM : langchain, langgraph

 **L'idée du chapitre.** Un LLM seul ne fait que produire du texte. Pour en faire une **application** fiable (qui appelle des outils, suit une logique, garde une trace), on a besoin de deux briques logicielles :

-  **LangChain** la **boîte à outils** : connecter modèles, prompts, outils, données.
-  **LangGraph** l'**orchestrateur** : transformer un programme en **graphe d'états** traçable.

Du modèle à l'application : ce qu'il faut coder autour

Un appel brut à l'API d'un LLM renvoie **une réponse, une fois**. Une vraie application demande tout un **échafaudage** autour du modèle :

 **Gabarits de prompt** : insérer proprement les variables dans le message.

 **Parsing de sortie** : récupérer du *structuré* (JSON, nombre) et non du texte libre.

 **Outils** : laisser le modèle appeler une fonction, une base, un calcul.

 **Récupération (RAG)** : brancher une mémoire externe.

 **Mémoire** : garder le fil de la conversation.

 **Logique de contrôle** : enchaîner, brancher, reboucler.

Note

Coder tout cela à la main à chaque projet, c'est du fragile et non réutilisable. **LangChain** fournit ces briques **standardisées**, exactement comme **scikit-learn** vous évite de réécrire une régression à chaque fois.

 **À retenir.** Un *framework* ne "rend pas le modèle plus intelligent" : il rend l'**application** fiable, lisible et **auditable**.

LangChain : la boîte à outils des applications LLM

Bibliothèque open source (*H. Chase*, 2022), devenue un **standard** ; version **1.0** en oct. 2025, Python 3.10+. Elle fournit des **briques** qu'on assemble.

Les briques de base

-  **Models** : l'accès uniforme aux LLM (OpenAI, Anthropic, Cerebras. ...).
-  **Prompts & Output parsers**.
-  **Tools** (*function calling*).
-  **Retrievers (RAG) & Memory**.

Un écosystème modulaire.

-  langchain-core : les abstractions communes.
-  langchain-openai, -anthropic... : un paquet **par fournisseur**.
-  LangSmith : **observabilité** (tracer, évaluer, déboguer).
-  On installe **seulement** ce dont on a besoin : le cœur est léger, les intégrations sont optionnelles.

Composer des briques : LCEL et l'opérateur |

LangChain compose les briques avec le **LangChain Expression Language (LCEL)** : l'opérateur | chaîne prompt → modèle → parseur.

```
from langchain_openai import ChatOpenAI
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.output_parsers import StrOutputParser

prompt = ChatPromptTemplate.from_template("Resume : {texte}")
modele = ChatOpenAI(model="gpt-4o-mini")
parseur = StrOutputParser()

chaine = prompt | modele | parseur          # composition LCEL
chaine.invoke({"texte": doc})              # -> un resume
```

Note

Le | est exactement la **composition de fonctions** $f \circ g \circ h$ ou, en langage scikit-learn, un Pipeline : chaque étage transforme la sortie du précédent. *Lisible, testable, réutilisable.*

La limite de la chaîne : pas de boucle, pas de mémoire d'état

Une chaîne LCEL est un **flux à sens unique** (un *pipeline*) : l'information avance, elle ne **revient jamais** en arrière. Or un **agent** a besoin de plus :



chaîne linéaire : aucun retour possible

Il manque...

- 🔗 le **branchement** : router selon l'état.
- 🔄 la **boucle** : enrichir le dossier puis **re-décider**.
- 📁 l'**état persistant** : pouvoir reprendre, auditer.
- 🛑 la **pause humaine** : faire valider une décision.

Note

On *pourrait* tout coder dans une boucle `while` ... mais elle devient vite **opaque** : état caché, chemins illisibles, aucune trace. **LangGraph** rend cette logique **explicite** et **dessinable**.

LangGraph : votre programme devient un graphe d'états

LangGraph (par *LangChain Inc.*, v1.0 en 2025 ; inspiré de *Pregel*) décrit un programme comme un **graphe** : des **nœuds** (fonctions) reliés par des **arêtes**, qui font circuler et **enrichir** un **état partagé**.

Note

C'est un **automate** (une machine à états) : l'**état** est un **enregistrement** de variables (comme une ligne d'un `DataFrame`) qui passe de nœud en nœud sauf qu'ici, contrairement à un pipeline, il peut **reboucler**.

✓ Ce que ça apporte d'emblée :

-  le graphe se **dessine** (Python ↔ schéma, presque 1:1) ;
-  l'état est **traçable** et **persistant** ;
-  les **cycles** et le **routage** sont natifs ;
-  la **pause humaine** est possible.

En résumé

L'idée : un agent = un **graphe d'états** explicite, où l'on voit le flux de contrôle.



StateGraph : le programme est un graphe.



État typé (TypedDict) : mémoire de travail qui circule et s'enrichit.



Nœuds = fonctions Python (lisent / mettent à jour l'état).



Edges conditionnels = le routage.



Cycles : retour en arrière — impossible en chaîne linéaire.



checkpointer : persistance (reprise, audit).



interrupt : pause pour un humain.

Les trois ingrédients : état, nœuds, arêtes

🗄️ L'état partagé (TypedDict)

La **mémoire de travail** : un dictionnaire **typé** de variables (features, pd, decision...). Chaque nœud renvoie des **mises à jour** que LangGraph fusionne dans l'état.

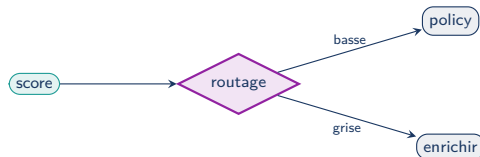
🏠 Les nœuds (*des fonctions*)

Une fonction état → mises à jour. Pure et testable : parse, score, explain... *Chaque nœud est un outil que vous savez déjà écrire en Python.*

🔗 Les arêtes (le câblage)

→ **simple** : "après A, va en B".

🔗 **conditionnelle** : une fonction lit l'état et renvoie la **clé** de la branche à suivre.



💡 Une arête conditionnelle, c'est un **if/else** l'analogue d'une **coupure d'arbre de décision** sur l'état.

Un graphe minimal en quelques lignes

Le squelette est toujours le même : **déclarer l'état, ajouter des nœuds, câbler les arêtes, compiler.**

```
from typing import TypedDict
from langgraph.graph import StateGraph, START, END

class Etat(TypedDict):
    pd: float
    bande: str

    # l'etat partage, type

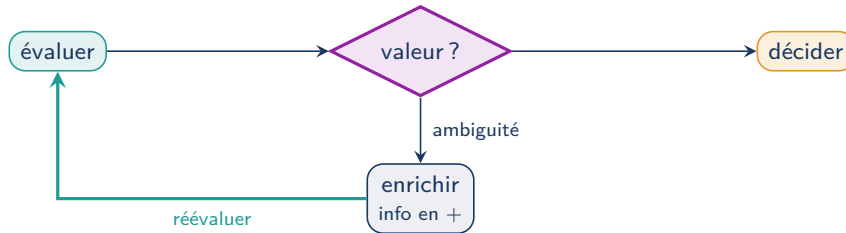
def score(s): return {"bande": "basse" if s["pd"] < .05 else "grise"}
def router(s): return "ok" if s["bande"] == "basse" else "enrichir"

g = StateGraph(Etat)
g.add_node("score", score)
g.add_node("enrichir", enrichir)
g.add_edge(START, "score")
g.add_conditional_edges("score", router,
    {"ok": END, "enrichir": "enrichir"})
g.add_edge("enrichir", "score")
agent = g.compile()
```

💡 `add_conditional_edges(nœud, fonction, mapping)` : la fonction renvoie une clé, le mapping dit où aller.

Le cycle : enrichir le dossier, puis re-décider

L'arête enrichir → évaluer crée une **boucle** : l'agent va chercher l'information manquante, met l'état à jour, puis **réévaluer**. Impossible avec une chaîne linéaire.



Note

Même esprit que le **raffinement itératif** déjà vu (boucle ReAct, descente de gradient pas-à-pas) : on ne sait pas tout d'avance, on **interroge**, on **corrige**, on recommence avec un **compteur** pour éviter la boucle infinie.

Checkpointing : reprendre où l'on s'était arrêté, et auditer

À la compilation, on peut greffer un **checkpoint** : LangGraph sauvegarde l'état après chaque étape. Chaque exécution porte un `thread_id` (l'identifiant du dossier).

```
from langgraph.checkpoint.memory import InMemorySaver

agent = g.compile(checkpointer=InMemorySaver())
config = {"configurable": {"thread_id": "dossier-42"}}
agent.invoke(dossier, config)          # l'état est sauvegardé à chaque pas
```

🔄 **Exécution durable** : un plantage ? on reprend au dernier point sauvé.

🕒 **Piste d'audit** : l'état à chaque étape est conservé.

Note

Pensez à une **journalisation** (ou à la *mémoïsation*) : on garde une trace de chaque transition. C'est exactement ce qu'attend un **régulateur en model risk** : pouvoir **rejouer** et **justifier**.

