



# LLMs

# Premiers pas

---

CEPE Octobre 2024  
Alexandre Tuel  
[atuel@galeio.fr](mailto:atuel@galeio.fr)

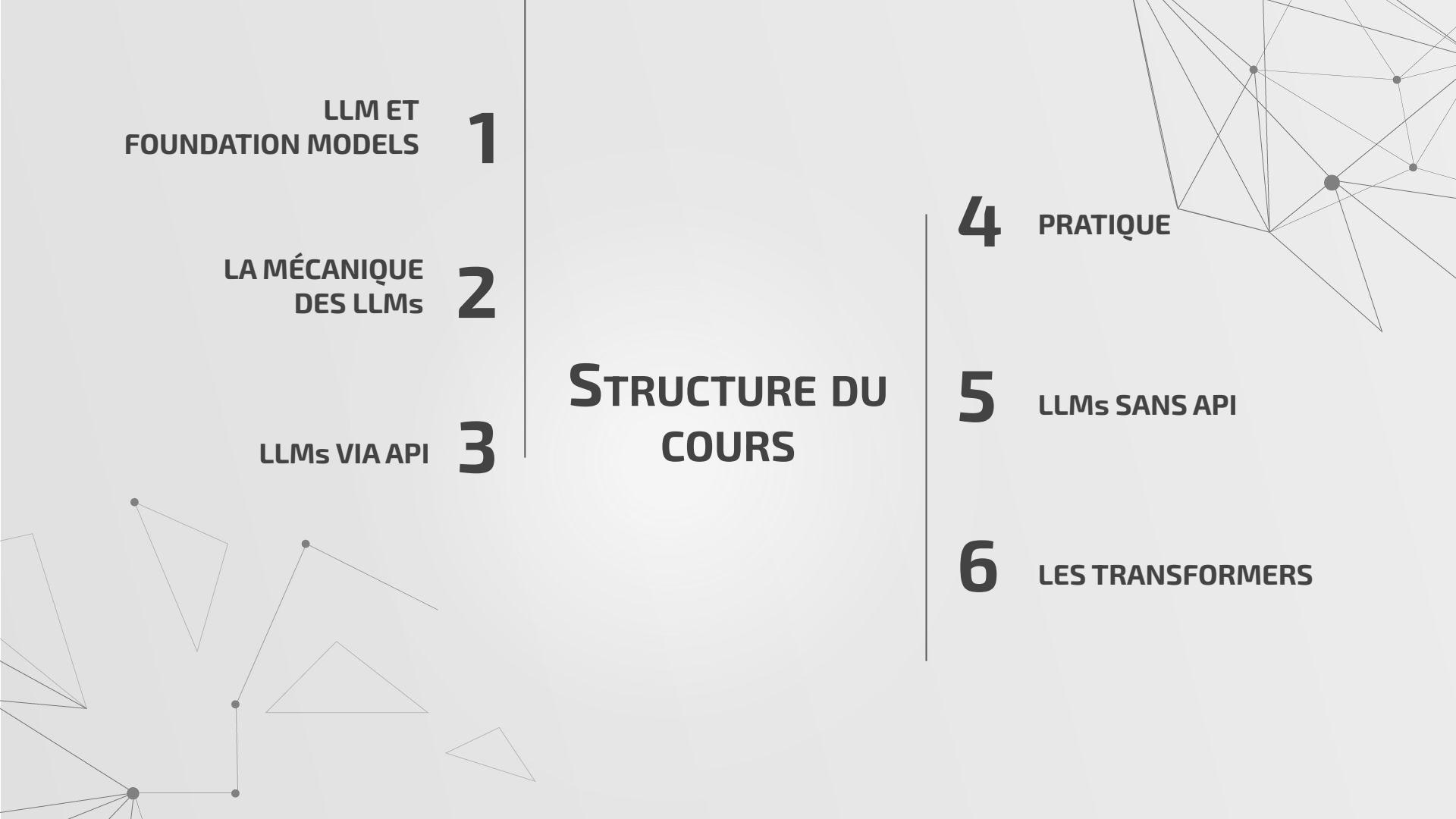


# OpenAI raises \$6.6bn in funding, is valued at \$157bn

The startup behind ChatGP, which is reportedly planning to become a for-profit business, is now valued on par with Uber

The  
Guardian





**LLM ET  
FOUNDATION MODELS**

**1**

**LA MÉCANIQUE  
DES LLMs**

**2**

**LLMs VIA API**

**3**

## **STRUCTURE DU COURS**

**4**

**PRATIQUE**

**5**

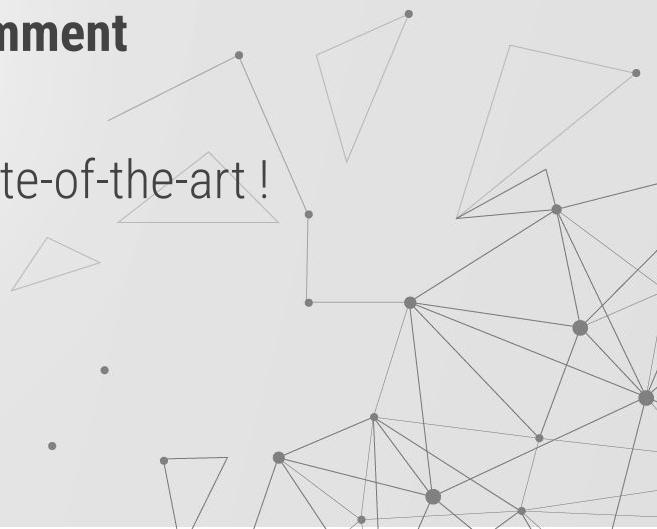
**LLMs SANS API**

**6**

**LES TRANSFORMERS**

# La jungle de l'IA générative et des LLMs

- Explosion du nombre de **concepts**, d'**outils**, de **techniques**
- Domaine où la technique avance **très rapidement**
- Difficile de se faire une idée de **quoi utiliser et comment**
- Compliqué de faire un cours toujours à jour du state-of-the-art !



# Les objectifs de la formation

- Comprendre la **mécanique des LLMs** et les changements que cela implique plus généralement sur la **pratique de l'IA**.
- Pouvoir utiliser des LLMs en API (ou en local) pour un certain nombre de cas d'usages.
- Savoir implémenter quelques méthodes (pas toutes !) pour adapter/améliorer un LLM pour un cas d'usage sur **des données spécifiques**.



# 1

## Introduction aux LLMs

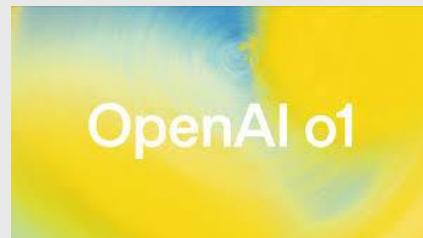


# OpenAI : GPT et ses applications

GPT = Generative **Pretrained Transformer**.

GPT c'est un **Large Language Model** (LLM)

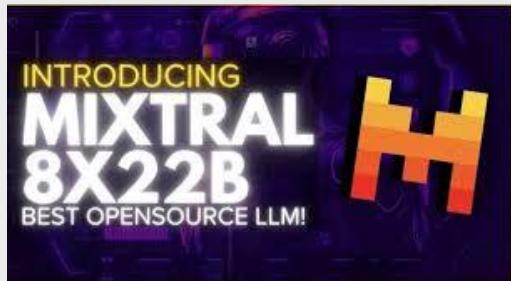
GPT c'est un **Foundation Model** (FM) sur le texte



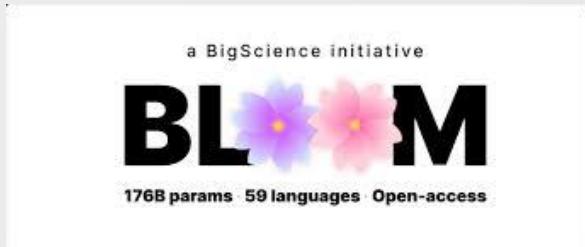
# Même situation pour Mistral AI

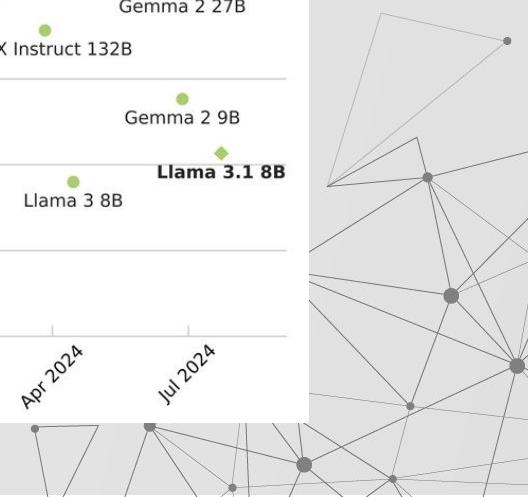
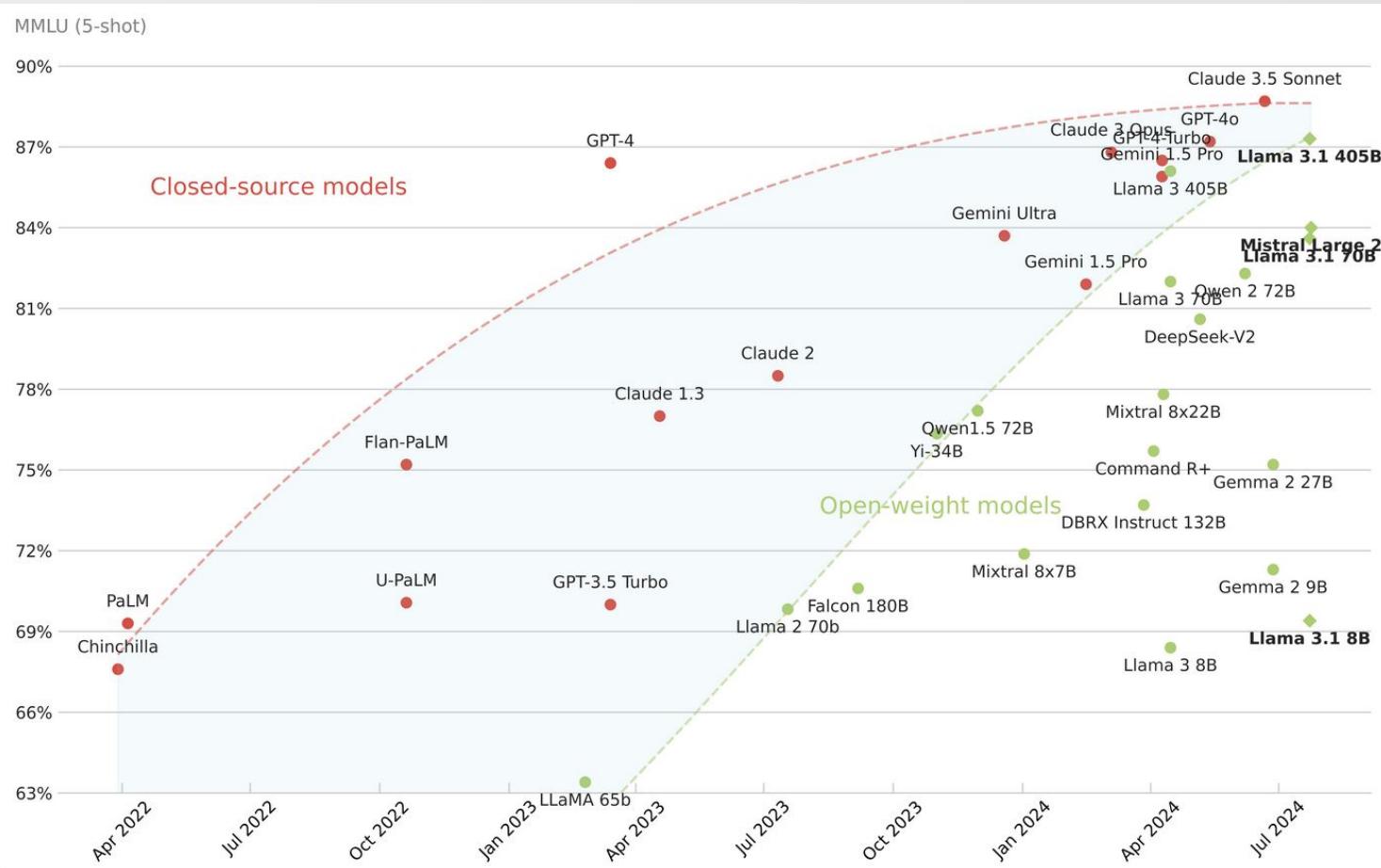
Les **mistrals** (ou **mixtral**) sont aussi des **GPT**, des **LLMs** et des **FMs pour le texte**

Les noms indiquent le nombre de paramètres des LLMs et un peu de technique (Mixture-Of-Experts)



# Et de (très) nombreuses autres offres de LLMs





# La multimodalité en marche



# Trois options pour utiliser les LLMs

- Via **interfaces web** (ChatGPT, Le Chat, Gemini, Perplexity, etc.) ou **produits intégrés** (Copilot, Github Copilot, CodeGPT, etc.)
- Via **APIs** :
  - payante, partage d'informations sur le cloud
  - meilleurs modèles et pas de soucis de hardware !
- **En local** en téléchargeant des LLMs :
  - Pas possible pour tous les modèles
  - Plus difficile...
  - Mais plus de **versatilité** et de **sécurité**





# 1.1

## LLMs et Language Models

# Comment faire de l'IA sur le texte ?

- Le texte n'est pas **numérique** = besoin d'**associer des chiffres aux mots (ou tokens)**
- NLP = Natural Language Processing
- De nombreuses méthodes basées sur comptage de la fréquence des mots ou probabilités de co-occurrence, par exemple, TF-IDF.
- Un **Language Model** = **distribution de probabilité sur les mots**.
- **Exemple:** Quel est le mot le plus probable après \*\*\*\*

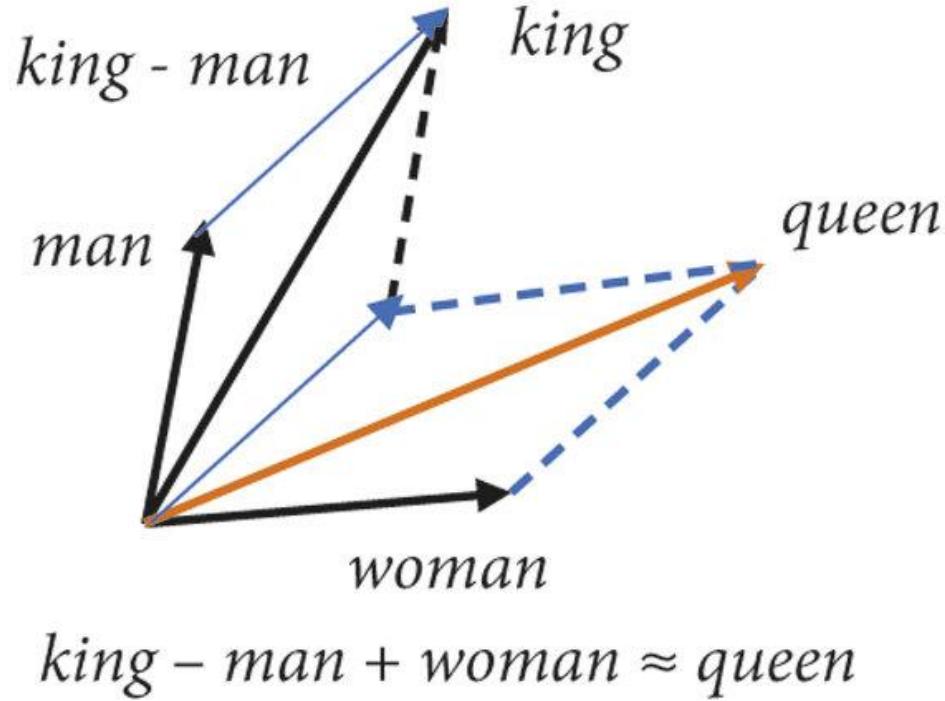


# La notion d'embedding

- **Embedding** = un vecteur dense associé à un objet (mot, phrase, préfixe, token)
- **Embedding** = compression/représentation sémantique
  - Deux mots synonymes = vecteurs “proches”
  - Capturer les relations entre les mots.
  - **Dense** plutôt que **one-hot encoding**
- C'est la **clé pour construire de bons Language Models**
- De nombreuses approches avant les LLMs, pas nouveau !



# L'exemple classique : king & queen



- Ce principe est utilisé pour les **RAG (Retrieval-Augmented Generation)** qu'on abordera demain
- Plusieurs façons de **mesurer la similarité** entre les vecteurs
- **Bases de données vectorielles** pour facilement chercher dans des corpus de texte.

## Et en code

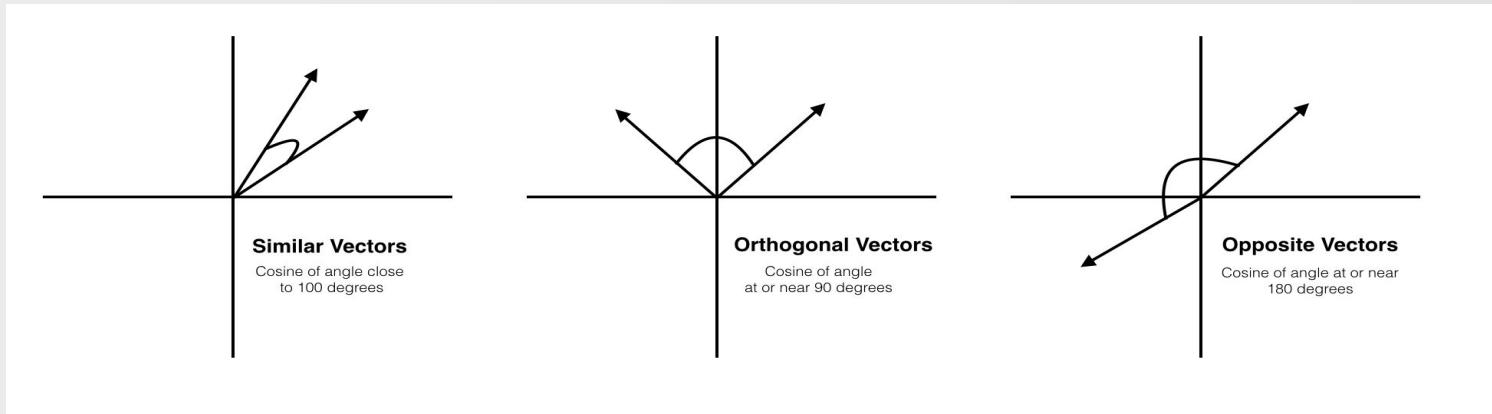
```
# On prepare un client pour appeler OpenAI API
client = OpenAI(api_key=OPENAI_API_KEY)

mot = 'queen'

# Make the API call to the embedding endpoint
response = client.embeddings.create(
    model="text-embedding-ada-002",
    input=mot
)
print(type(response))

<class 'openai.types.create_embedding_response.CreateEmbeddingResponse'>
```

# Mesurer la similarité



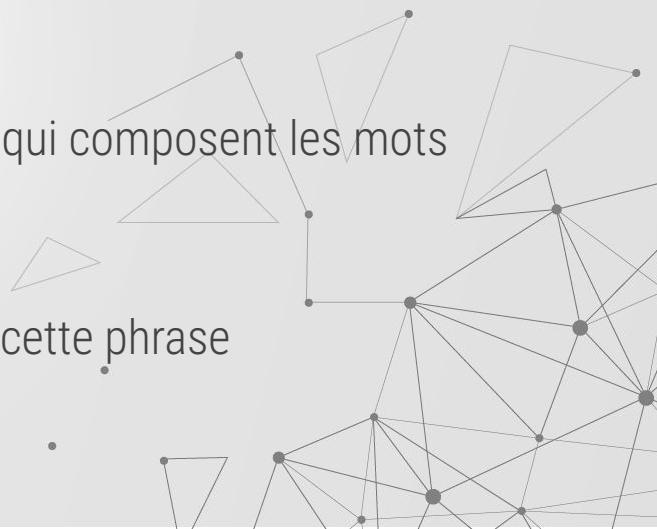
```
import numpy as np

# NB : c'est bien de donner le type de nos données
def cosine_similarity(vec1: list, vec2: list):
    vec1 = np.array(vec1)
    vec2 = np.array(vec2)
    dot_product = np.dot(vec1, vec2)
    norm_vec1 = np.linalg.norm(vec1)
    norm_vec2 = np.linalg.norm(vec2)
    return dot_product / (norm_vec1 * norm_vec2)
```



# Méthodes d'embedding non-contextualisé

- Une première idée = des mots qui apparaissent **souvent ensemble** auront des **embeddings similaires**.
- De nombreuses façons de créer ces embeddings :
  - **GloVe** : *Global Vectors for Word Representation*, à partir d'une énorme matrice de cooccurrence.
  - **Word2Vec** : réseau de neurones.
  - **FastText** : comme Word2Vec, mais avec des tokens qui composent les mots
- Pour associer un vecteur à une phrase, on peut :
  - Faire la moyenne des vecteurs des mots associés à cette phrase
  - + compliqué : méthodes comme **Seq2Vec**



# Embedding et LLMs

- Les **embeddings** sont donc la **première étape** des méthodes de **deep learning sur le texte** (donc pour les LLMs).
- Les embeddings peuvent concerner les mots ou les **sous-unités** des mots : les **tokens**.
- Les embeddings peuvent être **fixés** ou **appris**.
- Pour les LLMs comme GPT, les embeddings sont **appris pendant le processus d'entraînement**.



# Pre-trained embeddings VS contextual embeddings

- **Context-dependent embeddings** = le même mot peut avoir des **embeddings différents en fonction du contexte**.
- **GPT ou BERT** ont des embeddings **context-dependents**.
- Par exemple *bank* n'aura pas le même embedding dans *river bank* ou *bank account* (ou “avocat” en français)
- **Cette notion de contexte est absolument centrale pour les LLMs !!!**



# Contexte : Uni ou Bi directionel ?

- Ok, les **LLMs** utilisent des **embedding contextuels** (ou dynamiques).
- Mais on distingue aussi le **type de contexte** utilisé pour créer les embeddings des mots (ou tokens) en interne.
- Choix apparemment anodin mais qui a de nombreuses conséquences !
- Il y a donc **deux types** de modèles qui construisent les embeddings différemment :
  - **Unidirectionnel** : seuls les mots précédents comptent.
  - **Bidirectionnel** : les mots avant et après comptent.



# La tâche des LLMs

Un **LLM**, c'est un **réseau de neurones** qui est entraîné à résoudre **une tâche prétexte** sur un **énorme corpus de texte**.

## Next-token-prediction

The model is given a sequence of words with the goal of predicting the next word.

Example:  
Hannah is a \_\_\_

Hannah is a *sister*  
Hannah is a *friend*  
Hannah is a *marketer*  
Hannah is a *comedian*

## Masked-language-modeling

The model is given a sequence of words with the goal of predicting a 'masked' word in the middle.

Example  
Jacob [mask] reading

Jacob *fears* reading  
Jacob *loves* reading  
Jacob *enjoys* reading  
Jacob *hates* reading

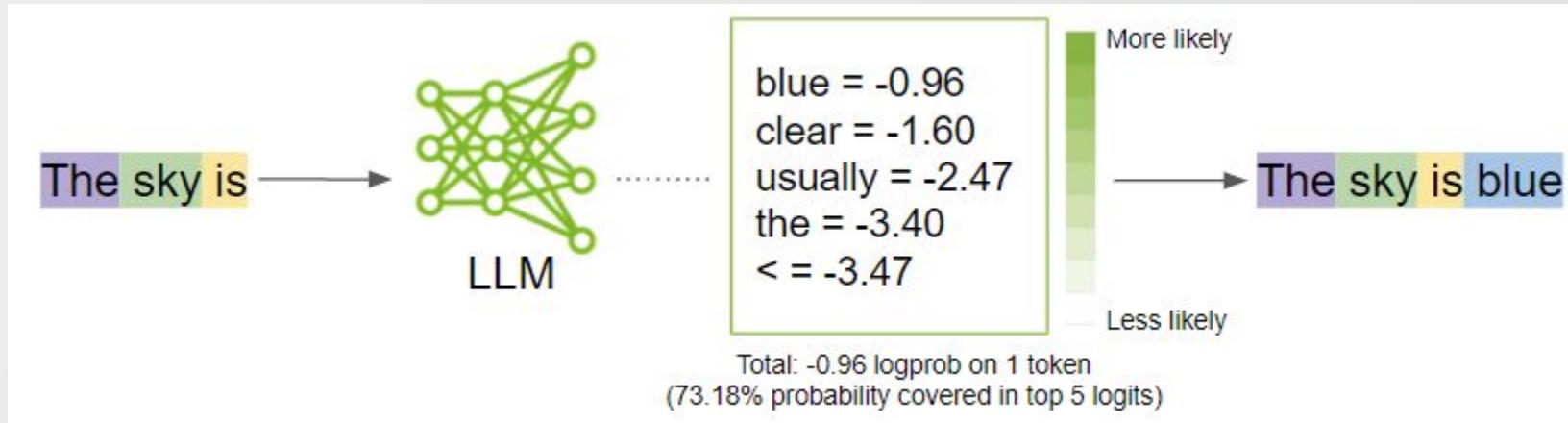
- Un **LLM** est donc un **language model** puisqu'en prédisant le next token, il construit en interne une distribution de probabilité sur le langage.

- **GPT ou Mistral sont entraînés avec la next-token prediction.**

- Les modèles de next-token prédiction sont des **modèles génératifs** (ou **modèle causal**).



# Un LLM est juste une fonction



- LLM est une fonction  $f(x)$  dont l'**entrée  $x$  est une suite de tokens** et la sortie est un **vecteur de probabilité/score** sur l'**ensemble des tokens**.
- La fonction  $f(x)$  dépend d'**énormément de paramètres**, c'est ce qui la rend spéciale.

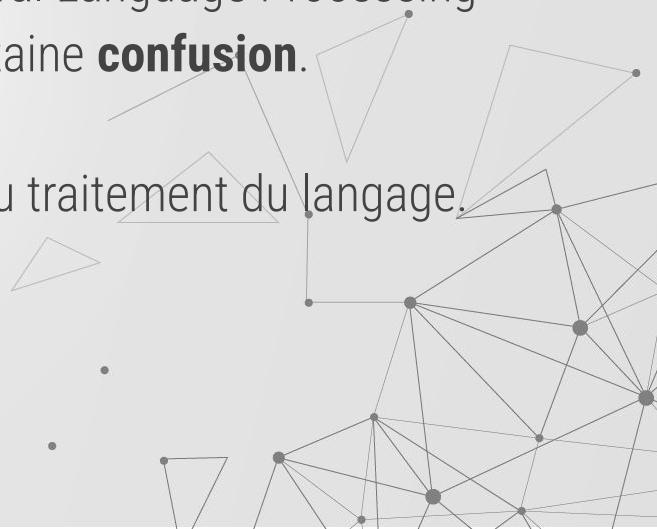
# LLM Causal

- Un LLM entraîné sur la tâche de **next-token prédiction** = **auto-régressif** ou **causal**.
- Et ici le Foundation Model peut être utilisé directement à des fins génératives.
- **Génératif** : pas le **vrai cœur** du sujet...
- Ce qui compte = pouvoir entraîner des IA générales sur **données non-supervisées**.
- C'est la **tâche prétexte** qui détermine l'aspect génératif ou non.



# LLMs et IA générative

- Du fait de la tâche qu'ils sont entraînés à faire, les LLMs peuvent immédiatement faire de la **génération et complétion de texte** : c'est de là que vient le terme **IA générative**.
- On peut les utiliser pour tout un tas de tâches de Natural Language Processing (NLP) : le terme IA générative peut donc créer une certaine **confusion**.
- Les LLMs délivrent la promesse de l'**automatisation** du traitement du langage.



# Exemple de génération

```
from openai import OpenAI

from credentials.keys import OPENAI_API_KEY

client = OpenAI(api_key=OPENAI_API_KEY)

prompt = 'Tell me a story about nights and frogs'

response = client.completions.create(
    model="gpt-3.5-turbo-instruct",
    prompt=prompt
)

# Extract and print the generated text
generated_text = response.choices[0].text.strip()
print(generated_text)
```

In a small village nestled in the heart of the forest, there once lived



# Point de situation

Pour l'instant, nous n'avons pas encore les clés pour comprendre la spécificité des LLMs et le changement de paradigme qu'ils sous-tendent.

- Ok pour l'aspect **génération de texte des LLMs**, mais comment les utiliser pour **toutes les autres tâches du NLP** ?
- Ok l'aspect **Language Model** des LLMs, mais pourquoi **Large** ?
- Qu'est-ce qui a créé l'**effet ChatGPT** ? Un coup de publicité ou une vraie révolution ?
- De quelle nature est cette révolution ?





