Quelques notions d’IA…

Jacques Duverger
jacques.duverger1@gmail.fr

Table des matières

Notes préliminaires :

*Le terme **IA** (Intelligence artificielle, AI en anglais) est trés largement utilisé dans le public et dans l'industrie, je l'utiliserai donc dans ce document, toutefois, il serait préférable d'utiliser le terme anglais de **machine learning** parce qu'il n'y a pas d'intelligence dans les outils de l'IA mais plutôt de l'acquisition et de la gestion de connaissances.*

L'objectif de ce document est de montrer que l'IA, bien qu'un peu compliqué, ce n'est pas de la magie !

Les annexes techniques sont a disposition du lecteur curieux, leur lecture est tout a fait facultative. Elles ont pour objectif de montrer que les techniques mises en œuvre dans l'IA sont connues. Elles sont le résultat d'échanges entre les scientifiques dans le monde entier au cours des dernières années, de publications, et sont pour la plupart dans le domaine public.
En fait, la principale difficulté pour mettre en œuvre un outil d'IA (particulièrement l'IA générative) est au niveau des ressources énormes et des infrastructures nécessaires pour faire son apprentissage et pour l'exploiter.

Le lecteur qui n'est pas intéressé par les détails techniques pourra se contenter de la lecture du

chapitre 4 « Les générateurs de textes»

qui porte sur la catégorie reine des applications IA grand public.

1. Un peu d'histoire

La préhistoire de l'IA commence entre autres avec

a) les **systèmes experts** (100 % algorithmiques),
Exemple de **règles** :

R1 : SI la voiture ne démarre pas ET les phares sont faibles
ALORS la batterie est déchargée
R2 : SI la voiture ne démarre pas ET les phares sont normaux
ALORS problème de démarreur
R3 : SI la batterie est déchargée
ALORS recommander de recharger ou remplacer la batterie
R4 : SI problème de démarreur
ALORS recommander de vérifier le démarreur
R5 : SI la voiture ne démarre pas ET il n'y a aucun bruit
ALORS suspecter un problème électrique.

b) les **moteurs d'inférence** (langage PROLOG….)
Exemple de faits et règles en **PROLOG** :

```
% Faits
aime(paul, pizza).
aime(marie, pizza).
aime(julie, chocolat).
% Règle : deux personnes sont amies si elles aiment la même chose
amis(X, Y) :-
aime(X, Z),
aime(Y, Z),
X \= Y.
```

Question : ?- amis(paul, marie). → réponse : **true** (vrai)

c) la **recherche opérationnelle** (optimisations basées sur des calculs matriciels….)
Exemple gérer la livraison de marchandises :

Une entreprise a :
des usines
des magasins
avec des Stocks disponibles dans les usines
des besoins dans les magasins
des couts de transport par unité d'une usine a un magasin

L'obectif est de **Minimiser** le coût total de transport

Le véritable ancêtre de l'IA moderne est le **perceptron** (modèle de neurone artificiel simple)

Dans l'IA moderne, on peut distinguer plusieurs catégories d'outils pour des besoins différents mais la plupart sont basées sur le **deep learning** avec la mise en œuvre de **réseaux de neurones profonds** (multi couches).
Ils utilisent des modèles différents :

- Les outils de classification
  - reconnaissance d'objets dans une image
  - traitement d'images médicales
  - reconnaissance de caractères (OCR)
  - ….

- Les outils de génération d'image

- Les outils de génération de texte

Certain de ces outils peuvent être sollicités via un interface commun (chat, API…), le modèle sous-jacent est dit **multimodal**.

Les modèles utilisés par ces outils sont détaillés dans ce document.

Au dela des outils ci-dessus (les plus connus du grand public), il y a un certain nombre de domaines émergents qui utilisent des modèles adaptés ou complètement originaux, on peut citer pour information:

- Modèles génératifs multimodaux
  - Objectif : texte + image + vidéo + audio + 3D intégrés. Modèles utilisés : Transformers multimodaux, Diffusion models (image/vidéo), Audio-language models
- Agents autonomes & systèmes multi-agents
  - Objectif : IA capables de planifier, utiliser des outils, collaborer.
  - Modèles utilisés : LLM avec mémoire externe,Tool-use training
- Raisonnement mathématique & logique formelle
  - Objectif : démontrer des théorèmes, résoudre des problèmes olympiques, vérifier des preuves formelles.
  - Modèles utilisés : LLM avancés avec chain-of-thought, Modèles avec recherche arborescente (Tree Search / MCTS), Intégration avec proof assistants (Lean, Coq), Modèles hybrides neuro-symboliques
- Découverte de médicaments & recherche moléculaire

- Objectif : générer des molécules, prédire structures protéiques, accélérer la découverte de médicaments.
- Modèles utilisées : Graph Neural Networks (GNN)

- Robotique autonome & IA incarnée (Embodied AI)
  - Objectif : robots adaptatifs capables d'apprendre en environnement réel.
  - Modèles utilisés : Reinforcement Learning (RL), Imitation Learning, Vision-Language-Action Models (VLA), World Models

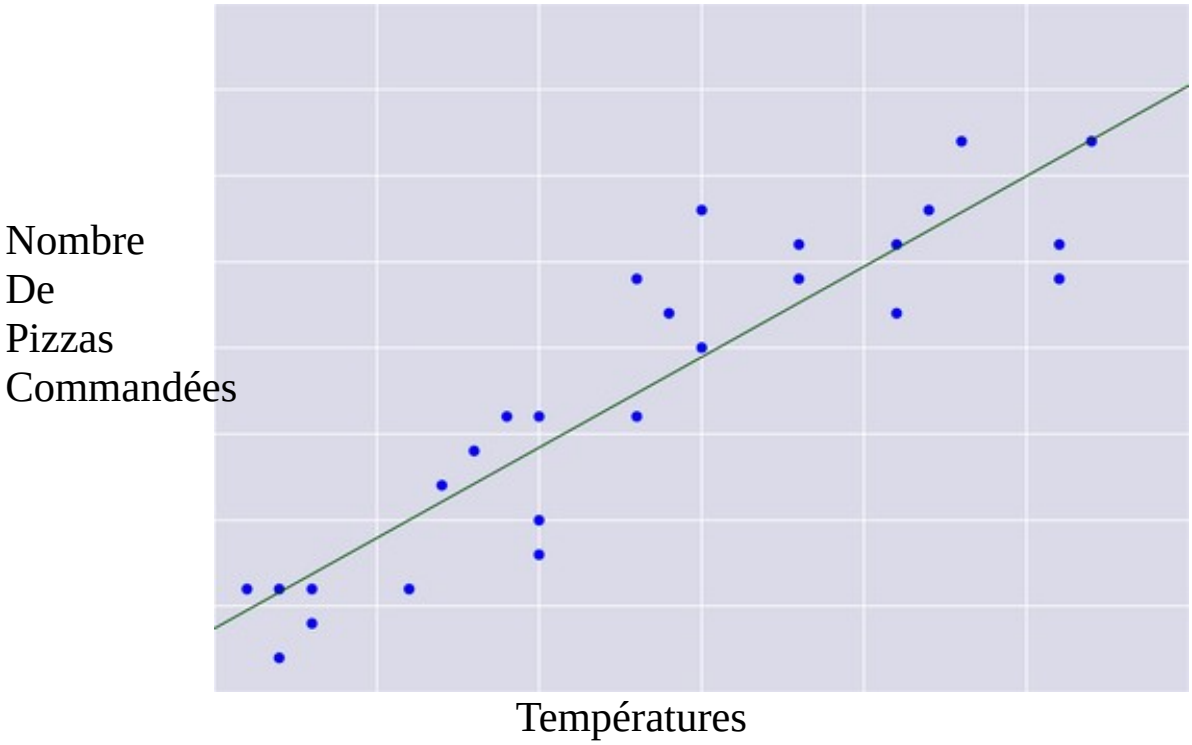
- IA scientifique (Physics AI, Climate AI, Materials AI)
  - Objectif : découvrir de nouvelles lois physiques, matériaux, modèles climatiques.
  - Modèles utilisés :Physics-Informed Neural Networks (PINNs)