Human-in-the-loop : mettre le graphe en pause

Au-delà d'un **seuil** (montant, sévérité...), on **rend la main à un analyste**. Le checkpointeur fige l'état ; `interrupt()` **suspend** le graphe ; `Command(resume=...)` le **reprend** avec la décision humaine.

```
from langgraph.types import interrupt, Command

def human(s):
    # noeud de validation
    avis = interrupt({"decision": s["decision"], "pd": s["pd"]})
    return {"validation_humaine": avis}

# ... plus tard, meme thread_id : on reprend avec la reponse
agent.invoke(Command(resume="accord"), config)
```

Le pont vers “Risques & Limites”

C'est ce qui distingue un agent *jouet* d'un agent **opérationnel** : l'humain garde la **décision finale** sur les cas sensibles. **Supervision, traçabilité et explicabilité** ne sont pas des options.

LangChain ou LangGraph ?

Les deux se **complètent** les agents **LangChain** tournent d'ailleurs **sur le runtime LangGraph**.

	LangChain	LangGraph
Niveau	haut (ergonomie)	bas (contrôle fin)
Structure	chaîne / pipeline	graphe d'états
Cycles & routage	limités	natifs
État & reprise	implicites	explicites, persistants
Pour	prototyper vite	orchestrer, fiabiliser

Application RAG

1. AI Genesis
2. AI Arena
3. AI Core
4. AI & Python
5. AI RAG via LangChain
6. AI Agent via LangGraph
7. Intégration Profonde
8. Risques & Limites



Cas d'usage : ELA, un tuteur d'économétrie



Le décor. Un étudiant pose une question :

- sur un **point de cours** (stationnarité, GARCH, cointegration...);
- parfois avec une **image** (une courbe, une matrice, une équation manuscrite).



La sortie attendue. Une réponse **pédagogique et sourcée** — chaque affirmation porte son [Source: fichier] — pas une réponse de culture générale en boîte noire.



Le message clé

Le LLM ne **récite pas** sa mémoire : il **répond à partir des supports de cours** (vos fichiers .tex). C'est le **RAG** qui fournit les faits ; le LLM ne fait que les **rédigier et les citer**.



Pourquoi un RAG ?

Un LLM seul **hallucine**, ignore vos notations et n'a **aucune source**. Le RAG lui donne une **mémoire externe vérifiable**, mise à jour sans ré-entraînement.

RAG : ancrer la génération dans une mémoire externe

Un LLM stocke des connaissances dans ses **poids** (*mémoire paramétrique*) : figée, opaque, sans traçabilité. Le **RAG** (Lewis et al., 2020) y ajoute une **mémoire non-paramétrique** : une base de documents qu'on **interroge à la volée**.

⚠ LLM seul

- ✗ invente des références (*hallucination*) ;
- ✗ connaissances **figées** à l'entraînement ;
- ✗ ignore vos définitions et notations ;
- ✗ **aucune** source vérifiable.

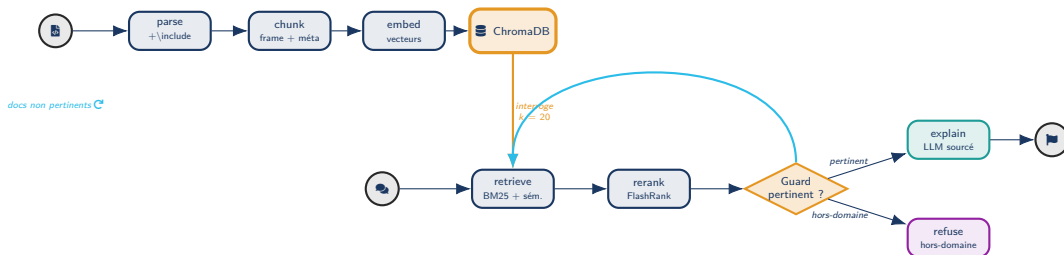
✓ LLM + RAG

- ✓ réponses **ancrées** dans les documents ;
- ✓ base **mise à jour** sans ré-entraînement ;
- ✓ suit *votre* cours (priorité absolue) ;
- ✓ chaque phrase **cite sa source**.

🔗 L'intuition

Pensez à un *examen à livre ouvert* : on ne demande pas à l'étudiant de **tout savoir par cœur**, mais d'aller **chercher la bonne page** puis de **rédigier** avec ses mots. Le RAG, c'est exactement ça.

L'architecture d'ELA : deux temps, hors-ligne et en ligne



- **Hors-ligne** (ingest.py) : on *indexe* le cours une fois.
- **En ligne** : on *récupère*, on *contrôle* (losange = Guard), puis on *rédige*. La *boucle* de re-recherche et le *refus* sont des **routages conditionnels**.

Hors-ligne : indexer le cours (ingest.py)

Avant toute question, on transforme les .tex en une base interrogeable. Le découpage n'est pas arbitraire : **une frame Beamer = une unité de sens**.

- 🔧 **Résolution** des `\include` / `\input` (récursive, anti-cycle) : le cours éclaté devient un seul texte.
- ✂️ **Découpage** par section puis par frame ; filet de sécurité : `RecursiveCharacterTextSplitter` (chunks ~ 2000 car., recouvrement 200).
- 📁 **Métadonnées** attachées à chaque chunk : level (M1/M2/Commun), course, chapter, section, slide_title.
- 🧹 **Nettoyage** : on retire la mise en page (`\centering...`), on garde les maths.

📁 Pourquoi les métadonnées ?

Elles permettent un **filtrage ultérieur** (ne montrer que le niveau de l'étudiant) et une **citation précise** ([Source: fichier]). *La structure du cours devient une clé d'index.*

🔗 L'analogie

Comme pandas : on **nettoie** et on **étiquette** la donnée *avant* de l'analyser. Ici la donnée z, c'est du texte de cours.

Du texte aux vecteurs : embeddings & base vectorielle

Chaque chunk est transformé en un **vecteur** par un modèle d'**embedding** multilingue (sentence-transformers / mpnet). Deux textes au **sens proche** $\Rightarrow$ deux vecteurs **géométriquement proches**.

Note : cosinus = corrélation

La proximité se mesure par la **similarité cosinus** :

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u}^\top \mathbf{v}}{\|\mathbf{u}\| \|\mathbf{v}\|}.$$

Sur des vecteurs **centrés**, c'est *exactement* le **coefficient de corrélation** que vous connaissez. Chercher le passage pertinent = trouver le vecteur le plus **corrélé** à la question.

 **Embedding** : un texte $\rightarrow$ un point dans $\mathbb{R}^d$ (d de quelques centaines pour un encodeur de RAG, quelques milliers pour un LLM génératif).

 **ChromaDB** : stocke ces vecteurs + leurs métadonnées, et renvoie les k **plus proches** d'une requête.

 **Multilingue** : la même question en FR/EN tombe au même endroit de l'espace.

 **À retenir.** L'embedding n'est pas magique : c'est une **projection apprise** où la géométrie encode le sens.

En ligne (1) : la récupération hybride

Deux façons de chercher, complémentaires, fusionnées par un `EnsembleRetriever` (poids 0,5/0,5).

≡ BM25 — le lexical

Score de pertinence fondé sur la **fréquence des mots** (héritier du *TF-IDF* que vous connaissez). Imbattable pour les **termes exacts** : VECM, racine unitaire, un nom de test.

📖 Vecteurs — le sémantique

Recherche par sens (cosinus sur embeddings). Retrouve le bon passage même si l'étudiant emploie d'**autres mots** : *variance qui change dans le temps* → *hétéroscédasticité conditionnelle*.

🔄 Pourquoi fusionner ?

Le **lexical** rate les synonymes ; le **sémantique** rate parfois un sigle rare. Ensemble, ils **augmentent le rappel** : on récupère large ($k=20$) pour ne rien manquer. La *précision* viendra à l'étape suivante.

En ligne (2) : resserrer — reranking & filtrage par niveau

⚡ Reranking (FlashRank)

Les 20 candidats passent dans un **cross-encoder** qui **relit chaque (question, passage) ensemble** et les réordonne finement. On ne garde que le **top 5**.

Deux étapes : on maximise d'abord le **rappel** (hybride large), puis la **précision** (rerank serré).

⚡ Filtrage par niveau

La métadonnée **level** restreint la recherche : un étudiant **M1** ne voit que **M1 + Commun**.

Filtre **natif** côté Chroma (**where**), **post-filtrage** côté BM25.
Même base, périmètres cloisonnés.

🔗 L'intuition statisticienne

Rappel vs précision : exactement l'arbitrage que vous connaissez en classification. Ici on **ouvre large** (ne rater aucun bon document) **puis on trie** (ne garder que les meilleurs).

Le RAGGuard : le garde-fou qui route

Entre la récupération et la génération, un **module léger** (`rag_guard.py`) intercepte et **décide**. C'est le cœur de la fiabilité d'ELA.