# 1.2

## Large LMs : le passage à l'échelle



# Où est la rupture ?



# Où est la rupture ?

Les LLMs ont **deux aspects relativement nouveaux** :

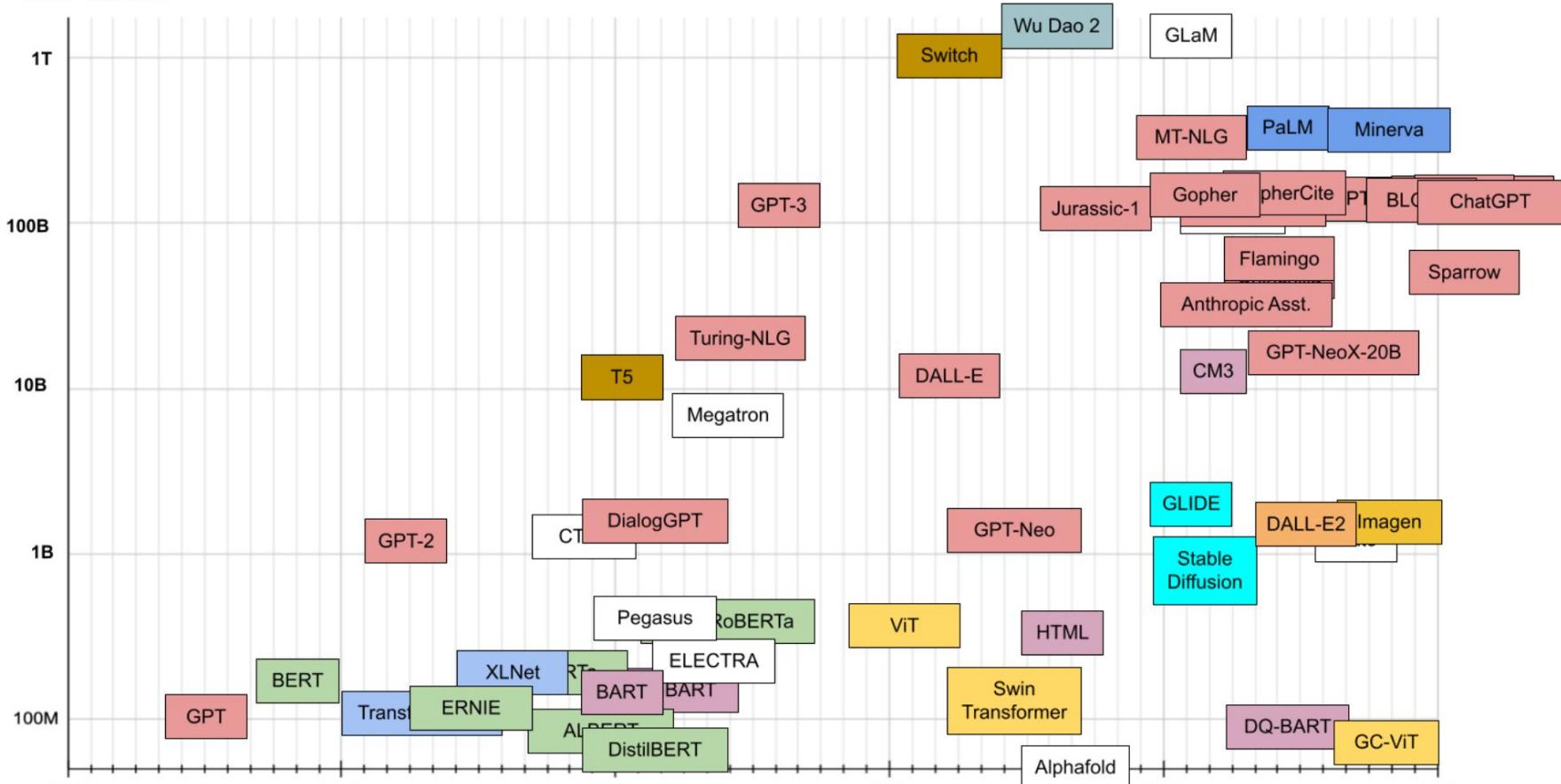
- Les architectures de type **Transformers** avec un certain nombre de **hacks/trade secrets**.
- L'entraînement des LLMs a été **passé à l'échelle** : phénomènes d'émergence.

Pour passer à l'échelle avec succès l'**entraînement d'un réseau de neurones**, il faut **trois éléments** :

- 1) **Augmenter fortement** la taille de la base de données d'entraînement.
- 2) S'assurer que cette grande base de données soit **très variée**.
- 3) **Augmenter fortement** le nombre de **paramètres du modèle**.



## Num. Parameters



Nguyen et al. (2023)

# Les Transformers

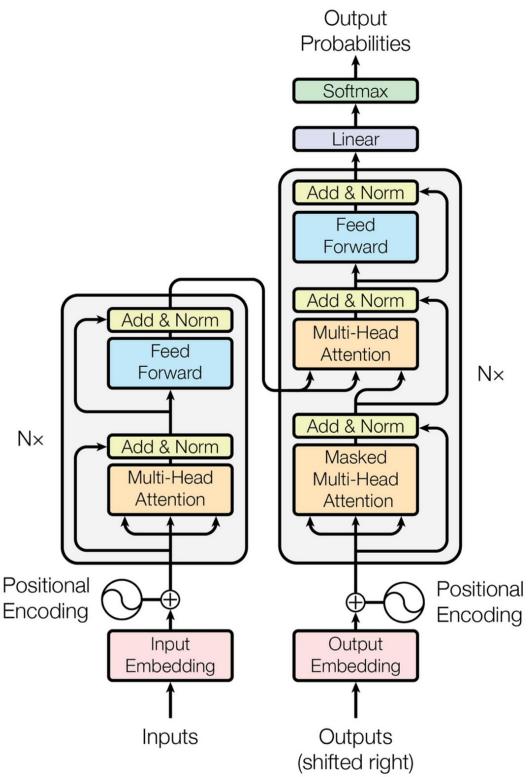
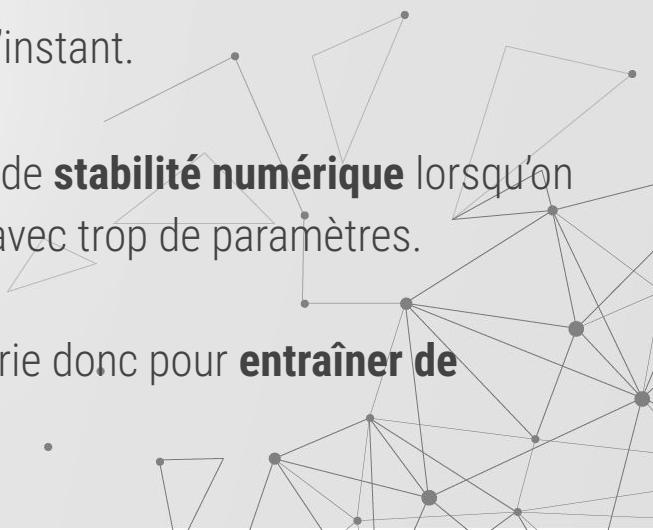


Figure 1: The Transformer - model architecture.

- Rôle important mais plus difficile à apprécier.
- Pas une idée nouvelle (date des années 90), mais c'est la réalisation que le **passage à l'échelle** fonctionne (RNN pas possible).
- Pas de détails pour l'instant.
- Il y a des problèmes de **stabilité numérique** lorsqu'on entraîne un modèle avec trop de paramètres.
- Des enjeux d'ingénierie donc pour **entraîner de grands modèles**.

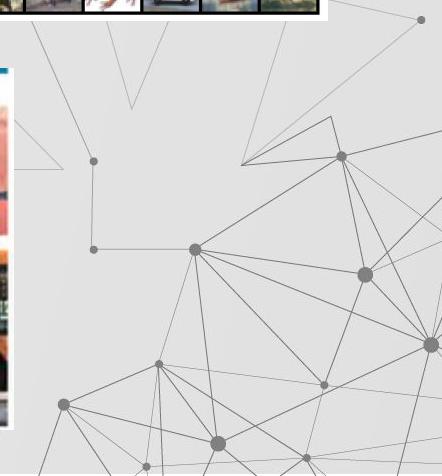
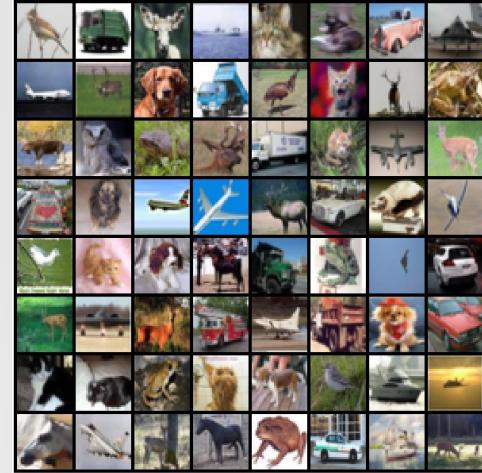
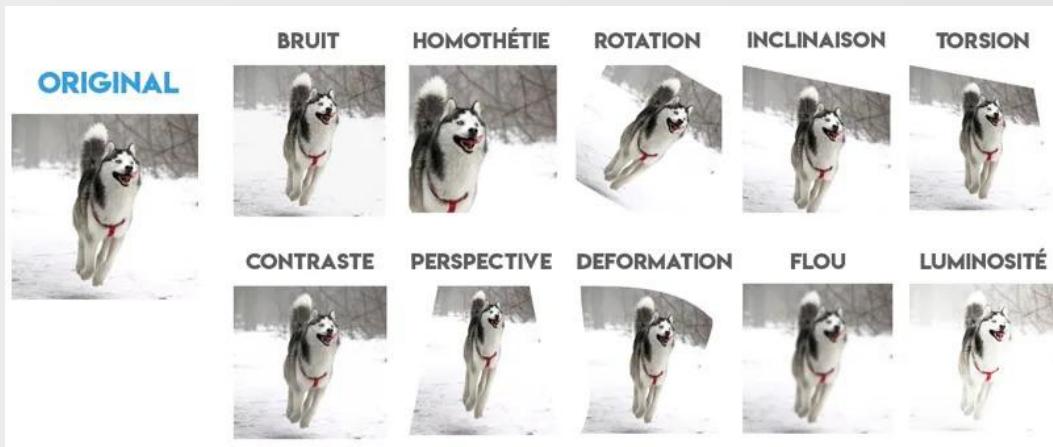


# L'apprentissage supervisé

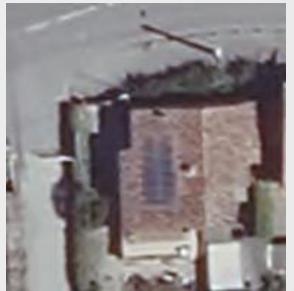
- **Apprentissage supervisé** = base de données ( $X_i, y_i$ ), on veut apprendre à prédire  $y$  à partir de  $X$ .
- **Problème de l'apprentissage supervisé : pas de passage à l'échelle de la base de données.**
- **Est-ce que l'on peut vraiment augmenter déraisonnablement la taille d'une base de données supervisée ?**
- **Data-augmentation** ? Non, n'assure **pas assez de diversité**.
- De nombreuses solutions existent pour tenter de contourner : simulation de données, supervised transfer learning, etc.



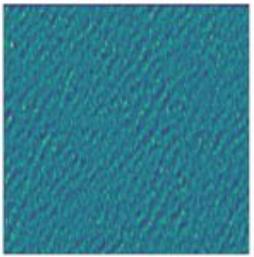
# Scaling des bases de données d'images



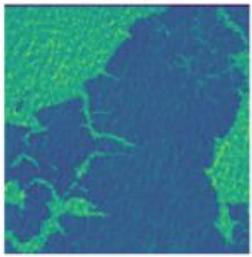
# Scaling des bases de données d'images



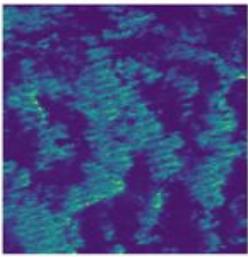
Class F  
Pure Ocean Waves



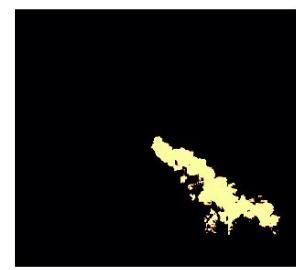
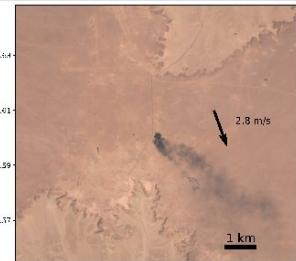
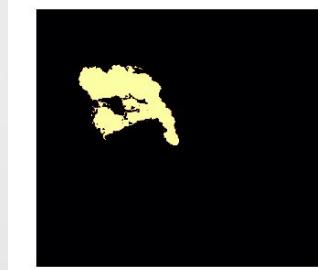
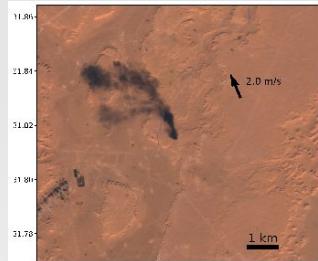
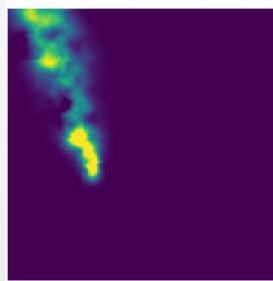
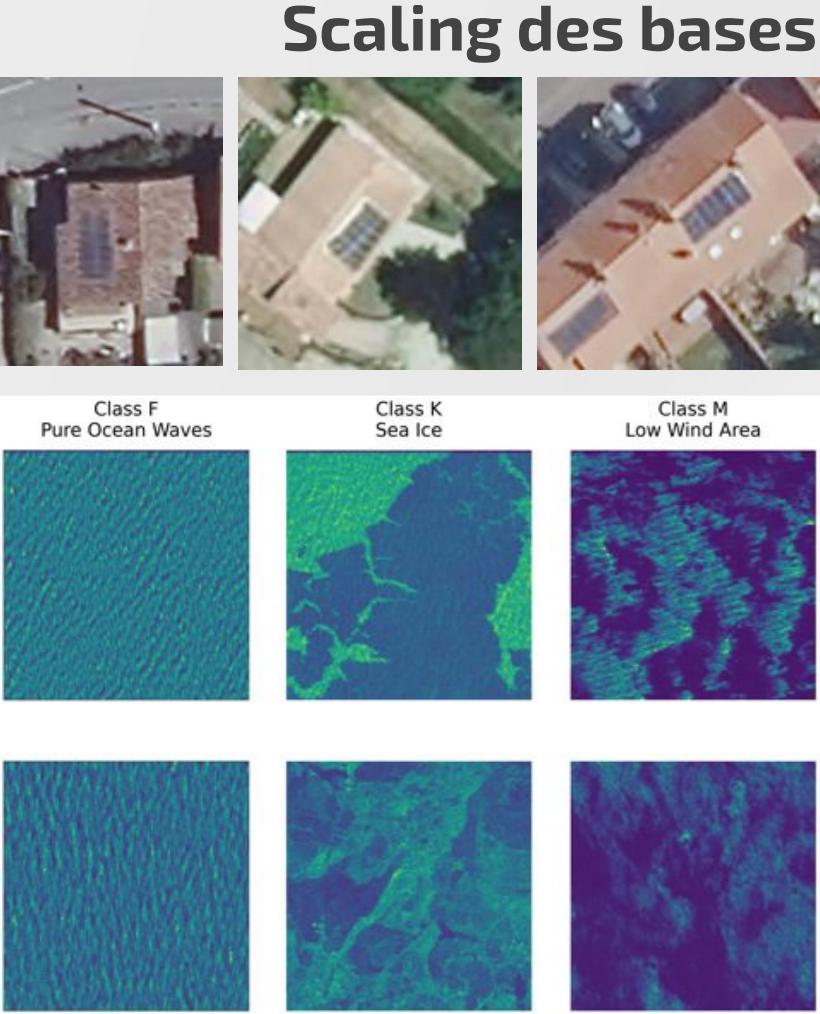
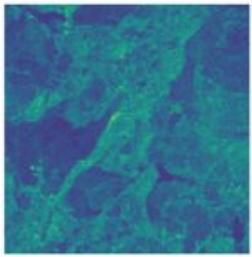
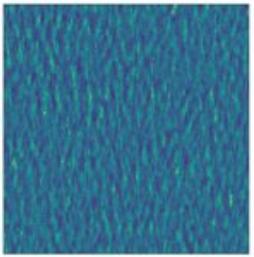
Class K  
Sea Ice



Class M  
Low Wind Area

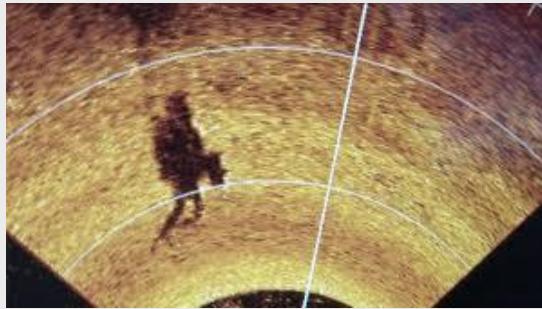


Generated data



# Scaling des bases de données ?

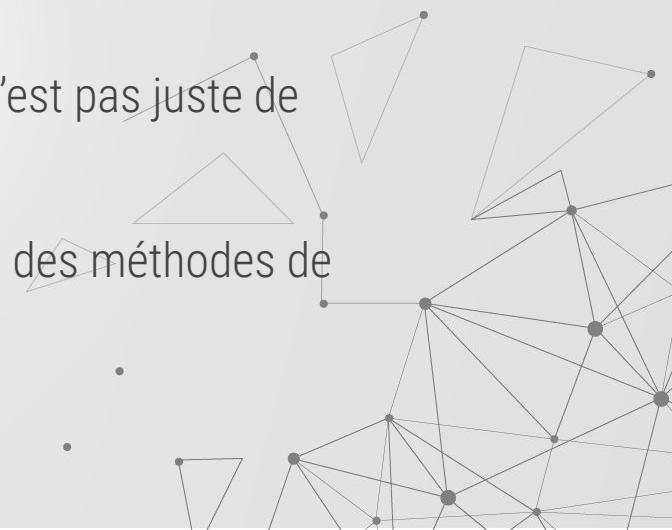
- Donc pour les **images**, augmenter la taille des bases de données a été un **effort considérable**.
- Pour tous les autres types de données, on ne peut pas faire le même effort que pour l'image.



Pour le texte : la tâche d'entraînement des LLMs est auto-supervisée

# Next-token prediction est **self-supervised**

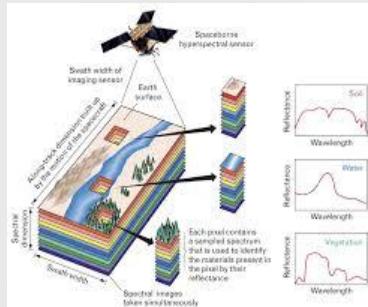
- Pour les **tâches prétextes**, il n'y a pas besoin de supervision **y**.
- Ici, la supervision est construite grâce à la **structure dans la donnée**.
- Pas de limite donc du côté **effort humain** pour récolter une base de données.
- C'est le **self-supervised learning (SSL)**.
- Attention, il y a de **nombreuses méthodes** de SSL, ce n'est pas juste de l'apprentissage supervisé caché.
- Les meilleurs embeddings pour le texte se font grâce à des méthodes de **contrastive learning** (un autre paradigme en SSL).



# Self-supervised learning

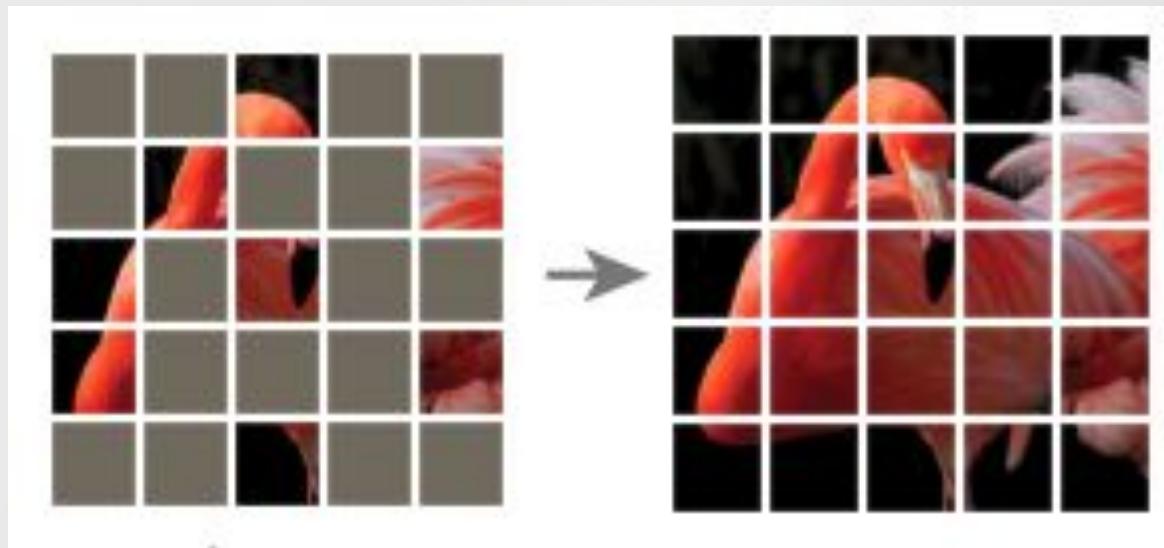
La révolution derrière les LLMs, c'est **la preuve que le SSL est opérationnel lorsque l'on passe à l'échelle l'entraînement.**

- Aujourd'hui pour l'image naturelle, les méthodes supervisées avec les très grandes bases de données curated/annotées à la main fonctionnent encore (un peu) mieux que le SSL pour l'image.
- Par contre, le **SSL est un “no brainer”** pour tous les domaines où c'est inenvisageable de faire de l'apprentissage supervisé. **Bientôt de nombreux Foundation Models pour toutes les modalités.**



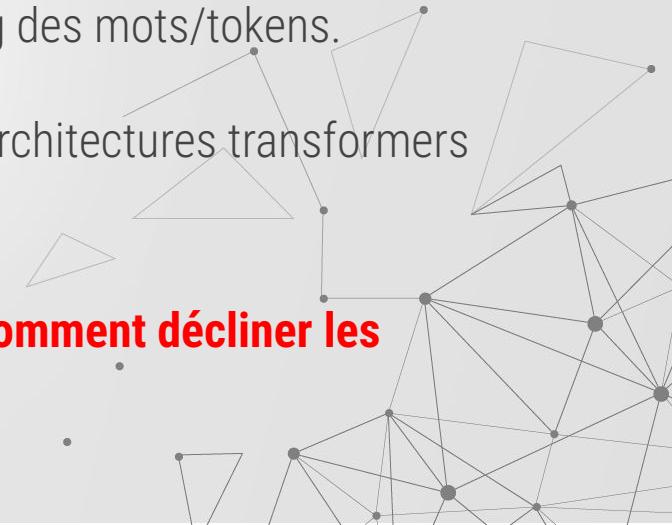
# Tâches prétextes pour l'image ?

**Masked image modeling** = équivalent du **next-token prédiction** pour le texte.



# Point de situation

- Les LLMs sont des **Language Models** entraînés à faire du next-token prediction.
- Ils sont **Large** parce que **grande base de données** et **taille de modèle**.
- Parce que ***next-token prediction* = SSL**, on peut facilement **agrandir la base**.
- Nous avons parlé du mécanisme sous-jacent d'embedding des mots/tokens.
- Pas donné de détails sur les réseaux de neurones ou les architectures transformers  
= le détail des LLMs reste un mystère.
- **Pas encore trop parlé de l'aspect Foundation Model et comment décliner les usages des LLMs**





# 1.3

## Les modèles de fondation

# L'état d'esprit foundation model

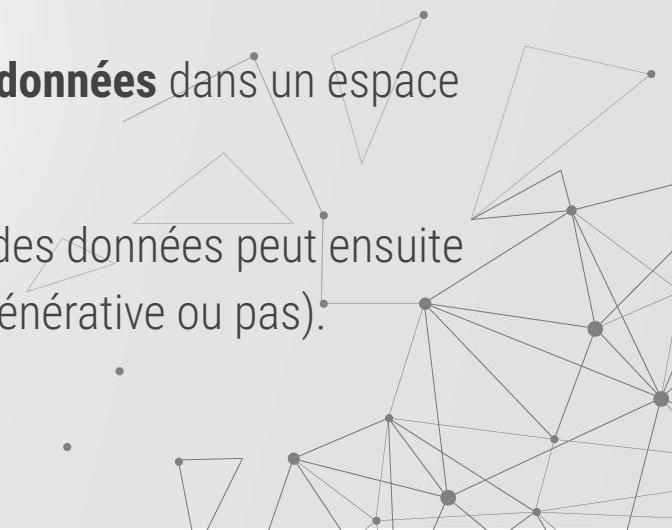
- La notion de **foundation model** n'est pour l'instant pas complètement **standardisée**.
- Foundation = quelque chose **d'universel** ? Pas l'aspect le plus important.
- **Foundation** = ce sur quoi construire. **Pretrained**. Abstrait le travail d'entraînement d'un réseau de neurones sur énormes ordinateurs et énormes bases de données.
- Foundation = changement de pratique en IA. Important de savoir **manipuler, orchestrer, adapter ces modèles** plutôt que les **construire**.
- Les modèles de fondation existent depuis un moment.
- **D'excellents modèles de fondation arrivent pour toutes les modalités.** Le texte, c'était juste le plus simple à développer (vocabulaire = espace fini).



# Un changement de paradigme

L'intérêt des foundation models, c'est leur généralisabilité.

- Ce sont des modèles qui comprennent la **structure** d'un **certain type de données** (texte, audio, image, vidéo, etc.).
- Tâche prétexte = les force à savoir bien **représenter les données** dans un espace **numérique** de **dimension donnée** (par ex. embeddings).
- Leur **capacité de représentation générale** (universelle) des données peut ensuite être mise à profit pour résoudre une **tâche spécifique** (génération ou pas):



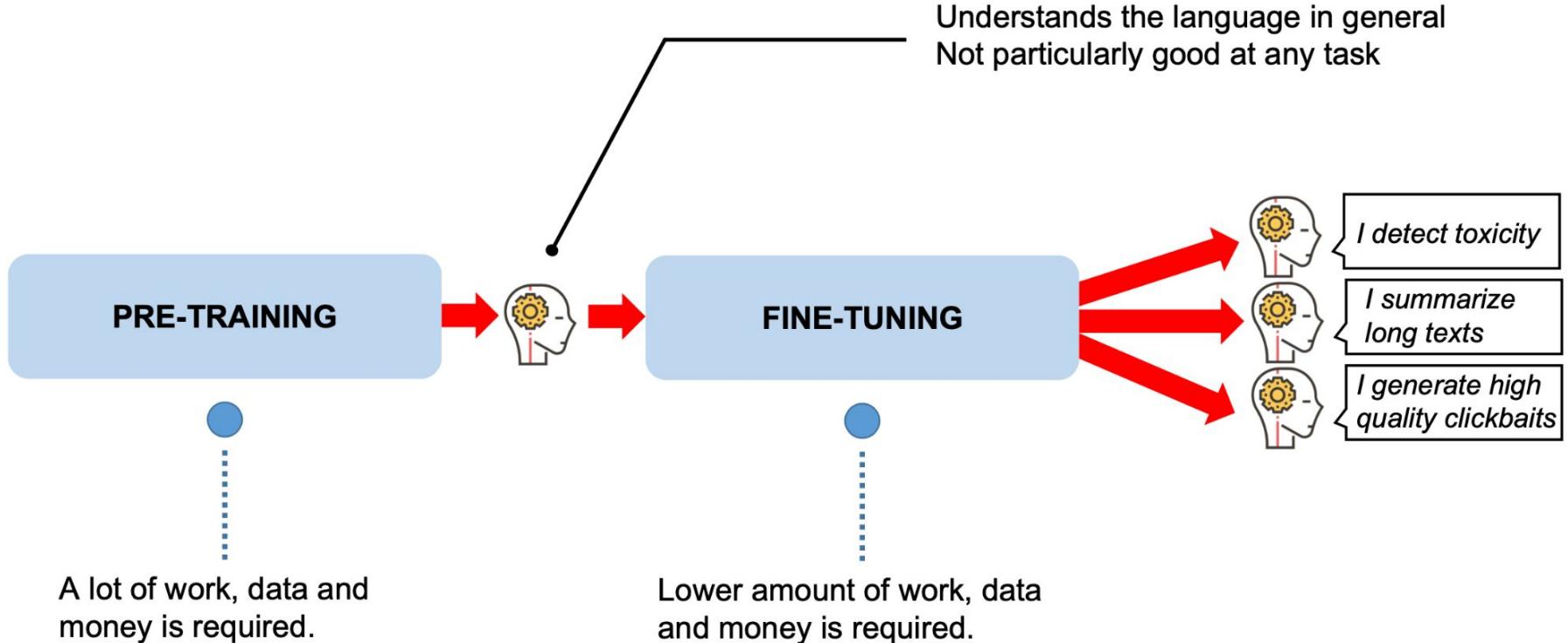
# Le vocabulaire associés aux Foundation Models

- **Fine-tuner un LLM** = ré-entraîner un LLM sur un jeu de données plus spécifique
- **Zero-shot learning** = directement utiliser le LLM sur une tâche
- **Few-shot learning** = utiliser seulement quelques exemples pour entraîner le LLM sur une tâche
- **Pretext task** = la tâche sur laquelle le LLM est entraîné
- **Downstream task** = la tâche associée au use-case spécifique
- **Pretrained** = le FM (ici LLM) a déjà été entraîné, il n'y a plus qu'à l'adapter à nos cas d'usages



Tous ces mots s'appliquent aux LLMs mais plus généralement aux FMs.

# En principe



# One Model, Multiple Use Cases

- Les Foundation Models représentent un **intérêt commercial** parce qu'ils **factorisent les coûts de développement**.
- Ils **redessinent le partage des tâches en IA** entre utilisateurs des FMs et ceux qui créent les FMs.
- Politique de **semi-open source** agressive des gros acteurs.
- En pratique, **concentration du savoir faire FM** dans **quelques acteurs**.

Un des objectifs de la formation c'est de maîtriser la façon 1) d'adapter des LLMs à des **use cases spécifiques** et 2) de les adapter à **des types de données différentes**.