- IA neuro-symbolique
  - Objectif : combiner logique formelle et réseaux neuronaux.
  - Modèles utilisés : LLM + moteurs logiques, Graph reasoning networks, Differentiable logic systems
- IA pour cybersécurité & bio-ingénierie
  - Applications : Détection d'attaques, Génération de protéines,Simulation biologique
  - Modèles utilisés : GNN, LLM spécialisés sécurité, Transformers bio-séquences, Modèles évolutionnaires

2. Les outils de classification

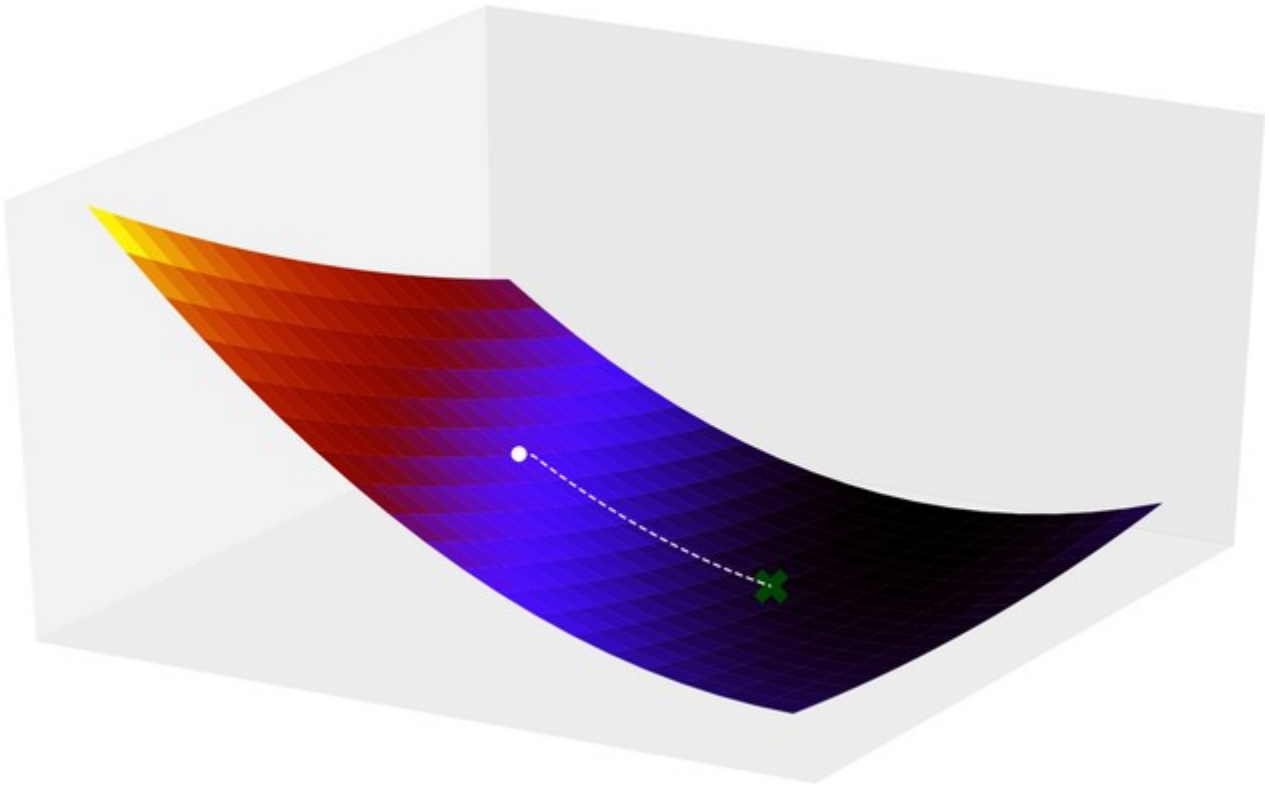
Utilisés lorsqu'on veut reconnaître un objet (voiture, animal,..), détecter une tumeur sur une image médicale (IRM) ou numériser un texte manuscrit que l'on a scanné.

Au début il y a la **droite de régression**….
Par exemple, si le nombre de pizza commandées par les clients a une pizzeria est fonction de la température extérieure, la droite permet de **prédire** les commandes en fonction de la température, elle est générée a partir des échantillons (température, nombre de pizza) prélevés sur une période significative.
Note pour les collégiens: On définit les paramètres de la droite avec la méthode des moindres carrés qui minimise les distances des points a la droite.



Cette solution est limitée a un seul paramètre et ne fonctionne que pour un phénomène **linéaire**.

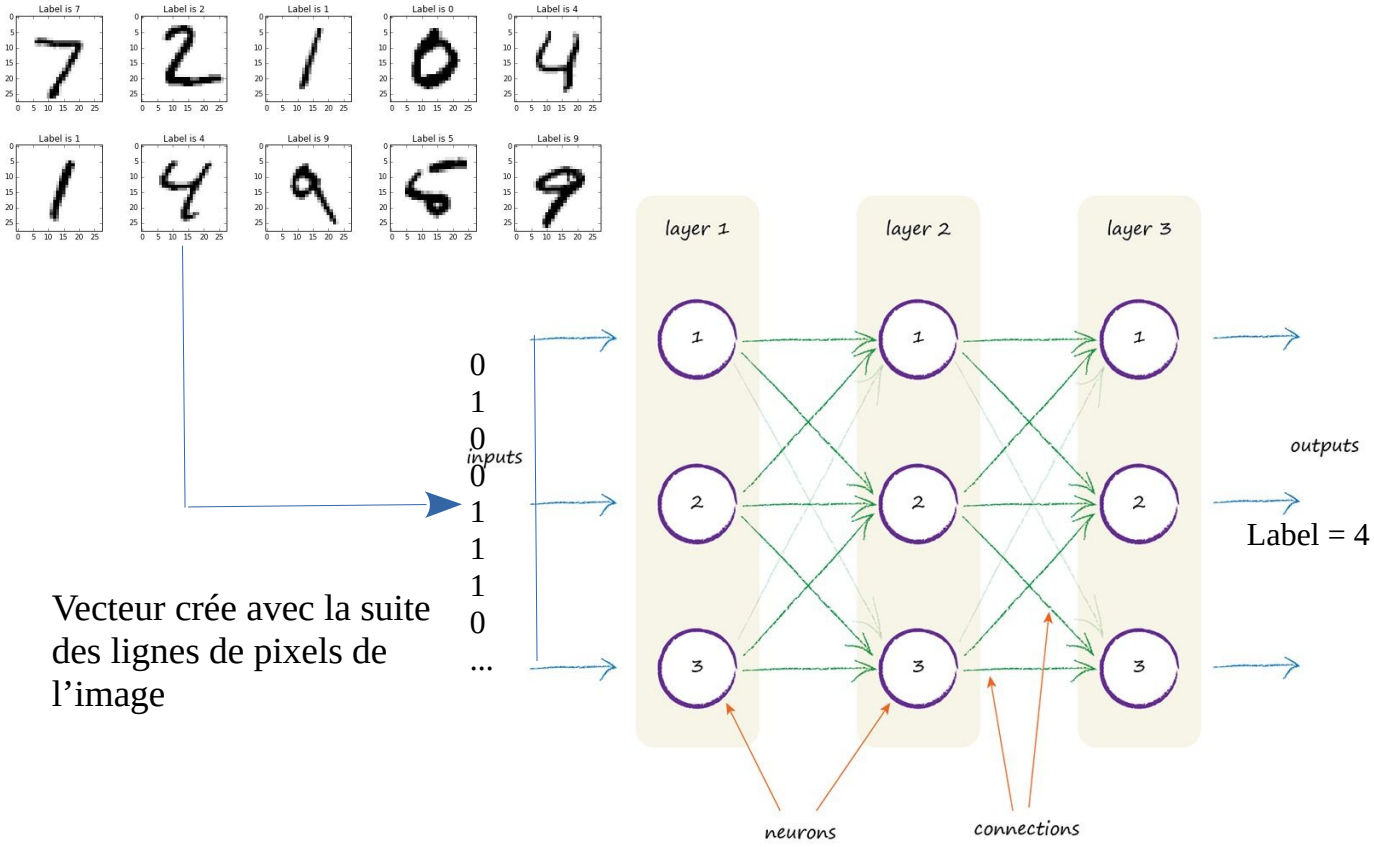
Et puis il y a les solutions de type **machine learning** ou on peut prédire une valeur en fonction de plusieurs paramètres (par exemple nombre de pizza commandées fonction de la température, des précipitations, de la date….. On peut représenter ces dépendances par une surface (un **gradient**), ici a 3 dimensions, au-delà ce ne sera plus visualisable :