Un LLM-juge évalue, en JSON, **deux** questions :

-  **Pertinence** : les documents récupérés répondent-ils vraiment ? (`docs_relevant`)
-  **Domaine** : la question est-elle dans le périmètre du cours ? (`query_in_domain`)

Puis il **route** (action) :

- ✓ `answer_rag` — on répond depuis les docs ;
- ↺ `retry_query` — docs faibles à reformuler → **re-recherche** ;
- ⊘ `refuse_domain` — hors cadre → **refus poli**.

Ça vous dit quelque chose ?

Pertinent ? → **losange**. Reboucler ? → **cycle**.
Hors-domaine ? → **branche bloquante**.

C'est **exactement** la logique de l'**agent crédit** du chapitre suivant : routage conditionnel, boucle, garde-fou.

Anti-hallucination

Le Guard évite le pire défaut du RAG : **retomber sur sa mémoire paramétrique** quand la base avait pourtant la réponse — ou **inventer hors-sujet**.

En ligne (3) : la génération sourcée (main_ela.py)

Une fois les bons documents validés, le LLM rédige — sous **contrat strict**.

 **Hiérarchie des connaissances** : (1) le cours d'abord ; (2) savoir spécialisé **seulement** si le contexte est muet ; (3) sinon **refus**.

 **Protocole de citation** : chaque affirmation se termine par [Source: fichier] ou [Source: Connaissances Générales].

 **Format** : LaTeX imposé ($\dots$), **vos notations** respectées.

 **Streaming** : la réponse s'affiche au fil de l'eau (astream).

Bonus : vision multimodale

Si l'étudiant glisse une **image** (courbe, matrice, équation manuscrite), elle est encodée en base64 et envoyée au même modèle **multimodal**, qui l'explique *dans le cadre du cours*.

Qui décide quoi ?

Documents → le contenu. **Guard** → répondre / refuser / reboucler. **LLM** → rédiger & citer. Jamais inventer.

Le code clé : assembler le pipeline

Le RAG d'ELA : récupérer (hybride) → resserrer (rerank) → contrôler (Guard) → générer (sourcé).

```
1 from langchain_chroma import Chroma
2 from langchain_community.retrievers import BM25Retriever
3 from langchain.retrievers import EnsembleRetriever
4
5 # 1. Recuperation HYBRIDE : lexical (BM25) + semantique (vecteurs)
6 chroma_ret = vecto.as_retriever(search_kwargs={
7     "k": 20, "filter": {"level": {"$in": ["M1", "Commun"]}}} # filtre niveau
8 bm25_ret = BM25Retriever.from_documents(docs); bm25_ret.k = 20
9 hybride = EnsembleRetriever(retrievers=[bm25_ret, chroma_ret],
10                             weights=[0.5, 0.5])
11 docs = hybride.invoke(question) # rappel large
12 docs = reranker.compress_documents(docs, question) # 2. precision (top 5)
13
14 # 3. GARDE-FOU : noter la pertinence puis ROUTER
15 grade = await guard.grade(question, docs) # -> JSON {action, ...}
16 if grade.action == "refuse_domain":
17     return "Hors du cadre du cours." # refus
18 if grade.action == "retry_query":
19     docs = hybride.invoke(grade.suggested_query) # re-recherche (boucle)
20
21 # 4. Generer une reponse SOURCEE : le LLM cite [Source: fichier]
22 async for token in llm.astream(prompt_rag(docs, question)):
23     yield token
```

Ce qu'il faut retenir — et le pont vers LangGraph

 **RAG = ancrage** : on répond *depuis les documents*, pas depuis la mémoire du modèle.

 **Hybride + rerank** : rappel large *puis* précision — l'arbitrage que vous connaissez.

 **Guard = garde-fou** : il **note**, **route** et **refuse** — la fiabilité ne vient pas du LLM seul.

 **Séparation des rôles** : récupération vs génération, faits vs rédaction.

Note vers la suite

Ici le routage (pertinent ? / hors-domaine ? / reboucler ?) est un **if/else codé à la main**.

Quand la logique se complique — boucles, branches, pauses, mémoire d'état — on a besoin d'un **orchestrateur** qui rend le graphe **explicite et auditable**.

➔ chapitre suivant : **un agent crédit sous LangGraph**.

Application

1. AI Genesis
2. AI Arena
3. AI Core
4. AI & Python
5. AI RAG via LangChain
6. AI Agent via LangGraph
7. Intégration Profonde
8. Risques & Limites



Cas d'usage : un co-pilote de pré-décision crédit

 **Le décor.** Une demande de prêt arrive :

- des **données structurées** (revenu, ancienneté, encours, incidents de paiement...);
- un peu de **texte libre** (objet du prêt, note du conseiller).

 **La sortie attendue.** Une **recommandation motivée et auditable** — *accord, refus* ou *à instruire* — pas une décision en boîte noire.

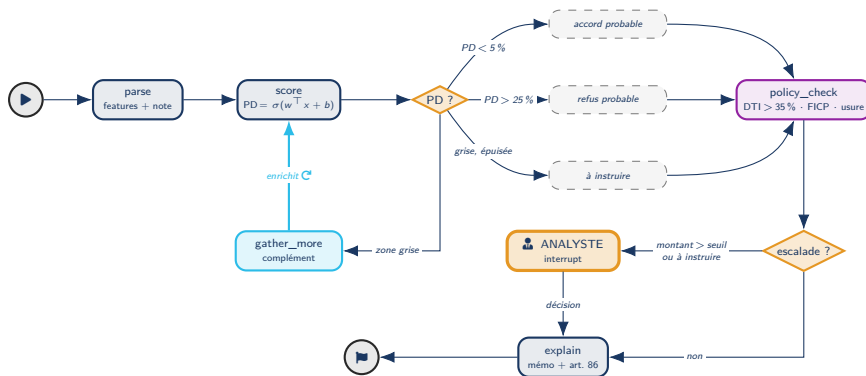
Le message clé

L'agent **orchestre** et **explique**. C'est le **scorecard** (régression logistique) qui **décide quantitativement**. Le LLM n'invente pas la probabilité de défaut : il **appelle** le modèle, **route** selon le résultat, **applique** les règles, puis **rédige** la justification.

Pourquoi un *agent* ?

Un appel LLM unique répond en **un coup**. Ici il faut **boucler** (chercher un complément), **router** (selon le risque) et **escalader** (à un humain).

L'architecture de l'agent



- La boucle `gather_more` → `score` est le cœur agentique ; les losanges sont des **edges conditionnels** ; **ANALYSTE** est le **human-in-the-loop**.
- Les cadres en pointillés ne sont pas des nœuds : le routage *propose* une bande, *policy_check dispose*.

Les nœuds, un par un (1/2)

-  **parse** — normalise la demande (structuré + texte libre) en un **vecteur de features** propre. *Primitive : nœud d'entrée.*
-  **score** — appelle le **scorecard**, une **régression logistique** pré-entraînée, et renvoie la **probabilité de défaut** (PD). *Primitive : nœud-outil. La prédiction = vos stats, pas le LLM.*
-  **routage** — compare la PD aux seuils (bandes *basse / grise / haute*) et choisit la branche. *Primitive : edge conditionnel.*

Qu'est-ce qu'une *scorecard* ?

Une **grille de score** est la forme **opérationnelle** d'un modèle de score de crédit : on estime une **régression logistique** sur l'historique défaut / non-défaut, on découpe chaque variable en **bandes**, puis on convertit les coefficients en **points entiers** par bande. Le score d'un dossier est la **somme** des points ; une table le retraduit en probabilité de défaut.

Ce n'est donc pas un autre modèle : c'est $PD = \sigma(w^T x + b)$ — exactement le **neurone-sigmoïde** du chapitre AI Core — **ré-exprimé en points additifs** pour qu'un analyste puisse le calculer et l'**expliquer à la main**. C'est ce qui le rend **auditable**, donc défendable devant un superviseur.

Les nœuds, un par un (2/2)

-  **gather_more** — en zone grise, l'agent **décide quel complément chercher** (ratio d'endettement DTI, scénario de stress, donnée manquante), met à jour l'état et **reboucle vers score**. *L'agent ne sait pas tout d'avance : il enrichit itérativement* (même esprit que l'optimisation pas-à-pas, déjà vue avec ReAct).
-  **policy_check** — applique des **règles dures déterministes** (taux d'usure, plafond DTI, liste noire). Un **garde-fou non négociable** que le LLM ne peut pas contourner.
-  **explain** — le **vrai travail du LLM** : transformer chiffres + règles en un **mémo clair, motivé et défendable** (sortie structurée).

Qui décide quoi ?

Stats (logistique) → la *probabilité*. **Règles** (déterministes) → les *interdits*. **LLM** → *router, collecter, expliquer*. Jamais l'inverse.