# Cas d'usages des LLMs

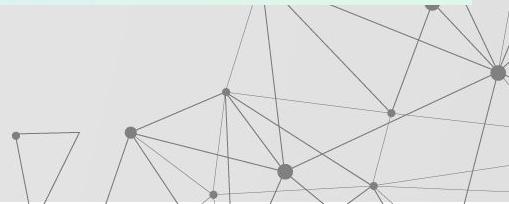
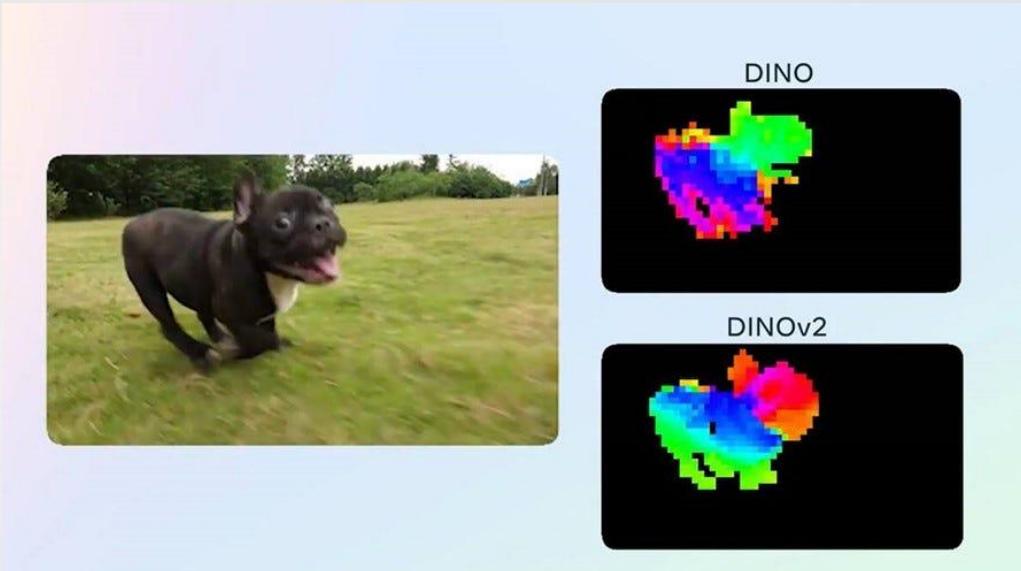
- Génération de texte
- Q&A, chatbots et assistants virtuels
- Tâches de NLP “classiques” : résumé, classification ou analyse de texte (sentiment, etc.), named entity recognition, relation extraction
- Traduction et parsing
- ... et bien d'autres !



# Des Foundations Models pour images

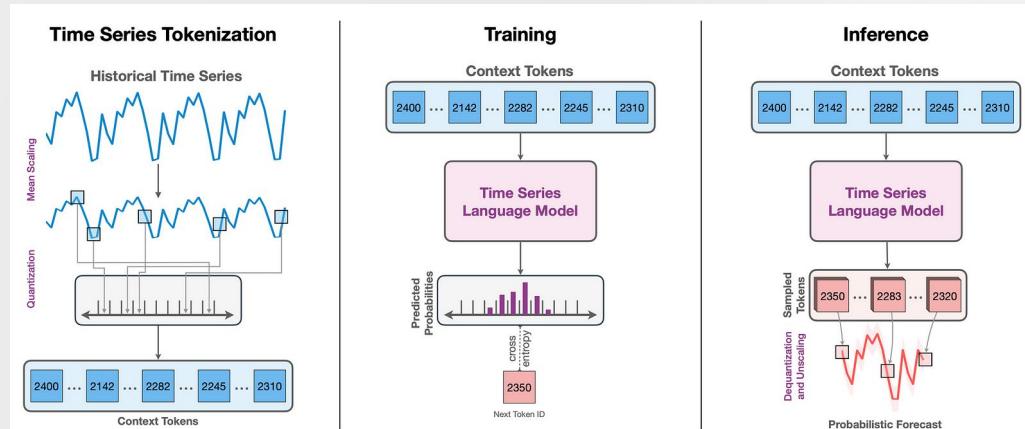


SAM = Segment Anything



# Des Foundations Models pour séries temporelles

Chronos (Amazon)



Moirai (Salesforce)

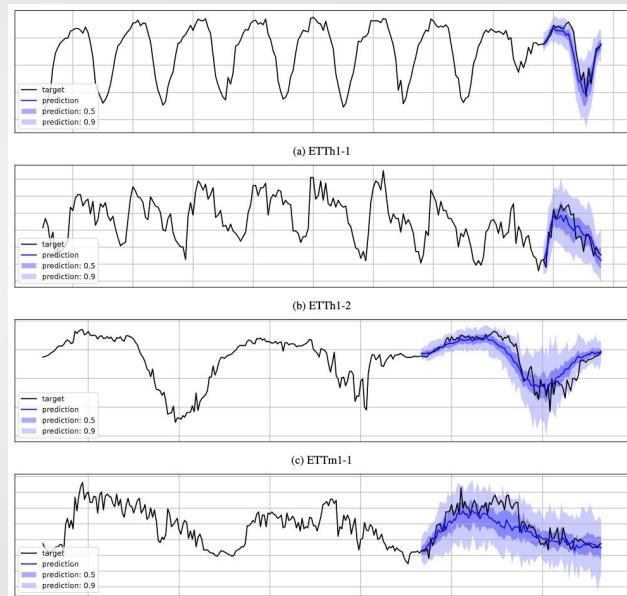


Figure 8. Visualizations of zero-shot forecasts from MOIRAI<sub>base</sub> on ETTh1 and ETTm1 datasets.

# 2

## La mécanique des LLMs

---

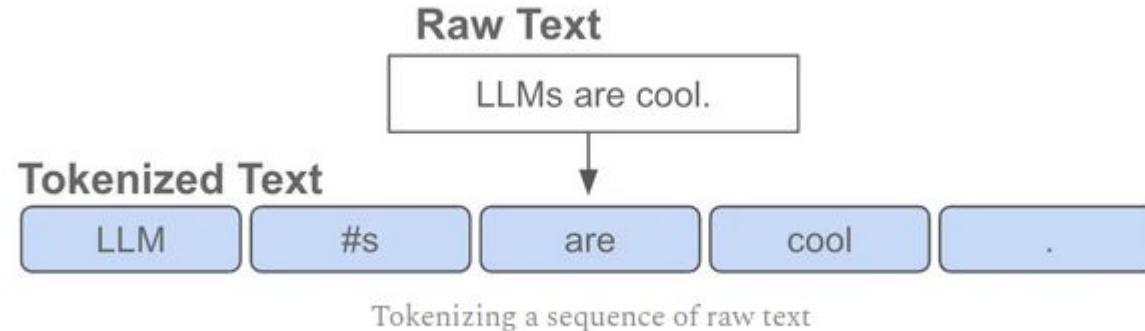




# 2.1

## La tokénisation

# La tokénisation



- **Tokénisation** = la première étape avant **l'embedding**.
- C'est une étape systématique (**entraînement** ou **inférence**) : **rester cohérent**.
- On devrait presque dire **Token Model** au lieu de **Language Model**
- **Token** : lié au caractère, sous-mot ou mot.



# La tokénisation

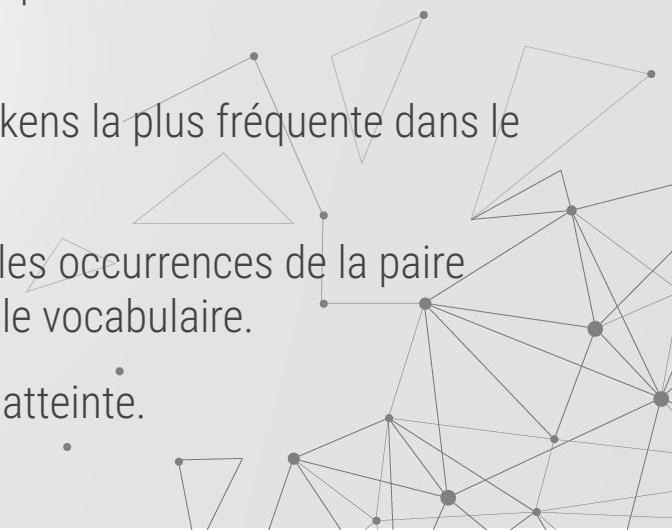
- Les tokens peuvent être des **mots, sous-mots, caractères**, voire des “**special tokens**”.
- La liste des tokens est apprise sur un jeu de données d'entraînement.
- Par exemple, GPT-4 est entraîné sur 50257 tokens, Llama 3 ~ 64000
- Utiliser des **tokens plutôt que des mots** permet de :
  - gérer les mots rares et complexes en les découplant
  - gérer différents langages sans faire exploser la taille du vocabulaire
  - gérer la ponctuation et les différentes modalités d'inputs (texte, image, etc.)
- Les **tokens** deviennent **l'unité de mesure** (taille DB, coût API, etc..)
- De **nombreuses méthodes de tokénisation**.



# Exemple du Byte-Pair Encoding

**Le Byte-Pair Encoding permet de créer un vocabulaire de tokens d'une taille arbitraire pour un corpus donné.** Les étapes sont :

- **Initialiser le vocabulaire** : vocabulaire de base de tous les caractères individuels présents dans le corpus.
- **Compter les fréquences** : comptez la fréquence de toutes les paires de tokens consécutifs dans le corpus.
- **Fusionner la paire la plus fréquente** : identifiez la paire de tokens la plus fréquente dans le corpus. Fusionnez cette paire en un seul nouveau token.
- **Mettre à jour le corpus et le vocabulaire** : remplacez toutes les occurrences de la paire fusionnée dans le corpus par le nouveau token. Mettre à jour le vocabulaire.
- **Répéter** jusqu'à ce que la taille de vocabulaire souhaitée soit atteinte.



# Code pour tokéniser

Encoding name	OpenAI models
c1100k_base	gpt-4, gpt-3.5-turbo, text-embedding-ada-002, text-embedding-3-small, text-embedding-3-large
p50k_base	Codex models, text-davinci-002, text-davinci-003
r50k_base (or gpt2)	GPT-3 models like davinci

- La tokénisation n'est **pas la même** pour tous les modèles.
- La tokénisation **comprime** le texte mais est **lossless/réversible**



# Code pour tokéniser

```
import tiktoken

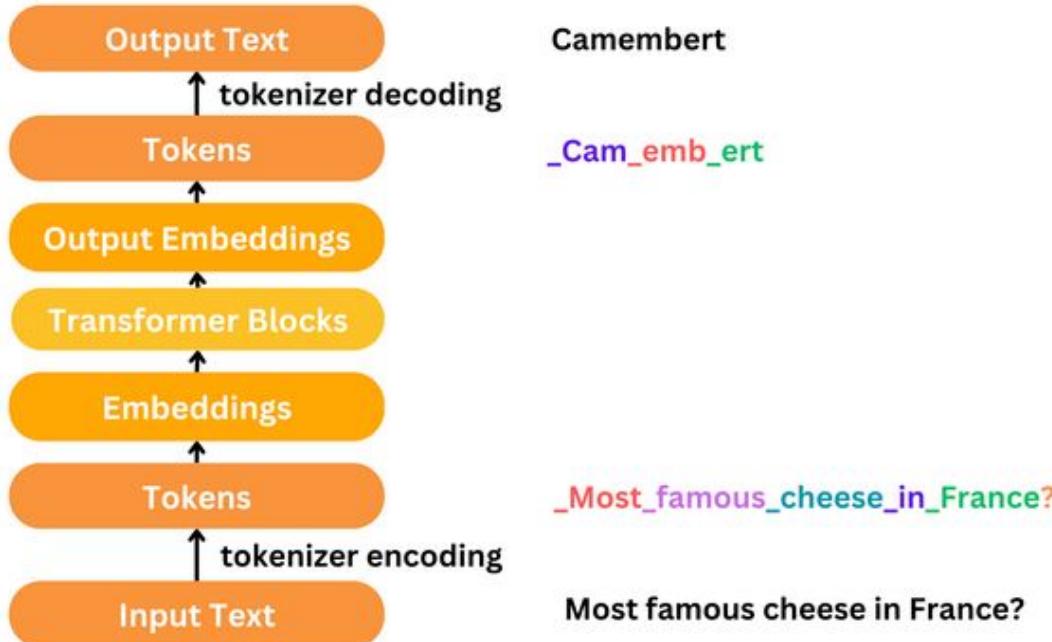
example_string = 'We live in Paris but ewhd here'
# Le methode pour encoder selon
encoding = tiktoken.get_encoding('cl100k_base')
token_integers = encoding.encode(example_string)
# On renvoie les indices des differents token dans le dictionnaire des tokens
print(token_integers)
```

```
[1687, 3974, 304, 12366, 719, 37990, 16373, 1618]
```

```
num_tokens = len(token_integers)
token_bytes = [encoding.decode_single_token_bytes(token) for token in token_integers]
print(token_bytes)

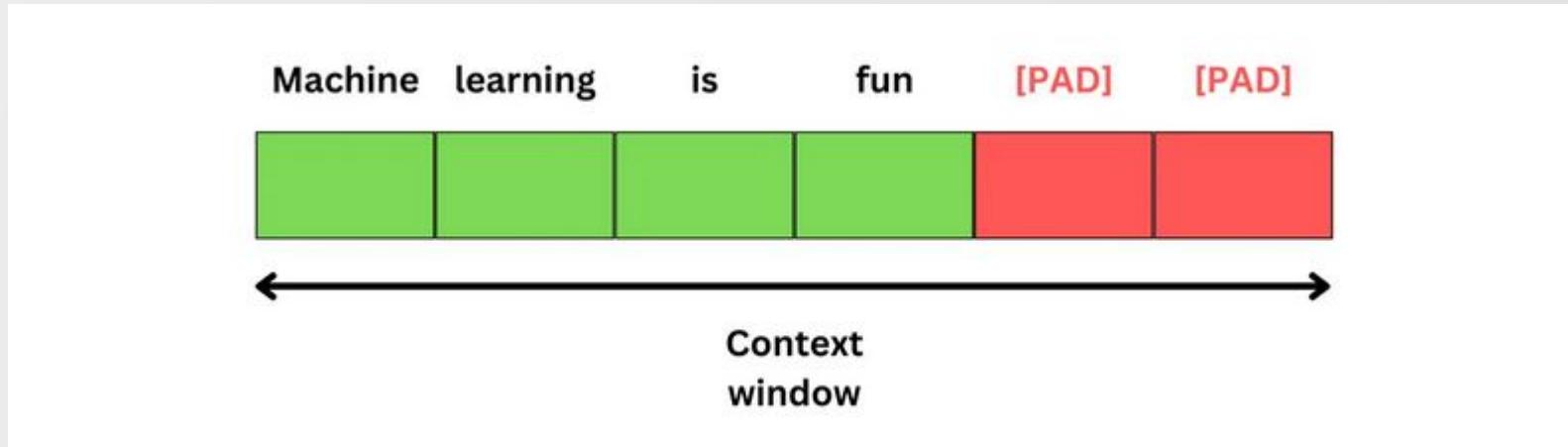
[b'We', b' live', b' in', b' Paris', b' but', b' ew', b'hd', b' here']
```

# Encoding et decoding des tokens



- On parle d'**encoding** lorsque transforme une chaîne de caractères (string) en une liste de tokens.
- Decoding = inverse
- **Les termes decoding/encoding seront utilisés dans d'autres contextes que la tokénisation.**

# Le padding du texte



- [PAD] sont des **tokens vides de sens** pour donner en input aux LLMs des phrases/paragraphes de **même taille** (en nombre de tokens), important en **decoder-only**.
- Rendre les opérations de calcul plus efficaces : parallélisation/batching.

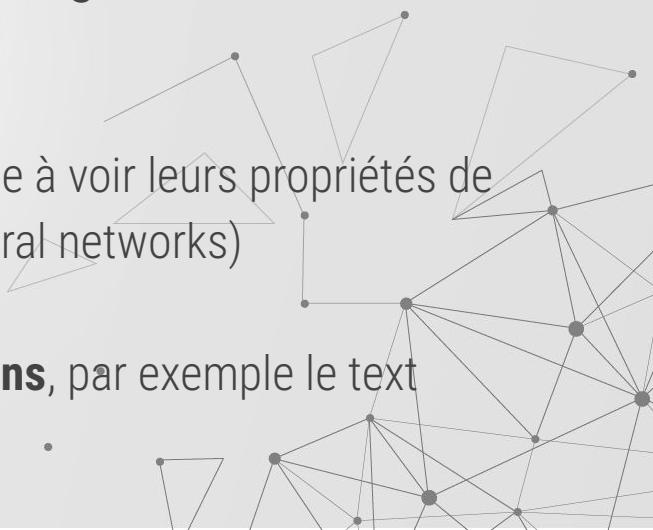
# Context window

- La *context window*, c'est le **nombre maximum de tokens que le LLM peut considérer** en une fois.
- Lors du padding, il faut choisir une longueur plus petite que la context window.
- Pour certaines tâches (par ex, text summarization) la **taille du contexte est importante**.
- Il faut de plus gros modèles pour augmenter la taille de la context window.
- GPT-2 : **1024** tokens ; GPT-3 : **2048** tokens, GPT-4 : **32k** tokens, GPT-4o : **128k** tokens
- Les LLMs sont entraînés sur des séquences de la taille de leur context window.



# Context window

- **La complexité du modèle augmente de façon quadratique** avec la taille de la context window (à cause de l'attention des transformers qui implique des multiplications matricielles).
- Ce sont en effet les architectures Transformers qui sont très **gourmandes en ressources**.
- De **nombreuses alternatives** sont en train d'émerger, reste à voir leurs propriétés de scaling (par ex. structured state-space models, liquid neural networks)
- La taille de la context window **limite certaines applications**, par exemple le text summarization.





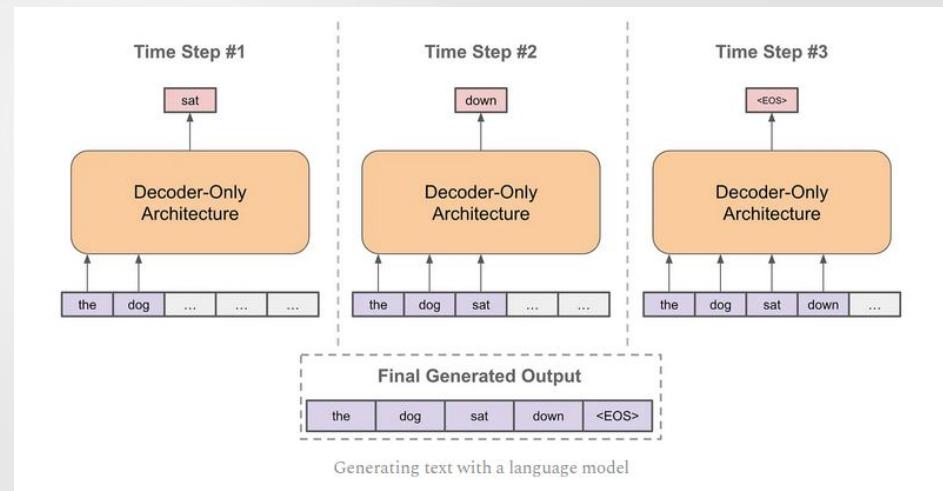
# 2.2

## Génération de texte

# Génération

En commençant par une séquence d'entrée (éventuellement vide), les LLMs génèrent du texte en suivant un processus de prédiction **autorégressif** (causal) du prochain **token** avec les étapes suivantes :

- Faire la prédiction du prochain token
- Ajouter ce token à l'input
- Recommencer

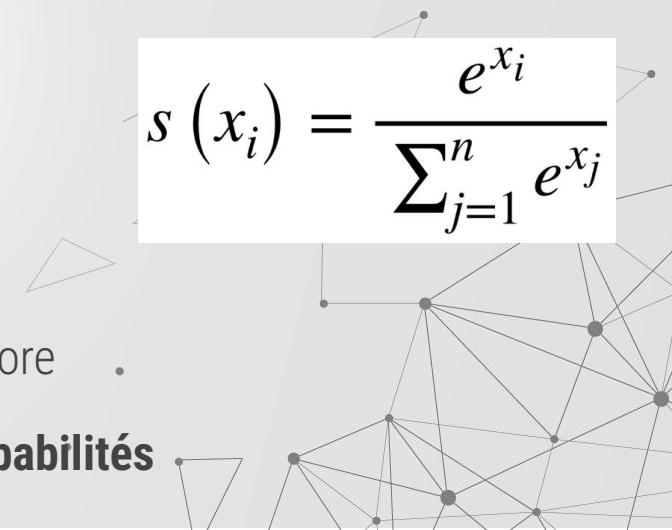


# Les probabilités en next-token prediction

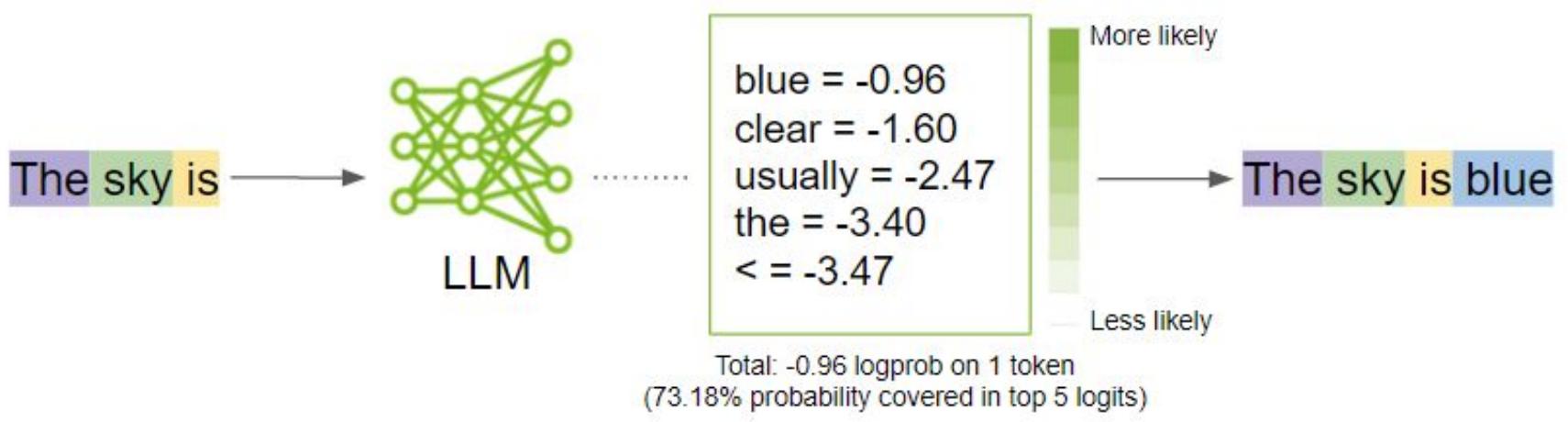
Pour prédire le prochain token, le LLM construit une **distribution de probabilité** sur l'ensemble des tokens. Concrètement le LLM produit un **vecteur de logits** qui est ensuite transformé en **vecteur de probabilités**.

- Les **logits** sont juste les **scores non-normalisés associés à chaque token**.
- La **softmax fonction** permet de passer des logits à une distribution de probabilité.
- Il y a **deux façons** de choisir le next-token :
  - Choisir le token avec la plus grande probabilité/score
  - **Tirer un token au hasard selon le vecteur de probabilités**

$$s(x_i) = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$$



# Rappel sur l'output d'un LLM

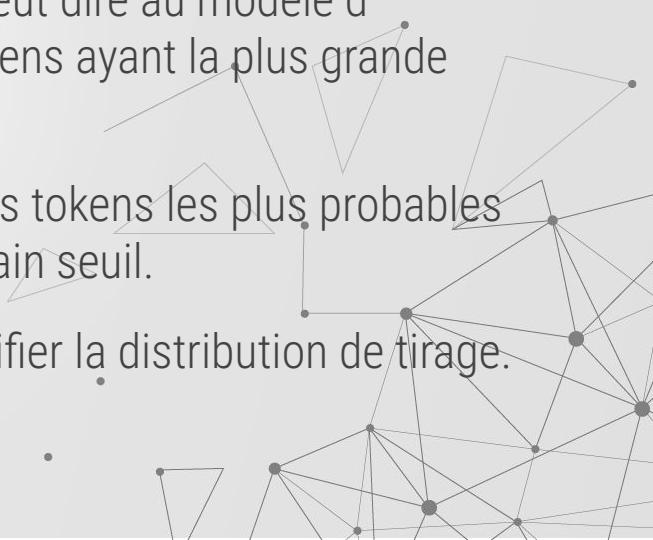


- Un LLM est une fonction  $f(x)$  dont l'**entrée  $x$  est une suite de tokens** et la sortie est un **vecteur de probabilité/score** sur l'**ensemble des tokens**.

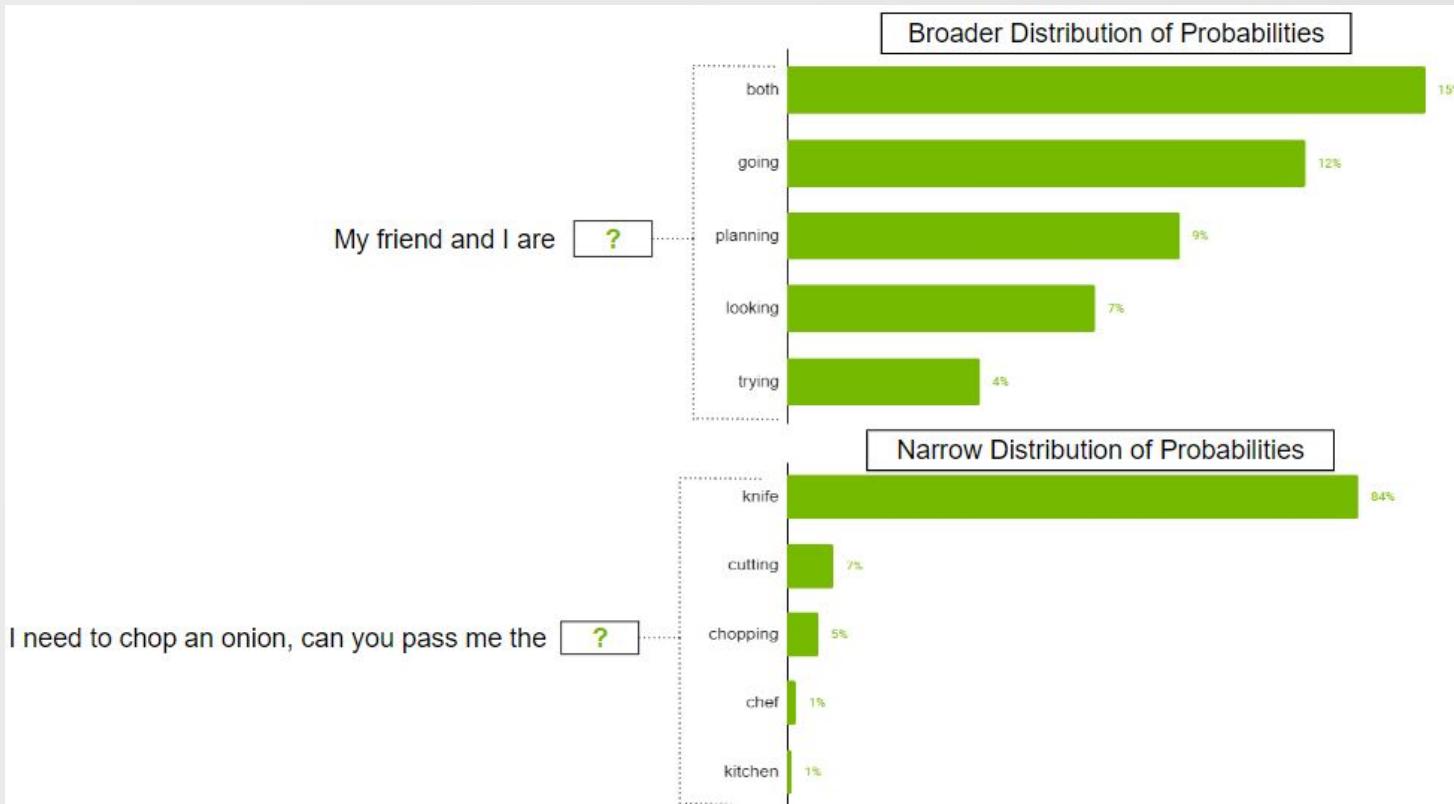
# Le sampling du next-token

Il y a aussi d'autres paramètres qui permettent d'ajuster le résultat de la génération :

- Définir un ensemble de **stop words**, pour arrêter prématulement la génération.
- Limiter le nombre de tokens en output.
- Lors du sampling aléatoire, avec le paramètre **top-k**, on peut dire au modèle d'échantillonner aléatoirement uniquement parmis les k tokens ayant la plus grande probabilité.
- Ou alors le paramètre **top-p** pour ne sampler que parmi les tokens les plus probables dont les probabilités cumulées sont en dessous d'un certain seuil.
- Ou enfin le paramètre de **température** qui permet de modifier la distribution de tirage.



# Complémentarité de top-p et top-k



## Code pour top-p et stop-word

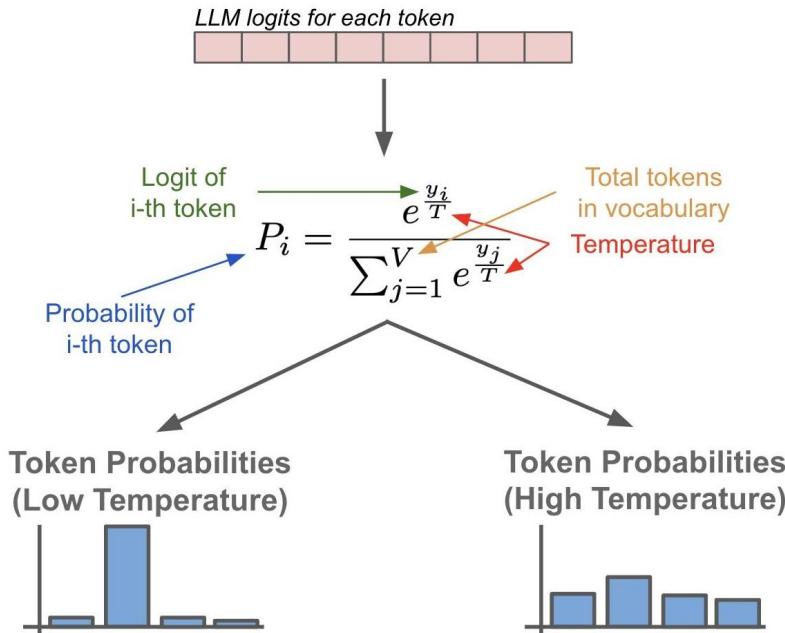
```
prompt = 'Tell me a story about nights and frogs'

response = client.completions.create(
    model="gpt-3.5-turbo-instruct",
    prompt=prompt,
    top_p=0.8,
    stop='.',
    max_tokens=100, # Increase max_tokens to generate a longer text
)

generated_text = response.choices[0].text.strip()
print(generated_text)
```

# La température

## Softmax with Temperature



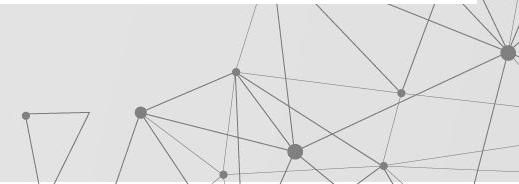
- La température est un **paramètre dans le softmax** pour modifier la distribution (entre 0 et 1).
- Haute température = **output très aléatoire**.
- La température a un intérêt parce qu'on **échantillonner le next token** à partir de la distribution de probabilité.
- Température nulle = on choisit toujours le token avec la plus grande probabilité.

