Notes - Introduction à l'intelligence artificielle et aux grands modèles de langage

Marius Garénaux-Gruau*, Corentin Val†- IRISA

Table des matières

Ces notes viennent en complément d'une présentation donnée lors d'une formation d'une journée sur les LLM. Ce document disponible sous la licence ***CC-BY-NC-SA****.*

Voici un rapide condensé de resources à consulter ; de la plus abordable à la plus approfondie. Elles ont été utilisées, entre autres, pour rédiger ce document. Vous pouvez les utiliser comme des compléments à ces notes, pour approfondir certains sujets ou tout simplement voir différents points de vue.

— (2) : Science Etonnante ; Vidéo accessible à un large public et complète sur ChatGPT.

*marius.garenaux-gruau@irisa.fr
†corentin.val@irisa.fr

A whimsical and creative image depicting a hybrid creature that is a mix of a waffle and a hippopotamus. This imaginative creature features the distinctive, bulky body of a hippo, but with a texture and appearance resembling a golden-brown, crispy waffle. The creature might have elements like waffle squares across its skin and a syrup-like sheen. It's set in a surreal environment that playfully combines a natural water habitat of a hippo with elements of a breakfast table setting, possibly including oversized utensils or plates in the background. The image should evoke a sense of playful absurdity and culinary fantasy.

Figure 1 – Un exemple d'image générée à partir d'un texte par Stable Diffusion (1)

— (3) : Rapport du Sénat sur l'I.A ; très complet, abordable et récent.
— (4) : Andrej Karpathy, présentation abordable et complète (en Anglais) du fonctionnement des LLM.
— (5) : Formation FIDLE ; en live ou rediffusion. Couvre un large éventail de notions, avec les TP correspondants.
— (6) : Panoram'IA : Emission mensuelle de l'IDRIS sur l'actualité en IA
— (7) : Umar Jamil, beaucoup de ressources vidéo, et des exemples de codes sur le fonctionnement détaillé de nombreux modèles. Niveau relativement avancé, mais vaut le détour.
— (8) : Article de recherche récent couvrant assez largement les LLM.

1 Introduction à la notion de modèle d'IA avec un exemple

La notion d'Intelligence Artificielle est de plus en plus présente dans le quotidien de nombreuses personnes. Des algorithmes génératifs (génération d'image à partir de texte, agents conversationnels, …) aux systèmes de recommandation (Spotify, Instagram, …) ; les exemples sont nombreux et de plus en plus impressionnants. Afin de prendre les décisions les plus appropriées (comment utiliser - ou **ne pas utiliser** - ces modèles, lesquels choisir dans quel cadre, …), il est nécessaire de bien comprendre les enjeux liés à l'utilisation et au déploiement de ces outils.

Il est difficile de trouver une définition rigoureuse et précise de l'Intelligence Artificielle. Pour cause, on peut aborder cette notion sous de nombreux aspects : scientifique, économique, culturel, technologique, juridique,… Trouver une définition englobant tous ces exemples n'est alors pas si simple… On se réfère au très bon rapport de l'OPECST (Office Parlementaire d'Evaluation des Choix Scientifiques et Technologiques) sur les nouveaux développements de l'intelligence artificielle (3), page 41 (1.B.1) ; pour une explication plus détaillée sur ce sujet.

On gardera à l'esprit dans ce document que les modèles d'Intelligence Artificielle dont on va parler sont d'abord des **programmes informatiques**. Ils permettent d'automatiser des tâches.

1.1 Qu'est-ce qu'un modèle ?

La notion de modèle est centrale. Pour bien la comprendre, on présente dans un premier temps un exemple simple : le lien entre consommation électrique et température extérieure.

Consommation électrique en Bretagne

On récupère des données de consommation quotidienne (9) et de température moyenne (10) en Bretagne, depuis 2016. On affiche ensuite un point pour chaque relevé, sa coordonnée sur l'axe horizontal étant la température du jour, et sa coordonnée sur l'axe vertical étant la consommation totale quotidienne en Bretagne. Ceci produit un nuage de points illustrant le lien entre consommation électrique et température (Figure 2).

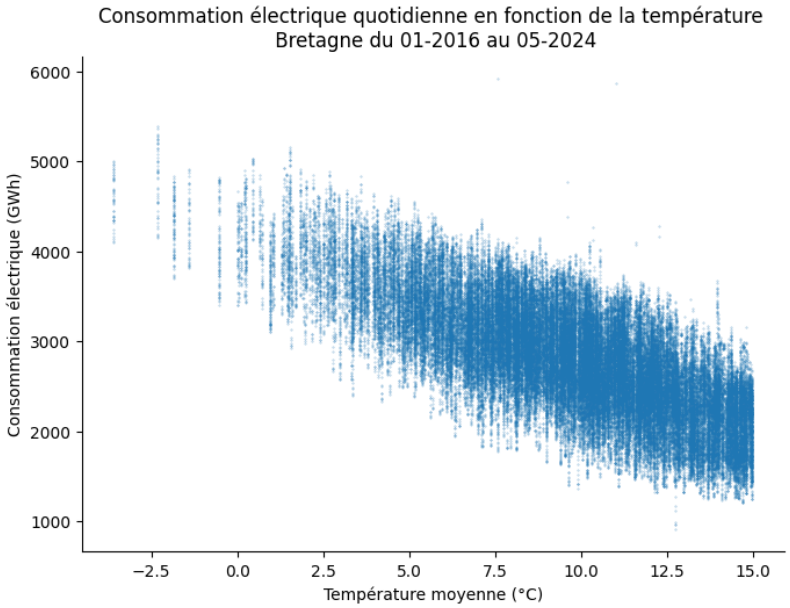

Figure 2 – Consommation électrique et température en Bretagne depuis 2016

Imaginons désormais que l'on souhaite avoir une idée de la consommation électrique pour toute la Bretagne demain ; en sachant qu'il fera en moyenne 10°C (avec une prédiction météo par exemple). On peut alors construire un programme informatique qui essaie de faire cette prédiction. Pour cela, nous allons d'abord **modéliser** le lien entre température et consommation électrique.

On observe une tendance globale : lorsque la **température diminue**, la **consommation augmente** à un rythme **constant**. C'est très probablement lié au chauffage - mais on ne cherche pas spécialement à comprendre ce lien ici. Forts de cette observation, on **choisit** de **modéliser** le lien entre température et consommation par une **droite**. Parmi toutes les droites possibles, on **choisit** celle qui est la plus proche du nuage de point [1] (Figure 3).

Selon notre modèle, s'il fait 10°C demain, la consommation moyenne sera d'environ 2800 GWh. On a bien sûr fait beaucoup d'approximations sur le chemin, avant d'arriver à cette valeur ; mais on a une idée de cette consommation. On a posé les bases d'une démarche scientifique, et on peut commencer à améliorer certains points pour avoir un "meilleur" modèle :

1. Bien qu'elle semble floue, la notion de "proche du nuage de point" a un sens très précis ici. Pour un point donné, on peut calculer sa différence de hauteur avec la droite. Si l'on fait la moyenne de ces différences avec tous les points, on obtient un nombre positif. Dire que la droite est proche du nuage de points veut dire ici que ce nombre est le plus petit possible. Il existe d'autres manières de mesurer la distance entre la droite et les points, et **choisir** une manière de mesurer cette distance est souvent une partie très importante des modèles.

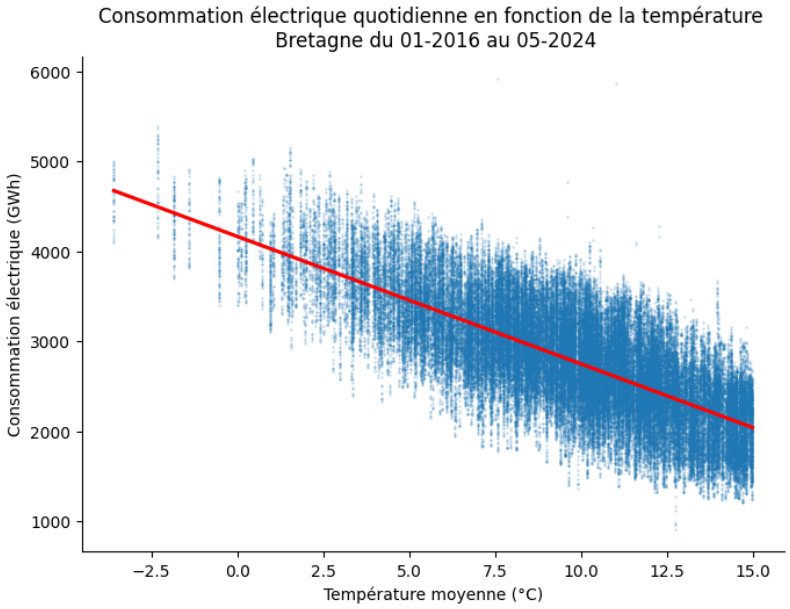

Figure 3 – Consommation électrique et température en Bretagne depuis 2016

- on a **choisi** de modéliser notre problème avec une droite (un autre modèle aurait peut-être mieux convenu),
- on a **choisi** une droite parmi toutes celles possibles (peut-être que la droite qui modélise le mieux notre phénomène est une autre),
- les données ne sont peut-être pas toutes bonnes (il y a peut-être des erreurs de mesure),
- notre modèle prédit tout le temps la même consommation pour une température donnée, la réalité est plus **complexe** ; il pourrait être intéressant de complexifier le modèle pour qu'il colle mieux aux données (par exemple en rajoutant de l'aléatoire ou d'autres paramètres en entrée, …)
- …

On retiendra qu'un **modèle** est un **support théorique** qui nous permet de reproduire un phénomène, de mieux le comprendre et éventuellement de l'automatiser *via* un programme informatique.

Les modèles d'IA fonctionnent pour la plupart en "entrée"-"sortie" : on leur donne quelque chose (par exemple une description textuelle d'image), et ils le transforment en autre chose (une image). Notre modèle, utilisé pour prédire la consommation électrique à partir de la température, ne déroge pas à cette règle.

Le modèle YOLO (11) (Figure 4) prend en entrée une image, et produit en sortie des "boîtes" annotées : un chien, un vélo, une voiture. Il modélise le fait de reconnaître des objets dans une image (capacité plutôt traditionnellement attribuée à des humains).

Le modèle AlphaFold (Figure 5)prédit la structure des protéines en 3 dimensions. Il prend en entrée une séquence d'acides aminés (en pratique c'est simplement une suite de lettres) ; et la transforme en des coordonnées dans l'espace. Ces coordonnées sont celles de la protéine constituée de cette séquence d'acides aminés. La structure en 3 dimensions de la protéine détermine sa fonction (par exemple (12) : transporter de l'oxygène dans le sang).

IA générative

Le terme IA générative revient beaucoup dans le débat public. Un modèle d'IA est dit génératif lorsqu'il produit du contenu : texte, image, vidéo, audio. On n'essaie pas ici de donner une définition rigoureuse au risque de s'empêtrer dans les sables mouvants de l'approximation. Les LLM, que l'on décrit dans ce document sont

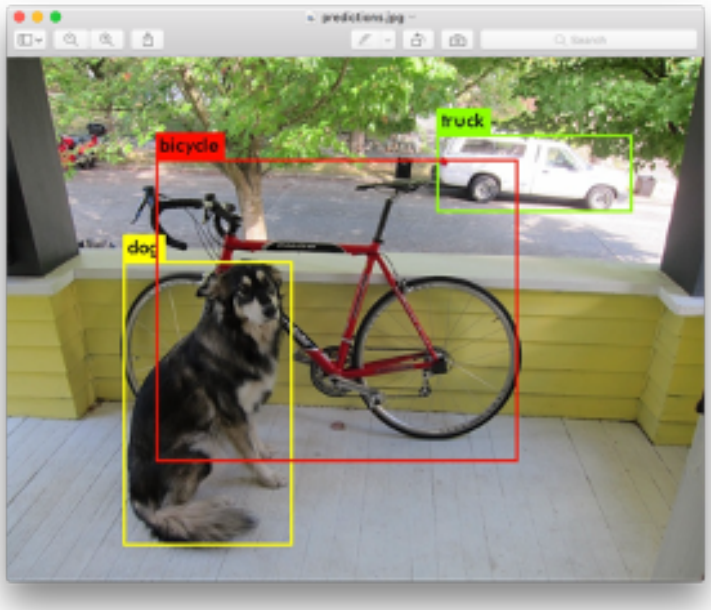

Figure 4 – Un exemple de segmentation d'image avec le modèle YOLO (You Only Look Once) (11)

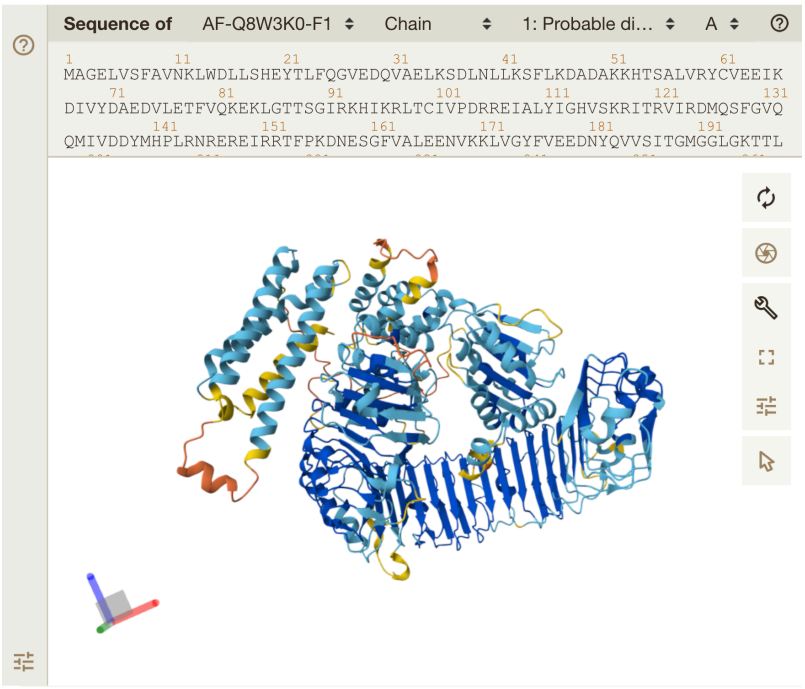

Figure 5 – AlphaFold - Probable disease resistance protein At1g58602 (13)

des modèles d'IA générative, mais on peut aussi citer Stable Diffusion (à l'origine de l'hippopotame - Figure 1 du début du document).