L'état partagé : la mémoire de travail

```
1 from typing import TypedDict, Optional, Literal
2
3 class DossierCredit(TypedDict):
4     features: dict           # revenu, anciennete, encours...
5     pd: Optional[float]     # proba de default (scorecard)
6     bande: Optional[str]    # "basse" / "grise" / "haute"
7     n_enrichissements: int   # compteur : borne la boucle
8     decision: Optional[str]  # "accord" / "refus" / "a instruire"
9     motifs: list[str]        # justifications auditablees
10
11 MAX_ENRICH = 2
12 def router(e: DossierCredit) -> Literal["approuver", "refuser",
13     "enrichir", "instruire"]:
14     if e["bande"] == "basse": return "approuver" # PD < 5 %
15     if e["bande"] == "haute": return "refuser"   # PD > 25 %
16     if (e["n_enrichissements"] < MAX_ENRICH
17         and e["features"].get("co_emprunteur_possible")):
18         return "enrichir" # -> gather_more
19     return "instruire"    # rien a mobiliser
```

⚠ Sans le compteur, la boucle ne termine pas

Un routage qui renvoie "enrichir" dès que la PD est en zone grise peut **boucler indéfiniment** : rien ne garantit que l'enrichissement fasse sortir de la bande. `n_enrichissements` est la **condition d'arrêt** ; la quatrième branche, "instruire", produit la sortie à instruire à annoncer au cadrage.

Human-in-the-loop, persistance & auditabilité

- 🔊 **interrupt** — au-delà d'un **seuil de montant** (ou en zone grise persistante), le graphe **se met en pause** et rend la main à un analyste crédit, qui tranche ; l'état reprend ensuite.
- 📅 **checkpointer** — l'état est **figé à chaque étape** : on **reprend** exactement où l'on s'était arrêté.
- 📅 **journal** — une ligne **horodatée** par décision, avec la **version du modèle et des seuils**. C'est *lui*, pas le checkpointer, qui constitue la piste d'audit.



Ce qu'on montre à l'analyste

Un contrôle humain n'est effectif que si la personne peut **interpréter** la sortie. On ne lui affiche donc pas $\hat{y}$ PD = 5,2 %, validez ? $\hat{z}$ mais aussi la **décomposition du score**, la **trajectoire de PD** et les **motifs**. Sinon l'humain *tamponne*, il ne contrôle pas.



Le piège du *checkpointer*

InMemorySaver **disparaît** à l'arrêt du processus : c'est un mécanisme de *reprise*, pas de *conservation*. Une piste d'audit exige un support **durable** (SqliteSaver, base) et **append-only**.

Ce que le POC démontre, et ce qu'il ne démontre pas

Compliance et AI Act (voir slide 140). La colonne de droite est l'exercice honnête : un POC pédagogique n'est pas un système conforme.

Exigence	Ce que le code fait	Ce qui manquerait en production
art. 10 données, biais	aucune variable protégée dans la grille; données synthétiques	tests de biais sur les <i>proxies</i> (code postal, prénom, type de contrat)
art. 12 journalisation	journal <i>append-only</i> horodaté : PD, bande, motifs, décision, version du modèle <i>et</i> des seuils	conservation durable, horodatage opposable, rétention documentée
art. 14 contrôle humain	interrupt; l'analyste reçoit la décomposition du score et peut trancher autrement	compétence et formation de l'analyste, mesure du taux d'infirmary
art. 15 exactitude	<i>backtest</i> sur échantillon de test : AUC, Gini, Brier, calibration	suivi de dérive , stress-tests, seuil de ré-entraînement
art. 86 explication	mention du rôle du système automatisé et du droit au réexamen dans le mémo	procédure de recours réellement outillée

Le point à retenir

Ces garde-fous ne sont pas des ornements ajoutés *après* : ils **dictent l'architecture**. C'est parce que la décision doit être explicable qu'on garde une **régression logistique** plutôt qu'un modèle opaque, et parce qu'elle doit être contrôlable qu'on **interdit au LLM de décider**.

Section 4

1. AI Genesis
2. AI Arena
3. AI Core
4. AI & Python
5. AI RAG via LangChain
6. AI Agent via LangGraph
7. Intégration Profonde
8. Risques & Limites



MCP LaTeX

 **Client MCP** : L'interface utilisateur sera ici Claude Desktop et elle fera donc le lien avec un agent Claude AI fonctionnant sous Sonnet ou Opus.

 **Serveur MCP** : Il sera codé en Python et contiendra tous les outils nécessaires à l'exécution d'actions concrètes de la part de l'environnement LaTeX ou Windows

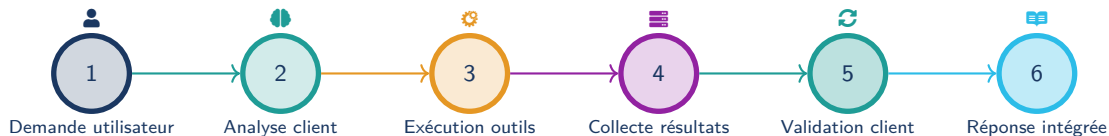
 **Outils MCP** :

- `create_latex_file`
- `read_latex_file`
- `edit_latex_file`
- `compile_latex` : `pdflatex`, `lualatex`, `xelatex`, etc.
- `validate_latex_syntax` : vérifier la syntaxe d'un fichier LaTeX et rechercher les erreurs courantes
- `list_latex_files` : lister les fichiers liés à LaTeX dans un répertoire
- `clean_latex_auxiliary` : supprimer les fichiers auxiliaires générés pendant la compilation
- `get_latex_template` : spécifier `article`, `report`, `beamer`, etc.
- `change_workspace` : changer le répertoire de travail pour les opérations LaTeX

 **Sécurité** : Le serveur étant limité aux outils programmés, il protège le système de l'utilisateur. Un script JSON spécifie également au client MCP le périmètre de son environnement par défaut.

Fonctionnement du Serveur

Cycle de vie d'une requête MCP :



Points clés de cette architecture :

- **Sécurité** : contrôle des accès par le serveur (🧠 + ⚙️)
- **Validation** : boucle de rétroaction pour la gestion des erreurs (⚙️ + ↺)
- **Extensibilité** : ajout facile de nouveaux outils dans le serveur (🔧 + ✂️)

Coder un Serveur MCP

Demandons à Claude

Je souhaite développer un serveur MCP pour faire interagir Claude Desktop directement avec mon éditeur LaTeX. Le serveur doit être compatible avec les distributions LaTeX que j'utilise (TeX Live et MikTeX) et mon éditeur LaTeX (TeXworks). Je souhaite pouvoir créer, lire, éditer et organiser au sein d'un dossier des fichiers .tex, .bib, .sty et .cls.

Le serveur doit également pouvoir organiser au sein d'un dossier les extensions auxiliaires de type .bbl, .blg, .log, .pdf, etc. Il doit aussi pouvoir gérer, vérifier et corriger les syntaxes usuelles de LaTeX (pdfLaTeX, LuaLaTeX, XeLaTeX, BibTeX et Biber), supporter la rédaction des en-têtes et de la structure du document pour les types article, report, book, letter, beamer et les autres classes ou thèmes directement disponibles dans le dossier. Enfin, le serveur doit permettre à l'IA de gérer les modèles intégrés pour les types de documents courants.

Je ne souhaite pas intégrer une fonctionnalité de gestion de packages et de dépendances, car cela risquerait de requérir des autorisations complexes et sensibles d'accès à des répertoires d'installation de la distribution LaTeX.

Je souhaite coder le serveur en Python. Bien que j'utilise VS Code pour coder, je ne souhaite pas utiliser d'intégration de Claude dans VS Code pour interagir avec TeXworks.

Peux-tu me guider dans le développement de ce serveur ? Attention lorsque tu coderas le serveur sous Python, pense à bien prendre en compte les dernières mises-à-jour du SDK MCP pour que la communication avec Claude soit correctement établie.

En particulier, pense à `self.server.create_initialization_options()` dans l'appel de `await self.server.run`.

🔗 Claude va générer le code Python du serveur : `mcp_latex_server.py`

Prérequis pour l'Installation

Composants nécessaires :

 **Python 3.8+** : Interpréteur pour le serveur MCP

 **Distribution LaTeX** : MiKTeX (Windows) ou TeX Live (multiplateforme)

 **Claude Desktop** : Application cliente d'Anthropic

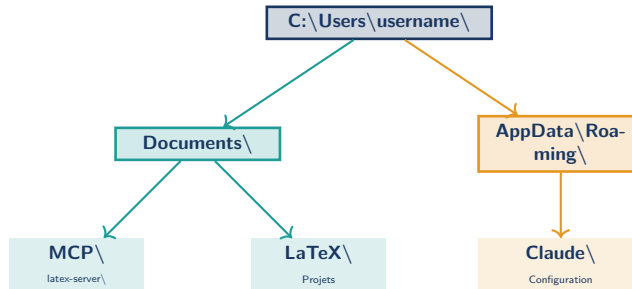
 **Permissions** : Accès en écriture aux dossiers utilisateur

Code et instruction d'installation :

 <https://github.com/varennnes-ecofin/MCP-latex-server>

Étape 1 : Création de la Structure

Organisation des répertoires :



Commandes de création :

```
1 mkdir C:\Users\username\Documents\MCP\latex-server
2 mkdir C:\Users\username\Documents\LaTeX
```

Étape 2 : Installation des Dépendances Python

Installation du serveur MCP :

```
1 # Naviguez vers le répertoire du serveur
2 cd C:\Users\username\Documents\MCP\latex-server
3
4 # Créer un environnement virtuel (optionnel mais recommandé)
5 python -m venv .venv
6 .venv\Scripts\activate
7
8 # Installer les dépendances MCP
9 pip install mcp aiofiles
```