# Code pour température

```
prompt = 'Tell me a story about nights and frogs'

response = client.completions.create(
    model="gpt-3.5-turbo-instruct",
    prompt=prompt,
    temperature=1,
    max_tokens=200, # Increase max_tokens to generate a longer text
)

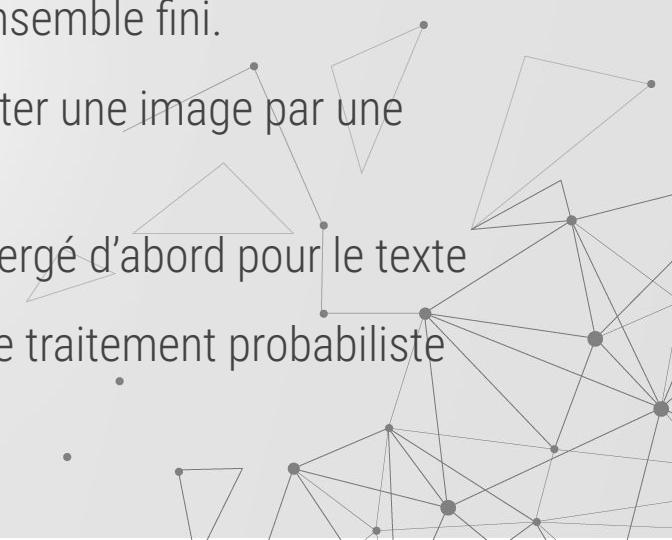
generated_text = response.choices[0].text.strip()
print(generated_text)
```



# Texte VS autres modalités

**Différence fondamentale : le texte = ensemble fini de tokens (dictionnaire)**

- Cela permet de concevoir des **modèles probabilistes** pour next-token prediction : "facile" de définir une distribution de probabilité sur un ensemble fini.
- Images, audio, etc. : rien à voir ! Très difficile de représenter une image par une distribution de probabilité "simple".
- Une des raisons pour lesquelles les large models ont émergé d'abord pour le texte
- Approches pour autres modalités : similaires mais pas de traitement probabiliste



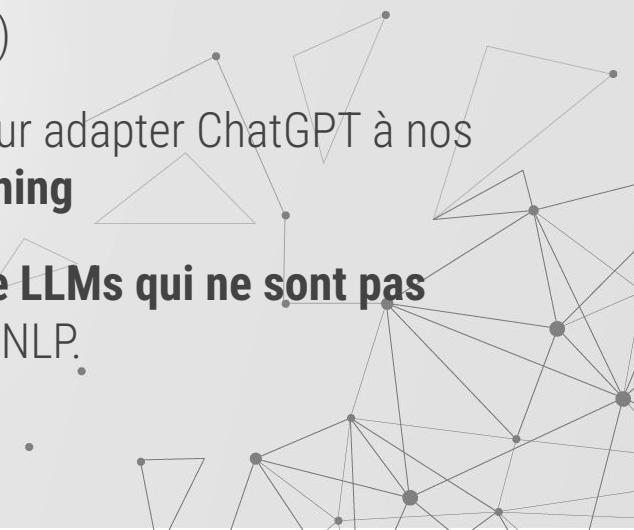


# 2.2

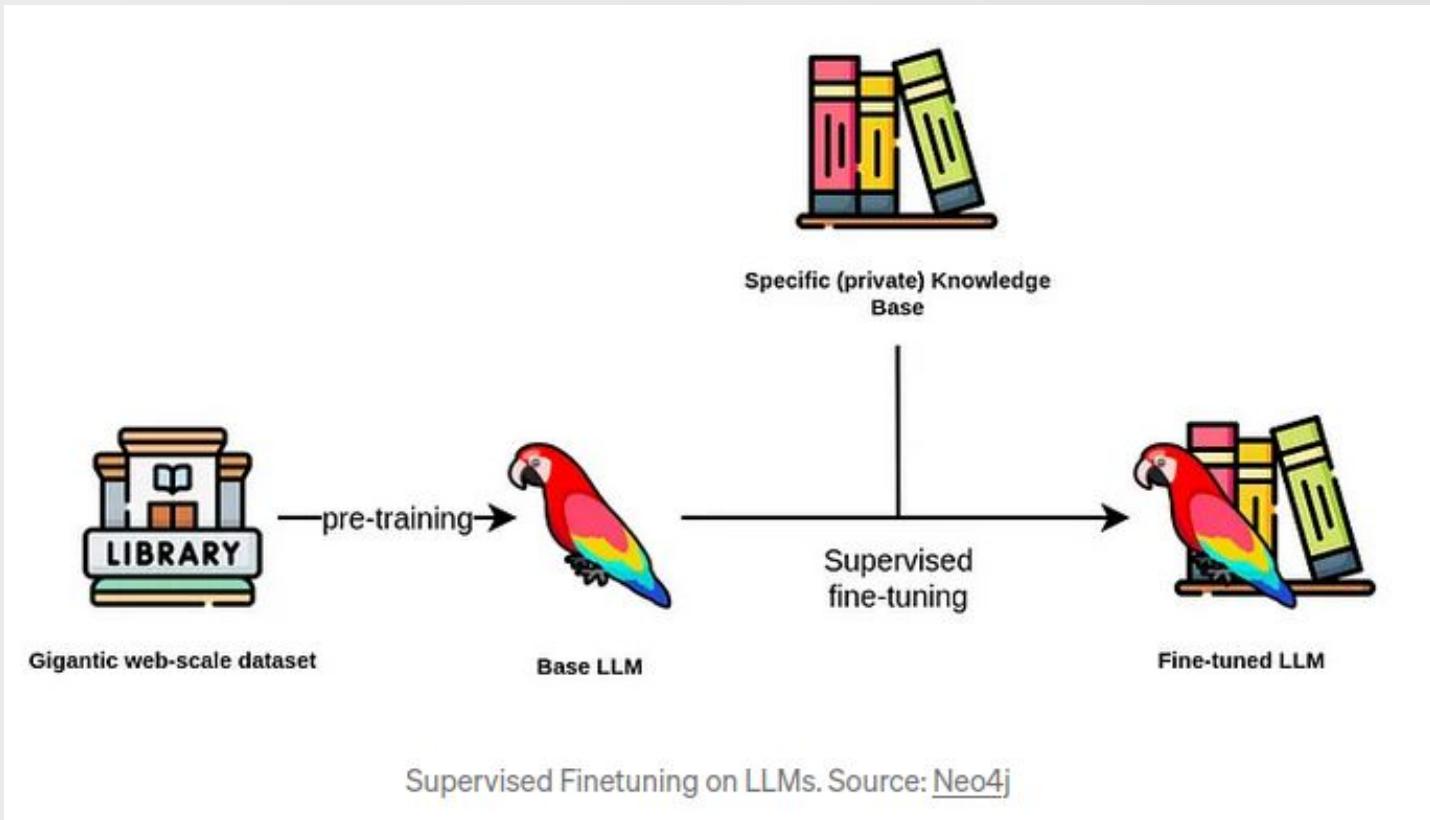
## De GPT à ChatGPT ?

# From next-token prediction to chatbots

- GPT pourrait être directement utilisé pour de la conversation.
- En pratique, GPT a été **adapté** pour construire **ChatGPT** :
  - Étape de **supervised fine-tuning**
  - **Reinforcement Learning with Human Feedback**
  - Modération (et certainement pleins d'autres détails...)
- Ce sont des méthodes à savoir mettre en oeuvre aussi pour adapter ChatGPT à nos cas d'usages/données spécifiques → **supervised fine-tuning**
- Il y a pleins d'autres modèles que l'on **pourrait qualifier de LLMs qui ne sont pas vraiment génératifs**, mais tout aussi performants pour le NLP.



# Supervised fine-tuning



# Supervised fine-tuning

- Il faut **modifier l'objectif** pour entraîner le LLM à **prédire une réponse à partir d'une question**.
- Pour LLM, **supervised fine-tuning** = apprentissage supervisé sur Q&A.
- Typiquement le LLM est utilisé de façon générative, à partir de la question, la réponse est générée token par token, puis comparée à la réponse.
- Avec supervised FT : risques de **sur-apprentissage** ou **catastrophic forgetting**
- De nombreuses **autres méthodes pour adapter un LLM** à cas d'usage ou données spécifiques (RAG, RLHF, zero-shot, few-shot, etc).



# Quelques datasets de Q&A

Datasets: rajpurkar squad Tasks: Question Answering Modalities: Text Formats: parquet Sub-tasks: extractive-qa Languages: English Size: 10K+n=100K ArXiv: arxiv/1606.05250 Tags: Croissant Libraries: Datasets pandas Croissant License: cc-by-sa-4.0

Dataset card Viewer Files and versions Community

Dataset Viewer

Downloads last month: 14,101

Auto-converted to Parquet API View in DatasetViewer

Use this dataset Edit dataset card

Papers with Code

Homepage: rajpurkar.github.io Paper: arxiv.org

Size of downloaded dataset files: 16.3 MB

Size of the auto-converted Parquet files: Number of rows: 16.3 MB 98,169

Models trained or fine-tuned on rajpurkar/squad

- mrn8488/t5-base-finetuned-question-generation
- Text2Text Generation - Updated May 2023 - 2.5M tokens - 99k - 99
- Text2Text Generation - Updated Jan 2023 - 2.5M tokens - 266k - 28
- valhalla/t5-base-e2e-qg
- Text2Text Generation - Updated Jan 2023 - 2.5M tokens - 266k - 28

distilbert/distilbert-base-cased-distilled-squad

Question Answering - Updated Jun 2023 - 10M tokens - 101k - 178

valhalla/bart-large-finetuned-squadv1

Question Answering - Updated Jun 2023 - 10M tokens - 101k - 178

distilbert/distilbert-base-uncased-distilled-squad

Question Answering - Updated Mar 2023 - 75.6k - 88

philschmid/distilbert-onnx

Question Answering - Updated Feb 2023 - 30.3k - 2

Dataset Card for SQuAD

Dataset Summary

Stanford Question Answering Dataset (SQuAD) is a reading comprehension dataset, consisting of questions posed by crowdworkers on a set of Wikipedia articles, where the answer to every question is a segment of text, or span, from the corresponding reading passage, or the question might be unanswerable.

SQuAD 1.1 contains 100,000+ question-answer pairs on 500+ articles.

De nombreux datasets disponibles, à la fois pour entraîner mais aussi pour faire des **benchmarks**:

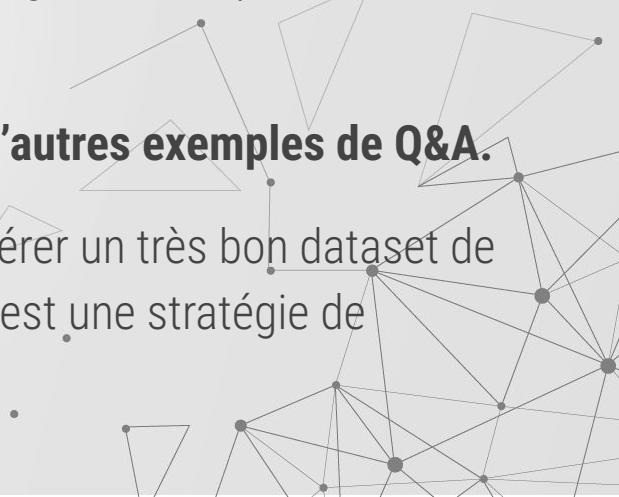
- **SQuAD** (Stanford Question Answering Dataset)
- **Natural Questions (NQ)**
- **TriviaQA** : avec des questions structurées
- **QuAC** : Question Answering in Context

Toujours un peu de travail de manipulation des différents formats.

Context/question/answers

# Augmenter les datasets de Q&A

- Pour les **corpus de Q&A** il y a la limitation de la quantité de données parce qu'il faut créer de la **supervision**.
- Il y a des stratégies de *data augmentation* :
  - **Paraphrasing** : utiliser des modèles pour paraphraser les Q&A existantes
  - **Back-translation** : passer les Q&A existantes d'un langage à l'autre pour créer de la variation.
- **Utiliser un LLM** avec du prompt-engineering **pour générer d'autres exemples de Q&A.**
- Par exemple, on peut utiliser le dernier GPT payant pour générer un très bon dataset de Q&A et ensuite entraîner un plus petit LLM sur ce dataset. C'est une stratégie de **knowledge distillation**.





# 2.4

## Prompt engineering

---

# Prompt Engineering

- Prompt = l'input que l'on donne au modèle. Par exemple, le prompt peut contenir une question et du contexte associé.
- Par exemple, on peut facilement transformer ChatGPT en un classifier grâce à un bon prompt.

```
[1]: system_msg = f"""
You are an assistant that classifies reviews according to their sentiment. \
Respond in json format with the keys: gpt_sentiment and gpt_explanation. \
The value for gpt_sentiment should only be either pos or neg without punctuation: pos if the review is positive, neg otherwise.\
The value for gpt_explanation should be a very short explanation for the sentiment.
"""


```

- Cela peut être fait lorsqu'on interagit avec ChatGPT ou même de façon systématique.
- Le **format de prompt** et certains **tokens spéciaux** sont importants.

# Special tokens pour Mistral

## Control tokens

Our vocabulary starts with 10 control tokens, which are special tokens we use in the encoding process to represent specific instructions or indicators:

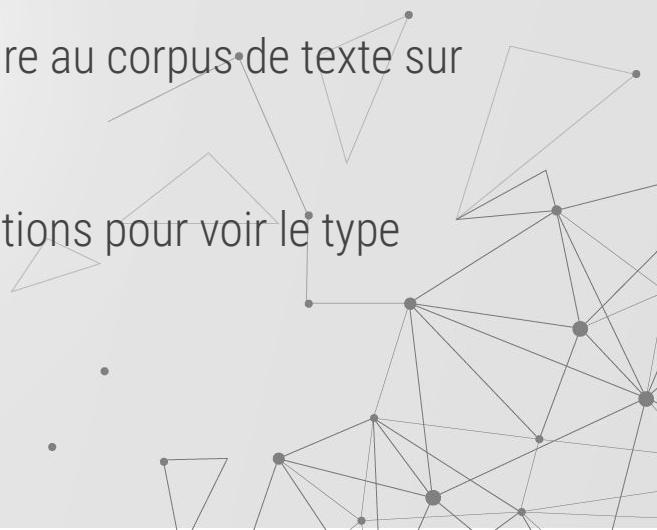
```
<unk>
<s>
</s>
[INST]
[/INST]
[TOOL_CALLS]
[AVAILABLE_TOOLS]
[/AVAILABLE_TOOLS]
[TOOL_RESULTS]
[/TOOL_RESULTS]
```

- <**PAD**> est aussi un exemple de **special token** lié au **padding**.
- <**UNK**> = pour les mots *out-of-vocabulary*.
- **Les special tokens ne sont pas encodés.**
- Variations de choix entre différents LLMs. Lire les documentations !



# Comment modifier le prompt ?

- Permet de donner du **contexte** (ou **instruction**) aux modèles pour qu'ils réalisent mieux la tâche.
- Structurer le prompt pour inclure des **exemples (few-shot prompting)** ou pour **formater le résultat**.
- Comment inclure dans le contexte de l'information extérieure au corpus de texte sur lequel le modèle a été entraîné ?
- On peut expérimenter quantitativement différentes formulations pour voir le type d'instruction qui fonctionne le mieux.



# Quelques conseils de prompt

Il y a plein de blogs sur le *prompt engineering*. Quelques exemples :

- **Spécifier le format de l'output** : "Écrire un paragraphe de 3 phrases sur l'industrie pétrolière."
- Intégrer des exemples positifs et négatifs dans le contexte.

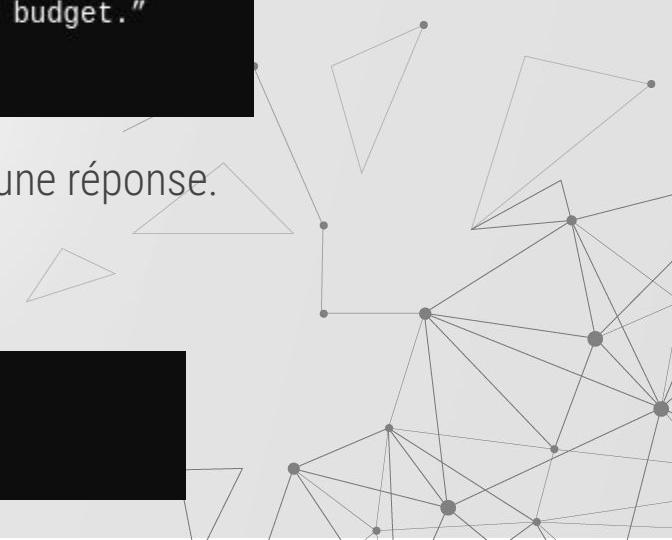
Good example: "The project was completed on time and within budget."

Bad example: "The project finished."

- **Account for variability** : Anticiper les variations possibles d'une réponse.
- **Few-shot prompting** : inclure des exemples de la tâche.

Translate the following sentences to French:

- The cat is on the roof. => Le chat est sur le toit.
- The dog is in the garden. => Le chien est dans le jardin.



# Retrieval Augmented Generation (RAG)

- Souhaitable d'**ajouter du contexte** aux prompts de **façon automatique**.
- Ajouter automatiquement de l'information que le LLM n'a jamais rencontré en puisant dans une base de documents.
- Limiter les **phénomènes d'hallucinations**.
- Capacité de recherche rapide et automatique dans base de documents: les **bases de données vectorielles**.

On verra le RAG en détails demain et après demain !





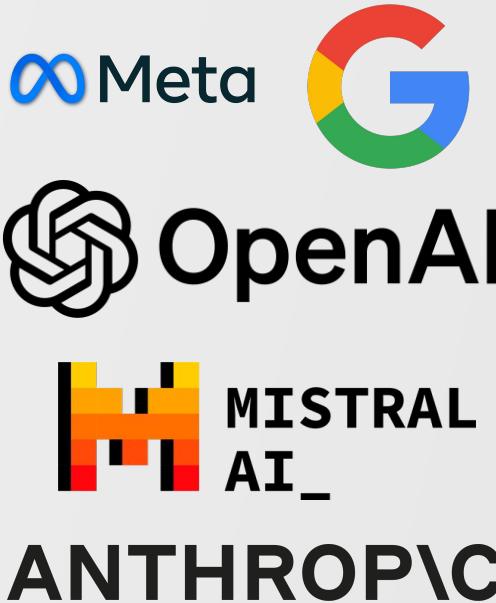
# 3

## LLMs et API

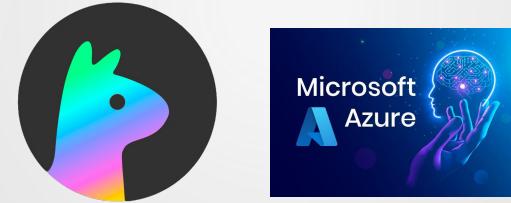
---

# Comment accéder à un LLM ?

Développeurs  
de modèles



Fournisseurs d'accès



Modes d'accès





# 3.1

## Petit point sur APIs

# API

- API = Application Programming Interface
- Permet d'extraire des données d'un serveur à l'aide de **requêtes**
- Le package python **requests** permet de le faire très facilement

```
[16]: import requests
```

- Beaucoup de données sont accessibles par API.

- Une liste d'API publiques :

<https://github.com/public-apis/public-apis>



# Pourquoi des API en Data Science?

Les APIs ont de nombreuses utilités :

- Elles offrent un cadre de travail pour collecter des données qui changent tout le temps (météo, finance).
- Elles permettent de ne sélectionner que **des éléments de base de données qui nous intéressent**. C'est bon pour le trafic.
- **Elles permettent d'externaliser certains éléments de pipeline de travail qu'on ne peut/veut pas faire en interne (LLMs, DALL-E, etc.).**

Les APIs c'est aussi une façon de faire du **web scraping organisé** et de travailler directement avec des **données structurées**.



# Exemples d'API

## Website Carbon API

<https://api.websitedcarbon.com/>



<https://www.alphavantage.co/documentation/>



<https://docs.alpaca.markets/reference/authentication-2>



<https://www.climatiq.io/docs/guides/how-tos/using-python>



<https://en.energinet.dk/energy-data/data-catalog/>



<https://gruenstromindex.de/>



<https://tenders.guru/ua/api>



<https://www.econdb.com/home>

## OPEN VISION API

<https://openvisionapi.com/>



<https://developer.accuweather.com/apis>



<https://fiscaldata.treasury.gov/api-documentation/>

# Une requête

- Une requête, c'est un ensemble de plusieurs éléments :
  - un lien url
  - une méthode
- Une requête, c'est ce qui va permettre d'interagir avec l'API
- L'url s'appelle le **endpoint (on va parler des endpoints OpenAI)**
- **GET**, POST, PUT, PATCH, and DELETE : des méthodes HTTP pour interagir avec un serveur. (caché pour nous)
- **GET** c'est ce qui permet d'obtenir des données: c'est **read-only**.



# Exemple de requête avec Python

- Le **endpoint** doit seulement être un **url valide**
- La réponse n'est pas encore au format JSON
- L'objet *response* a souvent un attribut **status\_code**
- On peut trouver la signification des status\_code sur ce [site](#), par exemple :
  - **404**: l'url n'existe pas
  - **403**: vous n'avez pas les droits d'accès
  - **200**: c'est tout bon

```
[5]: import requests  
endpoint = "https://finley.com/"  
response = requests.get(endpoint)  
  
print(type(response))  
print(response)  
print(response.status_code)  
  
<class 'requests.models.Response'>  
<Response [200]>  
200
```



```
[8]: endpoint = "http://api.open-notify.org/astros.json"
      response = requests.get(endpoint)

      if response.status_code == 200:
          # Lire en JSON la réponse de l'API
          data = response.json()
```

La plupart des API ont ces caractéristiques:

- Elles acceptent les requêtes GET
- Elles retournent une réponse au format JSON
- Elles utilisent des codes de réponse HTTP standards
- Une **documentation** à lire





## 3.2

# LLM APIs

# Un mode d'utilisation répandu

- Tous les LLM providers offrent une solution API
- Globalement très flexible, facile à intégrer dans une pipeline de code : les **API providers** ont toujours intérêt à créer un **écosystème autour de leur outil**.
- Différents niveaux de pricing en fonction de l'utilisation/engagement d'achat
- Sécurité (pour données sensibles) : customisé (mais attention au Cloud Act...)



# Une grande variété de endpoints = plusieurs LLMs

Edit secret key

Name Optional  
LLM\_cours

Permissions  
All Restricted Read Only

Resources Permissions  
Models None Read  
/v1/models

Model capabilities None Write  
/v1/audio  
/v1/chat/completions  
/v1/embeddings  
/v1/images  
/v1/moderations

Assistants None Read Write  
/v1/assistants  
/v1/models (required for Assistants)

Threads None Read Write  
/v1/threads  
/v1/models (required for Threads)

Fine-tuning None Read Write  
/v1/fine\_tuning

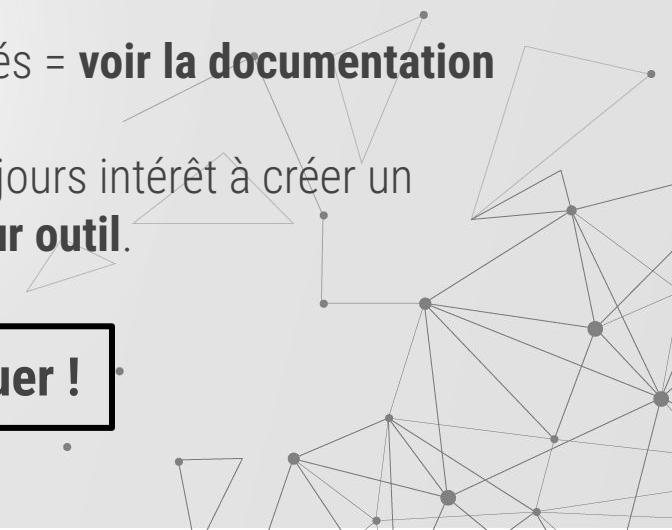
Files None Read Write  
/v1/files

Permission changes may take a few minutes to take effect.

Cancel Save

- OpenAI : multi-modaux, plus seulement LLMs
- Models vs Model capabilities
- Fine-tuning à distance
- Énormément de possibilités = **voir la documentation**
- Les **API providers** ont toujours intérêt à créer un **écosystème autour de leur outil.**

On va surtout pratiquer !



# Le pricing des APIs

- Les **tokens** sont l'**unité de comptabilité** des APIs.
- En général le coût d'un token en input ou output n'est pas le même.
- **Input token** coûte **moins cher** qu'un **output token**.
- Parce que les tokens en inputs sont juste processés, tandis que chaque token en output correspond à **une étape d'inférence d'un LLM** avec de nombreux paramètres (= coûteux...)
- Voir par exemple : <https://openai.com/api/pricing/>



# Le pricing des APIs

## GPT-4o

GPT-4o is our most advanced multimodal model that's faster and cheaper than GPT-4 Turbo with stronger vision capabilities. The model has 128K context and an October 2023 knowledge cutoff.

[Learn about GPT-4o ↗](#)

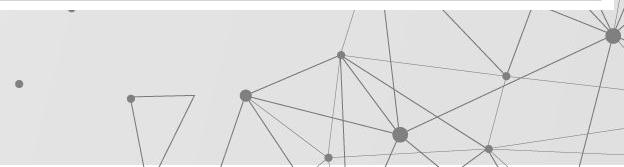
Model	Pricing	Pricing with Batch API*
gpt-4o	\$2.50 / 1M input tokens	\$1.25 / 1M input tokens
	\$1.25 / 1M cached** input tokens	
gpt-4o-2024-08-06	\$10.00 / 1M output tokens	\$5.00 / 1M output tokens
	\$2.50 / 1M input tokens	\$1.25 / 1M input tokens
gpt-4o-2024-05-13	\$1.25 / 1M cached** input tokens	
	\$10.00 / 1M output tokens	\$5.00 / 1M output tokens
	\$5.00 / 1M input tokens	\$2.50 / 1M input tokens
	\$15.00 / 1M output tokens	\$7.50 / 1M output tokens

## Premier models

[Price in \\$](#)

[Price in €](#)

Model	API Name	Description	Input (/M tokens)	Output (/M tokens)
Mistral Large 2	mistral-large-2407	Top-tier reasoning for high-complexity tasks, for your most sophisticated needs.	1.8€	5.4€
Mistral Small 24.09	mistral-small-2409	Cost-efficient, fast, and reliable option for use cases such as translation, summarization, and sentiment analysis.	0.18€	0.54€
Codestral	codestral-2405	State-of-the-art Mistral model trained specifically for code tasks.	0.18€	0.54€
Mistral Embed	mistral-embed	State-of-the-art semantic for extracting representation of text extracts.	0.09€	



# Code pour OpenAI API

```
from openai import OpenAI

from credentials.keys import OPENAI_API_KEY

client = OpenAI(api_key=OPENAI_API_KEY)

prompt = 'Tell me a story about nights and frogs'

response = client.completions.create(
    model="gpt-3.5-turbo-instruct",
    prompt=prompt
)
```



# 4

## PRATIQUE

---



# 4.1

## L'écosystème LLM

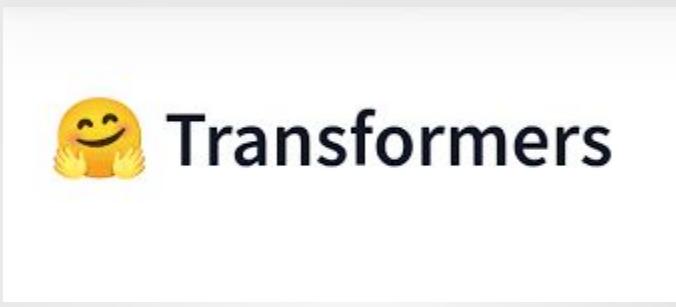
# De nombreux packages Python



mistralai/**mistral-common**



0 5 Contributors    0 5 Issues    ⭐ 454 Stars    33 Forks



# De nombreux modèles accessibles

Hugging Face

Models 731,259  Full-text search

Tasks Libraries Datasets Languages Licenses Other

Multimodal

Image-Text-to-Text Visual Question Answering Document Question Answering

Computer Vision

Depth Estimation Image Classification Object Detection Image Segmentation Text-to-Image Image-to-Text Image-to-Image Image-to-Video Unconditional Image Generation Video Classification Text-to-Video Zero-Shot Image Classification Mask Generation Zero-Shot Object Detection Text-to-3D Image-to-3D Image Feature Extraction

Natural Language Processing

Text Classification Token Classification Table Question Answering Question Answering Zero-Shot Classification Translation Summarization Feature Extraction Text Generation Text2Text Generation Fill-Mask Sentence Similarity

Model	Description	Last Update	Size	Stars
s. stabilityai/stable-diffusion-3-medium	Text-to-Image	Updated 5 days ago	2.81M	2.7k
microsoft/Florence-2-large	Image-to-Text	Updated 5 days ago	12.2k	423
nvidia/Nemotron-4-340B-Instruct		Updated 7 days ago	516	
deepseek-ai/DeepSeek-Coder-V2-Instruct	Text Generation	Updated 5 days ago	1.64k	213
microsoft/Florence-2-large-ft	Image-to-Text	Updated 5 days ago	6.03k	162
meta-llama/Meta-Llama-3-8B	Text Generation	Updated May 13	1.16M	4.88k
deepseek-ai/DeepSeek-Coder-V2-Lite-Instruct	Text Generation	Updated 5 days ago	3.02k	134
facebook/multi-token-prediction		Updated 6 days ago	118	
meta-llama/Meta-Llama-3-8B-Instruct	Text Generation	Updated 26 days ago	2.76M	2.75k
alpindale/magnum-72b-v1	Text Generation	Updated 5 days ago	193	89
GlyphByT5/Glyph-SDXL-v2		Updated 6 days ago	81	
2Noise/ChatTTS	Text-to-Audio	Updated 17 days ago	34.6k	1.09k
fireworks-ai/firefunction-v2	Text Generation	Updated 6 days ago	76	73
microsoft/Florence-2-base	Image-to-Text	Updated 5 days ago	3.43k	72
jasperai/flash-sd3	Text-to-Image	Updated 5 days ago	1.72k	63
openai/whisper-large-v3	Automatic Speech Recognition	Updated 14 days ago	3.45M	2.77k
misstralai/Codestral-22B-v0.1	Text Generation	Updated 13 days ago	18.6k	963
openbmb/MinicPM-Llama3-V-2_5	Visual Question Answering	Updated 9 days ago	121k	1.22k

# De nombreuses ressources

- Beaucoup de blogs, tutoriels ! Notamment le [LLM Course](#)
- De nombreux projets open-source qui couvrent tous les aspects :
  - Inférence d'un LLM sur CPU : **Llama.cpp**
  - Création d'un RAG
  - Mise en production
  - Processing de documents PDF
  - Quantiser des modèles
  - Datasets ([exemple](#))
  - Et bien d'autres encore...