Attention cependant à l'acronyme **IAG**. Il est fortement ambigu car il désigne plutôt l'IA Générale. L'IA Générale fait référence à un modèle (**fictif** pour le moment) capable d'effectuer "n'importe quelle tâche cognitive propre aux humains ou autres animaux"[2]. Si on veut absolument utiliser un acronyme pour désigner les IA génératives, on pourra - en dernier recours - utiliser : IAg.

Et les modèles de langage dans tout ça ?

Les modèles de langage, eux, modélisent … le langage naturel. C'est-à-dire qu'ils **comprennent** le sens d'une phrase, l'**importance** d'un mot par rapport à un autre ; et sont capables de faire des prédictions concernant ces mots. Par exemple, les Grands modèles de langage sont capables de répondre à une question ou d'interpréter si une phrase est plutôt positive ou négative. Cependant, modéliser le langage naturel est loin d'être facile, et on s'en rend compte en constatant le temps qui s'est passé avant l'apparition de robots conversationnels qui fonctionnent bien (environ 2018 avec GPT-2). Il existait avant cela d'autres modèles ; et tout un champ de recherche (*Natural Language Processing*). Ces modèles étaient cependant bien moins impressionnants que les grands modèles de langage de ces dernières années. La ruse qui a permis à ces modèles d'émerger est liée à un changement de point de vue :

> Au lieu d'expliciter les règles régissant le langage naturel, on construit des modèles qui sont capables "d'apprendre" à partir d'exemples de phrases.

En pratique, on parle de modèles de *Machine Learning* (Apprentissage Automatique / Apprentissage Machine). On décrit dans la suite comment ce genre de modèles fonctionnent.

1.2 Apprentissage Automatique (*Machine Learning*), entraînement

Les modèles de ***Machine Learning*** sont une sous-famille des modèles d'IA. Ce sont des modèles auxquels on va pouvoir apprendre à réaliser des tâches (par exemple, apprendre à déterminer si une photo donnée est une photo de voiture ou non). Souvent, cet apprentissage est effectué en utilisant des **données** (en suivant l'exemple, des milliers de photos dont on sait lesquelles contiennent des voitures). Ces données permettent **d'ajuster** (on parle **d'entraînement**) le modèle pour qu'il ***fonctionne bien***.

Dans notre exemple de consommation électrique, la droite est un modèle de Machine Learning ! On l'entraîne avec des données, en choisissant la droite la plus proche du nuage de point. Les paramètres que l'on ajuste ici sont la **hauteur** et la **pente** de la droite. Notre modèle a **appris** à prédire la consommation électrique à partir de la température.

Dans le processus d'entraînement d'un modèle de *Machine Learning*, une partie importante consiste à évaluer le modèle. En effet, **les paramètres** sont choisis pour que le modèle soit le meilleur possible **sur les données d'entraînement**. Par exemple, dans le cas de la droite, on minimise la distance moyenne entre les points et la droite. Si les données sont erronées (erreur de mesure, …) ; le modèle va être entraîné à prédire …des informations erronées (et il sera peut-être très bon pour prédire ces informations). La qualité des données est une partie très importante de ces modèles.

Nombre de paramètres

Là où une droite peut être définie par 2 nombres (sa hauteur et sa pente), les grands modèles de langage récents ont bien plus de paramètres (on compte en dizaines de **milliards**). C'est grâce à ces milliards de paramètres que l'on arrive à modéliser des choses aussi complexes que le langage naturel. C'est de là que vient l'adjectif "Grand" dans « Grand Modèle de Langage ».

L'essor de l'IA ces dernières années (depuis 2010) est dû en partie à des modèles avec un nombre très important de paramètres. Cet essor s'explique car ces modèles ont la possibilité de résoudre des tâches

2. https://fr.wikipedia.org/wiki/Intelligence_artificielle_g%C3%A9n%C3%A9rale#cite_note-:0-2

complexes (génération de texte, d'image, reconnaissance d'image, jouer et gagner à des jeux complexes comme le go, …). On parle de ***Deep Learning*** (**Apprentissage Profond**) pour une désigner une catégorie de modèles (les **réseaux de neurones**) ayant beaucoup de paramètres.

Les idées d'architecture de ces modèles sont plus anciennes que 2010, mais la **quantité de données** amenée par **Internet**, ainsi que le déploiement de **moyens de calcul** plus importants sont les facteurs qui ont permis l'entraînement et la popularisation de ces modèles. Un des points faible de ces modèles est qu'on ne comprend pas vraiment comment ils fonctionnent. En revanche, on sait observer qu'ils fonctionnent bien ! L'étude de ces modèles est devenue en partie une science expérimentale.

1.3 Les impacts humains et environnementaux des modèles

Le développement et l'utilisation des grands modèles que l'on vient de décrire ne se fait pas sans impacts. Des employés exploités pour visualiser des contenus toxiques (14) aux mini-centrales nucléaires produites à la chaîne pour alimenter des centres de données (15), le paysage n'est pas très beau à voir… L'article et la présentation vidéo des JRES (Journées Réseaux de l'Enseignement Supérieur) présentent très bien le sujet : (16).

Pour rentrer dans les détails, les modèles d'apprentissage profonds posent de sérieuses questions, et ce tout au long de leur cycle de vie. De la fabrication des composants électronique à l'utilisation finale des modèles, en passant par leur entraînement et leur mise à disposition ; chacune de ces étapes apporte son lot de limitations et de problématiques.

Fabrication des composants et durée de vie du matériel

La production des composants électroniques nécessaires à l'entraînement et à la mise à disposition de ces modèles nécessite des matières premières rares (on parle de terres rares). Ces terres rares sont parfois extraites dans des conditions désastreuses. Le tantale par exemple, utilisé dans les condensateurs ; est principalement produit en République Démocratique du Congo (43% en 2017 (17)). L'article de radio-canada (18) décrit une mine où l'extraction se fait manuellement, parfois par des enfants.

Une fois produites, les cartes graphiques ont *a priori* une durée de vie très courte (1 à 3 ans selon (19)), due à un taux d'utilisation très élevé. L'article cite un architecte de Google, selon qui il serait possible de prolonger cette durée - au doux prix de les utiliser moins souvent et donc d'amortir moins vite leur achat.

La question du recyclage des composants n'est pas du tout résolue ; alors que les cartes graphiques contiennent des produits dangereux pour l'environnement (20). Le rapport sur la durabilité de la compagnie Nvidia (21) (ayant un quasi-monopole sur les cartes graphiques utilisées pour l'entraînement des grands modèles de langage) reste bizarrement très timide sur le sujet ; malgré leur sublime phrase d'introduction : "Green isn't just our corporate color" (20).

Nettoyage et annotation des données

Comme on l'a vu, les modèles d'apprentissage automatique ont besoin de beaucoup de données. La qualité des données impactant directement la qualité des modèles, les entreprises qui développent ces derniers ont parfois recours à de méthodes de nettoyage et d'étiquetage des données **manuelles**. C'est le cas par exemple de l'entreprise OpenAI (développant ChatGPT) qui, en 2022, a sous-traité des tâches d'annotations d'images à une entreprise au Kenya (Sama). Les employés étaient payés autour de 2$ de l'heure ; pour visionner des contenus toxiques : abus sexuel d'enfants, suicide, torture, … Une enquête sur le sujet a été effectuée par le TIME magazine (14).

Des questions intéressantes sont à régler autour des sources de données et des droits d'auteurs. Il semble à peu près de notoriété publique que les modèles de langage utilisent des données sous droit d'auteurs (22).

Entraînement des modèles, mise à disposition

Les ordinateurs utilisés dans les phases d'entraînement et d'exploitation des modèles ont besoin d'électricité pour fonctionner, et de systèmes de refroidissements pour ne pas surchauffer. Ces consommations électriques sont loins d'être négligeables ; et même si l'accent est mis sur les émissions CO_2 [3], il ne faut pas oublier que ce n'est pas la seule manière de mesurer l'impact environnemental.

Les modèles génératifs sont tellement gros, que même une fois entraînés, le fait de transformer une entrée en sortie nécessite beaucoup de puissance de calcul, et donc d'électricité. La consommation en électricité des centres de calculs est tellement importante que Google envisage même d'y dédier des centrales nucléaires (15). Pour avoir un ordre de grandeur des consommations électriques des modèles (hors entraînement), on pourra consulter l'article (24). En plus de l'électricité, les centres de calculs ont besoin d'eau pour refroidir les composants (25).

1.4 *Facultatif - Zoom sur les réseaux de neurones*

Les réseaux de neurones sont une famille de modèles de ***Machine Learning***. Ils sont utilisés dans quasiment tous les grands modèles récents. Grossièrement, les réseaux de neurones sont découpés en couches. Pour transformer une entrée en sortie, les réseaux de neurones vont transformer successivement l'entrée avec chacune des couches. Chaque couche a ses paramètres (à la manière de notre droite), qui peuvent être réglés pour minimiser l'erreur du modèle sur des données d'entraînement. On parle de réseaux profonds (*Deep Learning*) lorsqu'ils ont beaucoup de couches.

Le succès des réseaux de neurones dans les modèles récents s'explique par plusieurs points :

- l'algorithme d'optimisation des paramètres de ces réseaux est très souple (méthode du gradient) ; et très bien implémenté (Automatic Differentiation (26))
- l'architecture en couche s'adapte à beaucoup de cas, et bénéficie d'une communauté *open-source* active (avec par exemple PyTorch (27)), ce qui favorise la réutilisation et le développement de nouveaux modèles,
- les réseaux de neurones fonctionnent bien pour modéliser beaucoup de phénomènes qu'on arrive pas à expliciter avec des règles (générer des images, du texte, …),
- les recherches sur ces modèles ont été (et sont toujours) accélérées par les financements privés (globalement toutes les grosses entreprises du web : les GAFAM [4] et autres entreprises américaines, des entreprises chinoises comme Alibaba, …), qui ont leurs propres laboratoires de recherche.
- …

2 Explication du fonctionnement des LLM

Un LLM (***Large Language Model***) est un modèle de *Machine Learning* qui comprend le langage naturel, et qui est capable de produire et d'analyser du texte. Comme on l'a décrit précédemment, ces modèles ont pour la plupart plusieurs dizaines de milliards de paramètres. La phase d'entraînement d'un tel modèle est très coûteuse en infrastructure [5] - elle demande beaucoup de matériel informatique - et en temps - les phases d'entraînement peuvent durer plusieurs mois.

Les robots conversationnels (*chatbots*) comme **ChatGPT** sont construits au-dessus de modèles appelés **modèles de fondations**. Afin de bien comprendre comment ces *chatbots* fonctionnent, on commence par présenter les **modèles de fondation**. On peut les voir un peu comme des **moteurs de voiture**. En suivant cette comparaison, les agents conversationnels peuvent être vus comme les différentes voitures que

3. Les papiers de recherches décrivant les phases d'entraînement des nouveaux modèles de langage diffusent parfois des estimations de CO2. Meta estime par exemple dans un papier que le développement d'une génération de modèles à émis autour de 1000 tonnes équivalent CO2 (23).

4. Désignait les entreprises américaines : Google, Amazon, Facebook, Apple, Microsoft. Aujourd'hui, certains noms ont changé : Google est devenu Alphabet et Facebook Meta. L'entreprise OpenAI (qui développe ChatGPT) a été rachetée par Microsoft.