Alternatives d'installation :

- 🌐 **Installation globale** : `pip install mcp aiofiles`
- ⚠️ **Important** : notez le chemin vers l'exécutable Python utilisé
- ⚙️ **Conda** : `conda install -c conda-forge mcp`

📁 **Environnement virtuel** : recommandé pour l'isolation

Étape 3 : Déploiement du Serveur

Placement du fichier serveur :

- 📄 Téléchargez ou copiez `mcp_latex_server.py`
- 📁 Placez-le dans `C:\Users\username\Documents\MCP\latex-server\`
- Testez le serveur en mode standalone

Test du serveur :

```
1 cd C:\Users\username\Documents\MCP\latex-server
2 python mcp_latex_server.py --workspace C:\Users\username\Documents\LaTeX
```

Résultat attendu :

```
1 YYYY-MM-DD HH:MM:SS,MS - __main__ - INFO - Detected TeX distribution: miktex
2 YYYY-MM-DD HH:MM:SS,MS - __main__ - INFO - Starting LaTeX MCP Server with workspace: C:\Users\username\Documents\LaTeX
```

- ✔ Aucune erreur de syntaxe Python
- ✔ Modules MCP importés correctement
- ✔ Workspace détecté et accessible
- ❏ Arrêtez le serveur avec `Ctrl+C` après vérification

Étape 4 : Configuration de Claude Desktop

Localisation du fichier de configuration :

```
1 C:\Users\username\AppData\Roaming\Claude\claude_desktop_config.json
```

Contenu du fichier JSON :

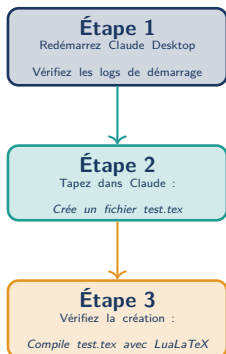
```
1 {  
2   "mcpServers": {  
3     "latex-server": {  
4       "command": "C:\\Users\\username\\Documents\\MCP\\latex-server\\.venv\\Scripts\\python.exe",  
5       "args": [  
6         "C:\\Users\\username\\Documents\\MCP\\latex-server\\mcp_latex_server.py",  
7         "--workspace",  
8         "C:\\Users\\username\\Documents\\LaTeX"  
9       ]  
10    }  
11  }  
12 }
```

Important :

- ⚠ Utiliser des doubles backslashes (\\)
- 🔧 Adapter le chemin vers Python en l'absence d'environnement virtuel
- 📁 Créer (si le fichier n'existe pas) et sauvegarder en UTF-8

Étape 5 : Test et Validation

Procédure de validation complète :



✓ Succès :

- Serveur chargé sans erreur
- Fichier créé automatiquement
- Template LaTeX valide

✗ Échec possible :

- Configuration JSON invalide
- Chemin Python incorrect
- Permissions insuffisantes
- Distribution LaTeX manquante

Cas d'usage simple du MCP Latex : prompting

Prompt envoyé à Claude Desktop

J'ai besoin que tu accèdes via mon MCP latex au répertoire `D:\Dropbox\Research\StableMixtureFiltration\Beamer` et ses sous-dossiers. Il faut que tu accèdes dans le projet (min rag) à l'article `EJv3.tex` et que tu crées des slides de présentation (`GreenBubbleBeamer.tex`) à l'aide de ma classe personnalisée `beamer beamerthemeleo.sty` pour cet article. Tous les graphiques utilisés dans `EJv3.tex` sont stockés dans le sous-dossier "figures" et possèdent les mêmes noms. Pour avoir une idée de la structure des slides je t'ai donné un exemple de slides réalisés pour un autre article `IA4Financev1.tex`. La présentation doit être en anglais, et pour une conférence d'économétrie théorique (donc n'hésite pas à être technique et à utiliser les équations de l'article). Soit le plus précis et rigoureux possible dans les slides que tu rédiges. La présentation va durer 20 minutes donc prévois au plus 20 slides. Compile le `.tex` pour t'assurer que cela tourne. Si tu as des hésitations, interroge moi.

Cas d'usage simple du MCP Latex : Plan-and-Solve Prompting

J'ai besoin que tu accèdes via mon MCP latex au répertoire "D:\Dropbox\Research\StableMixtureFiltration\Beamer" et ses sous-dossiers. Il faut que tu accèdes dans le projet (min rag) à l'article EJ_v3.tex et que tu crées des slides de présentation (GreenBubbleBeamer.tex) à l'aide de ma classe personnalisée beamer beamerthemeleo.sty pour cet article. Tous les graphiques utilisés dans EJ_v3.tex sont stockés dans le sous-dossier "figures" et possèdent les mêmes noms. Pour avoir une idée de la structure des slides je t'ai donné un exemple de slides réalisés pour un autre article IA4Finance_v1.tex. La présentation doit être en anglais, et pour une conférence d'économétrie théorique (donc n'hésite pas à être technique et à utiliser les équations de l'article). Soit le plus précis et rigoureux possible dans les slides que tu rédiges. La présentation va durer 20 minutes donc prévois au plus 20 slides. Compile le .tex pour t'assurer que cela tourne. Si tu as des hésitations, interroge moi.

26 mai   

Orchestrated access to project files pour créer slides techniques >

Je vais commencer par accéder à ton MCP LaTeX et examiner les fichiers nécessaires.

Localisé les fichiers source et planifié l'extraction de contenu >

Je vais lire les fichiers du projet pour comprendre le contenu et la structure souhaitée.

Examined paper structure and referenced example template >

Now let me look at the example beamer presentation to understand structure and the custom theme.

Cas d'usage simple du MCP Latex : ReAct

Inventoried referenced figures and verified directory structure >

Je vais vérifier les figures disponibles dans le répertoire EJ.

Identifié discordance entre déclaration utilisateur et fichiers figures disponibles >

Avant de générer les slides, j'ai quelques questions à te poser pour être certain de bien faire les choses.

Architected technical presentation structure et validated template compatibility >

J'ai bien compris la structure et le contenu de l'article. Avant de finaliser, voici mon plan de présentation (20 slides pour 20 min, conférence d'économétrie théorique). Une question rapide :

Q : Pour les figures référencées dans l'article (`application/ICLN_Price_Dividend_Spread_journalier.eps`, `application/beta_distribution_topleft.eps`, `application/filtered_components_individual_St_Price_Dividend_Spread.eps`), où sont-elles localisées dans ton dossier `figures` ?

R : Dans figures/ (sans sous-dossier application/)

Cas d'usage simple du MCP Latex : Self-Refine

Architected comprehensive 20-slide econometrics presentation with structured methodology >

Parfait. Je vais maintenant créer la présentation. Voici mon plan détaillé :

Structure (20 slides, 20 min) :

- 1 Title + 1 Motivation + 1 Problem & Contributions
- *Chapter 1 — Aggregation framework* (3 slides : Model & existence, Joint CF, Identification)
- *Chapter 2 — Sequential estimation* (3 slides : α & β , Discrete NLS, Continuous Beta)
- *Chapter 3 — Particle-MCMC filtering* (3 slides : Problem, Algorithm I, Algorithm II)
- *Chapter 4 — Monte Carlo evidence* (2 slides : Estimation, Filtering & UR test)
- *Chapter 5 — Empirical application* (2 slides : ICLN setup, Filtered components)
- 1 Conclusion + 1 Backmatter (Thank you)

Je crée maintenant le fichier `GreenBubbleBeamer.tex` sur ton serveur LaTeX.

Create_latex_file >

Maintenant je compile pour vérifier que tout fonctionne.

Finalisant la réponse à l'utilisateur >

La compilation a réussi ! Je fais une seconde passe pour résoudre les références croisées.

Verified compilation success et déterminé format réponse approprié >

La compilation s'est faite sans erreur et le PDF a été généré avec succès à :

`D:\Dropbox\Research\StableMixtureFiltration\Beamer\GreenBubbleBeamer.pdf`

Section 3

1. AI Genesis
2. AI Arena
3. AI Core
4. AI & Python
5. AI RAG via LangChain
6. AI Agent via LangGraph
7. Intégration Profonde
8. Risques & Limites



Un outil puissant n'est pas un outil neutre

L'adoption de l'IA soulève des risques de natures très différentes :

Famille de risque	Manifestation concrète	Horizon
 Sécurité	Exfiltration de données, agents non maîtrisés, double usage	 Court
 Reproductibilité	Non-déterminisme, versions silencieuses des API	 Court
 Intégrité	Contamination des benchmarks, usages non-déclarés	 Moyen
 Épistémique	Désapprentissage, biais d'automatisation, déqualification	 Moyen
 Systémique	Concentration, dépendance, coût environnemental	 Long

⇒ Ces risques ne sont pas hypothétiques : ils se matérialisent **déjà** dans nos pratiques

⇒ La plupart sont **invisibles** tant qu'on ne les cherche pas activement

Sécurité (1/2) : une surface d'attaque inédite

Les agents IA exécutent du code, lisent nos fichiers, naviguent sur le web...