Ils fonctionnent sur le même principe d'**apprentissage supervisé** et on utilise pour ce faire un **réseau de neurones** a plusieurs couches.

Par exemple on enregistre des milliers d'images de chiffres manuscrits en indiquant par chaque image quelle doit être la valeur de sortie correspondante, c'est la **labellisatio**n.

Dans la figure ci-dessous, une image du chiffre 4 est préparée – on va construire un vecteur d'entrée en ajoutant les lignes de pixel de l'image - et calculer les valeurs des nœuds du réseau qui vont donner le chiffre 4 en sortie du modèle.



Vecteur crée avec la suite des lignes de pixels de l'image

Ces systèmes de classification représentent un énorme progrès pour beaucoup de domaines de la science et du quotidien.
Elles ne demandent pas des ressources de calcul considérables et donnent des résultats spectaculaires.
L'analyse des images médicales (IRM…) est certainement une des applications les plus intéressantes, a noter que cette analyse est une aide pour le corps médical et en aucun cas un remplacement du praticien.

3. La génération d’images

Exemple d’image faite avec ChatGPT (en quelques secondes): question (**prompt**) : « *dessine des ruches au pied des montagnes avec un apiculteur* »

Il y a principalement 2 modèles utilisés :

3.1 GPT-Image : génération par diffusion (source chatGPT)

A. Pendant l'entraînement :

On prend des **millions d'images**
On ajoute progressivement du **bruit aléatoire** (elles deviennent de plus en plus floues / bruitées)
Le modèle apprend à faire l'inverse :
👉 **retirer le bruit étape par étape** pour retrouver l'image originale
En parallèle, il apprend à **lier les images à leurs descriptions textuelles**.

B. Génération d'une image a partir d'un prompt :

Le modèle :
Commence avec **du bruit pur** (une image totalement aléatoire)
Utilise le texte pour **guider le débruitage**
Améliore l'image **itération après itération**
Fait apparaître progressivement :

- les formes
- les couleurs
- les détails
- le style demandé

👉 À la fin, le bruit est devenu une image cohérente.

Ce mécanisme permet de générer des images, visages, textures ou œuvres artistiques.

3.2 Avec les GANs

Un GAN (Generative Adversarial Network) est une architecture d'apprentissage profond composée de deux réseaux de neurones entraînés simultanément.

3.3 Synthèse

La génération par diffusion est de plus en plus utilisée au détriment des GANs et est très efficace, les images générées sont très qualitatives bien que l'on puisse encore les distinguer du réel.

A noter que ces méthodes sont utilisées dans des logiciels de retouche d'image (comme Photoshop) pour des retouches ou des remplacement de partie d'image.

Les vidéos utilisent les modèles a diffusion qui intègrent le temps comme paramètre supplémentaire

L'usage de ces outils pose de quelques problèmes, on peut citer :

- génération de **fake news**, diffamation
- disparition de métiers tel que graphiste et donc perte d'expertise et d'imagination
- remplacement a moindre frais d'acteurs pat leur avatar
- manipulations divers (voir scandale des nus sur Grok)

4. Les générateurs de texte

La catégorie reine de l'IA est basée sur les **LLM** (**large langage model)**.

Le type le plus courant d'outil pour entraîner le modèle et pour l'exploiter est le **GPT** (**Generative pretrained transformer**).
Le **LLM** est entraîné grâce au **transformer** sur un (trés) grand nombre de textes (livres, article scientifiques, contenu de blogs, encyclopédies….) en plusieurs langues.
L'ensemble des textes participant a l'entraînement d'un LLM est le **corpus.**

L'entraînement par le **transformer** avec les textes du **corpus** permet de créer le **LLM** et de régler ses paramètres.
Cette entraînement se fait sur des **segment**s de texte que l'ordinateur peut absorber en une fois, le **corpus** entier sera traité dans une boucle.

Voir l'annexe A – Comment marche le transformer

En phase d'exploitation, le **prompt** (la question posée au modèle) est traitée par ce même **transformer** et génère la réponse mot par mot (en réalité **token** par token, le token peut être un mot ou un sous-ensemble de mot) en cherchant le mot (token) suivant le plus probable au sein du **LLM**. La réponse est ajoutée au prompt a chaque itération.
On peut dire de manière simplifiée qu'a chaque itération, le texte du prompt est comparé a des structures sémantiques similaires mémorisées dans le LLM, un mot (en réalité un **token)**, est alors généré et vient compléter la réponse.

La génération s’arrête soit parce que le modèle produit un token de fin (EOS), soit parce qu’une contrainte externe l’impose (limite maximum de tokens autorisée atteinte).

La définition en anglais de ce qu’est un LLM est :
A Statistical Next-Word Predictor
*que l’*on peut traduire par :
Modèle statistique de prédiction du mot suivant.
C’est bien ce qu’est UNIQUEMENT et STRICTEMENT un LLM cette définition permet de comprendre les limites du LLM, par exemple un LLM ne sait pas ni calculer (même pas une simple multiplication) ni raisonner en état. L’appel de fonctions et les agents permettent de palier certains de ces manques.

Commentaires complémentaires

- Le modèle peut être sollicite via un **API** (application programing interface) mais la plupart des utilisateurs utilisent une application de « chat » dans laquelle on pose une question.

  Voir Annexe B – Les APIs

- la qualité des réponses d’un LLM dépend de la taille du modèle (qui peut se chiffrer a plusieurs milliers de milliards de paramètres) de la taille du contexte (prompt et sortie) (400000 tokens pour chatgpt 5) mais SURTOUT de la qualité des textes d’entraînement.