5. Le modèle V3 de l'entreprise chinoise DeepSeek a *a priori* (c'est encore sujet à débat, à prendre avec précaution) nécessité bien moins de puissance de calcul que ses homologues pour leur entraînement, pour des tailles et des performances similaires (28). Ceci va peut-être rebattre les cartes sur les moyens de calculs nécessaires à l'entraînement de ces modèles.

l'on peut construire autour d'un moteur. Pour se repérer, l'agent conversationnel ChatGPT est construit au-dessus de modèles de fondations appelés GPT (*Generative Pre-trained Transformer*). Plusieurs modèles existent : GPT-2, GPT-3, GPT-4…

> Le terme *modèle de fondation* n'est pas propre aux modèles de langage ; et désigne en général un *grand* modèle dont l'entraînement est conséquent ; mais qui peut être **spécialisé** pour de nombreuses tâches différentes : génération de texte, calcul de similarité de deux phrases, agent conversationnel, …

2.1 Modèle de fondation, *tokens* et machines à camemberts

Les modèles de fondation sont le **noyau** des agents conversationnels comme *ChatGPT*. Le fonctionnement précis de ces modèles varie. Aussi, on présente ici les grandes idées des modèles déjà existants, mais il faut bien garder à l'esprit que ces domaines évoluent très vite.

On a vu dans la première partie de ce document que la plupart des modèles d'IA transforment une "entrée" en une "sortie". Les modèles de langage ne font pas exception. La plupart d'entre eux fonctionnent un peu comme le mécanisme d'autocomplétion des téléphones : on leur donne en entrée un début de phrase (une **séquence de mots**), qu'ils sont capables de transformer en la suite de la phrase. On parle de mécanisme auto-régressif (*auto-regressive* ou *causal* en anglais). D'autres modèles de langage sont plutôt capables de remplir des textes à trous (c'est le cas de BERT (29)). L'entrée des modèles est très souvent appelée ***"prompt"*** (en français, invite ou **instruction**).

Très concrètement, si on donne une séquence de mots au modèle ; la première étape sera de la convertir en *tokens* (**jetons** en Français). Les *tokens* sont l'unité de langage la plus fine qu'un LLM puisse comprendre. Ils sont de taille comparable aux mots, mais il faut absolument garder à l'esprit que le modèle utilise des *tokens* et non des mots voire des lettres. Une fois l'entrée découpée en *tokens* ; le modèle tente de prédire quel sera le prochain *token*. Pour résumer : "entrée" = "séquence de *tokens*", "sortie" = "*token* suivant". Le modèle a une liste de *tokens* qu'il peut comprendre ; que l'on appelle son "vocabulaire". La taille du vocabulaire varie selon les modèles ; en gros de 30 000 à plus de 100 000. Le découpage en *tokens* a l'avantage par rapport à un découpage en mots de ne pas créer des vocabulaires trop grands. Et il a l'avantage par rapport a un découpage en caractères de ne pas créer des séquences encodées trop longues. C'est une sorte de *juste milieu* entre ces 2 découpages (4).

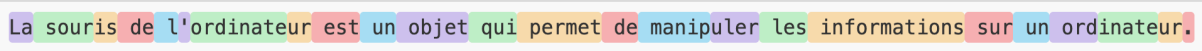

Figure 6 – Découpage d'une phrase en *tokens* (30)

Rentrons un peu plus dans le détail. Juste avant de générer un *token*, le modèle produit un **camembert** - ou *diagramme circulaire*. Ce camembert est en fait la sortie la plus utile du modèle [6]. Le modèle y a représenté les différents *tokens* pouvant apparaître à la suite de la séquence fournie en entrée. Seulement, la subtilité des LLM réside dans le fait qu'à chacun des *tokens* est assigné une probabilité d'apparition. Certains *tokens* ont plus de chance d'apparaître à la suite de cette séquence (Figure 7), et sont donc représentés avec une plus grande part de camembert.

Le point central de la génération de texte par les LLM est ici : le modèle **tire au hasard** le *token* suivant. Ce sont des modèles de nature **probabiliste**, qui sont tout à fait capables de produire 2 sorties différentes pour la même entrée. Différentes méthodes existent pour tirer au sort le prochain *token* ; chacune se basant sur le camembert. La méthode peut être choisie lors de la génération de texte, par les utilisatrices et utilisateurs des modèles. Citons en vrac différentes méthodes, directement issues de l'article (32) :

— *Greedy* (avare) : peut-être la moins utile, elle consiste à choisir le *token* avec la plus forte probabilité ; tuant le côté aléatoire du modèle ;

6. C'est la sortie la plus utile car elle contient plus d'information que simplement le *token* suivant. Ce camembert va pouvoir être utilisé pour spécialiser le modèle de fondation dans tout un tas de tâches variées.

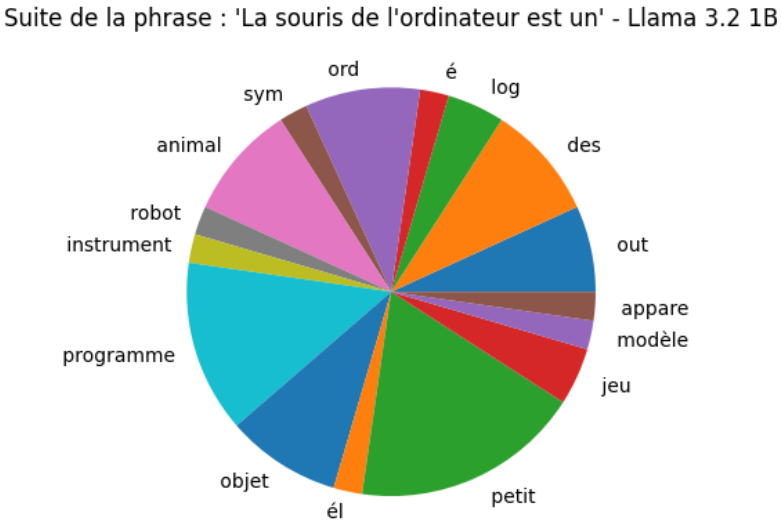

Figure 7 – Prédiction du prochain token - Llama 3.2 1B (31)

— *Beam search* (recherche par faisceau) : un peu plus évoluée, cette méthode consiste à regarder une étape plus loin. Elle choisit le couple des 2 *tokens* suivants ayant le plus de chance d'apparaître. Par exemple, pour la séquence : "La souris" ; le modèle prédit que 2 suites possibles sont "La souris de l'" et "La souris est un". Dans le premier cas, le *token* " de" avait une probabilité de 0.8, puis " l'" une probabilité de 0.3. Dans le second, le *token* " est" avait une probabilité de 0.6 et " un" de 0.9. Au total, la première séquence a donc une probabilité 0.8*0.3 = 0.24 d'apparaître, et la seconde 0.6*0.9 = 0.54. Le modèle choisit alors la seconde.
— *Sampling* (échantillonnage) : ici, chaque *token* peut être choisi, en fonction de sa probabilité : les *tokens* avec des plus fortes probabilité ont plus de chance d'être tirés. On peut imaginer qu'on fait tourner la roue (celle donnant les probabilités du prochain *token*) ; et qu'on choisit le *token* là où la roue s'arrête.
— *Top K* : même méthode que l'échantillonnage, mais on ne garde que les K *tokens* les plus probables.
— *Top p* : de manière un peu similaire à la précédente, cette méthode consiste à ne garder que les *tokens* les plus probables, pour que la somme de leurs probabilités dépasse *p*. On choisit ensuite le prochain *token* au hasard, comme dans la méthode "échantillonnage".

L'étape consistant à utiliser le modèle pour prédire un *token* est appelée **l'inférence**.

Dans l'exemple ci-dessus, le modèle a estimé que la suite la plus probable était "petit" ; mais que les mots "programme", "objet" ou "animal" pouvaient aussi apparaître. On observe ici la polysémie du mot "souris" ; tous les cas de figure sont envisagés par le modèle.

Pour prédire des textes plus longs, il suffit simplement de redonner au modèle la séquence qu'il vient de compléter (d'où le terme **auto-régressif**). Faisons cela sur l'exemple ci-dessus ; le modèle finit par prédire la phrase : "La souris de l'ordinateur est un objet qui permet de manipuler les informations sur un ordinateur.". On remarque sur les dernières prédictions de la phrase (Figure 8) que le modèle n'a plus aucun doute sur le fait que la phrase désigne une souris d'ordinateur et non l'animal.

Hallucination

Le LLM est donc simplement entraîné à prédire quel sera le prochain *token* suivant une séquence. Il n'a aucune notion de ce qui est **vrai** ou **faux** ; il a simplement une idée générale des contenus qu'on lui a montré lors de son entraînement. L'euphémisme désignant le fait qu'un LLM raconte complètement n'importe quoi est l' "hallucination". Les hallucinations ne sont pas étonnantes à ce stade de l'entraînement, la notion de

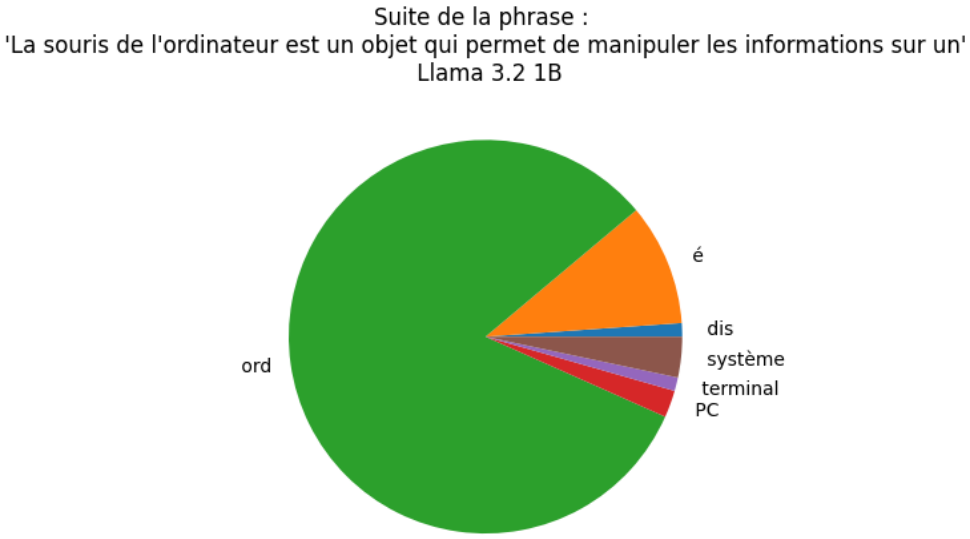

Figure 8 – Prédiction du prochain token - Llama 3.2 1B (31)

vérité peut être très loin de celle de *token* probable. Cependant, elle persiste jusqu'aux agents conversationnels ; et donc il ne faut absolument pas perdre de vue que toute information sortant d'un tel modèle **doit être vérifiée et sourcée**. Ainsi, toute information que l'on n'est pas capable de vérifier - ou de faire vérifier - doit être vue comme **inutilisable**. Pour contourner ces problèmes, les ***RAG*** (Retrieval Augmented Generation) sont une excellente parade : ils consistent à utiliser les LLM pour chercher dans une base de donnée ; puis à fournir la source de leur réponse.

Température

La température est un nombre que les utilisateurs et utilisatrices des LLM peuvent choisir pour modifier la génération de texte. Ce nombre représente à quel point le modèle est créatif dans ses réponses. En théorie, n'importe quel nombre positif peut être accepté, mais en pratique, les outils permettant d'utiliser les modèles restreignent aux nombres entre 0 et 1 (certains acceptent jusqu'à 2). La température modifie les probabilités d'apparition du prochain *token*, au moment où le modèle produit le camembert. Cependant, l'ordre des *tokens*, de celui ayant la plus faible probabilité d'apparition à celui avec la plus élevée, ne change pas avec la température [7]. Voir Figure 9 pour quelques exemples.

Une température proche de 0 va pénaliser les *tokens* avec des faibles probabilités d'apparition et favoriser ceux avec des fortes probabilités d'apparition : cela augmente le manichéisme du modèle. Une température de 0 rend le modèle complètement déterministe, il choisit nécessairement le *token* avec la plus haute probabilité d'apparition. Au contraire, une température plus éloignée de 0 va favoriser les *tokens* avec des faibles probabilités d'apparition, et pénaliser ceux avec de fortes probabilités d'apparition. En fait, avec une température au-delà de 2, le modèle assigne des probabilité tellement faibles aux *tokens* les plus probables que le texte généré n'a plus aucun sens (c'est une sorte de créativité extrême qui consiste à raconter littéralement n'importe quoi).

7. Pour que l'ordre des *tokens* change, il faudrait mettre une température négative. Dans ce cas, les *tokens* avec les plus faibles probabilités d'apparitions deviennent ceux avec les plus fortes, et *vice-versa*. Pas très utile… ça explique pourquoi les outils restreignent aux températures positives.