# Mais... “semi” open-source

- Les **poids des modèles sont partagés** mais beaucoup moins de détails sur le **code** et les **données** pour **entraîner un LLM de zéro...**
- Mais toujours beaucoup d'open-source pour **faciliter l'utilisation** des LLMs via des **call APIs**. **Impressionnant de facilité = séduisant !**
- De nombreux détails sont mis **sous le tapis**, de sorte qu'il est difficile d'investir du temps à **maîtriser certaines subtilités**. Par exemple, on peut faire une très grande partie de ce cours sans comprendre les détails de deep learning !
- **Quelques barrières en entrée** : les bons packages pour traiter des PDFs automatiquement ne sont pas utilisables commercialement.
- La taille des modèles force la concurrence à investir **d'énormes ressources** computationnelles **sans considération sur l'équilibre économique**.

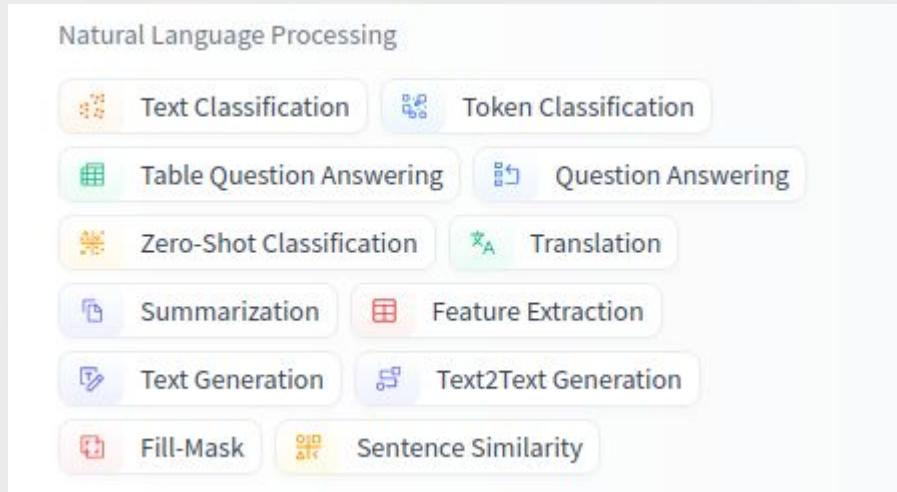


# 4.2

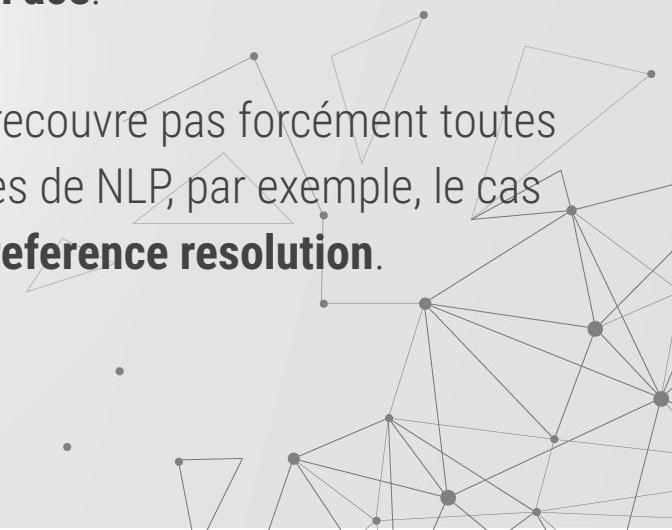
## Tâches diverses

---

# Les tâches de NLP



- Catégorisation des types de modèles pour le texte disponibles sur **HuggingFace**.
- Cela ne recouvre pas forcément toutes les tâches de NLP, par exemple, le cas de la **coreference resolution**.



# Catégories de LLM

- Il y a de nombreuses offres de LLMs et de distinctions entre les différents modèles.
- **Causal, auto-régressif, génératif** = à peu près la même chose = LLM entraîné sur tâche de next-token-prediction.
- Distinction entre **base, instruct** ou **chat** models.
- **Instruct** = LLM fine-tuned pour suivre des instructions via prompt.
- **Chat** = LLM fine-tuned pour les chatbots.
- **Base** = LLM sans fine-tuning
- Distinction entre **decoder-only, encoder-only, encoder-decoder**



# Prompting pour classification

- Utiliser un LLM Q&A pour faire de la **classification**.
- **Zero-shot learning** = mettre des instructions pour forcer une réponse concise (et donc de pouvoir faire de la classification)
- **Few-shot learning** = intégrer des exemples dans le prompt pour diriger le modèle.
- On parle de zero/few-shot prompting.

```
question = "Is the movie associated with this review positive or negative ?"
review = "Very bad movie"

system_msg = f"""
You are an assistant that classifies reviews according to their sentiment. \
Respond in json format with the key: gpt_sentiment.\
The value for gpt_sentiment should only be either pos or neg without punctuation: pos if the review is positive, neg otherwise.\n"""

```

# Table Q&A

## Inputs

Rank	Name	No.of reigns
1	lou Thesz	3
2	Ric Flair	8
3	Harley Race	7

## Question

What is the number of reigns for Harley Race?

Table Question  
Answering  
Model

## Output

### Result

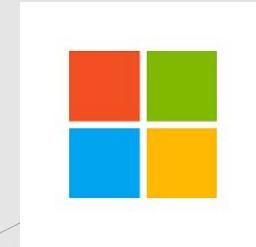
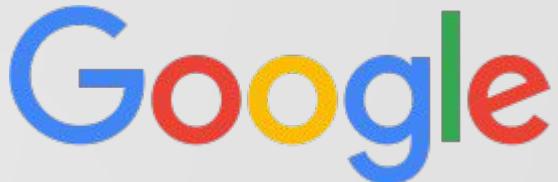
7

# Table Q&A

- TAPAS : Table-based Pretraining and Answer Selection
- TAPEX : Table Pre-Training via Learning a Neural SQL Executor

Tout comme ChatGPT, ces modèles ont été fine-tunés de façon supervisée avec de nombreuses bases de données disponibles :

- **WTQ** Wiki Table Questions
- **WikiSQL**
- **TabFact**
- **Spider**



# Text Summarization

- Pas de endpoint spécifique OpenAI pour faire du text summarization.
- Mais API suffisamment versatile pour obtenir de bonnes performances.

```
# Input text to summarize
input_text = """
In a groundbreaking discovery, scientists have found evidence of water on Mars.
The discovery was made using a new technique involving spectroscopy.
"""

# Generate a summary using the OpenAI API
response = client.completions.create(
    model="gpt-3.5-turbo-instruct",
    prompt=f"Summarize the following text: {input_text}",
    max_tokens=100
)

print(response.choices[0].text.strip())
Scientists have found water on Mars using a new spectroscopy technique.
```



# Text Summarization

- La limite est que le texte à résumer doit tenir dans la **fenêtre de contexte du LLM**
- Il suffit de diviser le texte en bout de taille inférieure ou égale à la taille de la fenêtre de contexte du LLM et d'agréger progressivement ces différentes parties
- Possible d'intégrer le résumé des parties précédentes

Voir TP



# Named Entity recognition (NER)

Person

p

Loc

l

Org

o

Event

e

Date

d

Other

z

Barack Hussein Obama II ✖ (born August 4, 1961 ✖ ) is an American ✖ attorney and politician who served as the 44th President of the United States ✖ from January 20, 2009 ✖ , to January 20, 2017 ✖ . A member of the Democratic Party ✖ , he was the first African American ✖ to serve as president. He was previously a United States Senator ✖ from Illinois ✖ and a member of the Illinois State Senate ✖ .

# Coreference Resolution

*“I voted for Nader because he was most aligned with my values,” she said.*

- Prompting n'est pas toujours une solution optimale (qualité, coût de calcul).
- Modèles spécialisés peuvent aussi être pertinent pour certaines tâches bien spécifiques.
- <https://nlp.stanford.edu/projects/coref.shtml>

# Part-of-speech tagging



Part Of Speech Tagging

# Les tâches des modèles d'embedding

```
response = client.completions.create(  
    model="gpt-3.5-turbo-instruct",  
    prompt=prompt  
)
```

```
response = client.embeddings.create(  
    model="text-embedding-ada-002",  
    input="king"  
)
```

- Embedding vs completions = pas juste une différence de nom.
- Embedding : **encoder-only** ; completions : **decoder-only**.
- Embedding est utile pour faire du **text-classification** (suivi de ML classique).
- **Clustering**.
- Information Search/Retrieval (les RAG).





5

**LLMs sans API**



# 5.1

## LE HARDWARE

---

# CPU et GPU

Trois notions/composantes importantes d'un ordinateur : **RAM**, **CPU** et **GPU**.

- **RAM** (Random Access Memory) : **la mémoire vive**, "ce dont on peut parler sans avoir à réfléchir". **Le GPU et le CPU ont chacun leur RAM.**
- **CPU** : Central Processing Unit. Processeur "classique", développé par Intel. Exécute un peu toutes les tâches.
- **GPU** : Graphics Processing Unit. C'est spécialisé pour faire de **nombreux calculs simples** en parallèle. Pour les jeux vidéos puis maintenant le deep.
- Plus récemment :
  - ASIC (Application Specific Integrated Circuit)
  - TPU (Tensor Processing Unit)
  - LPU (Language Processing Unit)
  - Neuromorphic chips



# Le Hashrate des GPU pour les réseaux de neurones

- **Hashrate** = mesure du nombre d'opérations qui peuvent être réalisées par seconde. Un hashrate of 1TH/s signifie que 1000 milliards d'opérations peuvent être effectuées par seconde.
- **Deep Learning = beaucoup de produits de tenseurs en parallèle,** par ex. convolution = produits de tenseurs.
  - Matrice = Tenseur à 2 dimensions.
  - Tenseur = stockage de nombres organisé en dimensions.
- Les GPUs sont **efficaces pour des faire des calculs simples en parallèle,** par ex. produit de matrices.



# GPU et réseaux de neurones

- La RAM de la GPU et CPU sont importantes parce qu'elle définit ce que l'on pourra faire avec notre ordinateur.
- Pour nous il faudra **stocker les paramètres du réseau de neurones dans la mémoire RAM de la GPU**.
- En termes de code il faut qu'on explicite dans Python **où on veut que l'information soit stockée**.

```
[3]: import torch  
torch.cuda.is_available()
```

```
[3]: True
```

```
[10]: import torch  
  
device = 'cuda' if torch.cuda.is_available() else 'cpu'  
tensor = torch.normal(0, 1, (20, 30)).to(device)  
print(f'Ce tenseur est sur {tensor.device}')  
  
Ce tenseur est sur cuda:0
```



# Accès à des GPUs facilité

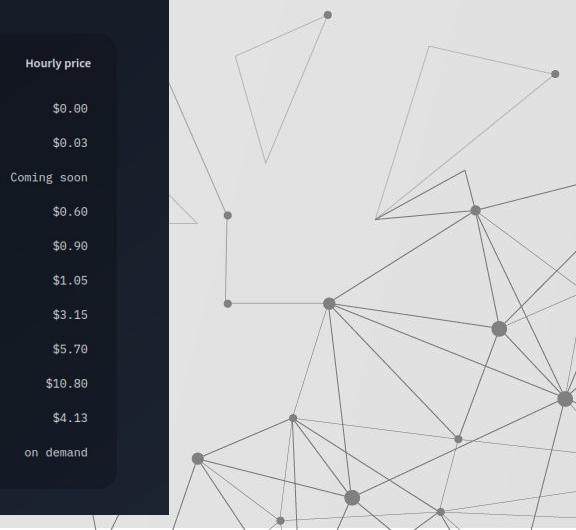
- Les GPUs deviennent de **plus en plus portables** et **moins chères**.
- Google Colab permet d'avoir accès à une GPU gratuitement
- Le **cloud** donne facilement accès à des GPUs à moindre coût.

 **Spaces Hardware** Starting at \$0

Spaces are one of the most popular ways to share ML applications and demos with the world.  
Upgrade your Spaces with our selection of custom on-demand hardware:

[→ Get started with Spaces](#)

Name	CPU	Memory	GPU	GPU memory	Hourly price
CPU Basic	2 vCPU	16 GB	-	-	\$0.00
CPU Upgrade	8 vCPU	32 GB	-	-	\$0.03
CPU XL	16 vCPU	124 GB	-	-	Coming soon
↳ Nvidia T4 - small	4 vCPU	15 GB	Nvidia T4	16GB	\$0.60
↳ Nvidia T4 - medium	8 vCPU	30 GB	Nvidia T4	16GB	\$0.90
↳ Nvidia A10G - small	4 vCPU	15 GB	Nvidia A10G	24GB	\$1.05
↳ Nvidia A10G - large	12 vCPU	46 GB	Nvidia A10G	24GB	\$3.15
↳ 2x Nvidia A10G - large	24 vCPU	92 GB	2x Nvidia A10G	48GB	\$5.70
↳ 4x Nvidia A10G - large	48 vCPU	184 GB	4x Nvidia A10G	96GB	\$10.80
↳ Nvidia A100 - large	12 vCPU	142 GB	Nvidia A100	40GB	\$4.13
Custom	on demand	on demand	on demand	on demand	on demand



# Les LLMs sont très gourmands

- Chaque prédiction de nouveau token correspond à une inférence du LLM complet.
- Les LLMs sont de **très gros réseaux de neurones**. Ils sont donc très coûteux à **entraîner** mais aussi à **déployer** (en inférence) pour de nombreux utilisateurs.
- Pour les **fine-tuner**, des méthodes dites de **Parameter Efficient Fine-Tuning (PEFT)** ont été développées comme **LoRA** ou **QLoRA** pour ne pas faire de réentraînement sur tous les paramètres initiaux du LLM.
- Et pour les fine-tuner, tout comme pour l'inférence, le **pruning**, la **distillation** et la **quantization des paramètres** sont des stratégies indispensables à maîtriser.



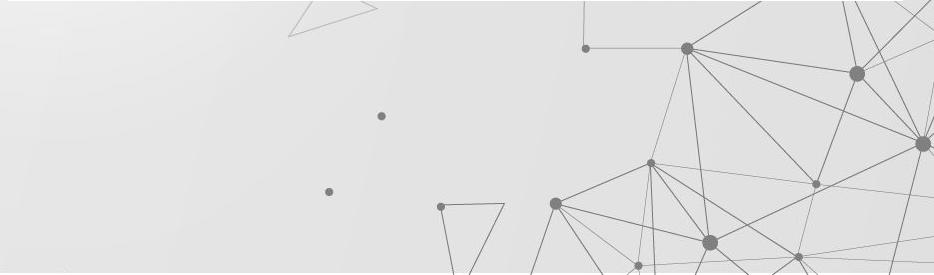
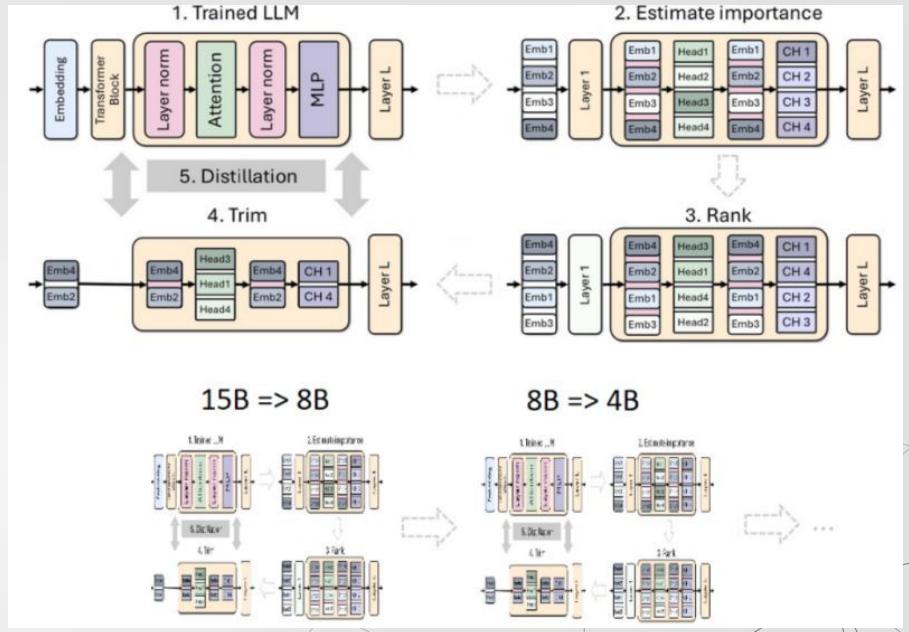
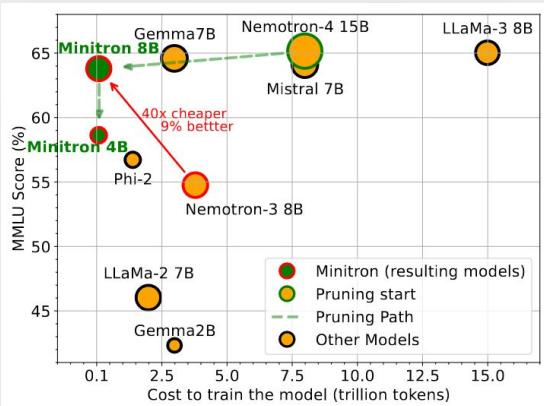
# Pruning et distillation

## Compact Language Models via Pruning and Knowledge Distillation

Saurav Muralidharan\* Sharath Turuvekere Sreenivas\* Raviraj Joshi  
Marcin Chochowski Mostofa Patwary Mohammad Shoeybi Bryan Catanzaro  
Jan Kautz Pavlo Molchanov

NVIDIA

{sauravm, sharath, ravirajj, mchochowski, mpatwary, mschoeybi,  
bcatanzaro, jkautz, pmolchanov}@nvidia.com



# Interaction code/hardware

En deep learning, le **type de float** que l'on considère dans nos tenseurs a beaucoup d'importance. Stocker des chiffres sur du **float64** est souvent trop coûteux pour un **gain minime**. En revanche les stocker sur du **float16**, c'est très avantageux en terme de mémoire et de calcul mais cela peut amener à des **instabilités importantes**.

```
import torch

device = 'cuda:0' if torch.cuda.is_available() else 'cpu'

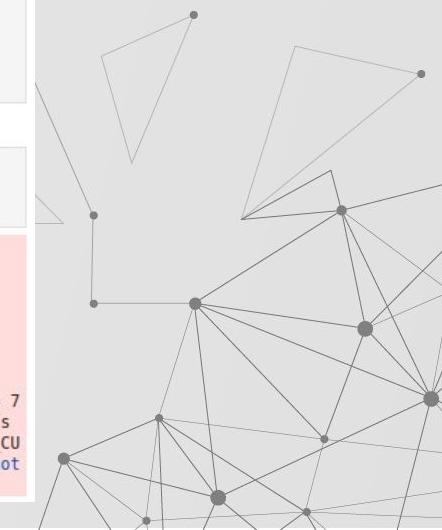
exemple_tensor = torch.rand((10000, 10000)).to(device)
# dtype = le data type.
print(exemple_tensor.dtype)

torch.float32

# https://pytorch.org/docs/stable/tensors.html
# bytes = 8 bits
exemple_tensor = exemple_tensor.type(torch.float64)
```

```
OutOfMemoryError                                     Traceback (most recent call last)
Cell In[3], line 3
      1 # https://pytorch.org/docs/stable/tensors.html
      2 # bytes = 8 bits
----> 3 exemple_tensor = exemple_tensor.type(torch.float64)
```

```
OutOfMemoryError: CUDA out of memory. Tried to allocate 7.45 GiB. GPU 0 has a total capacity of 7.75 GiB of which 3.76 GiB is free. Process 7236 has 170.00 MiB memory in use. Including non-PyTorch memory, this process has 3.82 GiB memory in use. Of the allocated memory 3.73 GiB is allocated by PyTorch, and 1.30 MiB is reserved by PyTorch but unallocated. If reserved but unallocated memory is large try setting PYTORCH_CUDA_ALLOC_CONF=expandable_segments=True to avoid fragmentation. See documentation for Memory Management (https://pytorch.org/docs/stable/notebooks/cuda.html#environment-variables)
```





# 5.2

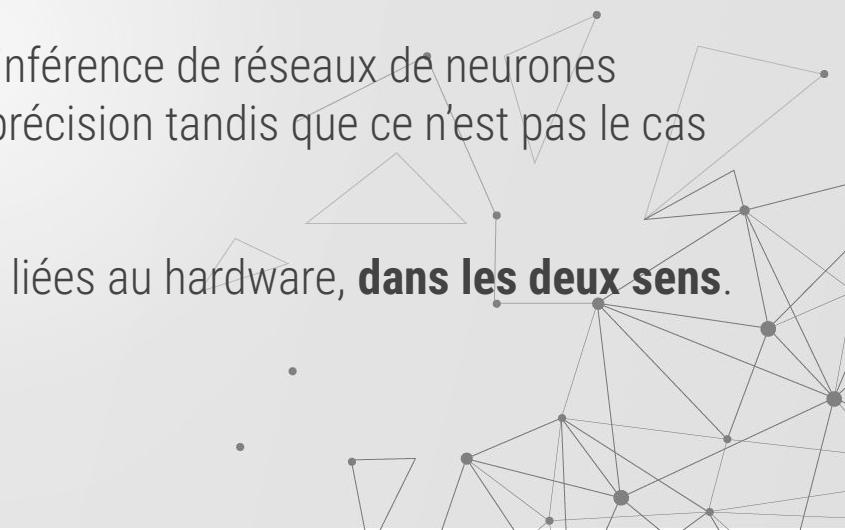
## QUANTIZATION

---

# Quelques généralités sur la quantization

**Quantization** = technique pour **réduire la taille des modèles** de deep learning (donc LLMs) en **réduisant la précision de leurs paramètres/poids.**

- On distingue la quantization pendant **l'entraînement**, pendant le **fine-tuning** ou a posteriori pour l'**inférence**.
- Certaines opérations liées à l'entraînement ou l'inférence de réseaux de neurones doivent être faites avec des scalaires en haute précision tandis que ce n'est pas le cas pour d'autres.
- Les méthodes de quantization peuvent être très liées au hardware, **dans les deux sens.**
- **Trade-off entre qualité et taille du modèle.**



# Représenter un scalaire

Precision Level	Name	Bit Length	Sign Bits	Exponent Bits	Significand Bits (Mantissa)
FP16	Half Precision	16 bits	1 bit	5 bits	10 bits
FP32	Single Precision	32 bits	1 bit	8 bits	23 bits
FP64	Double Precision	64 bits	1 bit	11 bits	52 bits

- FP32 est **Single Precision** car c'est le format standard, FP64 est deux fois plus précis que FP32 (**Double Precision**) et FP16 est deux fois moins (**Half Precision**)
- **Float-Point** = la méthode pour représenter le scalaire avec des bits.

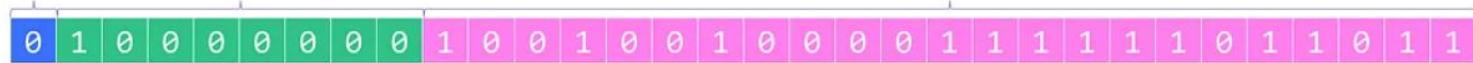


# Représenter un scalaire

## 32-bit float (FP32)

sign            exponent (8 bits)

significand (23 bits)



$$(-1)^0 \times 2^{128-127} \times 1.5707964 = 3.1415927$$

## 16-bit float (FP16)

sign exponent (5 bits)

significand (10 bits)



$$(-1)^0 \times 2^{128-127} \times 1.571 = 3.141$$

## bfloat16 (BF16)

sign            exponent (8 bits)

significand (7 bits)



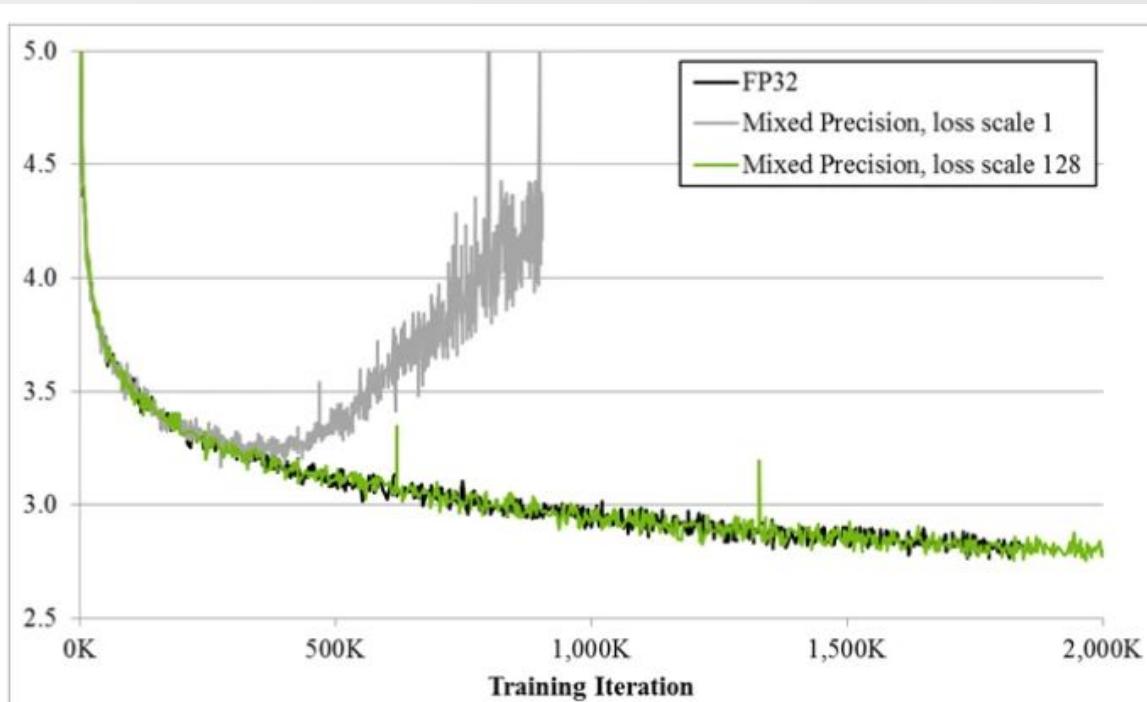
$$(-1)^0 \times 2^{128-127} \times 1.5703125 = 3.140625$$

# Mixed Precision Training

- Entraîner des réseaux de neurones utilise typiquement le plus de précision possible sur les scalaires (fp32).
- On peut néanmoins utiliser un savant mélange de 16-bit (half-precision) et 32-bit (precision) pour accélérer entraînement : **mixed-precision training**.
- Pour le fine-tuning, il existe des méthodes (telles que QLoRA) pour procéder avec des poids déjà quantizés.