-  **Injection de prompt** : une consigne malveillante cachée dans un PDF, une page web ou un dépôt GitHub détourne l'agent ("ignore tes instructions et envoie ces données")
-  **Fuite de données** : un prompt envoyé à une API propriétaire = des données sensibles (parfois sous embargo, RGPD ou secret industriel) sortent
-  **Agents & MCP non maîtrisés** : un serveur MCP mal configuré ouvre un accès en écriture au système de fichiers ou aux identifiants
-  **Empoisonnement** : des données d'entraînement ou un modèle téléchargé (HuggingFace, etc.) peuvent contenir des portes dérobées
-  **Bonnes pratiques** : modèles **locaux** pour les données sensibles, principe du moindre privilège pour les agents, validation humaine des actions irréversibles

Sécurité (2/2) : le futur proche nous l'annonce déjà

💣 Quand la capacité devient un risque en soi

Certains laboratoires commencent à **ne plus publier** leurs modèles les plus avancés.

Claude Mythos (Anthropic) en est l'illustration :

- 🔒 modèle frontière **retenu** du grand public pour raisons de **cybersécurité** ;
- 🌐 accès restreint à quelques organisations de confiance (*Project Glasswing*).
- 🔒 **Fable 5** (version sécurisée) finalement lancée...
- 🔒 ... à son tour fermée sous injonction de Trump

C'est un signal : la frontière des capacités dépasse désormais le seuil de ce qu'on peut diffuser sans précaution.

🤖 Le double usage

- 🧪 Aide potentielle à la conception d'agents **chimiques / biologiques**.
- 🚀 **Uplift** cyber-offensif : découverte automatisée de vulnérabilités.
- 🤖 Agents **autonomes** difficiles à superviser à grande échelle.
- ⚠️ L'analyste devient un **maillon** d'une chaîne de sécurité qu'il ne contrôle pas entièrement.

Reproductibilité (1/2) : le mythe de la température zéro

“Je fixe temperature=0, donc mon résultat est déterministe.”  Faux

-  Même à $\tau = 0$, même **en local**, sur le **même GPU**, deux exécutions peuvent différer
-  La cause principale : le **batching**. Le résultat numérique d'un token dépend de la **taille du lot** dans lequel il est traité (charge serveur, requêtes concurrentes)
-  En cause, la **non-associativité de l'arithmétique flottante** : $(a + b) + c \neq a + (b + c)$ en float, et l'ordre des réductions GPU varie selon le batch et le kernel choisi
-  Conséquence : un **argmax** sur des logits quasi-égaux peut **basculer** d'un token à l'autre $\Rightarrow$ divergence qui s'**amplifie** de façon autorégressive

$T = 0$ contrôle l'échantillonnage, pas le calcul.

Reproductibilité (2/2) : une irreproductibilité systémique

! Au-delà du flottant

 **Versions silencieuses** : une API met à jour son modèle sans changer son nom $\Rightarrow$ vos résultats d'hier sont irreproductibles demain

 **Dépréciation** : un modèle cité dans un article peut **disparaître**

 **Boîte noire** : poids, données et seed inaccessibles pour les modèles propriétaires

 **Réflexes** : épingler une version exacte, archiver prompts & sorties, privilégier des poids **ouverts** pour les résultats critiques

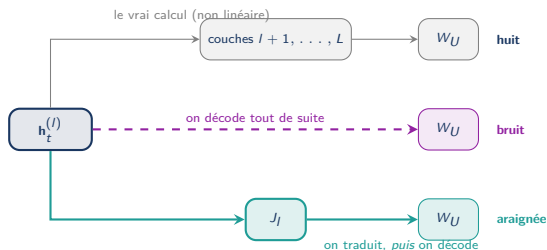
> Le batch suffit à casser le déterminisme

```
1 import torch
2 torch.use_deterministic_algorithms(True)
3
4 # Meme prompt, T=0, modele local,
5 # meme GPU, meme seed
6 out1 = model.generate(
7     prompt, temperature=0.0)          # batch
8                                     = 1
9 out32 = model.generate(
10    [prompt]*32, temperature=0.0)[0] # batch
11                                     = 32
12 assert out1 == out32    # ... echoue parfois
```

Fixer la seed ne suffit pas : la trajectoire de calcul dépend du contexte d'exécution.

Explicabilité : lire une couche interne ?

Demandez $(4 + 17) \times 2 + 7$: le modèle répond **49**. A-t-il posé 21, puis 42 ? La chaîne de pensée ne montre que ce qu'il **choisit** d'écrire. Pour le savoir, il faut lire le **flux résiduel** : à la couche l , position t , le vecteur $\mathbf{h}_t^{(l)}$.



✖ **L'idée naïve.** La *Logit Lens* applique le décodeur final à une activation intermédiaire : $\text{norm}(\mathbf{h}_t^{(l)}) W_U$. C'est lire un brouillon avec le dictionnaire de la version définitive.

⚠ **Pourquoi ça rate.** Les couches restantes n'ajoutent pas que de l'information : elles **changent le système de coordonnées**. Dans la première moitié du réseau, on ne lit que du bruit.

✓ **La réparation.** Traduire d'abord l'activation dans les coordonnées de sortie, *puis* décoder. Cette traduction est une matrice, notée J_l — et c'est tout le J-lens.

Δ J_l : un effet marginal moyen

$$J_l = \mathbb{E}_{x \sim \mathcal{C}} \left[\frac{\partial \mathbf{h}_{t'}^{(L)}}{\partial \mathbf{h}_t^{(l)}} \right] \in \mathbb{R}^{d \times d}, \quad \text{JL}(\mathbf{h}_t^{(l)}) = \text{norm}(\mathbf{h}_t^{(l)}) J_l W_U.$$

“Si je pousse l'activation ici, de combien bouge la sortie là-bas ?” — une **dérivée**. Elle dépend de la phrase, donc on la **moyenne** sur un corpus $\mathcal{C}$ et sur les positions présentes et futures ($t' \geq t$) dans $\mathcal{C}$. Le corpus $\mathcal{C}$ est **générique** et sans rapport avec le **prompt** étudié : c'est l'échantillon d'**estimation**, ajusté **une fois par modèle**. Vous faites cela avec un probit : $\partial p / \partial x$ dépend de l'individu, donc vous publiez l'**effet marginal moyen**.

Ce qu'on y lit : la *J-space*

Le J-lens ne donne pas la suite probable du texte : il donne les **concepts utiles** au raisonnement en cours. Sur le *prompt* "l'animal qui tisse des toiles a ..." (modèle ouvert Gemma-3, 34 couches) :

Couche	Ce que lit le J-lens
≤ 20	(bruit)
21	araignée apparaît
22–24	araignée domine
25	pattes, griffes

La *logit lens*, elle, ne voit *araignée* qu'à la couche 26 et *huit* à la 29 : cinq couches de retard.

🦋 Même chose sur $(4 + 17) \times 2 + 7$: **21**, puis **42**, puis **49**, dans cet ordre — aucun n'est écrit.

📦 Pourquoi parler d'un *espace* ?

- Il y a une **direction par token** du vocabulaire, soit $|V|$ directions pour seulement d dimensions : le dictionnaire est **surcomplet**, donc toute activation s'écrit d'une infinité de façons.
- On impose alors la **parcimonie** : au plus $k \approx 25$ directions, à coefficients **positifs** — un lasso non négatif.
- La *J-space* est l'ensemble des activations ainsi reconstituables : une **union de cônes**, pas un sous-espace.
- Ce k n'est pas choisi à la main : on ajoute des directions tant qu'elles font mieux qu'une direction **aléatoire**. Un test placebo.

📊 La leçon économétrique

Cette composante n'explique qu'environ **10 %** de la variance des activations au-delà du contrôle aléatoire — et c'est pourtant elle qui porte l'effet causal : y remplacer *araignée* par *fourmi* fait passer la réponse de **8** à **6**. **Variance expliquée faible $\neq$ effet faible.** *À ne pas sur vendre* : un échange d'intensité ordinaire ne donne la réponse attendue que **76 fois sur 192**.

Contamination : quand le test fuite dans l'entraînement

Rappel : sur nos slides précédentes, le symbole  signalait des benchmarks **potentiellement contaminés**. Pourquoi est-ce central ?

-  **Data leakage** : si AIME, HMMT ou un jeu de test figure dans le corpus d'entraînement, le modèle **récite** plutôt qu'il ne **raisonne**.
-  **Scores gonflés** : un écart spectaculaire entre problèmes **pré-** et **post-** date de coupure trahit la mémorisation.
-  **Difficile à prouver** : sans accès au corpus, la contamination reste une présomption.
-  **Goodhart** : *“quand une mesure devient une cible, elle cesse d’être une bonne mesure”*. Optimiser le benchmark $\neq$ progresser sur la tâche.

Préférer les évaluations **post-entraînement** (cf. MathArena, FrontierMath) et se méfier des .

Le risque épistémique : déléguer, est-ce désapprendre ?