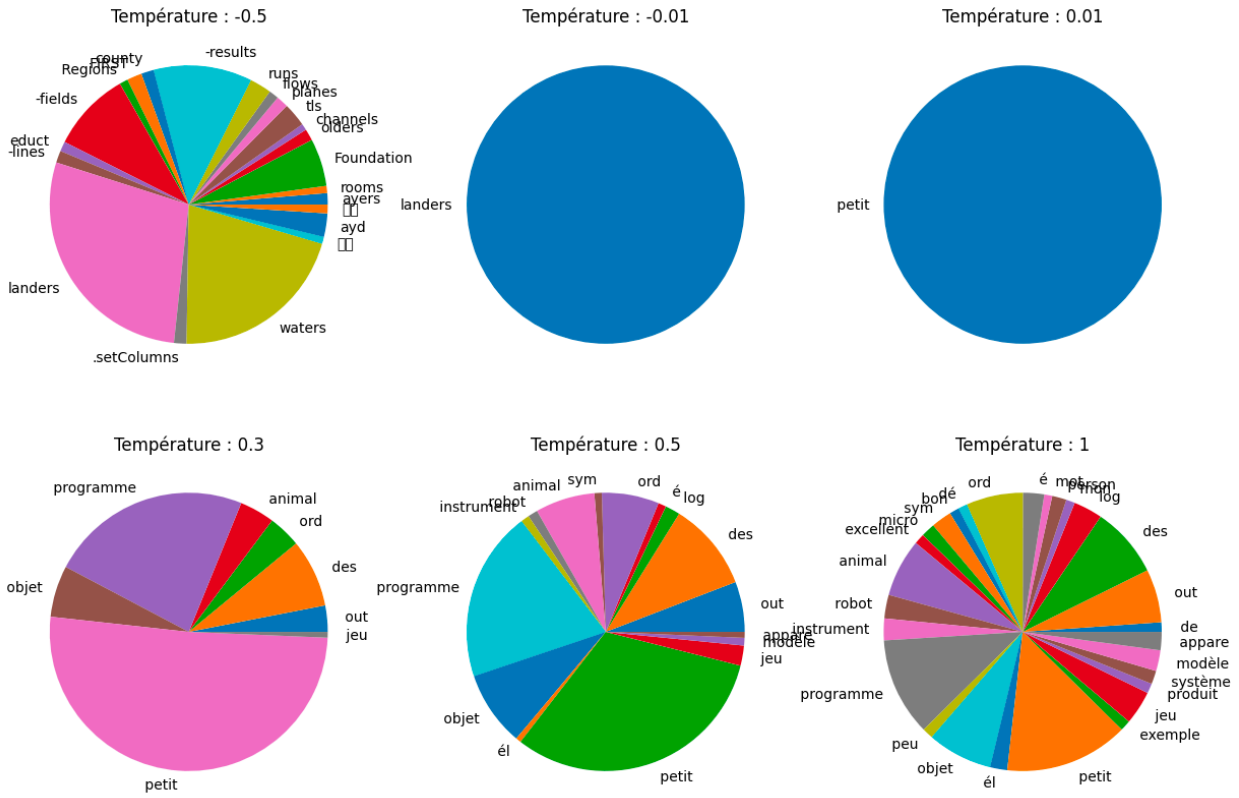

Figure 9 – Suite de la phrase : “La souris de l’ordinateur est un”, avec différentes températures - Llama 3.2 1B

2.2 Entraînement des modèles de fondation

Les grands modèles de langage sont des modèles d'apprentissage automatique : ils apprennent la structure du langage à partir de données textuelles. De la même manière que la droite de l'exemple initial est composée de deux paramètres, les modèles de fondation sont composés de milliards - dizaines / centaines de milliards - de paramètres. Si on initialise le modèle que l'on a présenté juste au dessus avec des paramètres quelconques, il prédit tout simplement n'importe quoi ! Dans la Figure 10 on présente le camembert produit par le modèle avec des paramètres quelconques. Pour information, en essayant de continuer la génération, nous avons obtenu la séquence suivante : "La souris de l'ordinateur estmpjesesper Annie ?Compar ?? clothes_sink QByteArray eitherenciónats Sustainabilityimanpyп ?? compatibility nth either". Clairement, le modèle n'est pas capable de produire du texte intelligible.

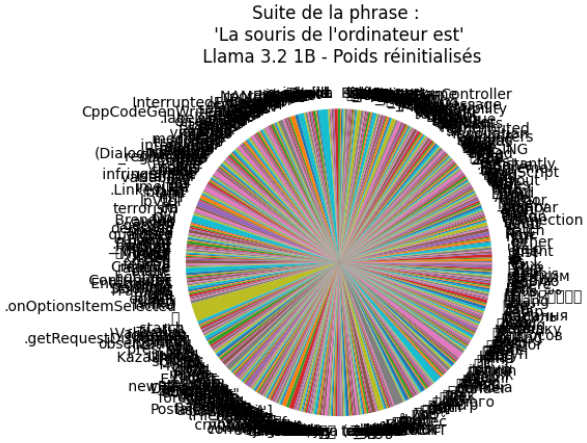

Figure 10 – Prédiction du prochain token avec des paramètres réinitialisés - Llama 3.2 1B (31)

L'entraînement du modèle de fondation consiste alors à trouver les meilleures valeurs possibles pour les paramètres, afin qu'il produise du langage cohérent (dans la partie précédente, les phrases sont produites avec le modèle entraîné bien sûr).

Comme dans le modèle de prédiction de température, la phase d'entraînement s'appuie sur des **données**. Et ça tombe bien, Internet contient énormément de contenus textuels ! En pratique, de grands jeux de données sont constitués à partir de données venant d'Internet : Wikipédia (33), GitHub (base de données de projets de programmation) (34), ArXiv (base de donnée d'articles scientifiques) (35), … On pourra consulter les Tables 1 et 2 de l'article (8) pour une vue d'ensemble des différents jeux de données utilisés pour l'entraînement des modèles. On peut également citer le jeu de données FineWeb (36) ; qui reconstruit des données semblables à celles de l'entraînement des LLM. Les données brutes sortant d'Internet ne sont pas utilisées directement dans l'entraînement. Un énorme travail de nettoyage de données (filtrage de contenus indésirables, gestions des contenus de différentes langues, déduplication, suppression d'informations personnelles, …) est fait en amont.

Pour les modèles Llama, les données nettoyées constituent moins de 5TB de stockage disque (!) (23). C'est très peu, mais ces données ne sont pas organisées, et il est potentiellement impossible de retrouver facilement une information dedans. Ces modèles sont alors entraînés avec environ 1 billion de *tokens* (mille milliards, 1 000 000 000 000). Une fois entraîné, on peut voir ces modèles comme un moyen de synthétiser ces données.

En pratique, l'entraînement se déroule de la manière suivante :

— On prend une partie du texte d'entraînement, découpé en *tokens*. Par exemple, 9 tokens consécutifs dans la page Wikipédia du gratin dauphinois : "Le", " grat", "in", " dau", "ph", "inois", " est", " un", " plat".

— On tronque cette séquence en ne gardant que les 8 premiers : “Le”, ” grat”, “in”, ” dau”, “ph”, “inois”, ” est”, ” un”.
— On donne ensuite cette séquence au modèle, qui prédit une distribution de probabilité pour le prochain *token* (c’est-à-dire, pour chaque *token* du vocabulaire, le modèle estime sa probabilité d’apparition à la suite de la séquence des 8 *tokens*). Cela nous donne un camembert, comme précedemment.
— Le camembert obtenu est comparé avec le résultat attendu : le *token* ” plat”.
— Les paramètres du modèle sont modifiés en fonction de cette comparaison.

Cette dernière étape est de loin la plus coûteuse en calcul. Elle est faite avec un algorithme (rétro-propagation de gradient) qui permet de calculer à quel point chaque paramètre doit être modifié pour améliorer la sortie du modèle. C’est loin d’être facile, d’autant plus que le nombre de paramètres est tellement grand que de nombreux problèmes se posent sur le chemin. Les implémentations de cet algorithme sont très astucieuses (auto-differentiation (26), (27)).

2.3 Du modèle de fondation à l’agent conversationnel

Le modèle de fondation, comme son nom l’indique, est à la racine de beaucoup d’applications différentes. Il comprend le langage naturel et est capable d’en générer. C’est donc une très bonne base pour produire tout un tas de modèles plus spécifiques. On peut par exemple spécialiser le modèle de fondation pour qu’il classifie si une phrase est positive ou négative [8] ; ou si un texte contient des données personnelles ou non. Une autre application serait d’avoir un modèle capable de calculer la similarité entre deux phrases [9]. Ici, on présente une de ces spécialisations, la création d’un **agent conversationnel**. C’est celle qui a vraiment popularisée les LLM, et qui est la plus utilisée aujourd’hui. Cependant, il ne faut pas oublier que, selon les usages, l’agent conversationnel n’est pas toujours la meilleure solution.

Afin d’obtenir ces agents conversationnels, plusieurs méthodes ont été imaginées par les développeurs et développeuses des modèles. Les différentes méthodes utilisées dépendent des organismes produisant les modèles, et ne sont pas toujours expliquées (il y a parfois une sorte de “secret de fabrication”). Ainsi, il convient de rester **prudent** sur toute cette partie : plus particulièrement que pour le reste du document, cette section ne contient pas de “vérité générale” qui s’applique et s’appliquera à tous les modèles ; mais plutôt quelques grandes idées que l’on retrouve dans plusieurs modèles (ceux qui ont publiés des informations sur leur processus d’entraînement). On cite le *survey* récent (37) qui présente de manière très claire les différentes méthodes pouvant être utilisées pour spécialiser un modèle de fondation.

Voici un exemple de complétion d’une séquence de *tokens* par un modèle de fondation (Llama 3.2 1B) :

```
Entrée :
"J'ai dans mon placard des tomates pelées, du riz et du poulet. Qu'est-ce
que je peux cuisiner ?"

Sortie du modèle de fondation (Llama 3.2 1B) :
"Je suis contente de pouvoir cuisiner! J'ai un petit gars qui a besoin
d'attention et de nourriture. Je vais le nourrir."
```

Et voici la sortie de l’agent conversationnel :

```
Sortie de l'agent conversationnel (Llama 3.2 1B Instruct) :
"C'est une question classique! Voici quelques idées de plats que vous pouvez
préparer avec ces ingrédients :
__Riz et poulet aux tomates__ :
Faites chauffer du riz et du poulet dans une poêle, ajoutez des tomates pelées
coupées en dés et du sel. Servez chaud."
```

Ces deux réponses sont complètement différentes. Dans la première, le modèle a été capable de produire du texte intelligible et cohérent avec l’entrée, mais il ne répond pas du tout à l’attente de l’utilisateur (on parle

8. https://huggingface.co/tabularisai/multilingual-sentiment-analysis
9. https://www.sbert.net/docs/sentence_transformer/training_overview.html

de **problème d'alignement**). Le second en revanche, y répond très bien. Nous allons expliquer comment on peut arriver à un tel agent.

Dans la littérature, le terme ***pre-training*** (pré-entraînement) est utilisé pour parler de l'entraînement du modèle de fondation, et ***post-training*** (post-entraînement) pour parler de la spécialisation du modèle de fondation en agent conversationnel [10]. Le pré-entraînement, que l'on a décrit juste avant, est coûteux (plusieurs mois, plusieurs millions - au grand minimum - de dollars d'investissement matériel). Le post-entraînement nécessite *a priori* bien moins de temps. Une fois le modèle complètement entraîné (ou même seulement avec le modèle pré-entraîné), il est possible de continuer à le spécialiser avec les méthodes présentées ici, pour qu'il intègre des données propres à un domaine. Cela permettra d'obtenir un modèle pour son propre cas d'usage (par exemple, retenir des acronymes, ou devenir spécialiste de la peinture flamande du 19e siècle). On parle dans ce cas de *fine-tuning* (ajustement). Voici un exemple avec les modèles Llama [11].

On trouve deux types de méthodes majoritaires dans le post-entraînement.

— Ajustement supervisé (*Supervised Fine Tuning (SFT)*),
— Ajustement par renforcement (*Reinforcement Learning (RL)*).

2.3.1 Ajustement supervisé

Cette phase consiste à continuer l'entraînement que l'on a décrit précédemment (du modèle de fondation), mais avec un nouveau jeu de données. Au lieu de prendre des suites de *tokens* issues d'Internet, on utilise des jeux de données de conversations entre un agent (aimable, près à aider les gens, …) et un utilisateur. À l'origine, ces jeux de données ont été constitués par des **experts humains**, payés pour répondre à des questions, afin que les échanges soient enregistrés dans le jeu de données. Désormais, il est très probable que ces jeux de données soient partiellement produits par des LLM (les experts humains peuvent être aidés dans la rédaction de leur réponse par des LLM, par exemple).

Ci-dessous, on montre un exemple de conversation, dont les réponses ont été remplies par des humains. On observe qu'une même question admet plusieurs formulations de réponse différentes.

text string	role string	lang string
Combien de livres Hunger Games existe-t-il ? ça fait environ combien de pages ?	prompter	fr
Il existe 4 livres Hunger Games: -Hunger Games (384 pages) -Hunger Games: L'embrasement (400 pages) -…	assistant	fr
Il y a 3 ouvrages principaux à la série Hunger Games, avec un préquel. En version française, ça…	assistant	fr
Il y a 3 livres principaux (1 prequel) et ça fait beaucoup de pages : Livre 1 : 399 Livre 2 :400 Livr…	assistant	fr

Figure 11 – Question - Réponse entre un utilisateur et un assistant. OASST1 (38) - Hugging Face

C'est aussi l'occasion de conditionner le modèle à formuler des réponses spécifiques dans des cas précis : dire qu'il ne connaît pas la réponse à une question, éviter de répondre aux questions sensibles, … L'objectif est de transformer le modèle de fondation, qui est une sorte de synthèse de tout internet, en une synthèse des différents experts ayant participé à rédiger les conversations d'entraînement.