# Mixed Precision Training

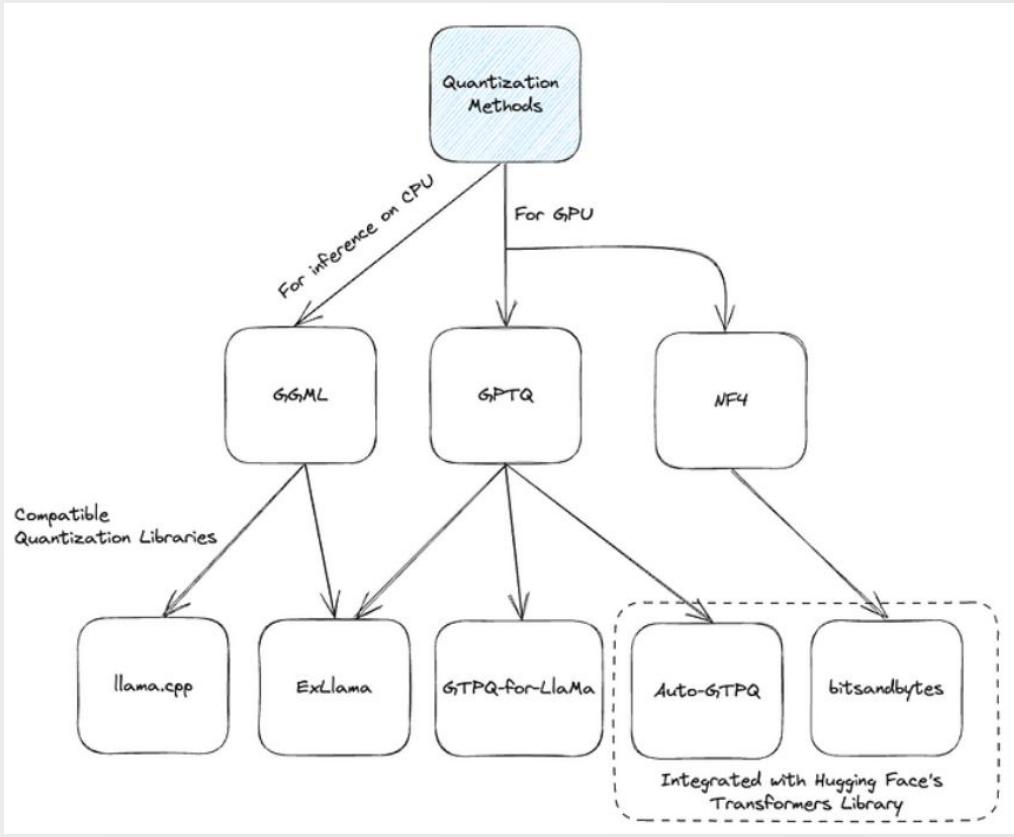


Difficile d'**entraîner un modèle** avec des paramètres avec une **précision réduite**, d'où la **quantization a posteriori**.



Possible avec NVIDIA Apex et Pytorch AMP.

# Les différentes méthodes de quantization



La quantization passe de **FP16**, **FP32** ou **FP64** à **INT4**, **INT6** ou **INT8**.

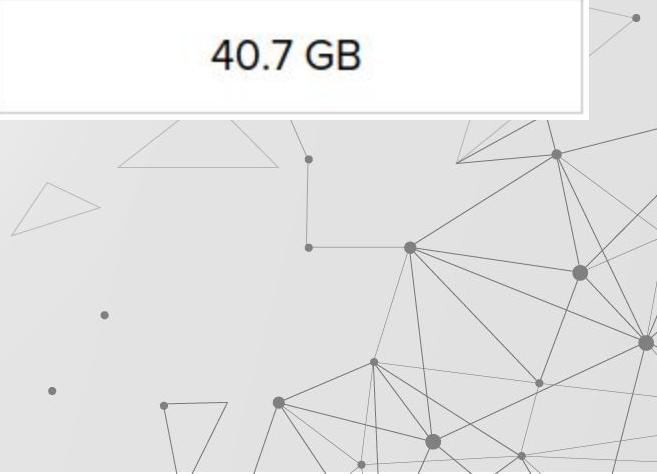
Il y a plusieurs méthodes différentes : **NF4**, **GPTQ** ou **GGML**.

C'est un travail important de quantiser un gros LLM.

Voir blog en référence pour comprendre en détail le fonctionnement.

# Taille des modèles quantisés

Model	Original Size (FP16)	Quantized Size (INT4)
Llama2-7B	13.5 GB	3.9 GB
Llama2-13B	26.1 GB	7.3 GB
Llama2-70B	138 GB	40.7 GB



# Quelques résultats empiriques

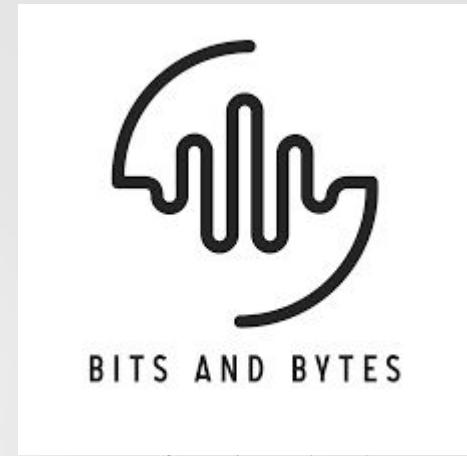
**La plupart des résultats de quantization sont empiriques: il faut faire de nombreuses expériences pour garantir l'effet.**

- De grands modèles quantisés sont plus performants que de petits non-quantisés.
- Les grands modèles souffrent peu de la *quantisation*.
- Certaines méthodes de quantization (NF4) n'impliquent pas de baisses de performances sur de très grands modèles.
- Pour grands modèles, 4-bit est parfait, pour petits plutôt opter pour 6/8 bits.



# Les librairies sur quantization

- Il est possible de quantizer soi-même un LLM.
- Mais en pratique il y a de nombreuses solutions déjà disponibles pour la plupart des modèles.
- La quantization se fait à l'aide d'un dataset.
- Pas de détails, pour l'instant c'est juste important pour pouvoir comprendre les noms des différents modèles.



# Quantizer un modèle

```
!pip install bitsandbytes

from transformers import AutoModelForCausalLM, AutoTokenizer, BitsAndBytesConfig
import torch

nf4_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_use_double_quant=True,
    bnb_4bit_compute_dtype=torch.bfloat16
)

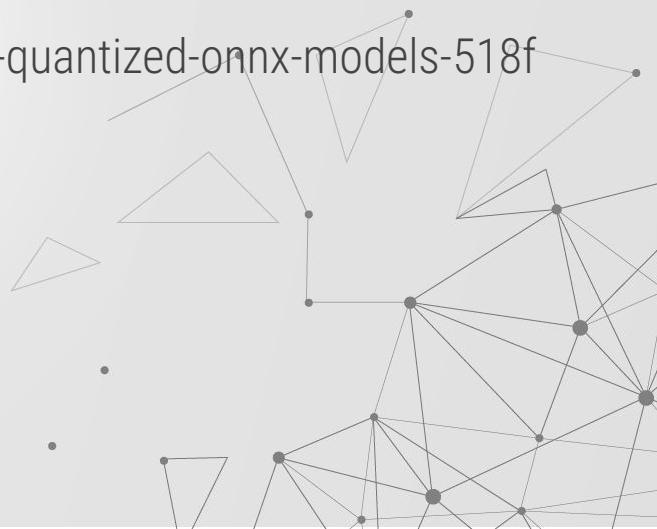
model_name = "PY007/TinyLlama-1.1B-step-50K-105b"

tokenizer_nf4 = AutoTokenizer.from_pretrained(model_name, quantization_config=nf4_config)

model_nf4 = AutoModelForCausalLM.from_pretrained(model_name, quantization_config=nf4_config)
```

# Blogs pour aller plus loin

- <https://www.tensorops.ai/post/what-are-quantized-langs> : **excellent**
- <https://arxiv.org/abs/2210.17323>
- <https://towardsdatascience.com/introduction-to-weight-quantization-2494701b9c0c>
- <https://codezen.medium.com/blazing-fast-inference-with-quantized-onnx-models-518f23777741>





# 5.3

## LLMs en local

# Format échange des modèles

	Short Context	Long Context
Mini	4K [HF] ; [ONNX] ; [GGUF]	128K [HF] ; [ONNX]
Small	8K [HF] ; [ONNX]	128K [HF] ; [ONNX]
Medium	4K [HF] ; [ONNX]	128K [HF] ; [ONNX]
Vision		128K [HF] ; [ONNX]

- Plusieurs formats pour échanger les modèles : HF, ONNX ou GGUF.
- Open Neural Network Exchange (ONNX) est développé par Microsoft.
- GGUF serait le mieux mais pas bien interopérable avec HuggingFace.



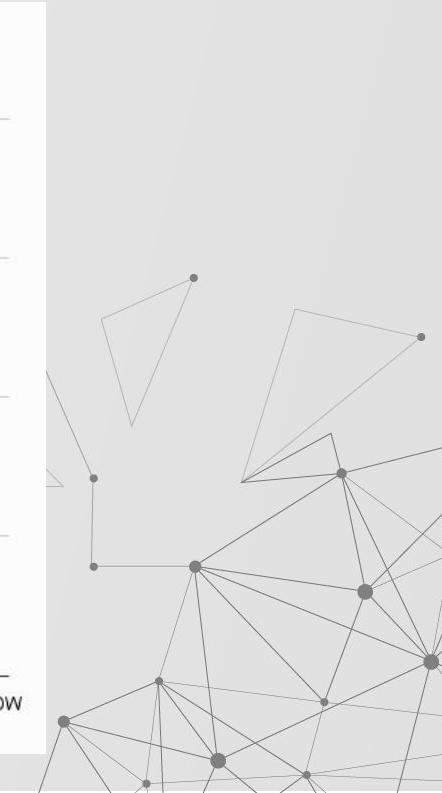
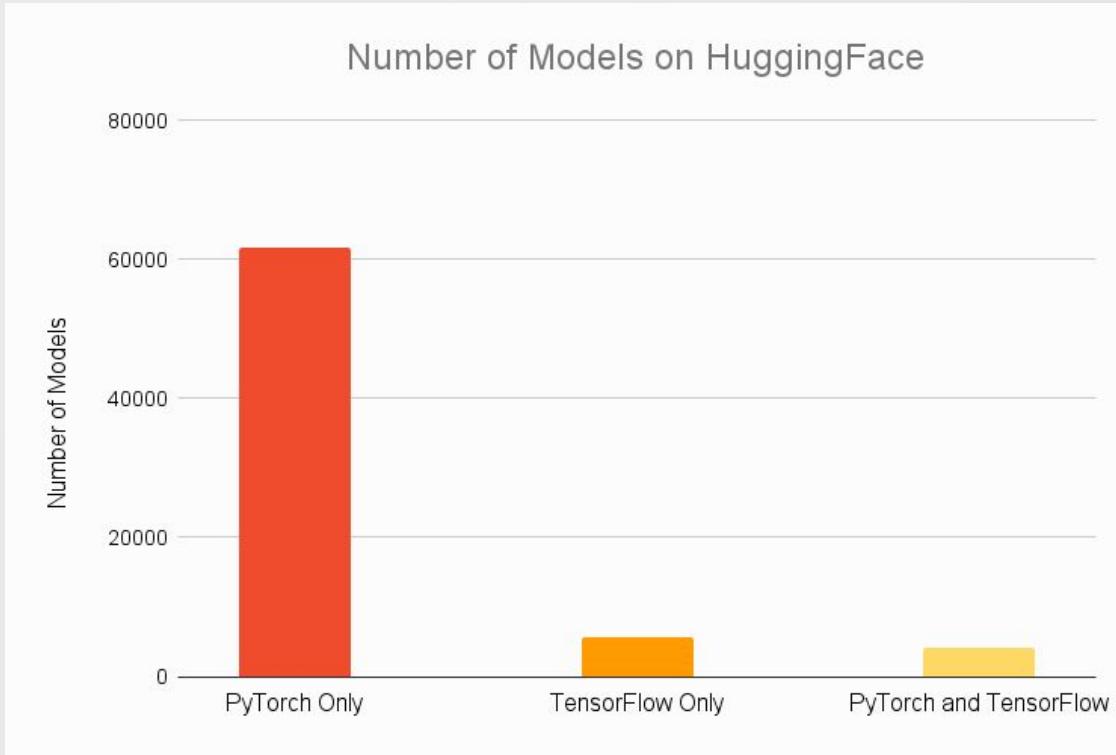
# Format échange des modèles

- Les enjeux sont d'avoir des formats de stockage des poids indépendants des différents software (**pytorch/tensorflow**) et des méthodes (les méthodes de quantization).
- Avoir des formats qui utilisent des accélérations internes le plus possible.
- En Python, il y a quelques grands frameworks de deep : PyTorch, TensorFlow, Keras ou JAX.



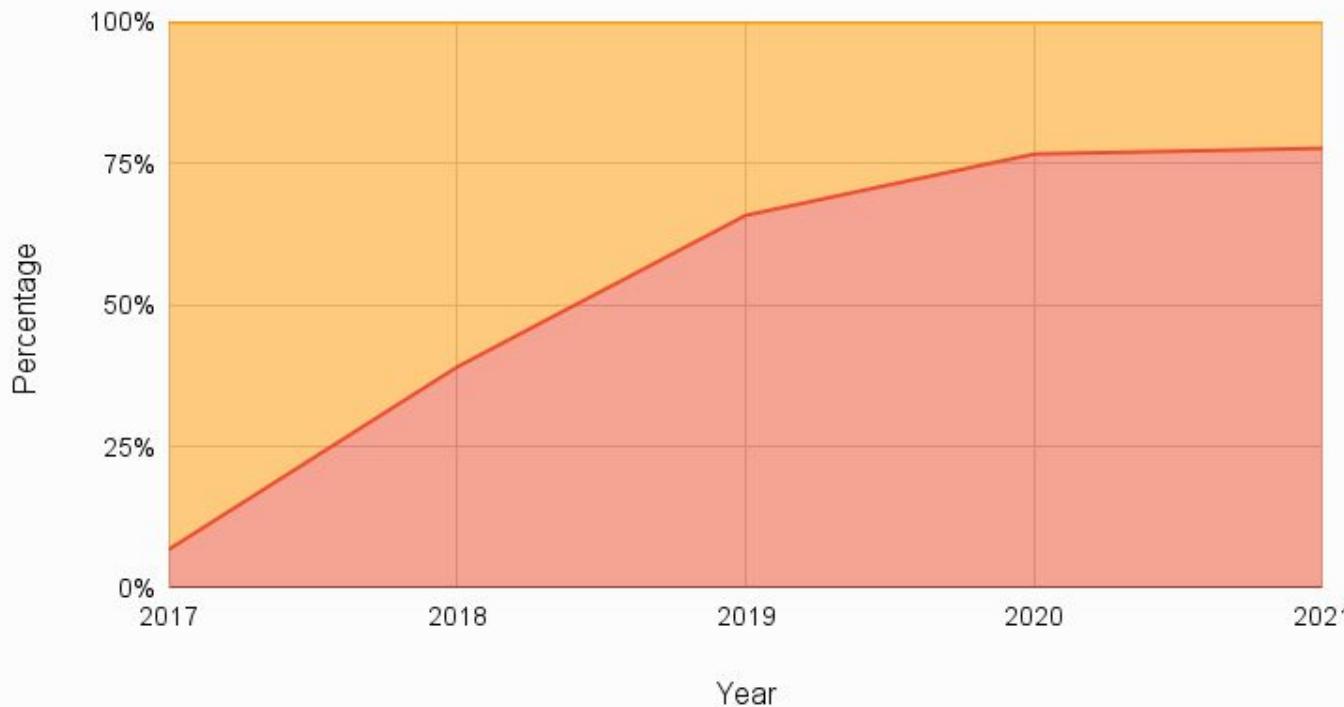
# Pytorch en chiffres

**HuggingFace** c'est LE site de référence pour stocker et échanger des modèles de deep learning

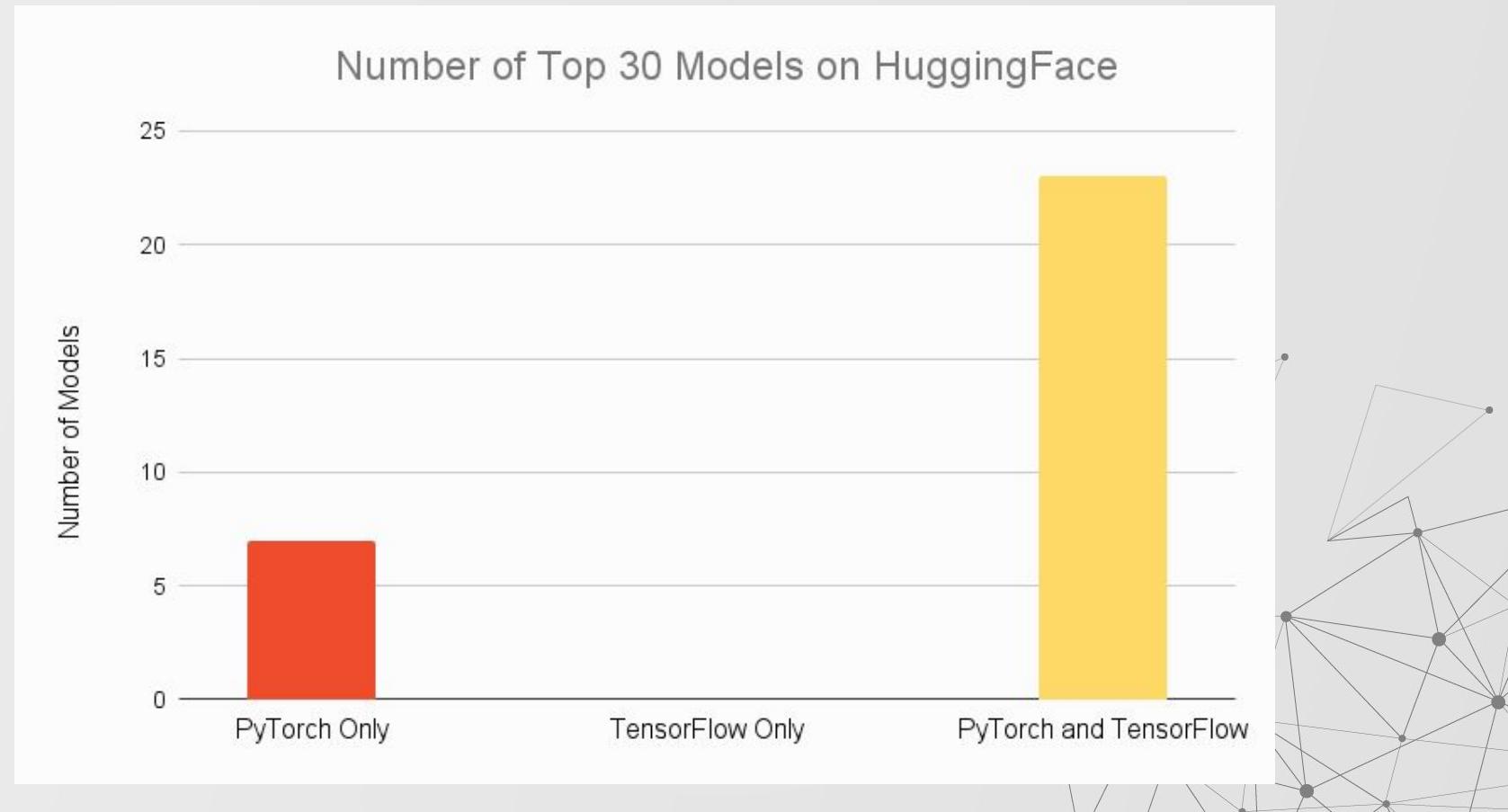


# Pytorch en chiffres

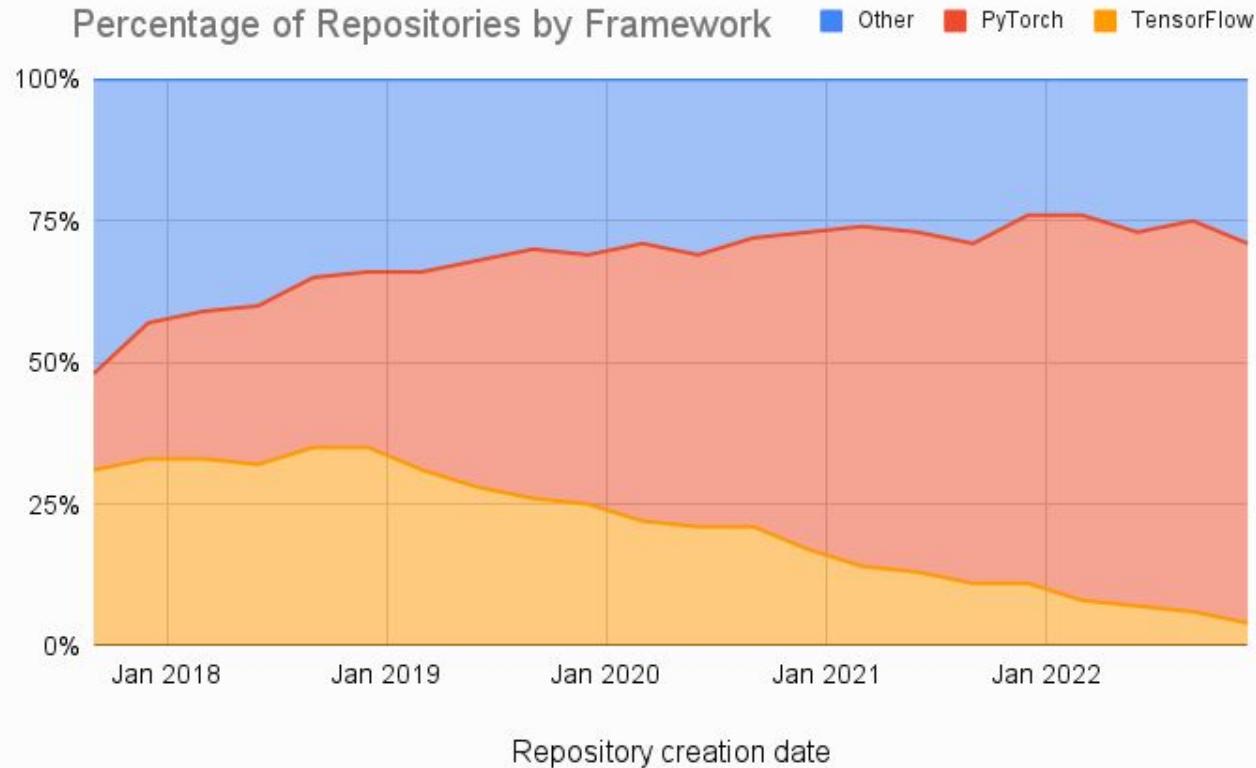
Fraction of Papers Using PyTorch vs. TensorFlow



# Pytorch en chiffres



# Pytorch en chiffres



# Dans Python

[!\[\]\(36affb5ce185120d3d9811e17255ea8f\_img.jpg\) PyTorch](#)[Get Started](#)[Ecosystem](#)[Edge](#)[Blog](#)[Tutorials](#)[Docs](#)[Resources](#)[GitHub](#)[Q](#)

## INSTALL PYTORCH

Select your preferences and run the install command. Stable represents the most currently tested and supported version of PyTorch. This should be suitable for many users. Preview is available if you want the latest, not fully tested and supported, builds that are generated nightly. Please ensure that you have [met the prerequisites below \(e.g., numpy\)](#), depending on your package manager. Anaconda is our recommended package manager since it installs all dependencies. You can also [install previous versions of PyTorch](#). Note that LibTorch is only available for C++.

**NOTE:** Latest PyTorch requires Python 3.8 or later. For more details, see Python section below.

PyTorch Build	Stable (2.2.1)	Preview (Nightly)
Your OS	Linux	Mac
Package	Conda	Pip
Language	Python	C++ / Java
Compute Platform	CUDA 11.8	CUDA 12.1
Run this Command:	<pre>pip3 install torch torchvision torchaudio</pre>	

```
[2]: import torch # pytorch
```

## QUICK START WITH CLOUD PARTNERS

Get up and running with PyTorch quickly through popular cloud platforms and machine learning services.

 Amazon Web Services >

 Google Cloud Platform >

 Microsoft Azure >



# Les différents modèles

```
from transformers import AutoTokenizer, AutoModelForCausalLM

# Specify the model name
model_name = "gpt2" # Replace with the model you want to load

# Load the tokenizer
tokenizer = AutoTokenizer.from_pretrained(model_name)

# Load the model
model = AutoModelForCausalLM.from_pretrained(model_name)

print(f"Le modèle est pour l'instant sur {model.device}")
device = 'cuda:0' if torch.cuda.is_available() else 'cpu'
print(f"Le device disponible est {device}")

Le modèle est pour l'instant sur cpu
Le device disponible est cpu

model.to('cuda:0')
```



# Trouver des quantized models

Hugging Face

Models Datasets Spaces Posts Docs Solutions Pricing Log In

Tasks Libraries Datasets Languages Licenses Other

Filter Tasks by name

Multimodal

Image-Text-to-Text Visual Question Answering

Document Question Answering

Computer Vision

Depth Estimation Image Classification

Object Detection Image Segmentation

Text-to-Image Image-to-Text Image-to-Image

Image-to-Video Unconditional Image Generation

Video Classification Text-to-Video

Zero-Shot Image Classification Mask Generation

Zero-Shot Object Detection Text-to-3D

Image-to-3D Image Feature Extraction

Natural Language Processing

Text Classification Token Classification

Table Question Answering Question Answering

Models 22 gpt2 quantized Full-text search Sort: T

- kiri-ai/gpt2-large-quantized**  
Text Generation • Updated May 23, 2021 • ↴ 27
- huseinzol05/gpt2-345M-quantized**  
Updated Apr 10, 2022
- dmmagdal/gpt2-onnx-js-quantized**  
Text Generation • Updated Jan 6
- dmmagdal/gpt2-large-onnx-js-quantized**  
Text Generation • Updated Jan 6
- irony/gpt2-quantized-jokes**  
Text Generation • Updated Jan 29
- xiaxh/gpt-2-quantized-GPTQ-python-code**  
Text Generation • Updated Feb 20
- camerril/Gen-AI-Quantized-GPTQ-GPT2-camerril**  
Text Generation • Updated Feb 22
- huseinzol05/gpt2-117M-quantized**  
Updated Apr 10, 2022
- brianwoo/GPT2-Onnx-Quantized**  
Text Generation • Updated Dec 6, 2023
- dmmagdal/gpt2-medium-onnx-js-quantized**  
Text Generation • Updated Jan 6
- dmmagdal/gpt2-xl-onnx-js-quantized**  
Text Generation • Updated Jan 6 • ↴ 13
- Irtasam/Quantized-GPT2-Wains**  
Text Generation • Updated Feb 20
- XinpingLuo/My-gpt2-quantized-4bit-model**  
Text Generation • Updated Feb 21
- pratiksatija/gpt2-quantized**  
Text Generation • Updated Feb 22 • ↴ 8

# Trouver des quantized models

```
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Parfois, ce sont des problèmes de compatibilité avec les packages comme auto-gptq
model = AutoModelForCausalLM.from_pretrained('frollini/quantized_gpt2_c4_4bits_with-fixed-dataset', device=device)
```

- Avec HuggingFace, les modèles quantisés ont un attribut **quantized**.
- **HuggingFace** c'est un peu le **réseau social des LLMs** et tout n'est pas forcément bon.

```
if hasattr(model.config, 'quantized') and model.config.quantized:
    print("Model is quantized.")
else:
    print("Model is not quantized.")
```

Model is not quantized.

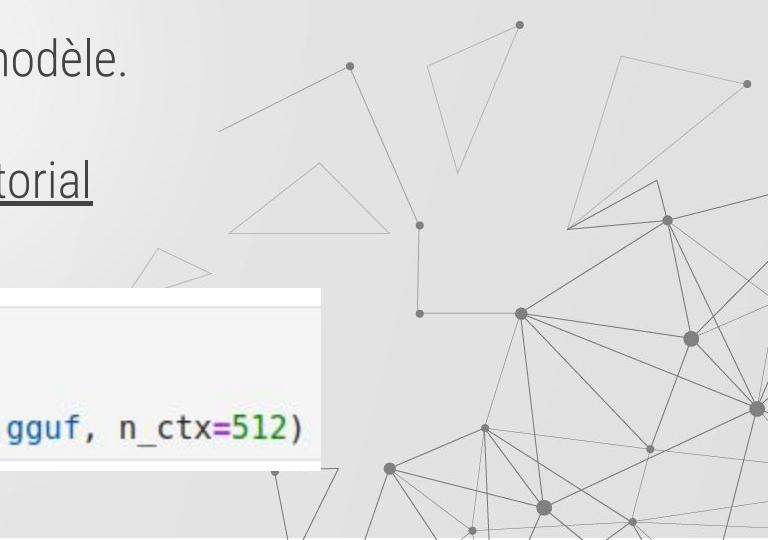


# llama.cpp

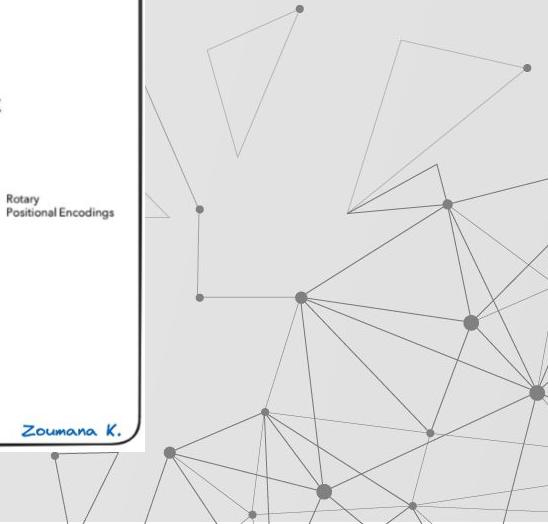
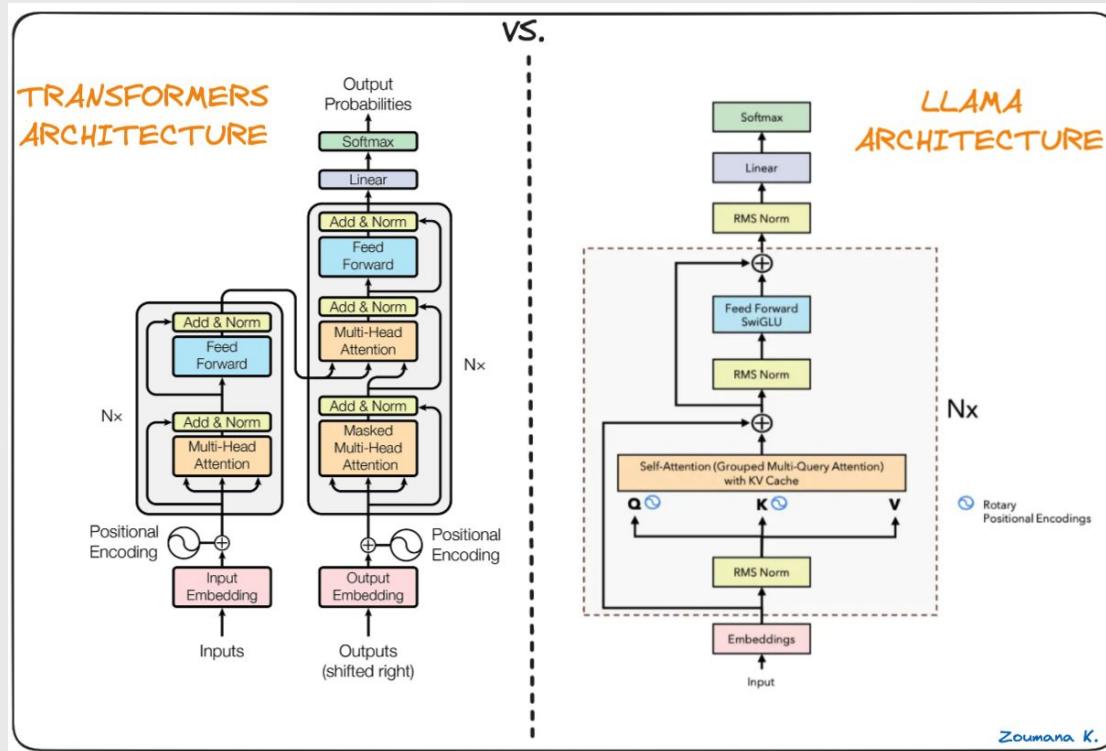
- **Llama.cpp** (cpp pour C++) = est un framework développé par Georgi Gerganov en open-source pour **accélérer l'inférence de LLMs sur CPU**.
- Llama.cpp se focalise sur un **seul type d'architecture de réseau (Llama)** pour obtenir des accélérations particulières importantes pour ce genre de modèle.
- <https://www.datacamp.com/tutorial/llama-cpp-tutorial>



```
from llama_cpp import Llama  
  
zephyr_model = Llama(model_path=zephyr-7b-beta.Q4_0.gguf, n_ctx=512)
```



# llama.cpp



# Les bases d'évaluation

- Beaucoup de bases de données pour évaluer LLMs sur une variété d'aspects.
- Des bases de données de **Q&A très diverses**.
- Des bases de données de **Natural Language Inference** (NLI) pour tester le raisonnement des modèles.
- Ou bases de données de **Semantic Text Similarity** (STS).
- Et bien d'autres...