- Les LLM ne sont pas **déterministes**, la même question posée plusieurs fois va générer des réponses différentes, le réglage d’un paramètre appelé **température** va permettre de brider un peu l’« imagination » du modèle.

voir ANNEXE E – Température et déterminisme

- Les LLMs ont de très fréquentes **hallucinations**, la recherche statistique ne permet pas toujours de trouver une réponse satisfaisante, une réponse farfelue est malgré tout fournie a l'utilisateur. Ceci peut être évité sur certains sujets si le texte d'entraînement dit explicitement qu'il n'y pas de réponse a cette question.

- Des filtres permettant de guider le LLM (rôles) et d'interdire certains sujets sont intégrés de plusieurs manières dans le modèle : on peut les mettre directement dans le texte d'entraînement., ou plus généralement dans la partie cachée du prompt (system prompt), dans le prompt lui même ou dans le programme en sortie de l'appel API.

- Les calculs nécessaires dans un LLM sont des opérations sur des tensors (matrices multidimensionnelles), ils sont exécutés dans des data centers avec des composants GPU (graphical processor unit) ou TPU (tensor processor unit) qui exécutent des milliers de calculs en parallèles. Une question envoyée a ChatGPT peut être équivalente en terme de calcul a des millions d'emails envoyés et a des centaine de requête google.
  NVIDIA est aujourd'hui le principal fournisseur de ces cartes GPU/TPU et le numéro 1 de la capitalisation boursière mondiale.

- Un LLM est incapable de raisonner en état. Toutefois on peut améliorer ces capacités de raisonnement par différents moyens : un des plus courant est le « Chain of Thought (CoT) », il consiste a forcer le modèle à raisonner étape par étape :
  Directive dans le prompt :« Explique ton raisonnement avant

de donner la réponse ».
d’autre méthodes consistent a passer par des outils externes (calculatrice, moteur de preuve, solveur logique, code métier...)

Voir ANNEXE D – Le LLM raisonne !

- On peut adapter un LLM a un métier ou a un besoin particulier. il s’agit alors de prendre en compte des textes (un corpus) qui concernent cette activité. Ces textes complémentaires sont ajoutés dans le prompt grâce au **RAG** (Retrieval Augmented Generation)

Voir ANNEXE C – Extension du prompt

- A ce jour de multiples LLMs sont disponibles avec des offres gratuites (chat) ou payantes (chat et APIs). On peut citer ChatGPT (openAI) , Gemini (Google), Claude (Anthropic), Llama (meta), Grok (XAI, E.Musk), Deepseek et Mistral (Le local de l’étape).
  On peut aussi utiliser des nanos LLM souvent gratuits qui peuvent être téléchargés sur un ordinateur et fonctionner de maniére autonome grace a des environnements d’exécution locaux (GPT4All, Ollama, LM Studio ...)

Les agents

Rappelons que un LLM est UNIQUEMENT et STRICTEMENT un *Modèle statistique de prédiction du mot suivant.*
Un LLM ne sait pas ni calculer (même pas une simple multiplication) ni raisonner en état.

On peut compléter le travail du LLM par l'appel de fonctions externes ou **agents**.
L'appel a ces agents peut être standardisé par l'utilisation du **MCP** (model context protocol) et permet de les solliciter de manière simple.
Par exemple la question « quelle est la température a Paris » va permettre au modèle d'appeler un **agent** spécialisé sur la météo avec les paramètres Paris et température et d'intégrer la valeur dans la réponse.

Un **agent** est en fait une fonction externe (un programme) a qui le LLM passe la main pour faire des calculs ou tout autre action programmée. Par exemple :
-envoyer un mail,
-mettre a jour le compte d'un client d'une banque,
-faire une recherche dans une feuille excel
-alimenter des informations produits depuis un catalogue web
-appeler un autre LLM….
les possibilités sont infinies

On peut aller jusqu'à la notion d'**agent autonome qui :**

- Peut fonctionner sans intervention humaine continue. Il prend ses propres décisions en fonction des informations dont il dispose.
- Collecte des informations sur son environnement via des capteurs ou des données d’entrée.
- Analyse ces informations et détermine quelles actions entreprendre pour atteindre ses objectifs.
- Interagit avec son environnement pour influencer l’état de celui-ci
- dans une certaine mesure, peut apprendre de ses expériences ou ajuster son comportement selon les changements dans l’environnement.

Ceci peut amener des résultats très intéressants et des gains de productivité dans beaucoup d’activités mais aussi alimenter les fantasmes les plus fous.

Un exemple concret récent où des fantasmes autour d’agents autonomes ont nourri des récits sensationnalistes dans l’actualité est celui de *Moltbook* : un réseau social d’agents IA qui aurait créé sa propre religion, crypto-monnaie et comportements “hostiles”. Sur Moltbook, des agents IA ont effectivement interagis et générés des contenus variés, mais ces comportements ne démontrent aucune réelle conscience ou intention propre — ce sont surtout des *imitations de phénomènes sociaux humains* reflétées par les modèles d’IA à partir des données qu’ils ont vues.

Voir ANNEXE H – Comment marchent les Agents ?

Les LLMs sont dénués d’intelligence mais fournissent cependant des résultats bluffants.
Regardons les bénéfices et tares de ces outils :

Les bénéfices….

Les LLMs sont de très bons assistant au quotidien :
« quels sont les effets secondaires de tel médicament ?»,
« comment je peux faire une superposition d’image dans tel logiciel de retouche photo ?»,
« je dois changer une roue de brouette, quel modèle prendre et ou le trouver ? ».

Ils permettent une aide au développement de programme informatique, très efficace pour coder un petit algorithme de quelques lignes, très utiles pour debbugger (corriger les erreurs).

Ils sont – ou vont devenir- une aide précieuse dans beaucoup de métiers du tertiaire: finance, compta, marketing, avocats….

Ils peuvent devenir une aide a la production ou a la maintenance en fournissant des **chatbots** dédiées a un métier, de plus ils animent les robots.

Ils sont en train de devenir des assistants efficaces pour les profession médicales.

Les inconvénients…

Il faut d'abord noter que les effets pervers des LLMs ne sont pas dus a leur machiavélisme ou a leur méchanceté (ce sont des machines), ils sont dus a l'usage de corpus tendancieux et a l'absence de filtres sécuritaire ou éthiques.

Le buzz et la peur. Il y a tous les jours des articles vantant les mérites de l'IA et ou l'on prétend qu'ils vont atteindre a court terme le niveau d'intelligence de plusieurs prix nobels réunis (*intelligence générale* et *singularité*), cela n'arrivera pas, du moins pas avec l'architecture du modèle GPT qui a des limites structurelles bien que nous n'en soyons encore qu'aux prémices de ses usages.
Ce buzz induit un enthousiasme débridé, une bulle (folie des valeurs IA sur le marché boursier et sur les investissements) qui peut exploser a tout moment ainsi que la peur de beaucoup de nos concitoyens qui se sentent dépassés.

Le « *vibe coding* » (développement rapide -et impulsif- par l'IA) est un exemple de bulle qui risque de faire quelques dégâts. On licencie les développeurs expérimentés et on fait faire du code par une IA, le résultat est souvent un code généré de mauvaise qualité.

La dégénérescence des LLMs est en cours, au fur et a mesure de l'ajout de textes redondants ou générés par l'IA (syndrome de la vache folle), les LLMs perdent rapidement de leur pertinence. Un exemple est l'ajout des contenus de Grokipéria (le wikipedia d'Elon Musk, généré par l'IA, biaisé et sectaire…).

Les pertes d'emploi et la perte d'expertise dans beaucoup de domaines.