La procédure d'entraînement est la même que pour le modèle de fondation (prédiction du camembert pour le prochain *token*, et ajustement des paramètres en fonction du *token* réel) ; mais il y a une nouveauté

10. Avec les termes *pre-training* et *post-training*, on se demande à quel moment se passe le "*training*".. En fait, l'entraînement est la combinaison du *pre-training* et du *post-training*.
11. https://github.com/meta-llama/llama-cookbook

importante : le rajout de certains *tokens* spéciaux. Ci-dessous, on a affiché l'encodage en *tokens* d'un échange utilisateur-assistant, juste avant qu'il soit donné au modèle pour l'entraînement.

```
<|im_start|>user<|im_sep|>Combien de livres Hunger Gam
es existe-t-il ? ça fait environ combien de pages ?<|i
m_end|><|im_start|>assistant<|im_sep|>Il existe 4 livr
es Hunger Games: -Hunger Games (384 pages) -Hunger Gam
es: L'embrasement (400 pages) -Hunger Games: La Révolt
e (339 pages) -Hunger Games: La Ballade du Serpent de
l'Oiseau Chanteur (1712 pages) 384+400+339+1712= 2835
pages (Le nombre de page est calculé à partir des édit
ions originales de ces livres, et non des versions fra
nçaises)<|im_end|><|im_start|>assistant<|im_sep|>
```

Figure 12 – Encodage d'un échange utilisateur-assistant (38) (39)

Pour que le modèle comprenne que c'est à lui de formuler une réponse, des *tokens* spéciaux sont rajoutés à son vocabulaire. Ils sont ensuite intégrés artificiellement aux données d'entraînement et, lors de l'apprentissage, le modèle comprend qu'il doit formuler une réponse juste après les *tokens* : "<|im_start|>", "assistant", "<|im_stop|>". Il existe d'autres *tokens* spéciaux ; par exemple pour utiliser des outils (4). En rajoutant un *token* encodant le fait d'aller chercher une information sur internet, il est possible de conditionner le modèle à produire ce *token* dès qu'il n'est pas sûr d'une réponse. Le serveur donnant accès au modèle peut alors détecter ce *token* et insérer dans le *prompt* le résultat de la recherche. Un autre exemple est un *token* signalant le début de l'exécution de code, généré par le modèle dès qu'il comprend qu'il faut utiliser du code pour répondre à une question.

À chaque fois, le modèle comprend quand utiliser ces *tokens* spéciaux car il a vu de nombreux exemples d'utilisation dans son jeu de données d'entraînement.

2.3.2 Ajustement par renforcement

L'objectif de l'ajustement par renforcement est de laisser le modèle s'exercer. L'idée est de laisser le modèle produire des réponses pour une question donnée, et de **juger** lesquelles sont les meilleures, afin d'encourager le modèle à produire ce genre de réponses à l'avenir.

Cependant, lorsqu'une question n'admet pas de bonne ou de mauvaise réponse, par exemple : "Fais moi une blague sur les ordinateurs", il est complexe de juger de la qualité d'une réponse. Une solution utilisée (notamment pour ChatGPT 3 (40)) est d'entraîner un **autre** modèle à prédire un score de qualité sur les réponse du LLM. Ces modèles sont appelés des *reward models* (modèles de récompenses) ; et sont entraînés avec des **retours humains** [12], pour donner des scores ressemblant à ce qu'un humain prédirait. En utilisant ce modèle auxiliaire, on peut dire au modèle de langage quelles réponses sont les plus appropriées parmi celles qu'il nous a données. Il a été observé que cela développe sa capacité à se comporter comme un humain (40).

3 Comment représenter des *tokens* avec des nombres ? Les *embeddings*

Dans cette partie, on présente comment les *tokens* sont utilisés par les modèles. C'est l'étape d'*embedding*, qui désigne la transformation des *tokens* en des séquences de nombres (**vecteurs**).

12. Le terme précis : RLHF, *Reinforcement Learning with Human Feedback*.

3.1 Vecteurs

On commence par expliquer brièvement ce qu'est un **vecteur** dans notre contexte. C'est en pratique tout simplement une liste de nombres. Par exemple :

$$\begin{bmatrix} 0 \\ 1.2 \\ 0.2 \end{bmatrix}$$

est un vecteur. La longueur de la liste (ici 3) est appelée **dimension du vecteur**. Chacun des élément du vecteur est appelé une "composante", ou "coordonnée". Un point crucial est qu'il est possible d'effectuer des opérations sur des vecteurs. On définit ainsi la somme de 2 vecteurs comme un nouveau vecteur, dont les composantes sont la somme des composantes des 2 vecteurs. Par exemple :

$$\begin{bmatrix} 0 \\ 1.2 \\ 0.2 \end{bmatrix} + \begin{bmatrix} 18 \\ 2.1 \\ 0.9 \end{bmatrix} = \begin{bmatrix} 18 \\ 3.3 \\ 1.1 \end{bmatrix}.$$

Un vecteur à 2 dimensions peut être **représenté** dans un plan, et à 3 dimensions dans **l'espace**. Prenons l'exemple (Figure 13) des deux vecteurs :

$$\begin{bmatrix} 3 \\ -2 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Une raison pour visualiser les vecteurs par une flèche issue de l'origine, qui pointe sur les coordonnées du vecteur, est que ça permet de bien comprendre ce que représente la somme de 2 vecteurs. Le vecteur obtenu en sommant les deux premiers correspond simplement à la composition des deux flèches. Il faut imaginer qu'un vecteur encode un déplacement, et que la somme encode la composition des déplacements.

Si le vecteur a plus de 3 dimensions, on peut garder l'intuition qu'il correspond à une flèche, mais dans un espace que l'on ne peut pas trop visualiser ! Malheureusement, c'est très souvent le cas…

3.2 Embedding

En traitement du langage, *l'embedding* correspond à la transformation du texte en **vecteurs**. La dimension des vecteurs varie, mais peut être de plusieurs centaines - jusqu'à plusieurs milliers ! Difficile alors de se représenter ces vecteurs. Cette représentation des *tokens* est utile car la traduction *tokens-vecteurs* est faite pour que 2 séquences de tokens sémantiquement proches (par exemple "chocolat" et "cacao") aient des vecteurs dont les directions sont proches. Les usages sont très nombreux et ne s'arrêtent pas à la génération de texte : calcul de similarité de 2 phrases, classification selon le sens d'un texte, recherche documentaire, … Selon l'usage, les vecteurs dont on a besoin peuvent être radicalement différents. On pourra consulter le site SBERT [13], présentant des implémentations.

> **Attention**, le terme *embedding* désigne à la fois l'action de transformer un *token* en un vecteur, et le vecteur obtenu.

On recense plusieurs méthodes pour transformer des *tokens* (ou directement des mots) en vecteurs. Certaines existaient avant la popularisation des LLM.

Par exemple Word2Vec (41) ; issu de chez Google en 2013. Word2Vec désigne deux modèles de *Machine Learning* ; capables de transformer des mots en vecteurs. Pour jouer avec ces représentations, vous pouvez regarder **Cemantix** [14], un jeu qui consiste tout simplement à deviner un mot, et qui s'appuie justement sur les

13. Sentence Transform (SBERT) : https://sbert.net/index.html.
14. Cemantix : https://cemantix.certitudes.org/.

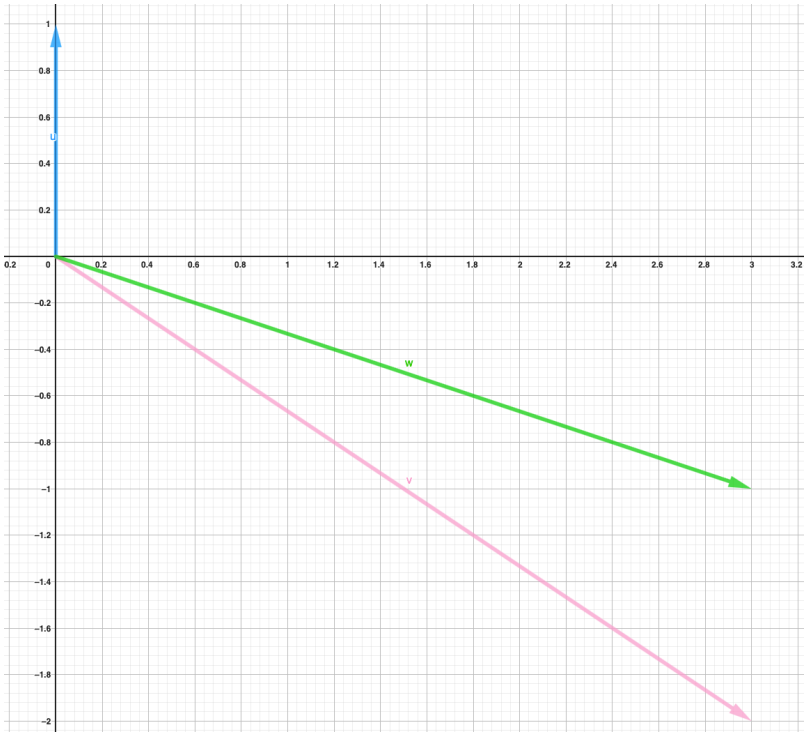

Figure 13 – Le vecteur vert est la somme du rose et du bleu. Il pointe sur l'endroit où on atterrit après avoir suivi le rose, puis le bleu.

distances entre vecteurs, obtenues avec Word2Vec. Il est intéressant de remarquer que ces deux modèles sont entraînés sur une autre tâche, et que la représentation des mots en vecteur est en quelque sorte une étape intermédiaire qui permet aux modèles de bien réaliser cette tâche. Le premier modèle est entraîné à remplir un texte avec un unique mot manquant, au centre (un peu comme les LLM finalement) ; et le second à trouver les mots en entourant un autre (41). Comme il est complexe de déterminer si un *embedding* est correct ; on est un peu obligé de passer par ce genre de chemin détourné pour entraîner des modèles d'*embeddings*. Remarquons qu'une fois les *embeddings* calculés pour tous les mots d'un vocabulaire, la traduction mot-vecteur est très rapide (il suffit de regarder dans le dictionnaire !). Dans le même style, on peut citer GloVe (42) ; de la même époque.

Dans le cas des LLM, les *tokens* sont transformés en vecteur dès leur entrée dans le modèle. Les composantes de chaque vecteur *d'embedding* du vocabulaire sont initialisées au hasard, puis **apprises** lors de l'entraînement : ce sont des paramètres du modèle. En un certain sens, comme avec Word2Vec, les *embeddings* sont une conséquence de l'entraînement du LLM à prédire le prochain *token* d'une séquence donnée. Pour mieux prendre en compte les informations sur la position de chaque *token* dans une phrase, les LLM ont en plus des "*positional embeddings*" (vecteurs de position). Ce sont d'autres vecteurs, appris lors de l'entraînement ou non, qui encodent la position d'un *token* dans une phrase. Là encore, plusieurs variantes existent (*positional embedding* (43), *rotary positional embedding* (44) (45) (46)).

Une des petites révolution des LLM consiste à utiliser une architecture appelée "transformer" (43), qui permet aux *embeddings* de chacun des *tokens* (sous forme de vecteur) d'être modifiés pour prendre en compte les autres *tokens* de la séquence. En utilisant des *transformers*, il est donc possible de faire évoluer les *embeddings* initiaux (appris pendant l'entraînement) pour qu'ils deviennent plus aboutis (deux *embeddings* du même mot seront alors différent selon le contexte). Ceci donne lieu aux modèles *d'embeddings* modernes ; dans lesquels l'étape de traduction doit être refaite à chaque fois que l'on veut transformer un *token* en vecteur dans un contexte précis. Impossible dans ce cas de créer un dictionnaire *tokens*-vecteurs ! Ces modèles sont alors plus coûteux que les anciens (Word2Vec par exemple) à utiliser, car les *transformer*

restent des architectures avec beaucoup de paramètres.

Récemment, un retour en arrière est observé : les *embeddings statiques* (47). Ils fonctionnent lors de la traduction comme les modèles style Word2Vec : ce sont des dictionnaires *tokens*-vecteurs. Un exemple est Model2Vec (48). C'est une technique permettant d'utiliser des modèles prenant en compte le contexte (*transformer*) pour créer un dictionnaire. Une fois créé, le dictionnaire ne permet pas de prendre en compte le contexte, et l'*embedding* d'un *token* est toujours le même. Néanmoins, la technique est suffisamment au point pour que cela n'affecte pas trop la qualité des *embeddings*.

4 Écosystème et historique des LLM

Les grands modèles de langage ont profondément transformé le domaine du traitement automatique du langage grâce à l'augmentation importante des ressources informatiques, à la disponibilité accrue de grandes bases de données textuelles et à l'innovation majeure des architectures Transformer. Cette partie explique clairement ce qui différencie les LLM et retrace leur évolution technologique et commerciale.