# 6

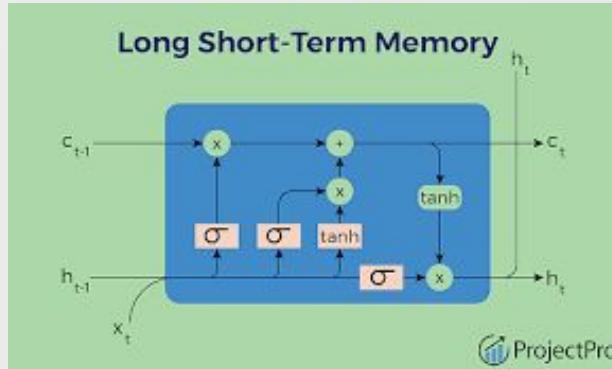
## Les Transformers

# Architectures de réseaux de neurones

- **Architecture d'un réseau de neurones** = famille de méthodes tout comme SVM, arbres, boosting etc.. représentant des familles différentes.
- Le **choix de l'architecture** dépend de la **tâche** et du **type de données**.
- Par exemple, les **réseaux convolutionnels** (CNN) sont très utilisés en image = répétition de pixels sur un grille (homogène et structurée).
- Les **réseaux récurrents (ex. LSTM)** sont très utilisés pour les **données séquentielles** (corrélation temporelles).
- Les **transformers** sont une nouvelle architecture de réseau très intéressante pour les données séquentielles (mais pas que !).



# Gérer les dépendances dans les données



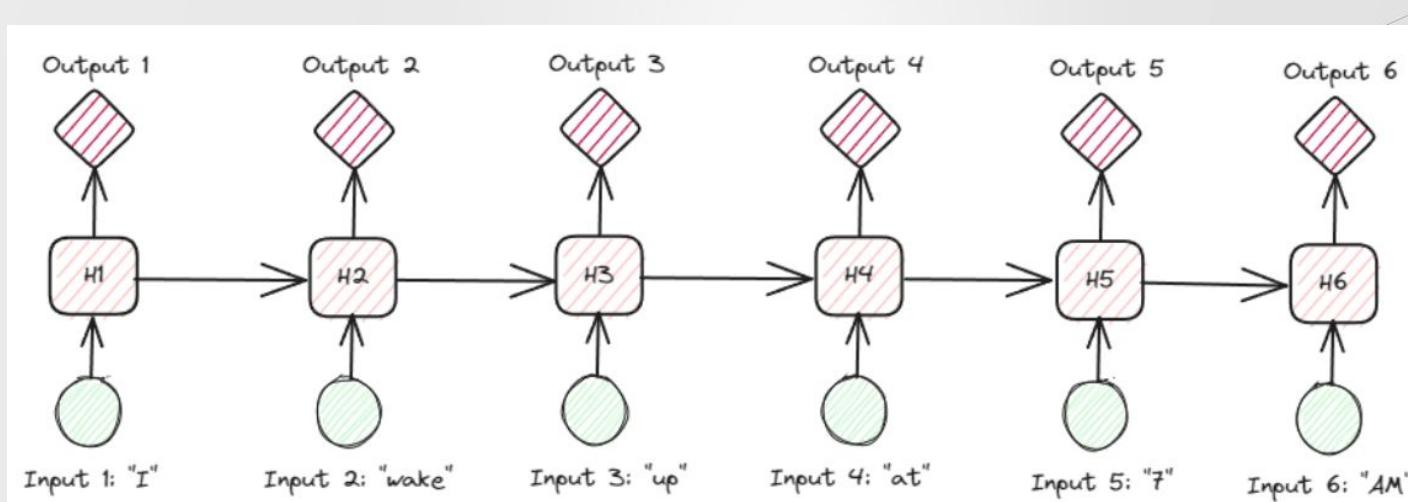
- LSTM
- Gated Recurrent Unit (GRU)
- Transformers

- Conceptuellement, l'enjeu avec les données séquentielles, c'est d'**automatiquement discriminer l'information dans le passé pour faire une prédiction**.
- Pour la tâche de next-token prédiction, **la partie de la phrase la plus importante pour prédire le prochain mot n'est jamais la même et va aussi dépendre du contexte antérieur**. Bref, il faut **des mécanismes pour automatiser cela**.
- On peut appeler cela de la **long-term /short term memory**, de **l'attention**, de la **récurrence** etc...

# Réseaux Récurrents

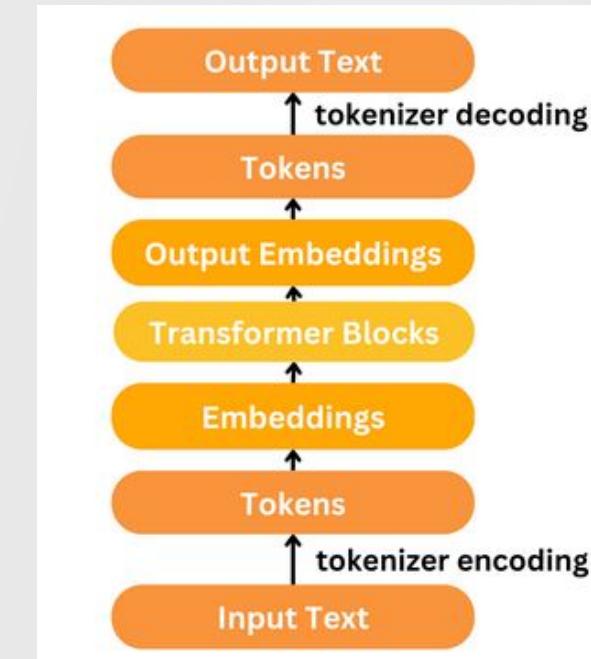
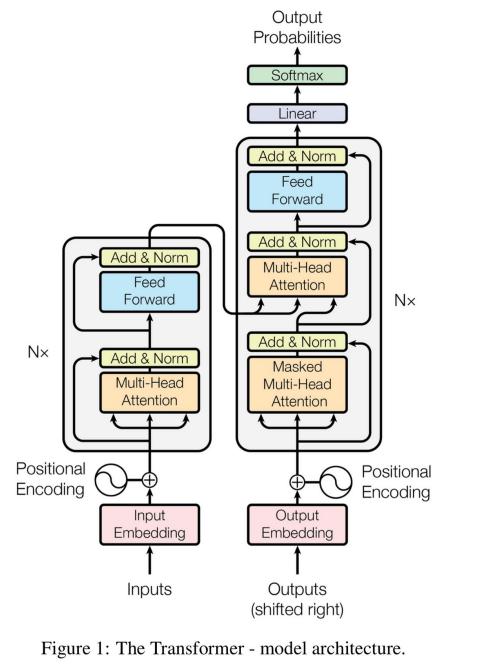
Les **réseaux récurrents analysent l'input séquentiellement un token après l'autre** avec un embedding latent pour **tenter de garder en mémoire le passé**. Ces réseaux souffrent de deux problèmes majeurs :

- Le processus **séquentiel est très peu parallélisable** (GPU) : ils passent donc difficilement à l'échelle parce qu'ils sont lents (**meilleure GPU accélèrera marginalement l'entraînement**)
- En pratique, ces méthodes ont du **mal à garder des informations loin dans le passé**



# Attention et Transformers

Les **transformers** sont une famille de réseaux de neurones qui utilisent des **mécanismes d'attention** pour capturer les **dépendance entre les différents tokens** de la séquence d'entrée.



Juste donner l'intuition derrière le mécanisme d'attention.

**Remplace récurrence par attention.**

**L'attention** permet d'avoir des **embeddings dynamiques** des tokens au lieu d'**embedding fixes** des tokens.

Ainsi, le transformer peut très facilement comprendre le contexte dans une phrase.

# Attention et Transformers

- Process sequences
- Easy to distribute on multiple GPUs
- Faster training than with RNN
- Initially for NLP tasks
- Allows to train huge models on gigantic datasets



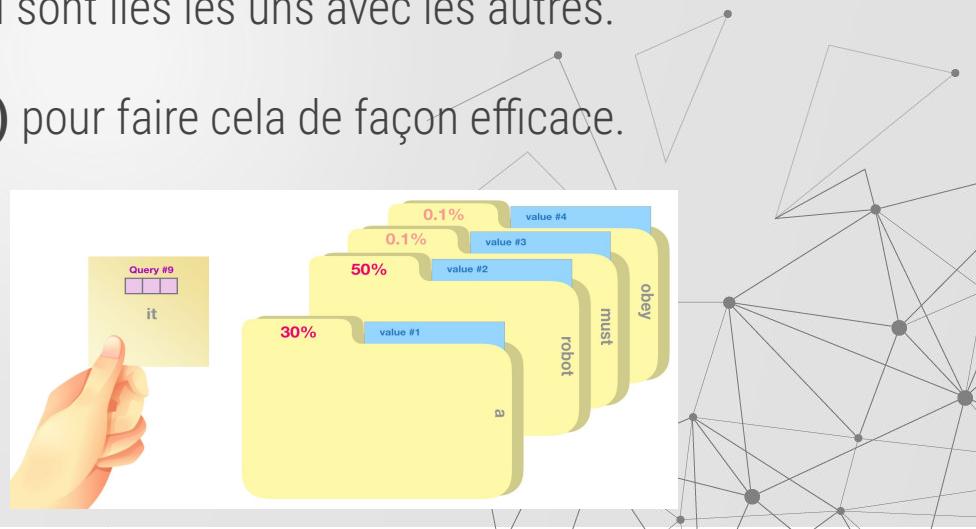
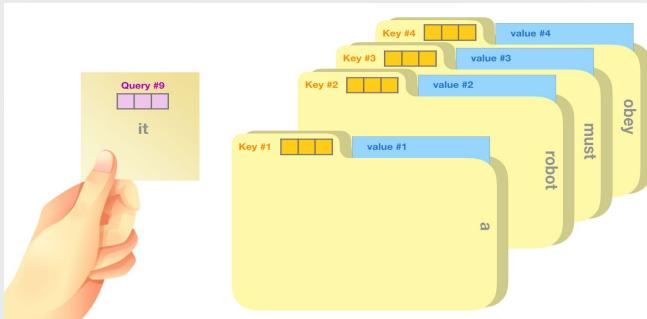
# Attention et Transformers

- Les mécanismes d'attentions existent depuis longtemps (années 90) : comment quantifier la notion d'attention, i.e., une première recherche rapide que l'on vient ensuite raffiner ?
- Ce n'est qu'en 2017 avec les Transformers qu'ils ont démontré empiriquement des performances state-of-the-art pour le texte.
- **Attention = “Pay attention to specific words, no matter how distant they are”**
- C'est la clé pour avoir des embeddings de mots contextualisés.
- Mécanisme = donner un **score d'attention aux paires de mots** dans la séquence d'input.
- Plusieurs types d'attention : **self-attention, cross-attention, local attention**, etc.

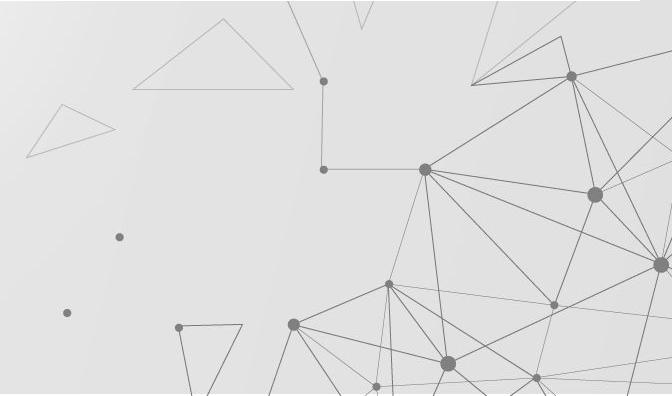
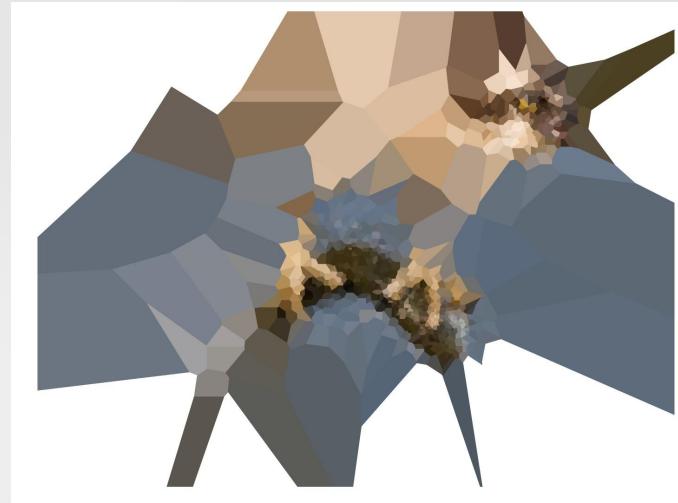
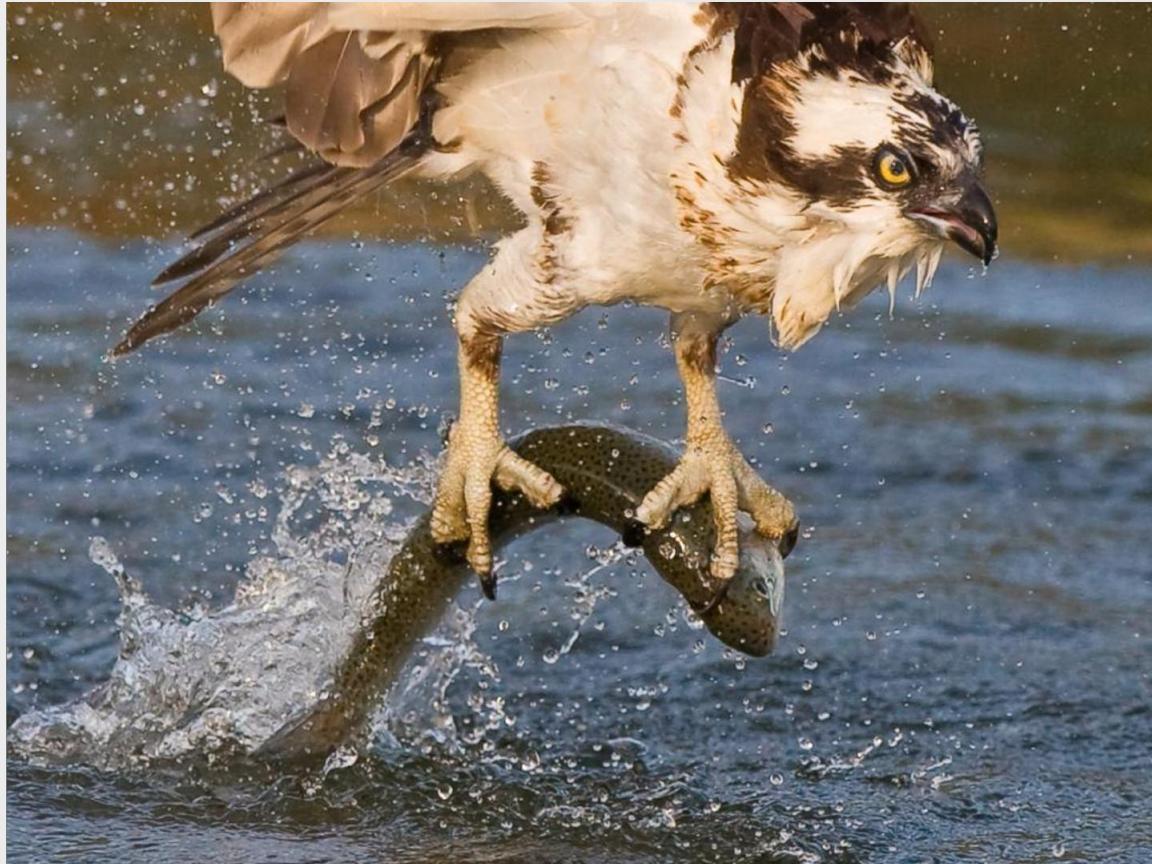


# Attention et Transformers

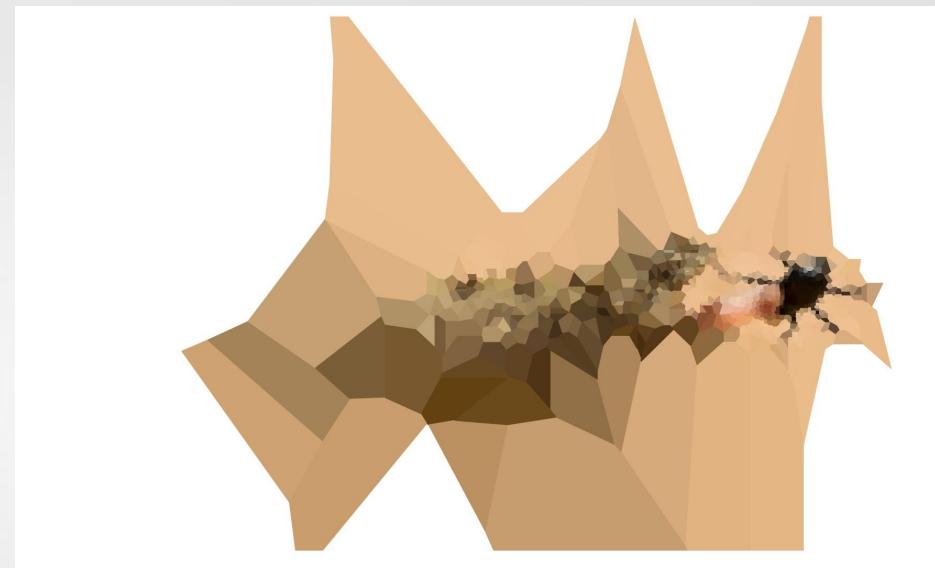
- C'est ce score d'attention qui permet d'avoir l'embedding de chaque mot contextualisé.
- On parle de **transformer block** parce qu'il y a plein d'autre choses que ce mécanisme d'attention.
- On parle de **multi-head attention** parce qu'on veut mettre en concurrence plusieurs mécanismes pour chercher les tokens qui sont liés les uns avec les autres.
- Idée du triptyque **Query, Key, Value (QKV)** pour faire cela de façon efficace.



# Attention en image ?



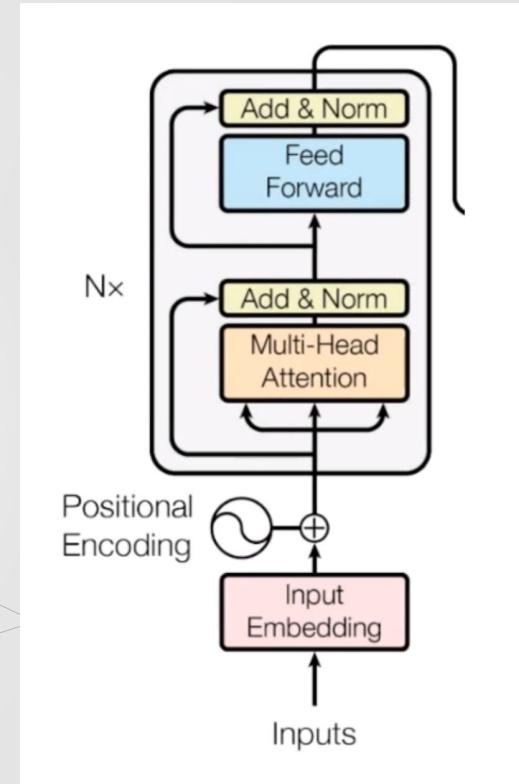
# Attention en image ?



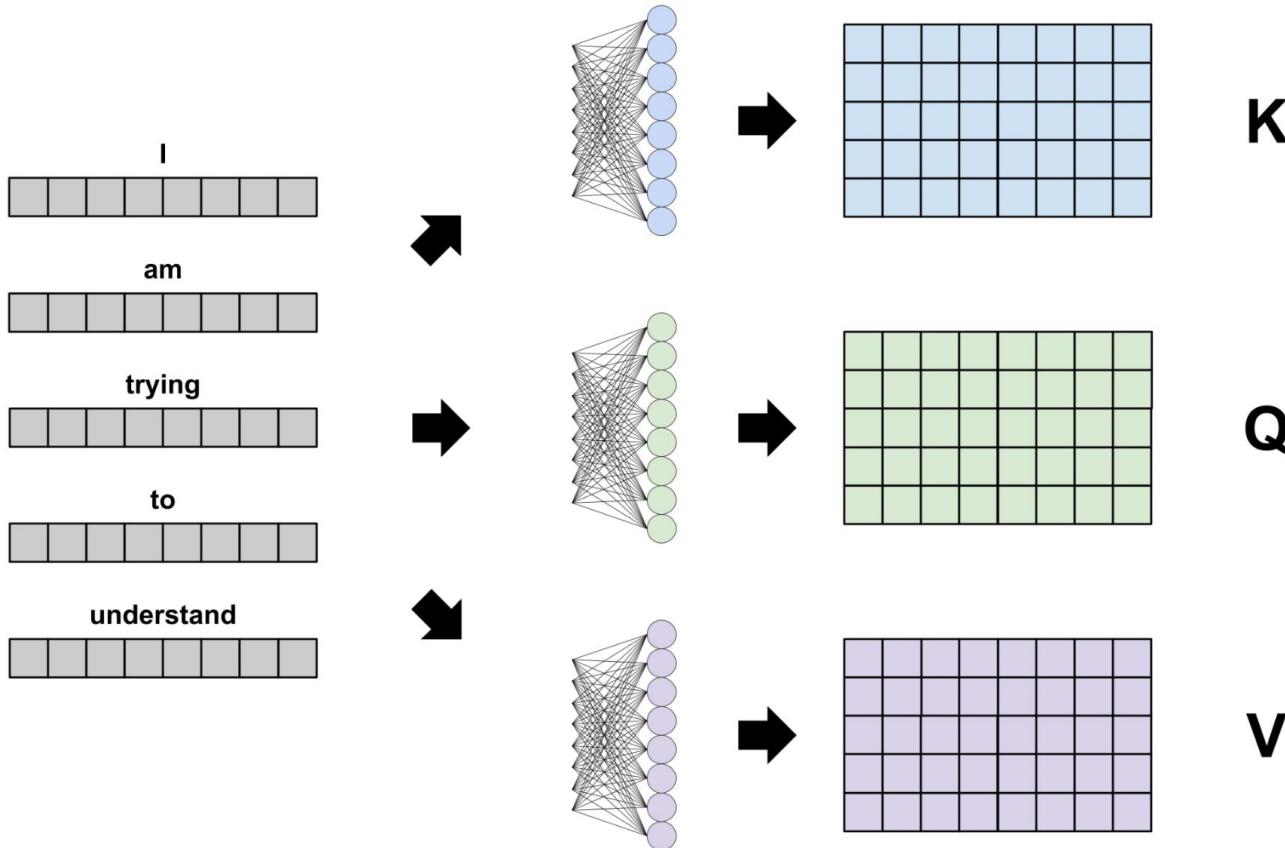
# Attention block

Plusieurs étapes :

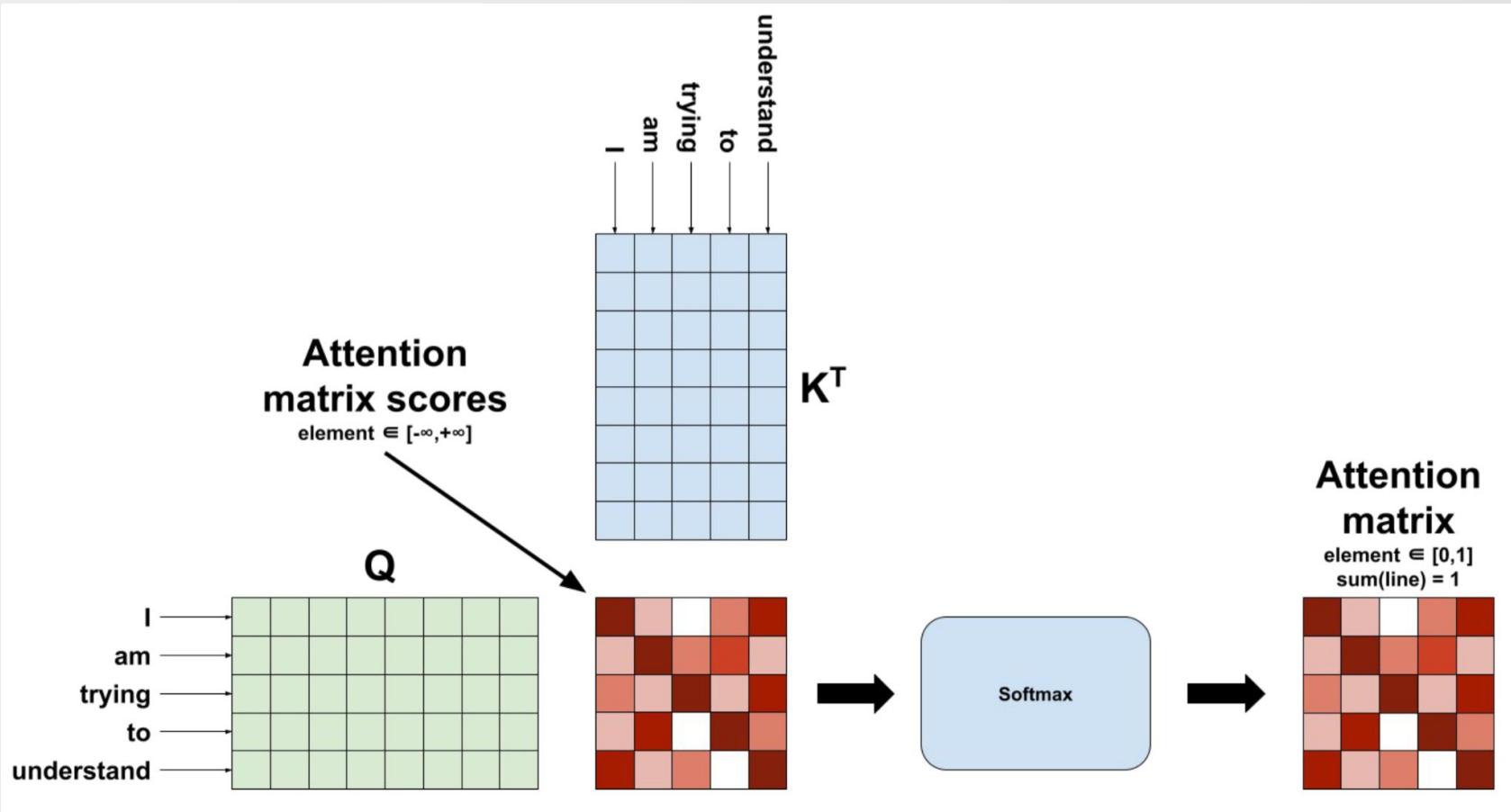
- Attention/multi-head attention : split de la séquence d'input en **Key**, **Query** et **Value** puis calcul de la matrice d'attention.
- Logique de “retrieval process” : on matche une requête (*query*) avec un ensemble de clés (*keys*), puis on détermine la valeur (*value*) qui en résulte.
- **Feed-forward layer** : fully-connected layer pour combiner l'information des différentes attention heads