La standardisation des pensées et pratiques. Les LLMs en cherchant les réponses les plus probables délivrent des réponses standard, bien pensantes….. quand elles ne sont pas manipulées et biaisées (Ex: posts du réseau X dans Grok)

Les datacenter, ogres nourris a l'électricité et a l'eau peuvent consommer 100 a 300MW en continu et plusieurs millions de litre d'eau par jour.

De plus en plus de sociétés se rendent compte qu'elles explosent leurs coûts lorsqu'elles déploient largement des agents IA en passant par un accès API au LLM, l'accès est alors facturé « au token ». Il y a un prix des tokens dits d'*input* – l'instruction que l'on donne au modèle –, relativement maîtrisable, puis un prix des tokens d'*output* – la réponse générée –, généralement deux à quatre fois plus élevé… et moins prévisible. Une société peut héberger un LLM dans ses propres systèmes pour palier a ce risque mais dans tous les cas, des coûts très importants sont a prendre en compte et a mettre en regard des bénéfices escomptés.

Un problème qui n'est pas directement lié au LLM mais plus a la pratique des concepteurs de LLM est celui des droits intellectuels pour les textes utilisés pour l'entraînement. L'utilisation des textes est très rarement rémunéré….. il y a actuellement des procès en cours sur ce sujet.

Les risques…

L’abêtissement générale d’une population qui commence a l’école ou chatGPT et consort remplacent les livres !!!!!
On peut citer Anthropic : “Les jeunes qui s’appuient fortement sur l’IA risquent de compromettre leur acquisition de compétences professionnelles”

la désinformation généralisée grâce aux « deep fake » qui fabriquent de fausses informations (quelquefois trés crédible) et aux réseaux sociaux qui les propagent.
On ne sait plus a qui faire confiance !

Le non déterminisme des LLM ajouté au vibe coding peut amener un développeur laxiste a programmer des actions non sécuritaires, non éthiques voir dangereuses.

La perte de contrôle par le citoyen d’une partie de sa vie ou l’IA se substitue a lui pour de multiples activités, surtout avec les agents dont l’action peut déraper s’ils ne sont pas conçus avec la sécurité et l’éthique en tête.

Le match entre les algorithmes et l'IA…

L'IA générative comme par exemple les **LLM**s de type GPT est largement connue et utilisée. Si l'on se contente de générer du texte, les résultats sont la plupart du temps correctes, et en tout cas vérifiables et modifiables par le lecteur.

Il n'en va pas de même lorsque ces outils complétés par des **agents autonomes** exécutent des taches administratives (par exemple mettre a jour un compte en banque) ou des actions sur des automatismes (par exemple actionner une pédale de frein). Le **non déterminisme** (voir annexe E) et les **hallucinations** rendent ces taches imprécises voir dangereuses.

Faire faire une action précise et motivée par une IA est comme vouloir faire labourer un champ par un cheval sauvage !

On peux prévoir un ensemble de gardes-fous (**guardrails**) tels que :

- le filtrage des entrées
- des directives au niveau du prompt,
- du code (programme) dans les agents qui valide les décisions fournis par l'IA…

mais ces solutions restent assez artisanales et compliquées a mettre en œuvre.

Lorsque l'on construit un système complexe et surtout critique, il faut vraiment choisir les taches a faire faire par l'IA d'une part et par les algorithmes (programmes) d'autre part afin que chacun travaille dans sa zone de compétence.

Un très bon exemple de cet équilibre est illustré dans la **conduite autonome** automobile :

A ce propos, deux acteurs dans ce domaine ont des approches opposées.

1. Tesla utilise un réseau de neurones massif qui apprend perception, compréhension, décision en une seule chaîne (IA de bout en bout). De plus la perception se base seulement sur des caméras.

2. Waymo a une approche modulaire

- perception (cameras+liDAR+radar + IA)
- prédiction (IA)
- planification (algorithmes)
- contrôle (algorithmes)

A ce jour (Mars 2026), les robot-taxis Waymo sont complètement opérationnels (sans chauffeur) a San Francisco (Bay Area), Phoenix, Los Angeles, Austin et Atlanta et plus a venir. La conduite autonome de Tesla dite FSD (Full self drive) est encore en tests, n'a encore reçue aucune certification et fait l'objet d'enquêtes aux états unis.

A noter que Waymo et Baidu sont au niveau 4 (sur 5) de la conduite autonome, Mercedes est au niveau 3. Ces 3 constructeurs utilisent des technologies équivalentes. Le FSD de Tesla est considéré être au niveau 2.

👉 Ceci illustre bien le fait qu'une chaîne de commande 100 % IA n'est pas prête a répondre aux impératifs de sécurité requis pour un système critique.

ANNEXE A - Le fonctionnement du transformer... ?

Attention, cette annexe est destinée aux lecteurs curieux et intrépides !

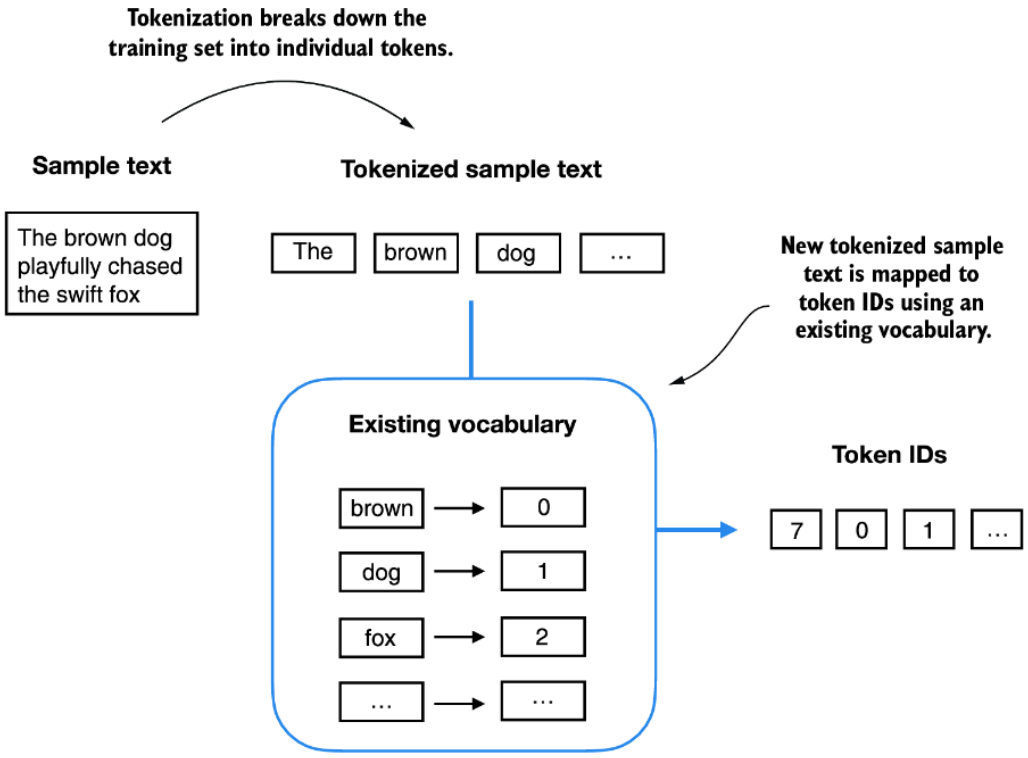
Rappel : Le type le plus courant d'outil pour 1) entraîner le modèle et 2) pour l'exploiter (inférence) est le GPT (**Generative Pretrained Transformer**).