4.1 Différencier les LLM

Les modèles de langage se distinguent notamment selon leur accès, leur taille, leur contexte et leur application.

Ouverture des modèles

Bien que de nombreux modèles de langage se revendiquent ouverts ou « open source », leur transparence réelle varie considérablement. L'ouverture d'un modèle implique plusieurs critères : disponibilité du code source, accès aux données et poids d'entraînement, existence de licences explicites, ainsi que qualité et exhaustivité de la documentation associée notamment concernant la phase d'entraînement du modèle.

Il apparaît que certains modèles offrent une transparence significative sur la plupart de ces critères (par exemple, OLMo (49) d'Allen AI, qui publie ses données d'entraînement, son code source et des documentation détaillées), alors que d'autres modèles populaires demeurent essentiellement opaques (DeepSeek R1 (50), dont les données d'entraînement restent fermées, ou Llama 3.3 (51), qui nécessite une licence spécifique et dont les données ne sont pas divulguées).

Ainsi, le degré d'ouverture influence directement la possibilité d'utiliser ces modèles dans des contextes scientifiques, éducatifs ou responsables.

Des ressources comme Opening Up ChatGPT (52) et l'Open Source AI Index (53) proposent des évaluations complètes et à jour du degré d'ouverture des modèles de langage. On pourra également consulter le tableau Figure 14.

Accès aux modèles

Les **modèles fermés** comme GPT-4 (54) ou Claude (55) sont accessibles uniquement *via* des interfaces propriétaires (API [15] ou applications web), sans possibilité d'accéder à leur code ou à leurs paramètres internes. Cela facilite leur utilisation mais limite la transparence. Les **modèles à tendance *open-source*** comme LLaMA (56) offrent un accès complet à leur code source et à leurs paramètres, permettant ainsi une utilisation plus flexible, notamment pour des développements spécifiques.

Taille et nombre de paramètres

15. API : *Application Programming Interface*, désigne ici une manière automatisée (machine à machine) d'accéder au modèle. Ceci permet par exemple de créer un programme informatique qui utilise de manière automatique des modèles hébergés ailleurs que sur son ordinateur.

Project	Availability						Documentation						Access	
	Open code	LLM data	LLM weights	RL data	RL weights	License	Code	Architecture	Preprint	Paper	Modelcard	Datasheet	Package	API
OLMo 7B Instruct	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	~
BLOOMZ	✓	✓	✓	✓	~	~	✓	✓	✓	✓	✓	✓	✗	✓
AmberChat	✓	✓	✓	✓	✓	✓	~	~	✓	✗	~	~	✗	✓
Open Assistant	✓	✓	✓	✓	✗	✓	✓	✓	~	✗	✗	✗	✓	✓
OpenChat 3.5 7B	✓	✗	✓	✗	✓	✓	~	✓	✓	✓	~	✗	✓	~
Pythia-Chat-Base-7…	✓	✓	✓	✓	✗	✓	✓	✓	~	✗	~	~	✓	✗
Cerebras GPT 111…	~	✓	✓	✓	✓	~	✗	✓	~	✗	✗	✓	✗	✓
RedPajama-INCITE…	~	✓	✓	✓	✓	~	~	~	✗	✗	✓	✓	✗	~
dolly	✓	✓	✓	✓	✗	✓	✓	✓	~	✗	✗	✗	✓	✗
Tulu V2 DPO 70B	✓	✗	~	✓	✓	~	~	~	✓	✗	~	~	✗	✓
MPT-30B Instruct	✓	~	✓	~	✗	✓	✓	~	✗	✗	~	✗	✓	~
MPT-7B Instruct	✓	~	✓	~	✗	✓	✓	~	✗	✗	✓	✗	✓	✗
trlx	✓	✓	✓	~	✗	✓	✓	~	✗	✗	✗	✗	~	✓
Vicuna 13B v 1.3	✓	~	✓	✗	✗	~	✓	✗	✓	✗	~	✗	✓	~
minChatGPT	✓	✓	✓	~	✗	✓	✓	~	✗	✗	✗	✗	✗	✓
ChatRWKV	✓	~	✓	✗	✗	✓	~	~	~	✗	✗	✗	✓	~
BELLE	✓	~	~	~	~	✗	~	✓	✓	✗	✗	~	✗	✗
WizardLM 13B v1.2	~	✗	~	✓	✓	~	~	✓	✓	✗	✗	✗	✗	✗
Airoboros L2 70B G…	~	✗	~	✓	✓	~	~	~	✗	✗	~	~	✗	✗
ChatGLM-6B	~	~	✓	✗	✗	✓	~	~	✗	~	✗	✗	✗	✓
Mistral 7B-Instruct	~	✗	✓	✗	~	✓	✗	~	~	✗	✗	✗	~	✓
WizardLM-7B	~	~	✗	✓	~	~	~	✓	✓	✗	✗	✗	✗	✗
Qwen 1.5	~	✗	✓	✗	✓	✗	~	~	✗	✗	✗	✗	~	✓
StableVicuna-13B	~	✗	~	~	~	~	~	~	~	✗	~	✗	✗	~
Falcon-40B-instruct	✗	~	✓	~	✗	✓	✗	~	~	✗	~	✗	✗	✗
UltraLM	✗	✗	~	✓	~	✗	✗	~	✓	✗	~	~	✗	✗
Yi 34B Chat	~	✗	✓	✗	✓	~	✗	✗	✓	✗	✗	✗	✗	~
Koala 13B	✓	~	~	~	✗	~	~	~	✗	✗	✗	✗	✗	✗
Mixtral 8x7B Instruct	✗	✗	✓	✗	~	✓	✗	~	~	✗	✗	✗	~	✗
Stable Beluga 2	✗	✗	~	✗	✓	~	✗	~	~	✗	~	✗	✗	~
Stanford Alpaca	✓	✗	~	~	~	✗	~	✓	✗	✗	✗	✗	✗	✗
Falcon-180B-chat	✗	~	~	~	~	✗	✗	~	~	✗	~	✗	✗	✗
Orca 2	✗	✗	~	✗	✓	✗	✗	~	~	✗	~	✗	✗	~
Command R+	✗	✗	✗	✓	✓	~	✗	✗	✗	✗	~	✗	✗	✗
Gemma 7B Instruct	~	✗	~	✗	~	✗	✗	~	✗	✗	✓	✗	✗	✗
LLaMA2 Chat	✗	✗	~	✗	~	✗	✗	~	~	✗	~	✗	✗	~
Nanbeige2-Chat	✓	✗	✗	✗	✓	~	✗	✗	✗	✗	✗	✗	✗	~
Llama 3 Instruct	✗	✗	~	✗	~	✗	✗	~	✗	✗	~	✗	✗	~
Solar 70B	✗	✗	~	✗	~	✗	✗	✗	✗	✗	~	✗	✗	~
Xwin-LM	✗	✗	~	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	~
ChatGPT	✗	✗	✗	✗	✗	✗	✗	✗	~	✗	✗	✗	✗	✗

Figure 2: Openness of 40 text generators described as open, with OpenAI's ChatGPT (bottom) as closed reference point. Every cell records a three-level openness judgement (✓ open, ~ partial or ✗ closed). The table is sorted by cumulative openness, where ✓ is 1, ~ is 0.5 and ✗ is 0 points. RL may refer to RLHF or other forms of fine-tuning aimed at fostering instruction-following behaviour. For the latest updates see: https://opening-up-chatgpt.github.io

Figure 14 – Tableau recensant le degré d'ouverture des grands modèles de langage entraînés par instructions - (52)

La taille d'un LLM impacte directement ses capacités et ses besoins en ressources. Les modèles **petits** (quelques millions à quelques centaines de millions de paramètres) fonctionnent facilement sur des ordinateurs standards et conviennent pour des tâches simples.

Les modèles **grands** (plusieurs dizaines à centaines de milliards de paramètres) nécessitent une infrastructure importante, et sont adaptés pour des tâches complexes comme la traduction ou la génération détaillée de texte.

Taille du contexte

Le contexte est la quantité de texte (en nombre de *tokens*) que le modèle peut traiter en une seule fois. Des modèles comme Gemini 1.5 (57) traitent des contextes allant jusqu'à un million de *tokens*, permettant ainsi de gérer des textes ou documents très longs.

Domaine d'application

Certains modèles sont spécialisés dans des domaines précis (médical, juridique, financier), offrant une grande précision mais limitée à ce secteur. D'autres modèles restent généralistes, permettant une utilisation polyvalente.

4.2 Historique des LLM

L'évolution des LLM peut être divisée en plusieurs grandes périodes historiques.

4.2.1 1990-2010 : Approches statistiques et neuronales

Les premiers modèles de langage s'appuyaient sur des n-grammes (58) (séquences de n mots consécutifs, permettant d'estimer la probabilité d'apparition des mots) (59). Ils exploitaient les statistiques de co-occurrence de mots, mais étaient limités aux séquences courtes en raison d'une complexité exponentielle (60). Vers la fin des années 1990, l'utilisation de réseaux neuronaux récurrents (RNN (61) - *Recurrent Neural Network*) – c'est-à-dire des modèles qui traitent l'information séquentiellement en conservant une mémoire des éléments précédents – a été introduite (61). Ces approches ont rapidement évolué avec l'émergence des LSTM (62) (Long Short-Term Memory, des réseaux capables de conserver l'information sur de longues périodes grâce à des mécanismes de mémoire) et, plus tard, des GRU (63) (Gated Recurrent Units) une variante simplifiée des LSTM utilisant des portes pour contrôler le flux d'information (62).

Les réseaux neuronaux récurrents comme les LSTM présentaient des difficultés d'entraînement sur de grands corpus, notamment à cause de la complexité des calculs nécessaires. L'arrivée des réseaux neuronaux à propagation avant (*feed-forward*), comme le Neural Network Language Model (NNLM) (64), a introduit les représentations distribuées des mots (*word embeddings*). Ces représentations vectorielles ont permis de saisir les similarités sémantiques et les relations complexes entre les mots, dépassant les limitations des approches statistiques traditionnelles basées sur les n-grammes.

Cette période est également liée à l'amélioration des techniques de lissage (pour attribuer des probabilités non nulles aux séquences jamais observées dans les données d'entraînement) pour les modèles n-grammes (58), et leur optimisation pour des corpus de plusieurs milliards de mots (65). Ensuite, Mikolov et al. (66) ont commencé à explorer les réseaux de neurones récurrents pour la modélisation du langage, introduisant notamment des architectures RNN optimisées qui réduisaient considérablement le temps d'entraînement tout en améliorant les performances prédictives. Ces travaux ont posé les fondations conceptuelles et techniques des futures architectures comme les GRU (67), en démontrant la viabilité des approches neuronales récurrentes pour capturer des dépendances à long terme dans le langage.

4.2.2 2011-2016 : Word Embeddings et avancées RNN

Word2Vec (41) et GloVe (42) ont introduit des méthodes pour convertir les mots en vecteurs denses, dans lesquels la proximité entre les vecteurs reflète la similarité sémantique des mots. Ces techniques, en permettant de placer les mots dans un espace numérique à dimension réduite, ont amélioré l'efficacité des réseaux neuronaux récurrents. La diffusion de bibliothèques d'apprentissage profond, tel que TensorFlow, a aussi accéléré ces progrès. Durant cette période, le mécanisme d'attention, procédé qui permet de pondérer l'importance de chaque élément d'une séquence lors de la prédiction en calculant des scores de pertinence, a commencé à être exploité dans des applications telles que la traduction automatique (67).

Avant ces innovations, les mots étaient souvent représentés sous forme "*one-hot*" (un vecteur avec un seul 1 à l'index du mot et des 0 ailleurs), ce qui ne permettait pas de capturer la similitude entre les mots et engendrait des vecteurs de très grande dimension. Les méthodes de *word embeddings* comme Word2Vec ou GloVe viennent plutôt fournir une représentation compacte et informative.

Avancées des RNN et mécanisme d'attention

L'intégration des word embeddings a amélioré les représentations d'entrée des réseaux de neurones récurrents (RNN), ce qui a permis d'optimiser leurs performances pour des tâches comme la classification de textes ou la traduction automatique. L'arrivée de frameworks tels que Theano, TensorFlow et PyTorch (27) a facilité l'expérimentation et la mise en œuvre de modèles plus avancés, comme les architectures LSTM (Long Short-Term Memory) et GRU (Gated Recurrent Units).

Cependant, ces modèles éprouvaient des difficultés à gérer des dépendances lointaines. Ils peinaient à se souvenir des informations antérieures et à relier des mots éloignés dans le texte. L'introduction du mécanisme d'attention, associée à la montée en puissance du calcul parallèle sur GPU, a ouvert la voie aux architectures sans récurrence, annonçant l'émergence des Transformers.

Le mécanisme d'attention (68) permet au modèle d'attribuer un score de pertinence à chaque mot d'une séquence. Contrairement aux méthodes antérieures, qui réduisaient l'ensemble de la séquence à un vecteur fixe, cette approche crée un vecteur contextuel différent pour chaque étape de la prédiction, en fonction de la position du mot. Ce procédé améliore la prise en compte des relations entre mots éloignés et, par conséquent, la qualité des prédictions.