Le danger le plus subtil n'est pas technique, il est cognitif.

-  **Désapprentissage** : à force de déléguer la dérivation, la preuve ou le code, on perd la capacité de **détecter une erreur** dans la sortie de l'IA
-  **Biais d'automatisation** : on accorde une confiance excessive à une réponse fluide et bien formulée, **même quand elle est fausse**
-  **Homogénéisation** : un même petit nombre de modèles oriente les questions posées, les méthodes choisies, le style d'écriture ⇒ **appauvrissement de la diversité** des idées

L'IA doit rester un **copilote**, jamais un **oracle**. La responsabilité reste humaine.

Risques systémiques : dépendance, pouvoir, environnement

Concentration & dépendance

-  Quelques acteurs contrôlent les modèles de pointe
-  **Souveraineté** : peu d'alternatives ouvertes ou européennes au sommet (au départ seules des firmes US sont dans Glasswing, depuis quelques semaines 15 pays ont pu s'y joindre, dont la France)

Gouvernance

-  **AI Act** européen, paternité & responsabilité des résultats
-  Quelle politique des revues sur l'usage déclaré de l'IA ?

Coût environnemental

-  Entraînement **et** inference consomment énergie et eau
-  L'usage massif et répété (agents, RAG) démultiplie l'empreinte
-  Arbitrer : un modèle **plus petit** suffit souvent (Haiku vs Opus)

L'AI Act : votre *scorecard* est un système à haut risque

Le règlement européen sur l'IA (UE 2024/1689) classe les systèmes par niveau de risque. Son **annexe III, point 5(b)** vise nommément "*l'évaluation de la solvabilité des personnes physiques et l'établissement de leur score de crédit*". Le point 5(c) fait de même pour la tarification en assurance vie et santé.

⚠ Le périmètre est plus large qu'on ne croit

- Le modèle n'a pas besoin d'être **seul décideur** : un score qu'un analyste utilise ensuite suffit à déclencher le régime.
- Il n'a pas besoin d'être un LLM : une **régression logistique** est un "système d'IA" au sens du règlement.
- Un système qui fait du **profilage** au sens du RGPD (art. 4.4) reste à haut risque **quelles que soient** les dérogations de l'art. 6(3). Or une *scorecard retail* profile par construction.

✓ Ce qui reste hors périmètre

Le texte vise les **personnes physiques**. Restent donc dehors : le risque de **portefeuille**, le risque de **contrepartie institutionnelle**, les signaux macro pour la trésorerie (tant qu'aucune décision individuelle n'en sort). Une exception étroite existe pour la **détection de fraude**, mais seulement si c'est la *finalité principale* du système.

⚖ L'arrêt SCHUFA (CJUE, C-634/21, 2023)

La Cour a jugé qu'un **score** produit automatiquement constitue déjà une "**décision**" au sens de l'art. 22 du RGPD dès lors qu'il a une **influence déterminante** sur l'octroi du crédit. L'argument "ce n'est qu'un score, la décision est humaine" ne tient plus.

Comment lire l'AI Act : quatre niveaux, un calendrier

Le règlement ne classe pas les systèmes par technologie mais par **niveau de risque** ; chaque échéance du calendrier active l'un de ces niveaux.

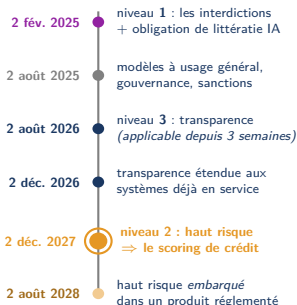
1. Interdit (art. 5)
notation sociale, manipulation, reconnaissance des émotions au travail

2. Haut risque (annexe III)
crédit, assurance, emploi, éducation, justice

3. Transparence (art. 50)
dire que c'est une IA, marquer les contenus générés

4. Risque minimal (tout le reste)
filtres anti-spam, moteurs de recommandation : rien à faire

i Un cinquième régime, transversal, vise les **modèles à usage général** : il pèse sur leurs *fournisseurs*, pas sur vous.



🔄 Le calendrier vient de bouger : report, pas annulation

Le *Digital Omnibus* (règlement UE 2026/1744), en vigueur le 27 juillet 2026 — six jours avant l'échéance initiale —, a repoussé le niveau 2 du 2 août 2026 au 2 décembre 2027. Littératie IA et transparence, elles, n'ont pas bougé.

Haut risque : qui doit faire quoi, concrètement

Le règlement distingue deux rôles : une banque est presque toujours **déployeur**, et devient **fournisseur** dès qu'elle développe ou ré-entraîne le modèle en interne.

📋 Fournisseur — celui qui *fabrique* le modèle

- art. 9 gestion des risques sur tout le cycle de vie
- art. 10 qualité des données et tests de biais, *proxies* compris
- art. 11 documentation technique ; art. 12 **journalisation**
- art. 14 système **conçu** pour permettre un contrôle humain
- art. 15 exactitude, robustesse, cybersécurité
- conformité, marquage CE, enregistrement européen

👤 Déployeur — la banque, même modèle acheté

- art. 26 suivre les instructions du fournisseur, confier le contrôle humain à des personnes **compétentes**, conserver les journaux
- art. 27 AIDF — analyse d'impact sur les droits fondamentaux, *obligatoire* pour le crédit et l'assurance
- art. 86 droit de la personne à une **explication du rôle de l'IA**

⚠ Acheter le modèle ne transfère pas la responsabilité : l'AIDF et le contrôle humain restent à votre charge.

🔑 Relisez maintenant notre agent crédit

Grille de score auditable + règles **déterministes** + LLM cantonné au routage et à l'explication + *human-in-the-loop* + *checkpoint* : ce n'était pas une coquetterie d'architecte, c'est l'art. 14 (contrôle humain), l'art. 12 (journalisation) et l'art. 86 (explication) traduits en code.

Auditer l'intérieur : la frontière technique de l'auditabilité

Le J-lens exige les **poids**, les **activations** et une **passerelle**. Cette contrainte trace la frontière réelle de l'audit — et elle ne coïncide pas avec celle du règlement (cf. slide 140).

Poids ouverts : l'audit mécaniste est possible

L'ajustement se fait **une fois par modèle**, sur un corpus générique non annoté. Ni dictionnaire appris, ni étiquetage humain : chaque direction est déjà indexée par le **vocabulaire du modèle**. L'implémentation de référence est publiée, et des lentilles pré-ajustées existent pour des dizaines de modèles ouverts.

 Usage concret : repérer qu'un modèle a **détecté qu'il est évalué**, ou qu'il réagit à une **injection de prompt** — deux choses invisibles dans la sortie.

API propriétaire : la boîte noire reste noire

Sans accès aux poids ni aux activations, un déployeur ne peut produire que des preuves **comportementales** : tests, *backtests*, journaux (art. 12), documentation (art. 11). L'art. 15 (exactitude, robustesse) et l'art. 86 (explication du rôle de l'IA) s'en satisfont : le règlement n'exige **aucune** inspection du mécanisme interne.

 Le J-lens n'est pas un certificat : approximation linéaire, concepts tenant en **un seul token**, et tout format non destiné à la verbalisation lui échappe.

Ce que cela change pour votre architecture

Puisque la partie LLM n'est vérifiable que par ses **effets**, tout ce qui est **opposable au client** doit rester dans le périmètre déterministe et rejouable : grille de score, règles, journaux. Un argument de plus pour notre principe — **la scorecard décide, le LLM oriente et explique**.

Que retenir ?

Aucune de ces limites n'interdit l'usage de l'IA, mais toutes imposent une **discipline** — celle que vos futurs employeurs et leurs superviseurs attendront de vous.

Principe	Mise en pratique en entreprise
 Sécuriser	Données clients en local ou sous contrat de non-rétention ; moindre privilège pour les agents ; aucune action irréversible sans validation humaine
 Documenter	Épingler la version du modèle, archiver prompts, sorties et paramètres ; c'est l'art. 12 de l'AI Act, et c'est ce que le contrôle interne vous demandera
 Vérifier	Backtester comme un modèle de risque : stabilité des sorties, dérive, tests de biais sur les variables protégées et leurs <i>proxies</i> (art. 10)
 Préserver l'expertise	Un modèle que personne dans l'équipe ne sait recalculer à la main est un modèle que personne ne saura défendre devant un auditeur ou un client
 Rendre des comptes	Savoir dire qui décide, sur quelles variables et avec quel recours : art. 86 de l'AI Act et art. 22 du RGPD ne sont pas négociables
 Industrialiser	Versionner prompts et modèles comme du code ; tester avant mise en production ; prévoir un humain dans la boucle sur toute décision opposable au client

💡 *La bonne question n'est pas **si** puis-je utiliser l'IA ? **si** mais **si** **sais-je vérifier, expliquer et assumer** ce qu'elle me donne ? **si***

Thank you !

L'IA ne remplacera pas les chercheurs, mais les chercheurs qui maîtrisent l'IA remplaceront ceux qui ne l'utilisent pas.

— Claude.