A1. Les traitements

Le **transformer** exécute différents traitement explicités ci-dessous :

a. la **préparation**
Le texte est préparé : nettoyage (élimination des tags, des non caractères….), découpage en morceaux de taille traitables par l'ordinateur,….).

b. la **tokenisation**
On crée un dictionnaire qui affecte un identifiant numérique (le **token id**) a un mot ou un sous ensemble de mot.
L'idée est de faire des compromis entre avoir des tokens qui ont un minimum de sens et limiter le nombre de tokens pour réduire la taille du LLM et donc la complexité des calculs.
Le dictionnaire moyen d'un LLM comprend environ 50000 tokens.



c. le **token embedding**

Le **token embedding** a pour objectif de transformer chaque token (mot, sous-mot ou symbole) en un **vecteur** numérique qui capture son sens et ses relations avec les autres tokens.

Cette représentation permet au modèle de traiter le langage mathématiquement, en comprenant les **similarités sémantiques** et le **contexte**, afin d'effectuer des tâches comme la prédiction, la classification ou la génération de texte.

Les différents vecteurs de la **séquence** sont empilés pour former une matrice.

Voir ANNEXE F – Les vecteurs de Tokens embeddings

c. les **embeddings positionnels**

On ajoute les éléments de position du token dans la séquence.

d. la **self attention**
L'objectif de l'algorithme de **self-attention** est de permettre au modèle de se concentrer sur les parties les plus pertinentes d'une **séquence** pour représenter chaque token, en tenant compte des relations et dépendances avec tous les autres tokens, quelle que soit leur distance. Il permet ainsi de capturer le contexte global, d'identifier les liens importants entre les éléments et d'améliorer la compréhension du sens.

Voir ANNEXE G – La self attention

e. la **back-propagation**
La back-propagation est l'algorithme qui permet à un réseau de neurones d'apprendre en remontant l'erreur de la sortie vers les paramètres internes pour les ajuster.

f. le **feed forward**
Le feed-forward est un mini-réseau de neurones appliqué à chaque token pour ajouter de la non-linéarité et enrichir les représentations.

g. la **projection finale**
La projection finale, c'est l'étape qui transforme la représentation interne du modèle en scores pour chaque token du vocabulaire. Autrement dit, c'est ce qui permet au modèle de choisir le prochain mot.

h. le calcul de la **perte** (**loss**)
le calcul de la perte, c'est le moment où le modèle découvre s'il s'est trompé ou non. Il mesure l'écart entre ce que le modèle prédit et la bonne réponse attendue

A2. L’apprentissage

L’apprentissage d’un **LLM** lui permet d’apprendre les **schémas linguistiques (patterns)** en analysant une énorme quantité de textes.

Pour les lecteurs très curieux, l’algorithme simplifié :

1 . COLLECTER et PRÉPARER dataset
nettoyage des textes/tokenisation

2. CRÉER le dataset d’entraînement
séquences de tokens, train / validation

3. INITIALISER modèle avec des poids aléatoires

POUR chaque époque (un passage sur la totalité du dataset)

```
POUR chaque batch(lot) du dataset

    4.1 TOKEN EMBEDDING
       tokens → vecteurs

    4.2 AJOUTER position_embedding

    4.3 PASSAGE dans couches TRANSFORMER
       - SELF-ATTENTION
       - FEED FORWARD

    4.4 PRÉDIRE prochain_token
       logits → softmax

    4.5 CALCULER loss
       cross_entropy(prediction, token_reel)

    4.6 BACKPROPAGATION
       calcul gradients

    4.7 METTRE_A_JOUR poids
        optimizer (Adam, SGD)

FIN POUR batch

11. ÉVALUER sur validation_set
```

FIN POUR époque → SAUVEGARDER modèle

A3. Le traitement d’un prompt (inférence)

En **inférence**, le LLM transforme un prompt en tokens, puis utilise la self-attention et ses poids figés pour prédire et générer un token à la fois, jusqu’à produire une réponse complète.

Note : Quand on pose une question dans le prompt, le modèle ne voit pas une question.

Il voit une **suite de tokens** qui ressemble statistiquement à des millions de questions déjà vues pendant l’entraînement.

Exemple : *« Pourquoi le ciel est bleu ? »*

Dans les données d’entraînement, ce genre de séquence est **très souvent suivi** par :

- une explication
- une structure causale (“parce que…”, “cela est dû à…”)

👉 **Le modèle n’initie pas une réponse**, il **continue un schéma conversationnel**.

Pour les lecteurs très, très curieux qui ont déjà passer la 1iere étape, l’algorithme simplifié :

1. RECEVOIR prompt_utilisateur

2. TOKENISER prompt texte → tokens

3. CHARGER modèle_entrainé

BOUCLE génération_token

```
4.1 TOKEN EMBEDDING
   tokens → vecteurs

4.2 AJOUTER position_embedding

4.3 PASSAGE dans couches TRANSFORMER
   - SELF-ATTENTION
   - FEED FORWARD

4.3 CALCULER logits
   → distribution_probabilite

4.4 SÉLECTIONNER prochain_token
   (sampling / greedy)

4.5 AJOUTER token à la séquence
   tokens = tokens + prochain_token

4.6 VÉRIFIER condition_arret
    SI token_fin OU longueur_max
        SORTIR boucle
```

FIN BOUCLE

5. DETOKENISER tokens → texte_final

ANNEXE B - Les **APIs**

Le modèle peut être sollicité via un API (Application Programing Interface) . Voici un exemple en *python*:

```
url= "http://localhost:4891/v1/chat/completions"
payload = {
   "model": "mistral-7b-instruct.gguf",
   "messages": [
      {"role": "system", "content": ""},
      {"role": "user", "content": "Qui est Balzac"},
   ],
   "temperature": 0.7,
   "max_tokens": 300
}
response = requests.post(url, json=payload)
..
data = response.json()
print(data["choices"][0]["message"]["content"])
```

Réponse :

```
E:\Python\AI> & E:\Python\Python38-32\python.exe e:/Python/AI/call_gpt4all.py
Balzac is the pen name of Émile-Auguste de Balzac, a French novelist and playwright born on May 25, 1799, in Tours, France. He was one of the most influential figures in French literature during his lifetime, and his works continue to be studied and admired today.....
```

L'exemple ci-dessus a été exécuté sur un PC puissant doté d'une carte graphique GPU/TPU. Le LLM est d'origine *Mistral.ai* et tourne dans un environnement local GPT4All.

ANNEXE C - Extension du prompt, Le RAG

Il est long et compliqué d'intégrer dans le LLM des faits et régles métier (c.a.d qui concerne une activité particulière, par ex. le métier d'avocat, aussi dans la plupart des cas on utilise l'extension du prompt pour ce faire (les textes métiers sont ajoutés au prompt). On construit un **PACK** (Prompt with All Corpus Knowledge)
Il n'est pas possible d'entrer ces textes en entiers, le prompt deviendrait trop volumineux et ne pourrait pas être traité par l'ordinateur.
On met alors en œuvre des solutions de types **RAG** (Retrieval Augmented Generation) qui consiste a analyser le **corpus** métier a faire un classement par mot clefs ou sémantique (beaucoup plus efficace) et a ranger ce texte dans une base de données (simple ou sémantique).

Lors de l'usage du LLM, le prompt (la question) est analysé et en fonction des mots clefs trouvés ou de la sémantique, un ou plusieurs articles de la base de données sont récupérés et ajoutés au prompt.
Des techniques d'**embedding** (voir annexe A) sont utilisées pour le classement et la recherche sémantique.