4.2.3 2017-2019 : Apparition des Transformers

L'introduction de l'architecture Transformer par Vaswani et al. (43), présentée dans « Attention is All You Need », représente une rupture par rapport aux approches séquentielles des RNN. Le Transformer exploite le mécanisme d'attention multi-têtes pour analyser simultanément toutes les positions d'une séquence, réduisant ainsi le temps d'entraînement et capturant les dépendances à longue distance. Le self-attention calcule pour chaque mot un score de pertinence par rapport aux autres, permettant de générer un vecteur contextuel adapté à chaque étape de prédiction. Cette structure encodeur-décodeur a ouvert la voie à des modèles de plus en plus performants, comme GPT-1 et BERT en 2018, suivis par GPT-2, XLNet et RoBERTa en 2019, qui ont tous contribué à améliorer les performances sur diverses tâches.

Modèles phares de cette période

GPT-1 (Generative Pre-trained Transformer 1) (69) : GPT-1 a été l'un des premiers modèles à démontrer l'efficacité du pré-entraînement massif d'un Transformer (en utilisant uniquement la partie décodeur) pour la génération de texte et le transfert vers d'autres tâches. Il utilise 117 millions de paramètres.

BERT (Bidirectional Encoder Representations from Transformers) (29) : BERT utilise la partie encodeur du Transformer et est pré-entraîné sur deux tâches : la prédiction de mots masqués ("Masked Language Modeling") et la prédiction de la phrase suivante ("Next Sentence Prediction"). Cette approche bidirectionnelle (considérant à la fois le contexte gauche et le contexte droit) permet à BERT de capturer des représentations contextuelles très riches. BERT existe en différentes tailles, la plus grande ayant 340 millions de paramètres (BERT-Large).

GPT-2 (70) : GPT-2 est une version beaucoup plus grande de GPT-1 (jusqu'à 1,5 milliard de paramètres). Il a été entraîné sur un corpus encore plus vaste et a démontré des capacités de génération de texte impressionnantes, parfois difficiles à distinguer d'un texte écrit par un humain.

XLNet (71) : XLNet est un modèle qui combine les avantages de l'approche autorégressive (comme GPT) et de l'approche bidirectionnelle (comme BERT). Il utilise une méthode de pré-entraînement appelée "Permutation Language Modeling" qui lui permet de prendre en compte toutes les permutations possibles de l'ordre des mots dans une phrase.

RoBERTa (Robustly Optimized BERT Pretraining Approach) (72) : RoBERTa est une version optimisée de BERT. Les auteurs ont montré qu'en modifiant certains hyperparamètres, en entraînant le modèle plus longtemps et sur plus de données, et en supprimant la tâche de "Next Sentence Prediction", ils pouvaient améliorer significativement les performances de BERT.

4.2.4 2020-2021 : Montée en puissance et adoption

Entre 2020 et 2021, les modèles de langage de très grande échelle émergent, qui, en combinant augmentation de taille et apprentissage en contexte, ont transformé l'approche des tâches de traitement du langage.

GPT-3 (73) (développé par OpenAI) a repoussé les limites de la taille des modèles en intégrant 175 milliards de paramètres, soit plus de 100 fois la taille des modèles précédents comme BERT-Large, et plus de dix fois la taille de GPT-2. L'apparition de ce modèle introduit des capacités inédites telles que le *few-shot learning* (voir ci-dessous), sans nécessiter un entraînement spécifique pour chaque tâche. Pour la première fois, un modèle pouvait s'adapter dynamiquement à de nouvelles tâches simplement à travers des instructions textuelles et quelques exemples dans le prompt, là où les approches précédentes nécessitaient systématiquement un *fine-tuning* avec des milliers d'exemples étiquetés pour chaque nouvelle application.

- **Zero-Shot** : Le modèle est capable d'effectuer une tâche sans avoir été explicitement entraîné sur des exemples de cette tâche. On lui donne simplement une description de la tâche en langage naturel (par exemple, "Traduire de l'anglais au français : cheese =>") et il génère la sortie attendue ("fromage").
- **One-Shot** : Le modèle reçoit un seul exemple de la tâche à accomplir (par exemple, "Traduire de l'anglais au français : sea otter => loutre de mer, cheese =>") avant de générer la sortie ("fromage").
- **Few-Shot** : Le modèle reçoit quelques exemples (typiquement entre 2 et une dizaine, voire une centaine) de la tâche (par exemple, plusieurs paires de mots anglais-français) avant d'effectuer la tâche sur une nouvelle entrée. L'idée est de montrer au modèle le format attendu de la sortie, ainsi que quelques exemples de la relation entrée-sortie.
- **In-context learning** : L'ensemble de ces méthodes (zero-shot, one-shot et few-shot) sont des formes de "in-context learning". Le modèle apprend à effectuer une tâche à partir d'instructions ou d'exemples fournis dans le contexte de la requête, sans modification de ses paramètres (contrairement au fine-tuning).

Ces approches réduisent considérablement, voire éliminent, le besoin de grands ensembles de données étiquetées pour chaque nouvelle tâche. Cela rend les modèles de langage beaucoup plus polyvalents et faciles à adapter à de nouveaux usages tout en se rapprochant de la capacité humaine à apprendre rapidement à partir de quelques exemples. Des modèles comme T5 (Text-to-Text Transfer Transformer) (74) et Megatron-LM (75) ont poursuivi la tendance à l'augmentation de la taille et des capacités.

Diversification et spécialisation des modèles

- T5 (Text-to-Text Transfer Transformer) (74) développé par Google, est un modèle unifié qui reformule toutes les tâches de NLP comme des tâches de texte à texte. Par exemple, la traduction est formulée comme "Translate English to German : That is good. =>", la classification de sentiment comme "Classify the sentiment of this sentence as positive or negative : I love this movie. =>", etc. Cette approche permet de pré-entraîner un seul modèle sur un mélange de tâches et de le fine-tuner facilement sur de nouvelles tâches. T5 a été pré-entrainé sur le corpus "Colossal Clean Crawled Corpus" (C4).

— Megatron-LM (75) (et développements ultérieurs) : Megatron-LM est un projet de NVIDIA visant à entraîner de très grands modèles de langage Transformer. Il se concentre sur les techniques permettant de distribuer l'entraînement sur de nombreux GPU, car les modèles de cette taille ne peuvent pas tenir sur un seul appareil.
— Codex (OpenAI) : Codex est un exemple de spécialisation des modèles de langage. Il est basé sur GPT-3, mais il est fine-tuné sur un grand corpus de code source provenant de dépôts publics (comme GitHub). Codex est capable de générer du code à partir de descriptions en langage naturel, d'aider à compléter du code, de traduire entre différents langages de programmation, et d'expliquer le fonctionnement du code. Il alimentait des outils comme GitHub Copilot.

4.2.5 2022 : Démocratisation et *open-source*

Fin 2022, ChatGPT, basé sur une version améliorée de GPT-3 (souvent appelée GPT-3.5 ou gpt-3.5-turbo), a permis de rendre les grands modèles de langage accessibles au grand public via une interface web intuitive. Contrairement aux versions antérieures, disponibles uniquement via une API payante pour les développeurs, ChatGPT offre une interaction en langage naturel qui ne requiert aucune compétence technique particulière. Les réponses, à la fois fluides et pertinentes – capables d'expliquer des concepts complexes, de rédiger des textes dans divers styles ou même de générer du code – ont suscité un véritable effet viral sur les réseaux sociaux, transformant ainsi l'usage des LLM en une expérience quotidienne.

2022 a également marqué l'accélération du mouvement open-source des LLM. Des projets comme BLOOM, développé par le collectif BigScience (rassemblant plus de 1000 chercheurs et entraîné sur le supercalculateur Jean Zay), offrent un modèle multilingue dont le code, les données d'entraînement et les poids sont entièrement accessibles, permettant de générer du texte dans 46 langues et 13 langages de programmation. De leur côté, les modèles tels qu'OPT de Meta AI – bien que les versions les plus volumineuses soient soumises à des licences restrictives – ainsi que LLaMA et StableLM (76), illustrent comment l'open-source contribue à démocratiser l'accès, accélère la recherche, et renforce la transparence des modèles en facilitant l'audit des systèmes.

4.2.6 2023-2024 : Multimodalité et spécialisation

À partir de 2023, les LLM intègrent la multimodalité, c'est-à-dire la capacité de traiter et générer simultanément du texte et des images. Par exemple, GPT-4, notamment dans sa version multimodale souvent désignée GPT-4V (pour « Vision »), peut analyser des images, répondre à des questions sur leur contenu ou générer des légendes explicatives. D'autres modèles, tels que Flamingo (77) de DeepMind ou PaLM-E (78) de Google, combinent des capacités linguistiques avec des encodeurs visuels pour étendre leur application à des domaines comme la robotique, tandis que LLaVA (79) émerge comme initiative open-source.

La concurrence s'est également intensifiée avec l'arrivée de nouveaux acteurs et l'émergence de modèles spécialisés. Des systèmes comme Claude d'Anthropic (55), Gemini de Google (57) et LLaMA de Meta (23), proposés en différentes tailles et souvent accompagnés de versions optimisées pour le dialogue, offrent des fenêtres de contexte considérablement étendues – jusqu'à 1 million de tokens pour Gemini 1.5 Pro (57) ou 200k pour Claude 3 (55) – permettant de traiter des textes plus longs et complexes. Des modèles dédiés à des tâches spécifiques, tels que ceux pour la traduction automatique, la génération de code (comme Codex (80) ou Code LLaMA (81)), la rédaction scientifique, l'analyse financière ou encore la réponse à des questions médicales (par exemple, Med-PaLM (82) de Google (82) et BioBERT (83)), ainsi que dans le domaine juridique avec LegalBERT (84), indiquent une spécialisation croissante des LLMs.

4.3 Accès et déploiement des LLM

Différentes méthodes existent pour utiliser ces modèles selon les besoins :

Interfaces en ligne :

Mistral, ChatGPT, Claude.ai ou Gemini proposent des interfaces faciles d'accès via navigateur ou API.

— Mistral : https://chat.mistral.ai/chat,
— ChatGPT :https://chatgpt.com/,
— Claude.ai : https://claude.ai,
— Gemini : https://gemini.google.com/?hl=fr.

Déploiement local :

Des outils simples (Ollama (85), LM Studio (86), llama.cpp (87)) permettent d'utiliser les modèles localement pour garantir confidentialité et contrôle.

— Ollama : https://ollama.com/search,
— LM Studio : https://lmstudio.ai/models,
— llama.cpp : https://github.com/ggml-org/llama.cpp.

API pour développeurs :

OpenAI, Anthropic, Google Gemini et Mistral AI proposent des API pour intégrer facilement ces modèles dans des applications.

— API de OpenAI : https://platform.openai.com/docs/overview,
— API de Claude par Anthropic : https://www.anthropic.com/api,
— API de Google Gemini : https://ai.google.dev/,
— API de Mistral AI : https://docs.mistral.ai/api/.

Plateformes de modèles :

HuggingFace et Replicate (88) facilitent la comparaison et l'évaluation des performances des différents modèles disponibles.

— HuggingFace spaces : https://huggingface.co/spaces,
— Replicate : https://replicate.com/explore.

L'écosystème des LLM est en constante évolution, apportant régulièrement de nouvelles possibilités ainsi que des défis techniques et éthiques à considérer.

5 Bibliographie

1) Esser, Patrick, et al. *Scaling Rectified Flow Transformers for High-Resolution Image Synthesis*. 2024, https://arxiv.org/abs/2403.03206.
2) Louapre, David. *Comment Fonctionne ChatGPT ?* https://scienceetonnante.com/2023/04/14/comment-fonctionne-chatgpt/. Accessed 4 Feb. 2025.
3) Sabatou, Alexandre, et al. *ChatGPT, Et Après ? Bilan Et Perspectives de l'intelligence Artificielle*. 2024, https://www.senat.fr/notice-rapport/2024/r24-170-notice.html.
4) Karpathy, Andrej. *Deep Dive into LLMs Like ChatGPT*. https://www.youtube.com/watch?v=7xTGNNLPyMI. Accessed 25 Feb. 2025.
5) CNRS. *Fidle*. 2025, https://fidle.cnrs.fr/w3/.
6) CNRS, IDRIS -. *Panoram'IA*. https://www.youtube.com/@idriscnrs. Accessed 12 Mar. 2025.
7) Jamil, Umar. *YouTube*. https://www.youtube.com/@umarjamilai. Accessed 5 Mar. 2025.
8) Liu, Yiheng, et al. *Understanding LLMs : A Comprehensive Overview from Training to Inference*. 2024, https://arxiv.org/abs/2401.02038.
9) *Consommation Quotidienne Brute Régionale*. https://odre.opendatasoft.com/explore/dataset/consommation-quotidienne-brute-regionale. Accessed 3 Feb. 2025.
10) *Température Quotidienne Régionale*. https://odre.opendatasoft.com/explore/dataset/temperature-quotidienne-regionale. Accessed 3 Feb. 2025.