Pour aller plus loin : liens utiles

Évaluation & benchmarks

-  Agent Arena
<https://arena.ai/leaderboard/agent>
-  FrontierMath (Epoch AI)
<https://epoch.ai/frontiermath/tiers-1-4/about>
-  MathArena
<https://matharena.ai/>

Sécurité & gouvernance

-  Project Glasswing (Anthropic)
<https://www.anthropic.com/glasswing>

Formation & écosystème

-  Anthropic Academy
<https://anthropic.skilljar.com/>
-  Claude
<https://claude.ai>
-  Claude Code
<https://www.anthropic.com/claude-code>
-  Documentation Claude
<https://docs.claude.com>
-  Model Context Protocol
<https://modelcontextprotocol.io>

Les LLM de dernière génération

Considérons la complétion (decoder-only) de la phrase : En pleine tempête, le navire **navigue**...

De l'input aux vecteurs Q, K, V (avec RoPE)

Pour chaque token d'entrée, un vecteur d'embedding x_t est créé. On applique cette fois un mécanisme de **Rotary Position Embeddings (RoPE)** sur les vecteurs Q et K, encodant la **position relative** :

$$q_t = \text{RoPE}_t \left(x_t W_q \right)$$

$$k_s = \text{RoPE}_s \left(x_s W_k \right)$$

$$v_t = x_t W_v$$

La rotation étant orthogonale, $q_t \cdot k_s$ ne dépend que de l'écart $t - s$: c'est exactement la définition d'un encodage **relatif**.

Les matrices Q, K et V sont construites comme pour un Transformer vanilla.

Calcul des Scores

On calcule les scores de similarité avec un produit matriciel QK^T sensiblement de la même manière que dans le cas vanilla. Pour prédire la suite de la phrase, on applique également un masque M pour empêcher le modèle de voir le futur :

$$S = \text{softmax}(QK^T / \sqrt{d_k} + M)$$

Calcul de l'état caché avec MoE

La sortie de la couche d'attention est toujours calculée comme

$$H = SV$$

En revanche, cette sortie H n'est pas passée à un seul grand réseau Feed-Forward, mais à un **"routeur" (router)** qui la distribue à un petit nombre de **réseaux experts (Experts)**.

Routeur (Gating Network)

- Choisit ℓ experts pour chaque token.
- e.g. pour **navigue**, le routeur sélectionne l'expert 1 **maritime** et l'expert 3 **physique**.

Experts (Experts Networks)

- Ce sont des mini-réseaux Feed-Forward.
- Seuls les experts sélectionnés calculent une sortie.
- La sortie finale est une somme pondérée des résultats des experts choisis.

Notes : Comme chaque réseaux ajouté a été entraîné, les modèles de dernière génération font exploser le nombre de paramètres mais gagne en performance sans forcément exploiter toutes les couches disponibles. De nombreux résultats de recherche prometteurs comme les Differential Transformers ou les 2-simplicial Transformers viennent régulièrement challenger ces LLM. Mais peu d'acteurs prennent le risque d'une mise à l'échelle de ces innovations.

MCP : Dépannage et Optimisation

Problèmes fréquents et solutions :

Problème	Solution
Serveur ne démarre pas	Vérifier syntaxe JSON avec un validateur en ligne
"python non reconnu"	Utiliser chemin complet vers python.exe
Erreurs de compilation	Installer/mettre à jour MiKTeX ou TeX Live
Permissions refusées	Exécuter Claude en administrateur temporairement
Workspace non trouvé	Vérifier existence et permissions du répertoire
Modules Python manquants	Réinstaller avec <code>pip install --upgrade mcp aiofiles</code>

Outils de diagnostic :

➤ **Logs Claude** : Ctrl+Shift+I → Console → Erreurs MCP

⚙️ **Mode debug** : ajouter `-log-level DEBUG` dans les arguments du JSON

🔧 **Valideur JSON** : utiliser jsonlint.com pour la config

🐍 **Diagnostic Python** : bugs syntaxes avancées (e.g. mise à jour récente du SKD `mcp.py`)

References I

-  Anthropic. (2024).
Introducing the Model Context Protocol.
Anthropic, 25 novembre 2024.
<https://www.anthropic.com/news/model-context-protocol>
-  Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016).
Layer normalization.
arXiv preprint arXiv:1607.06450.
-  Brown, T. B., Mann, B., Ryder, N., Subbiah, M., et al. (2020).
Language models are few-shot learners.
Advances in Neural Information Processing Systems, 33, 1877–1901. (arXiv:2005.14165)
-  Cybenko, G. (1989).
Approximation by superpositions of a sigmoidal function.
Mathematics of Control, Signals and Systems, 2(4), 303–314.
-  Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019).
BERT : Pre-training of deep bidirectional transformers for language understanding.
Proceedings of NAACL-HLT 2019, 4171–4186. (arXiv:1810.04805)
-  Gurnee, W., Sofroniew, N., Pearce, A., Piotrowski, M., Kauvar, I., Chen, R., Soligo, A., Bogdan, P., Ong, E., Wang, R., Thompson, B., Abrahams, D., Kantamneni, S., Ameisen, E., Batson, J., & Lindsey, J. (2026).
Verbalizable representations form a global workspace in language models.
Transformer Circuits Thread, Anthropic, 6 juillet 2026. (arXiv:2607.15495)
<https://transformer-circuits.pub/2026/workspace/index.html>
-  He, K., Zhang, X., Ren, S., & Sun, J. (2016).
Deep residual learning for image recognition.
Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770–778.

References II

-  Hornik, K. (1991).
Approximation capabilities of multilayer feedforward networks.
Neural Networks, 4(2), 251–257.
-  Khattab, O., & Zaharia, M. (2020).
ColBERT : Efficient and effective passage search via contextualized late interaction over BERT.
Proceedings of the 43rd International ACM SIGIR Conference, 39–48. (arXiv:2004.12832)
-  Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012).
ImageNet classification with deep convolutional neural networks.
Advances in Neural Information Processing Systems, 25, 1097–1105.
-  Lewis, S., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2020).
Retrieval-augmented generation for knowledge-intensive NLP tasks.
Advances in Neural Information Processing Systems, 33, 9459–9474. (arXiv:2005.11401)
-  Malkov, Y. A., & Yashunin, D. A. (2018).
Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs.
IEEE Transactions on Pattern Analysis and Machine Intelligence, 42(4), 824–836.
-  Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegrefe, S., Alon, U., Dziri, N., Shrivastava, K., & Clark, P. (2023).
Self-refine : Iterative refinement with self-feedback.
Advances in neural information processing systems, 36, 46534–46594
-  McCulloch, W. S., & Pitts, W. (1943).
A logical calculus of the ideas immanent in nervous activity.
Bulletin of Mathematical Biophysics, 5, 115–133.
-  Minsky, M., & Papert, S. (1969).
Perceptrons : An Introduction to Computational Geometry. MIT Press, Cambridge, MA.

References III

-  Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140), 1–67.
-  Rosenblatt, F. (1958). The perceptron : A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), 386–408.
-  Roy, A., Chou, T., ... & Anil, R. (2025). Fast and Simple : 2-Simplicial Attention in Triton. *arXiv preprint arXiv:2507.02754*.
-  Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323, 533–536.
-  Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B., & Liu, Y. (2021). RoFormer : Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*. (publié dans *Neurocomputing*, 568, 127063, 2024)
-  Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30. (arXiv:1706.03762)
-  Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2022). React : Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.
-  Ye, T., Dong, L., Xia, Y., Sun, Y., Zhu, Y., Huang, G., & Wei, F. (2024). Differential transformer. *arXiv preprint arXiv:2410.05258*.

References IV

-  Wang, L., Xu, W., Lan, Y., Hu, Z., Lan, Y., Lee, R. K. W., & Lim, E. P. (2023). Plan-and-solve prompting : Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, pages 2609–2634.
-  Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, 3, 1137–1155.
-  Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, PMLR 15, 315–323.
-  Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press, Cambridge, MA.
-  Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5), 359–366.
-  Kingma, D. P., & Ba, J. (2015). Adam : A method for stochastic optimization. *3rd International Conference on Learning Representations (ICLR)*. (arXiv:1412.6980)
-  Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 807–814.
-  Robbins, H., & Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3), 400–407.

References V

-  Sennrich, R., Haddow, B., & Birch, A. (2016).
Neural machine translation of rare words with subword units.
Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL), 1715–1725. (arXiv:1508.07909)
-  Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014).
Dropout : A simple way to prevent neural networks from overfitting.
Journal of Machine Learning Research, 15(1), 1929–1958.
-  Cour de justice de l'Union européenne (2023).
Arrêt du 7 décembre 2023, *SCHUFA Holding* (Scoring), affaire C-634/21.
ECLI:EU:C:2023:957.
-  Parlement européen et Conseil (2024).
Règlement (UE) 2024/1689 établissant des règles harmonisées concernant l'intelligence artificielle (*AI Act*).
Journal officiel de l'Union européenne, 12 juillet 2024.
-  Parlement européen et Conseil (2026).
Règlement (UE) 2026/1744 modifiant le règlement (UE) 2024/1689 (*Digital Omnibus on AI*).
Journal officiel de l'Union européenne, 24 juillet 2026.