ANNEXE D - Le LLM raisonne !

En pratique, les “capacités de raisonnement” d’un LLM viennent surtout de trois éléments :

a) **Les données d’entraînement**
On entraîne le modèle sur des exemples où des humains déroulent des étapes logiques (explications, démonstrations, chaînes de pensée, etc.). Le modèle imite ce style, il ne “raisonne” pas vraiment.

b) **La taille et l’architecture**
Plus le modèle est grand (paramètres, contexte), plus il peut capturer des patterns complexes. Ça donne l’impression de raisonnement, mais ça reste de la prédiction de texte.

c) **Le prompting et le fine-tuning**
Des techniques comme :

- *Chain-of-thought prompting* (forcer des étapes intermédiaires)
- *Instruction tuning*
- *RLHF*

aident à produire des réponses structurées, donc perçues comme “logiques”.

d) **Outils externes (optionnel)**
Si on veux obtenir de meilleurs résulats, on branche le modèle à :

- des moteurs de calcul
- des bases de connaissances

- des systèmes de vérification
  Là, on s'approche d'un vrai système de raisonnement…
  mais ce n'est plus juste le LLM.

En résumé, on ne rajoute pas du raisonnement → on entraîne un modèle à **imiter des traces de raisonnement**.

1. Le chain-of-though

Le **chain-of-thought prompting (CoT)** est une technique simple mais puissante : au lieu de demander directement la réponse, on pousse le modèle à **décomposer son raisonnement en étapes intermédiaires**.

Principe de base

Au lieu de :

"Combien font 17 × 24 ?"

On écris :

"Explique étape par étape comment calculer 17 × 24."

Le modèle va produire quelque chose comme :

- 17 × 24 = (17 × 20) + (17 × 4)
- = 340 + 68
- = 408

le modèle performe mieux quand il "écrit" son raisonnement.

Avec des variantes...

1.A Zero-shot CoT

On ajoute juste une instruction simple :

“Résous ce problème. **Explique ton raisonnement étape par étape.**”

C’est la version la plus simple — souvent déjà efficace.

1.B Few-shot CoT

On montre des exemples AVANT la question :

Q: Si j’ai 3 pommes et j’en ajoute 2, combien j’en ai ?

Raisonnement: 3 + 2 = 5

Réponse: 5

Q: Si j’ai 10 et j’en enlève 4 ?

Raisonnement: 10 - 4 = 6

Réponse: 6

👉 Le modèle imite la structure.

1.C Chain-of-thought + self-consistency

On fait générer **plusieurs raisonnements différents**, puis on prend la réponse la plus fréquente.

Exemple :

- réponse 1 → 408
- réponse 2 → 408
- réponse 3 → 392

  On garde 408.

En rélalité, le CoT n’est pas du vrai raisonnement, le modèle peut produire des étapes fausses mais convaincantes

2. L’instruction tuning

L’**instruction tuning** est une étape clé pour transformer un LLM “brut” en assistant utile. Là où le modèle de base prédit juste du texte, l’instruction tuning lui apprend à **suivre des consignes humaines**.

👉 L’instruction tuning consiste à l’entraîner sur des paires :

Instruction → Réponse attendue

Exemple :

Instruction: Traduis en français : "Hello world"

Réponse: Bonjour le monde

3. Le Reinforcement Learning from Human Feedback

Le **RLHF** est la couche qui transforme un LLM “correct” en assistant **aligné avec les préférences humaines** (utile, poli, sûr, pertinent).

Au lieu de dire au modèle *“voici la bonne réponse”*, on lui apprend :

“Parmi plusieurs réponses, lesquelles préfèrent les humains ?”

👉 On optimise **la qualité perçue**, pas juste l’exactitude.

ANNEXE E - Température et déterminisme

Lorsque le LLM (en fait le transformer) prédit le mot suivant pour un prompt donné, il vérifie, à partir de ses données d'entraînement, les probabilités du mot qui devrait venir ensuite. Le LLM sélectionne alors le mot le plus probable. Cependant, ce n'est pas tout à fait toujours le cas.

Lorsque nous envoyons un prompt à un LLM pour qu'il prédise le mot suivant, nous pouvons spécifier d'autres paramètres afin de lui donner des instructions supplémentaires sur la manière d'accomplir sa tâche. Ces paramètres sont appelés paramètres d'échantillonnage ; « échantillonner » signifie sélectionner une option aléatoire parmi plusieurs. Pour l'instant, nous allons nous concentrer sur un paramètre d'échantillonnage en particulier : la **température.**
La température peut être réglée dans une plage allant de 0 à 2.

Si la température est réglée sur 0, alors le LLM se comportera comme suit : en supposant qu'il existe un mot le plus probable (et qu'il ne s'agit pas d'une quasi-égalité ou d'un cas similaire), le LLM sélectionnera toujours le mot le plus probable. Cette approche est également appelée échantillonnage glouton (*greedy sampling*).

En revanche, si la température est supérieure à 0, le LLM introduit délibérément une certaine part d'aléatoire dans la manière dont il choisit le mot suivant.

ANNEXE F - Les vecteurs d’embeddings

Un vecteur d'embedding représente **le sens d'un mot sous forme de nombres**.

Chaque mot a une "carte d'identité" avec des centaines de caractéristiques mesurées par des nombres. Par exemple :

Pour le mot "chat" :

- Degré d'"animal" : 0.95
- Degré de "domestique" : 0.87
- Degré de "félin" : 0.93
- Degré de "petit" : 0.65
- Degré de "abstrait" : 0.01
- ... et des centaines d'autres caractéristiques

Pourquoi c'est utile ?

1. **Les mots similaires ont des vecteurs similaires** :
    - "chat" et "chien" auront des vecteurs proches (tous deux sont des animaux domestiques)
    - "chat" et "voiture" auront des vecteurs très différents
2. **Les relations sont préservées** :
    - Roi - Homme équivalent a Femme - Reine
    - Paris - France équivalent a Italie - Rome
3. **Le contexte est capturé** :

- Le mot "banque" aura un vecteur différent selon qu'il s'agit d'une banque financière ou de la banque d'une rivière

En résumé : Un vecteur d'embedding transforme un mot en une liste de nombres qui capture son sens, permettant à l'ordinateur de "comprendre" les relations entre les mots et leur signification.

Le nombre de caractéristiques est fixe pour un modèle donné.

Par exemple pour GPT-3.5 : **12,288 dimensions** pour le plus grand modèle.

Une caractéristique est toujours à la même position MAIS...

Ce que nous savons :

- La position 0 dans le vecteur "chat" correspond à la même "chose" que la position 0 dans le vecteur "chien"
- Les positions sont cohérentes entre tous les mots

Ce que nous NE savons PAS :

- **Les dimensions ne correspondent pas à des caractéristiques humainement compréhensibles** comme "degré d'animal" ou "taille"
- Les nombres sont appris automatiquement par le réseau de neurones
- Nous ne pouvons pas dire "la dimension 42 représente la couleur" ou "la dimension 123 représente la taille"

Exemple concret :

```
chat  = [0.23, -0.45, 0.67, 0.12, ...]
chien = [0.19, -0.41, 0.71, 0.08, ...]
table = [-0.82, 0.34, -0.15, 0.91, ...]
```

C'est comme si chaque mot était décrit par x (x dépend du modèle) caractéristiques mystérieuses" :

Les chercheurs peuvent parfois découvrir que certaines dimensions capturent des concepts particuliers (genre, temps, sentiment), mais c'est généralement difficile à interpréter car les dimensions travaillent ensemble de manière complexe.