11) Redmon, Joseph, et al. “YOLOv3 : An Incremental Improvement.” *arXiv*, 2018, https://pjreddie.com/darknet/yolo/.
12) Véry, Thibaut. *Alphafold2 Et La Prédiction de Structures de Protéines*. https://www.youtube.com/watch?v=YEHXMsowhxg. Accessed 6 Feb. 2025.
13) *AlphaFold - Probable Disease Resistance Protein At1g58602 - Google Deep Mind, EMBL-EBI*. https://alphafold.com/entry/Q8W3K0. Accessed 3 Feb. 2025.
14) https://time.com/6247678/openai-chatgpt-kenya-workers/. Accessed 4 Feb. 2025.
15) *Google Turns to Nuclear to Power AI Data Centres*. https://www.bbc.com/news/articles/c748gn94k95o. Accessed 6 Feb. 2025.
16) Lefèvre, Laurent, et al. *IA Générative Sobre : Un Oxymore ?* https://2024.jres.org/programme#modal-119. Accessed 6 Feb. 2025.
17) *Fiche de Criticité - Tantale*. https://www.mineralinfo.fr/sites/default/files/documents/2020-12/fichecriticitetantale-publique20200131.pdf. Accessed 6 Feb. 2025.
18) *Du Sang Dans Nos Cellulaires*. https://ici.radio-canada.ca/info/2019/05/coltan-republique-democratique-congo-mines-enfants/. Accessed.
19) *Tom’s Hardware*. https://www.tomshardware.com/pc-components/gpus/datacenter-gpu-service-life-can-be-surprisingly-short-only-one-to-three-years-is-expected-according-to-unnamed-google-architect. Accessed 6 Feb. 2025.
20) *NVIDIA*. https://www.nvidia.com/en-us/sustainability/product-recycling/. Accessed 6 Feb. 2025.
21) *NVIDIA Corporate Sustainability Report*. https://images.nvidia.com/aem-dam/Solutions/documents/FY2024-NVIDIA-Corporate-Sustainability-Report.pdf. Accessed 6 Feb. 2025.
22) *OpenAI Pleads That It Can’t Make Money Without Using Copyrighted Materials for Free*. https://futurism.com/the-byte/openai-copyrighted-material-parliament. Accessed 6 Feb. 2025.
23) Touvron, Hugo, et al. *LLaMA : Open and Efficient Foundation Language Models*. 2023, https://arxiv.org/abs/2302.13971.
24) Luccioni, Sasha, et al. “Power Hungry Processing : Watts Driving the Cost of AI Deployment ?” *The 2024 ACM Conference on Fairness, Accountability, and Transparency*, ACM, 2024, https://doi.org/10.1145/3630106.3658542.
25) *Consommation d’eau de ChatGPT*. https://apnews.com/article/chatgpt-gpt4-iowa-ai-water-consumption-microsoft-f551fde98083d17a7e8d904f8be822c4. Accessed 6 Feb. 2025.
26) Baydin, Atilim Gunes, et al. *Automatic Differentiation in Machine Learning : A Survey*. 2018, https://arxiv.org/abs/1502.05767.
27) *PyTorch*. https://pytorch.org/. Accessed 3 Feb. 2025.
28) DeepSeek-AI, et al. *DeepSeek-V3 Technical Report*. 2024, https://arxiv.org/abs/2412.19437.
29) Devlin, Jacob, et al. *BERT : Pre-Training of Deep Bidirectional Transformers for Language Understanding*. 2019, https://arxiv.org/abs/1810.04805.
30) *GPT Tokenizer*. https://gpt-tokenizer.dev/. Accessed 3 Feb. 2025.
31) *Llama-3.2-1B*. https://huggingface.co/meta-llama/Llama-3.2-1B. Accessed 4 Feb. 2025.
32) *How to Generate Text : Using Different Decoding Methods for Language Generation with Transformers*. https://huggingface.co/blog/how-to-generate. Accessed 7 Mar. 2025.
33) *Wikipédia*. https://wikipedia.fr/. Accessed 26 Feb. 2025.
34) *GitHub*. https://github.com/. Accessed 26 Feb. 2025.
35) *ArXiv*. https://arxiv.org/. Accessed 26 Feb. 2025.
36) *FineWeb*. https://huggingface.co/datasets/HuggingFaceFW/fineweb. Accessed 25 Feb. 2025.
37) Tie, Guiyao, et al. *A Survey on Post-Training of Large Language Models*. 2025, https://arxiv.org/abs/2503.06072.
38) Köpf, Andreas, et al. *OpenAssistant Conversations – Democratizing Large Language Model Alignment*. 2023, https://arxiv.org/abs/2304.07327.
39) *Tiktokenizer*. https://tiktokenizer.vercel.app/. Accessed 26 Feb. 2025.
40) Ouyang, Long, et al. *Training Language Models to Follow Instructions with Human Feedback*. 2022, https://arxiv.org/abs/2203.02155.
41) Mikolov, Tomas, et al. *Efficient Estimation of Word Representations in Vector Space*. 2013, https://arxiv.org/abs/1301.3781.
42) Pennington, Jeffrey, et al. “GloVe : Global Vectors for Word Representation.” *Empirical Methods in Na-*

tural Language Processing (EMNLP), 2014, pp. 1532–43, http://www.aclweb.org/anthology/D14-1162.
43) Vaswani, Ashish, et al. *Attention Is All You Need*. 2023, https://arxiv.org/abs/1706.03762.
44) Su, Jianlin, et al. *RoFormer : Enhanced Transformer with Rotary Position Embedding*. 2023, https://arxiv.org/abs/2104.09864.
45) Jamil, Umar. *LLaMA Explained : KV-Cache, Rotary Positional Embedding, RMS Norm, Grouped Query Attention, SwiGLU*. https://www.youtube.com/watch?v=Mn_9W1nCFLo. Accessed 27 Mar. 2025.
46) *Llama Nuts and Bolts - RoPE*. https://adalkiran.github.io/llama-nuts-and-bolts/10-ROPE-ROTARY-POSITIONAL-EMBEDDINGS/. Accessed 7 Mar. 2025.
47) *Train 400x Faster Static Embedding Models with Sentence Transformers*. https://huggingface.co/blog/static-embeddings. Accessed 7 Mar. 2025.
48) *Model2Vec : Distill a Small Fast Model from Any Sentence Transformer*. https://huggingface.co/blog/Pringled/model2vec. Accessed 7 Mar. 2025.
49) Groeneveld, Dirk, et al. *OLMo : Accelerating the Science of Language Models*. 2024, https://arxiv.org/abs/2402.00838.
50) DeepSeek-AI. "DeepSeek-R1 : Incentivizing Reasoning Capability in LLMs via Reinforcement Learning." *arXiv Preprint arXiv :2501.12948*, 2024, https://arxiv.org/abs/2501.12948.
51) Grattafiori, Aaron, et al. *The Llama 3 Herd of Models*. 2024, https://arxiv.org/abs/2407.21783.
52) *Opening up ChatGPTOpening up ChatGPT : Tracking Openness of Instruction-Tuned LLMs*. https://opening-up-chatgpt.github.io/. Accessed 18 Mar. 2025.
53) *Open Source AI Index*. https://osai-index.eu/the-index. Accessed 21 Mar. 2025.
54) OpenAI. *GPT-4 Technical Report*. 2023, https://arxiv.org/abs/2303.08774.
55) Anthropic. *Claude*. 2024, https://www.anthropic.com/news/claude-3-family.
56) Touvron, Hugo, et al. *LLaMA : Open and Efficient Foundation Language Models*. 2023, https://arxiv.org/abs/2302.13971.
57) Team, Gemini, et al. *Gemini : A Family of Highly Capable Multimodal Models*. 2024, https://arxiv.org/abs/2312.11805.
58) Chen, Stanley F., et al. "An Empirical Study of Smoothing Techniques for Language Modeling." *Computer Speech & Language*, vol. 13, no. 4, 1999, pp. 359–94, https://aclanthology.org/A98-1014.pdf.
59) Shannon, Claude E. "A Mathematical Theory of Communication." *Bell System Technical Journal*, vol. 27, no. 3, 1948, pp. 379–423, https://people.math.harvard.edu/~ctm/home/text/others/shannon/entropy/entropy.pdf.
60) Brown, Peter F., et al. "Class-Based n-Gram Models of Natural Language." *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, 1992, pp. 467–73, https://aclanthology.org/P92-1058.pdf.
61) Elman, Jeffrey L. "Finding Structure in Time." *Cognitive Science*, vol. 14, no. 2, 1990, pp. 179–211, https://doi.org/10.1016/0364-0213(90)90002-E.
62) Hochreiter, Sepp, et al. "Long Short-Term Memory." *Neural Computation*, vol. 9, no. 8, 1997, pp. 1735–80, https://www.bioinf.jku.at/publications/older/2604.pdf.
63) Cho, KyungHyun, et al. *Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation*. 2014, https://arxiv.org/abs/1406.1078.
64) Bengio, Yoshua, et al. "A Neural Probabilistic Language Model." *Journal of Machine Learning Research*, vol. 3, 2003, pp. 1137–55, https://www.jmlr.org/papers/volume3/bengio03a/bengio03a.pdf.
65) Brants, Thorsten, et al. "Large Language Models in Machine Translation." *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2007, pp. 858–67, https://aclanthology.org/D07-1090.pdf.
66) Mikolov, Tomáš, et al. "Recurrent Neural Network Based Language Model." *Proceedings of the 11th Annual Conference of the International Speech Communication Association (INTERSPEECH 2010)*, 2010, pp. 1045–48, https://www.isca-speech.org/archive/interspeech_2010/mikolov10_interspeech.html.
67) Bahdanau, Dzmitry, et al. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2014, https://arxiv.org/abs/1409.0473.
68) *Mécanisme d'attention*. https://ai.stackexchange.com/questions/21389/what-is-the-intuition-behind-the-attention-mechanism. Accessed 3 Feb. 2025.
69) Radford, Alec, et al. *Improving Language Understanding by Generative Pre-Training*. OpenAI, 2018, https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

70) Radford, Alec, et al. *Language Models Are Unsupervised Multitask Learners*. OpenAI, 2019, https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.
71) Yang, Zhilin, et al. *XLNet : Generalized Autoregressive Pretraining for Language Understanding*. 2019, https://arxiv.org/abs/1906.08237.
72) Liu, Yinhan, et al. *RoBERTa : A Robustly Optimized BERT Pretraining Approach*. 2019, https://arxiv.org/abs/1907.11692.
73) Brown, Tom, et al. "Language Models Are Few-Shot Learners." *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 1877–901, https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.
74) Raffel, Colin, et al. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer." *Journal of Machine Learning Research*, vol. 21, no. 1, 2020, pp. 5485–551, https://jmlr.org/papers/volume21/20-074/20-074.pdf.
75) Shoeybi, Mohammad, et al. *Megatron-LM : Training Multi-Billion Parameter Language Models Using Model Parallelism*. 2019, https://arxiv.org/abs/1909.08053.
76) Stability AI. *StableLM-3B-4E1T*. 2024, https://stability.wandb.io/stability-llm/stable-lm/reports/StableLM-3B-4E1T--VmlldzoyMjU4.
77) Alayrac, Jean-Baptiste, et al. *Flamingo : A Visual Language Model for Few-Shot Learning*. 2022, https://arxiv.org/abs/2204.14198.
78) Driess, Danny, et al. *PaLM-e : An Embodied Multimodal Language Model*. 2023, https://arxiv.org/abs/2303.03378.
79) Liu, Haotian, et al. *Visual Instruction Tuning*. 2023, https://arxiv.org/abs/2304.08485.
80) Chen, Mark, et al. *Evaluating Large Language Models Trained on Code*. 2021, https://arxiv.org/abs/2107.03374.
81) Rozière, Baptiste, et al. *Code Llama : Open Foundation Models for Code*. Meta AI, 2023, https://ai.meta.com/blog/code-llama-large-language-model-coding/.
82) Singhal, Karan, et al. *Towards Expert-Level Medical Question Answering with Large Language Models*. 2023, https://arxiv.org/abs/2305.09617.
83) Lee, Jinhyuk, et al. "BioBERT : A Pre-Trained Biomedical Language Representation Model for Biomedical Text Mining." *Bioinformatics*, vol. 36, no. 4, 2020, pp. 1234–40, https://doi.org/10.1093/bioinformatics/btz682.
84) Chalkidis, Ilias, et al. "LEGAL-BERT : The Muppets Straight Out of Law School." *Findings of the Association for Computational Linguistics : EMNLP 2020*, Association for Computational Linguistics, 2020, pp. 2898–904, https://doi.org/10.18653/v1/2020.findings-emnlp.261.
85) *Ollama - Run Llama 2, Mistral, and Other Models Locally*. https://ollama.ai/. Accessed 28 Mar. 2025.
86) *LM Studio - Discover, Download, and Run Local LLMs*. https://lmstudio.ai/. Accessed 28 Mar. 2025.
87) Gerganov, Georgi. *Llama.cpp - Port of Facebook's LLaMA Model in c/c++*. https://github.com/ggerganov/llama.cpp. Accessed 28 Mar. 2025.
88) *Replicate - Run Open-Source Models with an API*. https://replicate.com/. Accessed 28 Mar. 2025.