ANNEXE G - La self attention

Rappel : L'objectif de l'algorithme de **self-attention** est de permettre au modèle de se concentrer sur les parties les plus pertinentes d'une **séquence** pour représenter chaque token, en tenant compte des relations et dépendances avec tous les autres tokens.

L'algorithme permet au modèle de pondérer l'importance des éléments précédents d'une séquence pour décider du suivant : chaque **token** « regarde » les autres et attribue un poids plus ou moins fort à leur influence.

Voici comment cela fonctionne de maniére simplifiée:

- Le système de comparaison : Chaque **token** possède deux étiquettes : une Requête (**Q**, ce qu'il cherche) et une Clé (**K**, ce qu'il propose). On applique des **transformations linéaires** aux **tokens embeddings** (voir annexe F) pour créer ces vecteurs étiquettes. On utilise pour ce faire les matrices **Wq et Wk** qui sont générées a l'apprentissage. Ces matrices apprennent progressivement :
  1. quelles composantes des **embeddings** sont utiles
  2. comment transformer l'information pour détecter :
     - des relations syntaxiques
     - des dépendances longues
     - des relations sujet/verbe
     - des coréférences
     - etc.

- Le score de similitude : L'importance d'un token "A" pour un token "B" est définie par la ressemblance mathématique (le **produit scalaire**) entre la Requête de "A" et la Clé de "B". Plus le score est élevé, plus le lien est fort.

- La distribution du poids : Ces scores sont convertis en pourcentages (poids d'attention). Ces poids mesurent à quel point un mot doit prêter attention aux autres.
  Un token avec un poids élevé sera celui qui influencera le plus la nouvelle représentation du mot en cours de traitement, car il contient l'information la plus pertinente pour le contexte.

Par exemple, dans la phrase : **« *Le chat mange la souris parce qu'elle est petite.* »,** Voici comment l'importance pourrait être distribuée pour le mot **"elle"** :

Mot ciblé	Importance (Poids)	Raison
souris	**75%**	C'est l'antécédent direct (accord de genre).
petite	**15%**	L'attribut confirme l'identité du sujet.
mange	**7%**	L'action donne un contexte sur la relation.
chat	**3%**	Très faible importance (incohérence de genre).

Pourquoi c'est astucieux ?

Le modèle n'utilise pas de "règles de grammaire" figées.
Par exemple, Il a appris, en lisant des milliards de textes lors de l’apprentissage, que :

1. Les pronoms s'accordent souvent en genre avec le nom qu'ils remplacent.
2. Dans le contexte de la phrase « ***Le chat mange la souris parce qu'il a faim*** », celui qui a "faim (il)" est généralement le sujet ("chat").

C'est cette combinaison de syntaxe (le genre) et de sémantique (le sens) qui définit mathématiquement l'importance.

ANNEXE H - Comment marchent les Agents ?

Le concept d'agent permet a un LLM de faire appel a une fonction extérieure afin de compléter ces possibilités.

Par exemple, toute mention dans le prompt (la question) d'une opération d'addition (un signe : +, ou un mot : ajoute…) va permettre de faire appel a une fonction « **add »** qui aura été identifié comme un **outil** disponible auprès du modèle.

La logique (simplifiée) est la suivante :

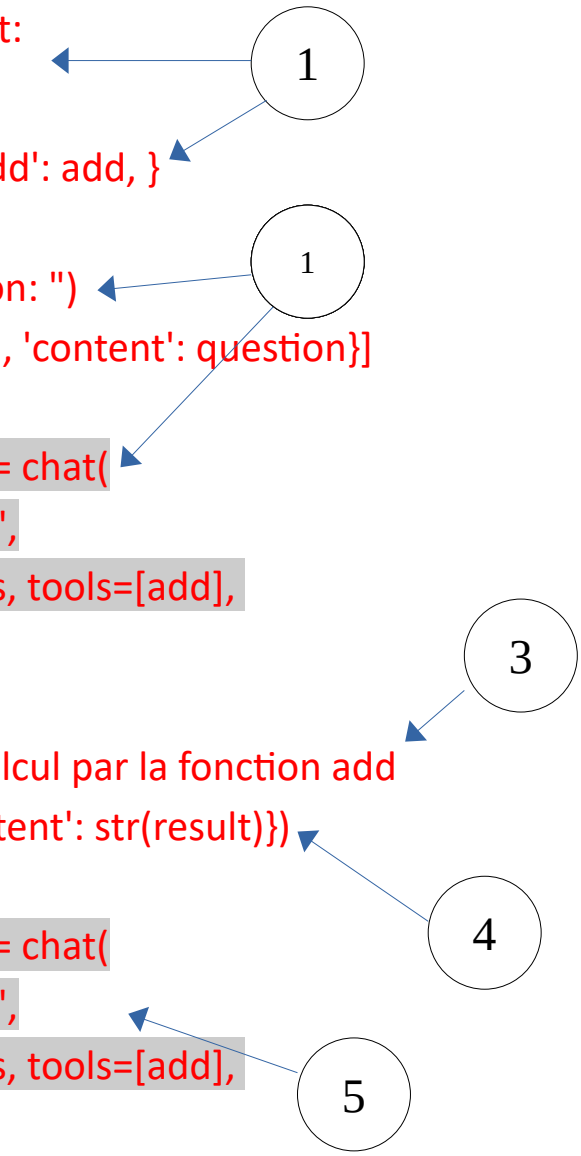
1. La fonction « **add** » est déclarée
2. on appel le LLM avec une question comme :
   how much is 3 + 5 (combien font 3+5)
3. le LLM associe (grâce a son apprentissage) le signe « + » a une addition et donc a la fonction « **add** », il rend une réponse qui demande l'usage de l'outil « **add** ».
3. Le programme traite cette réponse en appelant la fonction « **add** » (le programme qui a été associé a cette action) et donc qui sait faire ce calcul. Le programme donne un résultat.
4. on ajoute le résultat de la fonction au prompt initial
5. on appelle le LLM a nouveau qui répond avec la réponse finale
   The sum of 3 and 5 is 8 (la somme de 3 et 5 est 8)

Exemple de programme python qui utilise le LLM chinois *qwen* (dans un environnement *ollama*, local au PC).

```
...
def add(a: int, b: int) -> int:
        return a + b

available_functions = { 'add': add, }
...
question = input("Question: ")
 messages = [{'role': 'user', 'content': question}]
 ...
response: ChatResponse = chat(
        model='qwen2.5:7b',
        messages=messages, tools=[add],
)
 ...
result = ... # résultat du calcul par la fonction add
messages.append(... 'content': str(result)})
...
response: ChatResponse = chat(
        model='qwen2.5:7b',
        messages=messages, tools=[add],
)

...
print("Content: ", response.message.content)
```



Cet algorithme relativement simple peut être complexifié pour lui permettre d'appeler plusieurs d'outils différents dans la même séquence.

Bibliographie

Introduction to Artificial Intelligence (1985 !)
Charniak & McDermott

Programming in micro-PROLOG
Hugh de SARAM

Introduction to Operations Research
Hillier & Lieberman

Programming Machine Learning
Paolo Perrotta

Build a Large Language Model (From Scratch)
Sebastian Raschka

A Common-Sense Guide to AI Engineering
Jay Wengrow

Build a Reasoning Model (From Scratch)
Sebastian Raschka