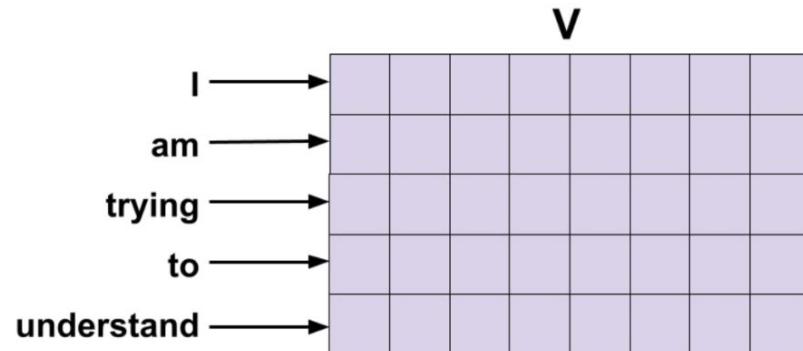
# L'attention en pratique



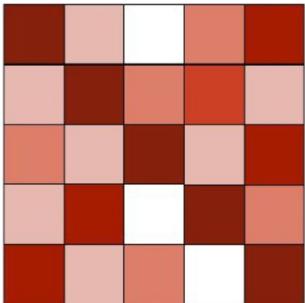
# L'attention en pratique



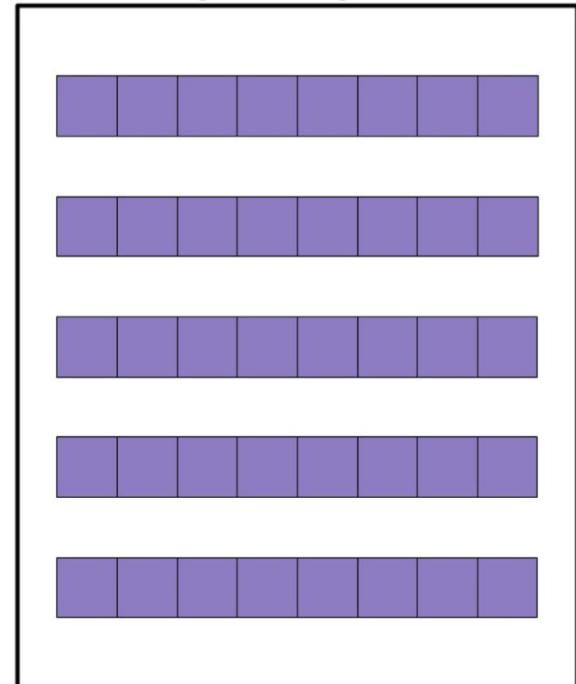
# L'attention en pratique



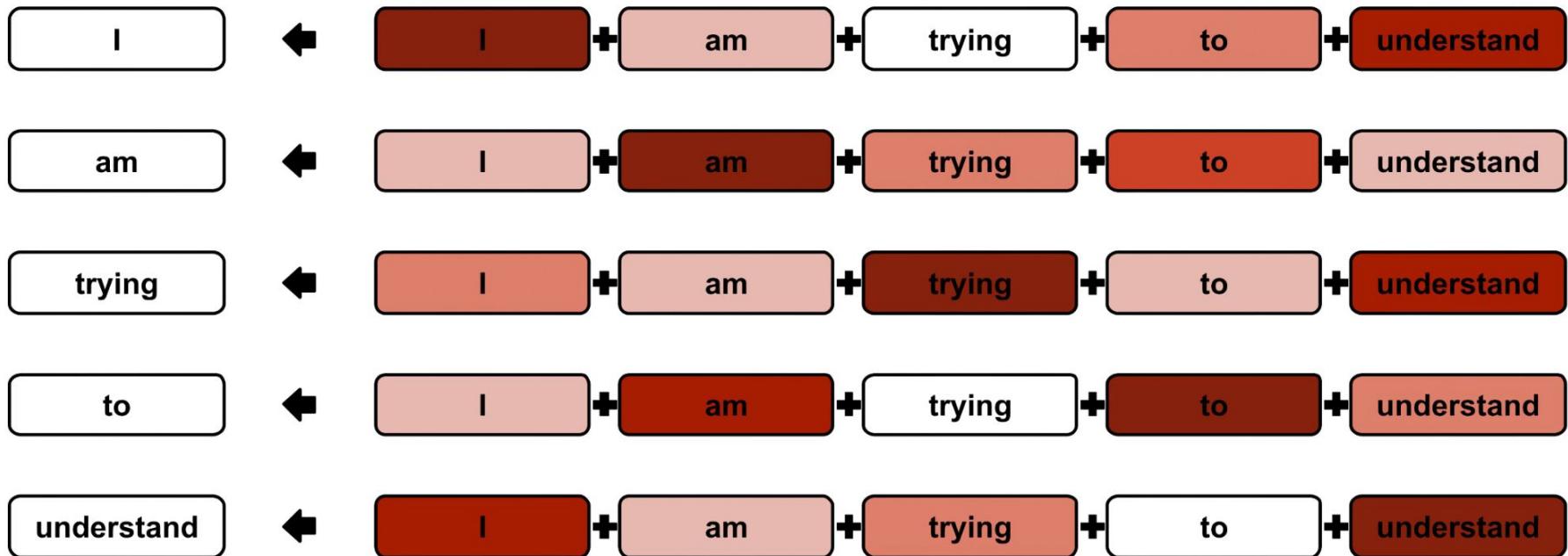
Attention matrix



Output sequence



# L'attention en pratique

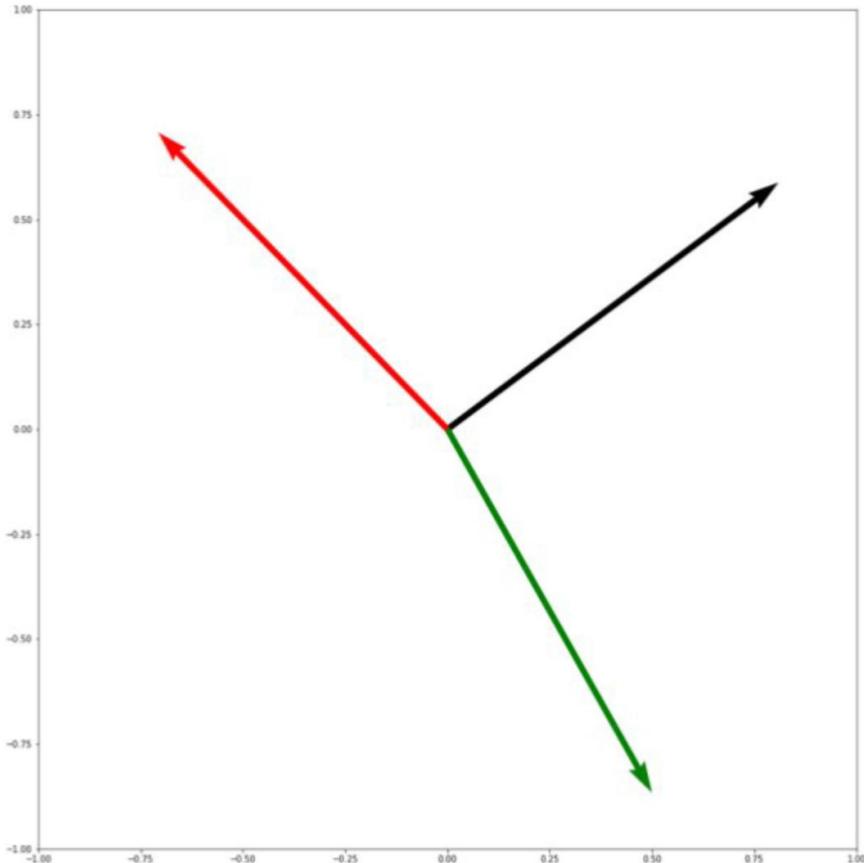


# Un exemple

The big dog



The : (0.50, -0.87)  
big : (-0.70, 0.70)  
dog : (0.81, 0.59)



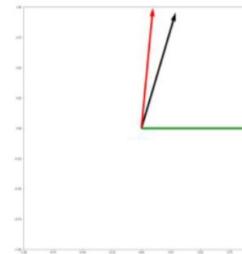
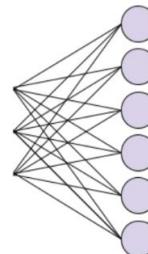
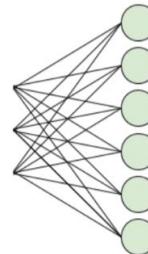
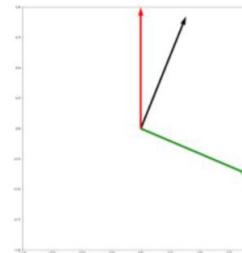
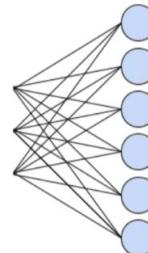
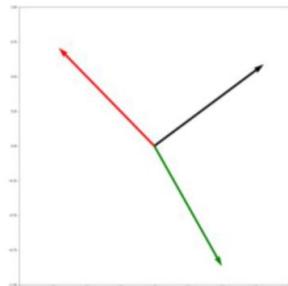
# Un exemple

$$\begin{bmatrix} 0.50 & -0.87 \\ -0.70 & 0.70 \end{bmatrix}$$

$$\begin{bmatrix} -0.70 & 0.70 \end{bmatrix}$$

$$\begin{bmatrix} 0.81 & 0.59 \end{bmatrix}$$

=



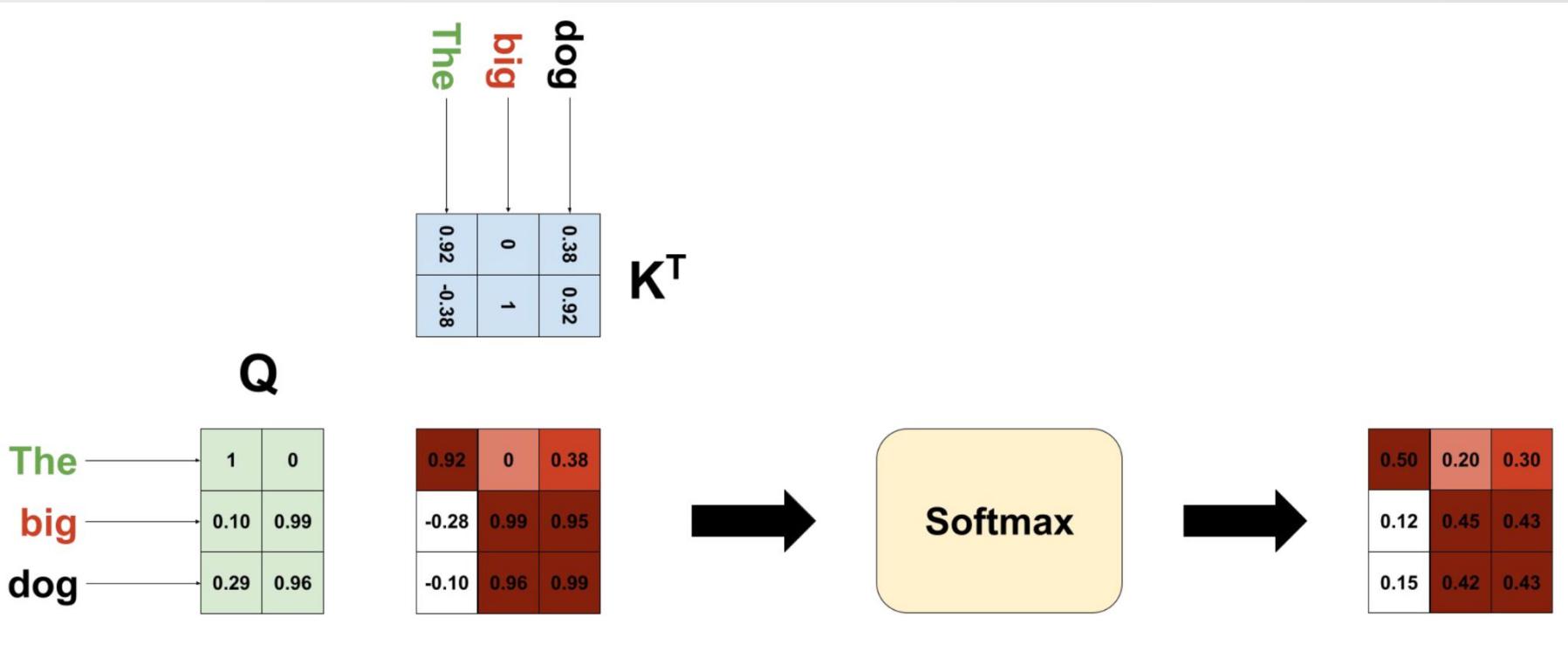
K

Q

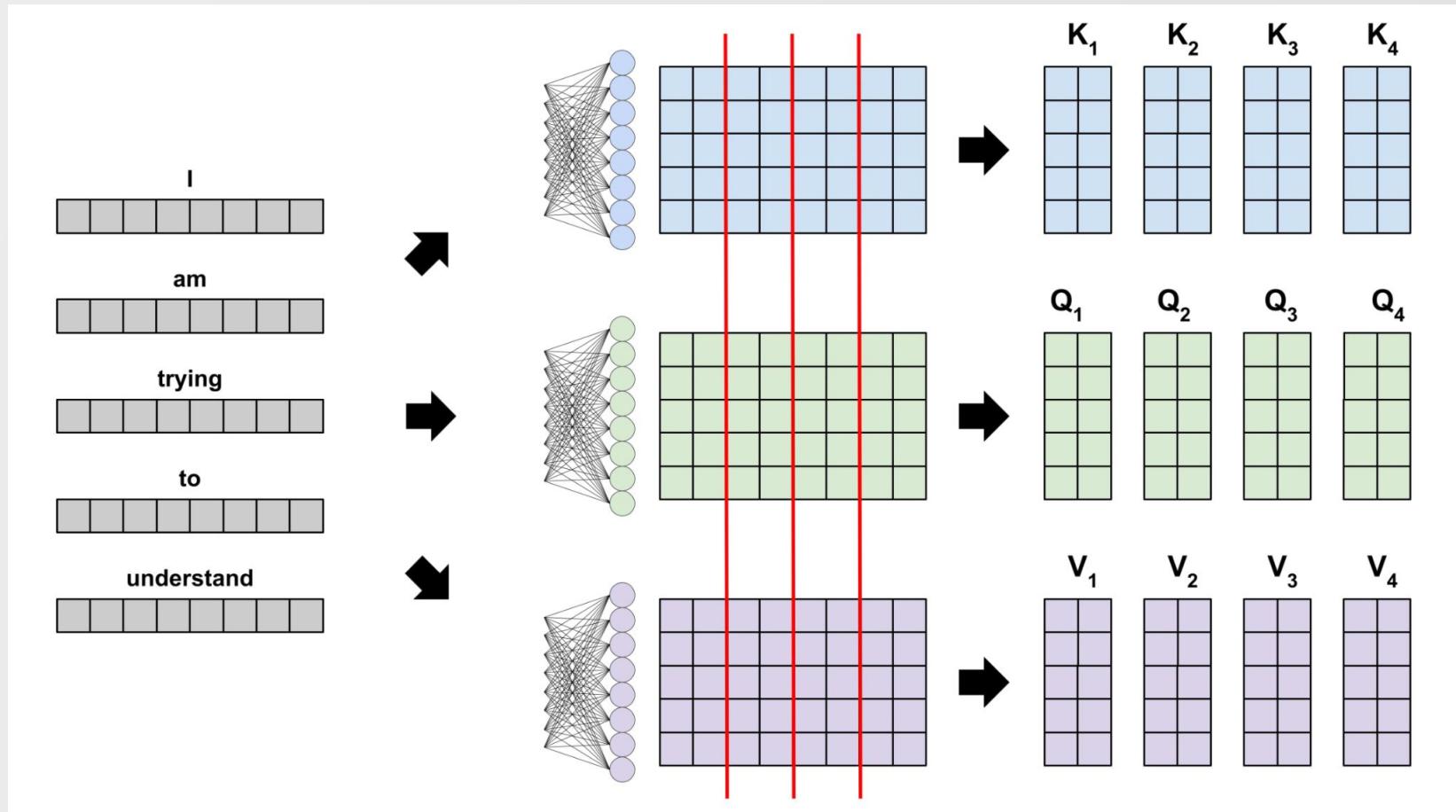
V



# Un exemple



# Multi-head attention



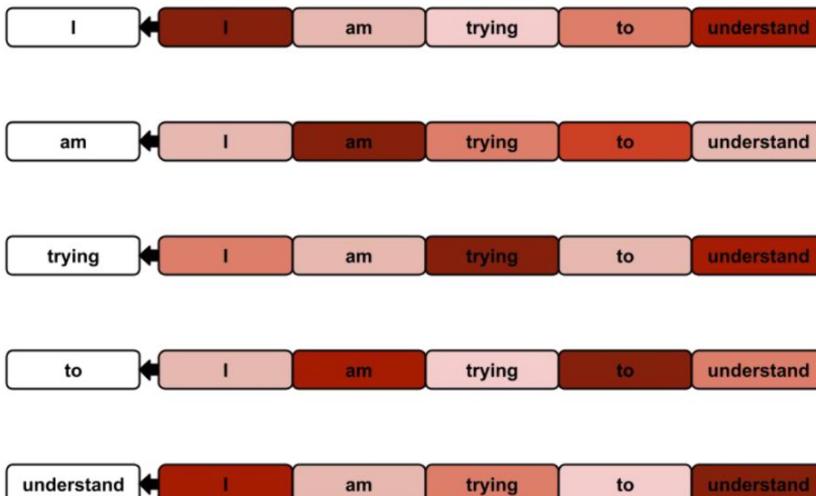
# Bilan

**Attention = moyenne pondérée des valeurs (Value) des tokens dans la séquence, où les poids (matrice d'attention) sont dépendent des relations entre les tokens de la phrase (query/key).**

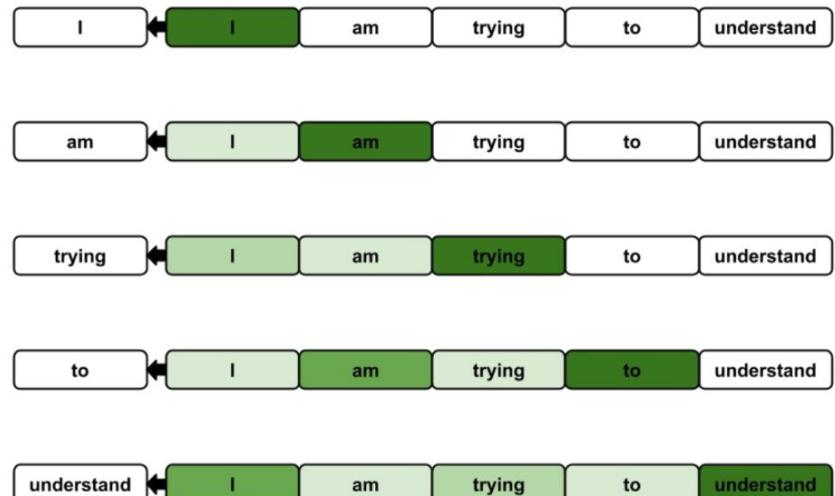


# Causal vs autre attention

## Bidirectional attention for Encoder



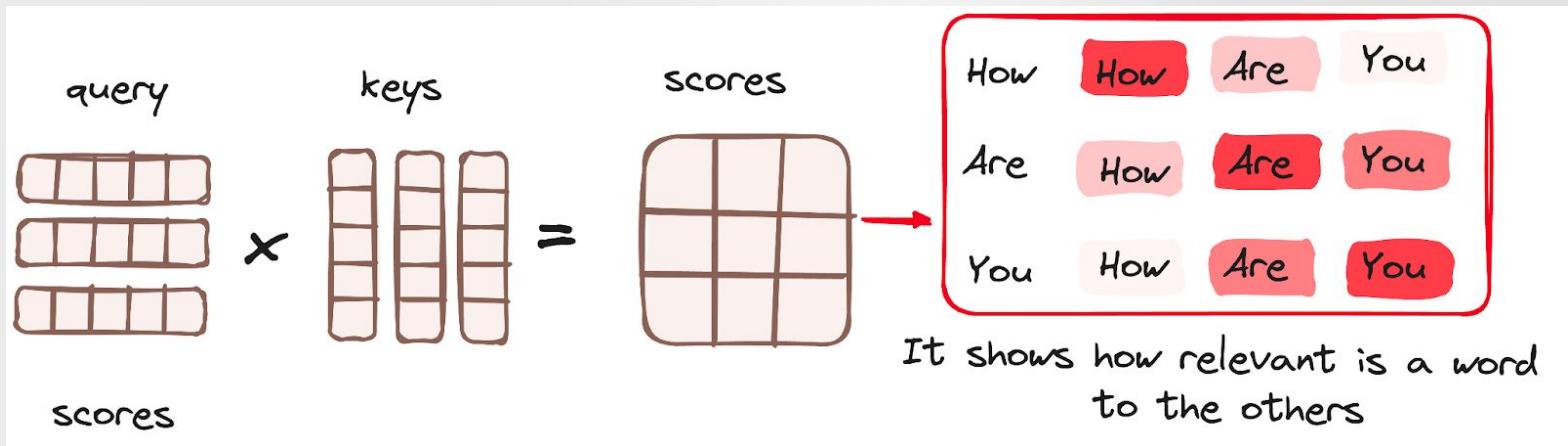
## Unidirectional attention for Decoder



VS

# Caveat avec l'Attention

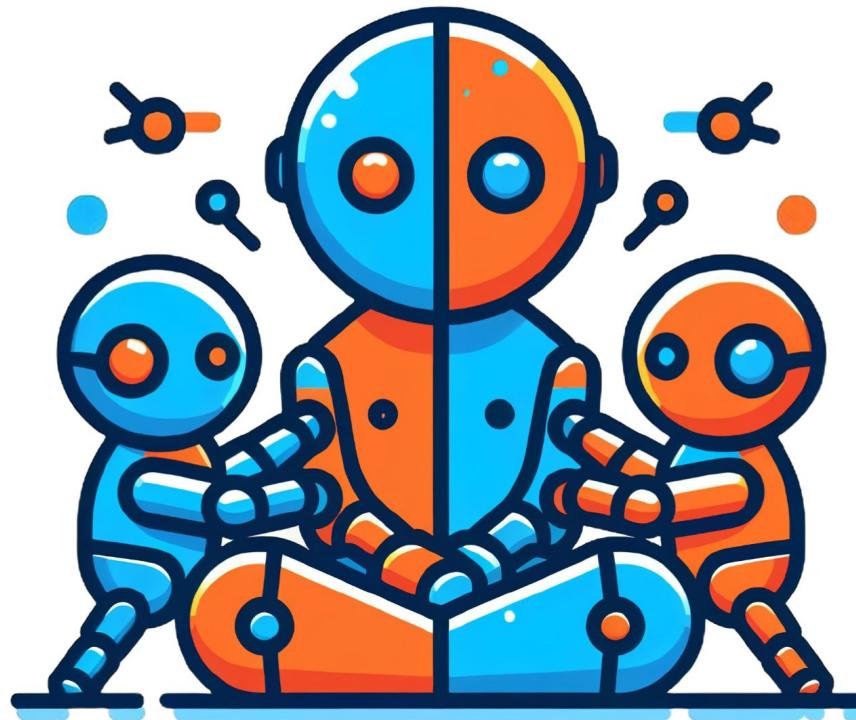
Le mécanisme de base de `self.attention` calcule un score d'attention pour **chaque couple de tokens dans la séquence d'entrée**, donc la complexité de ces calculs est **quadratique en la longueur de la donnée** en entrée (donc du contexte) mais c'est **parallélisable** !



# LLMs : les architectures

## Encoder-Decoder

Encoder

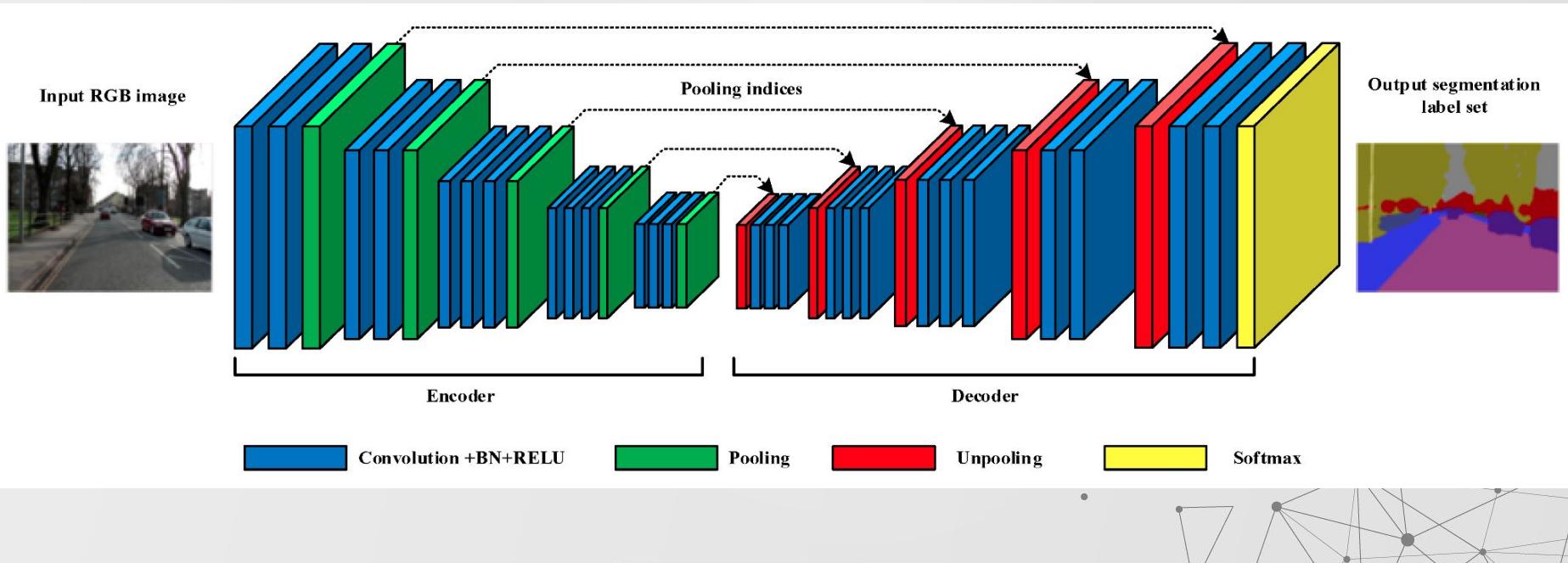


Decoder

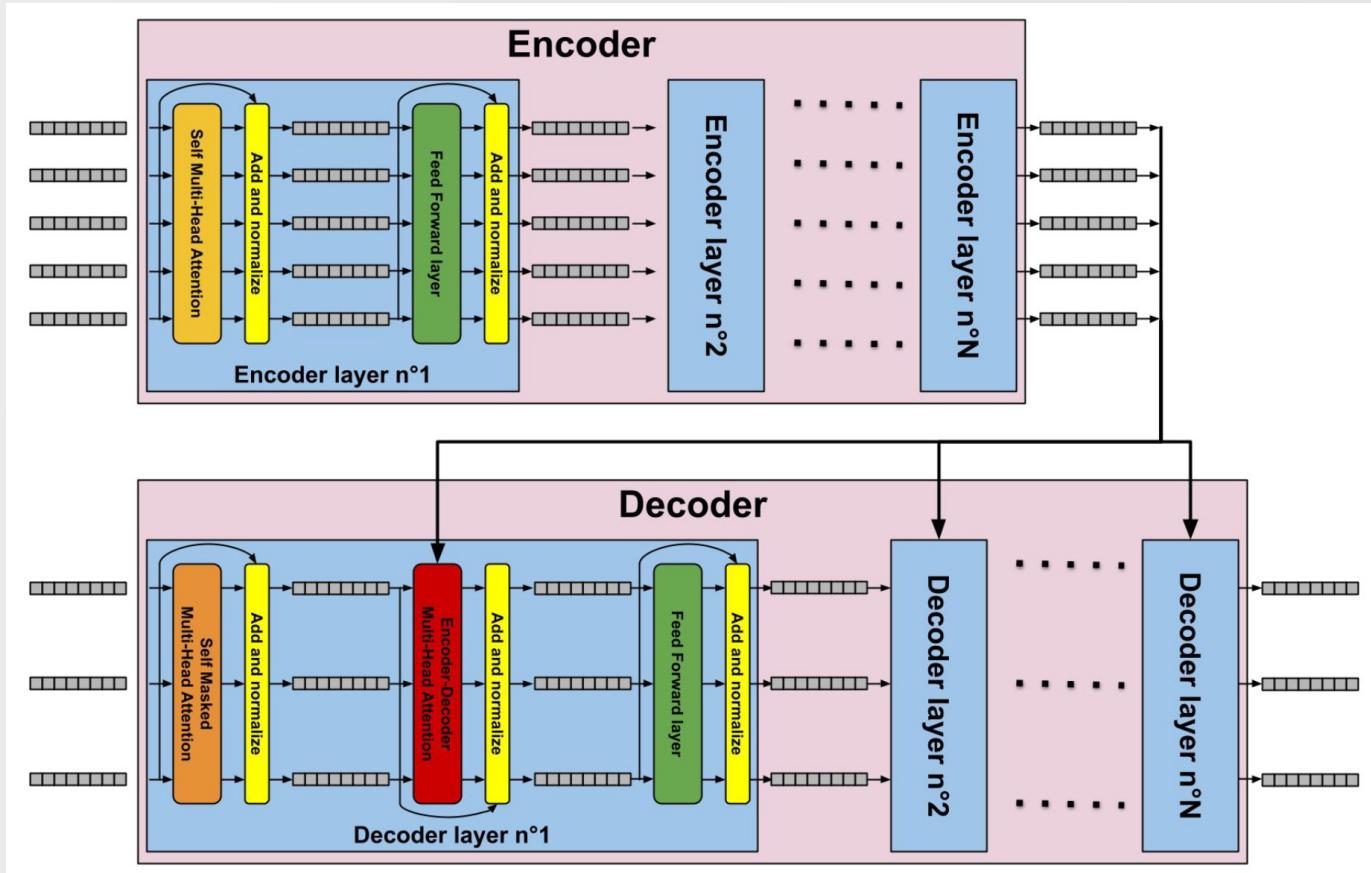


# LLMs : les architectures

Il existe encore une autre façon de catégoriser les modèles entre **encoder-decoder**, **encoder-only** et **decoder-only**.



# Encoder-Decoder (sequence-to-sequence)



# Encoder-Decoder

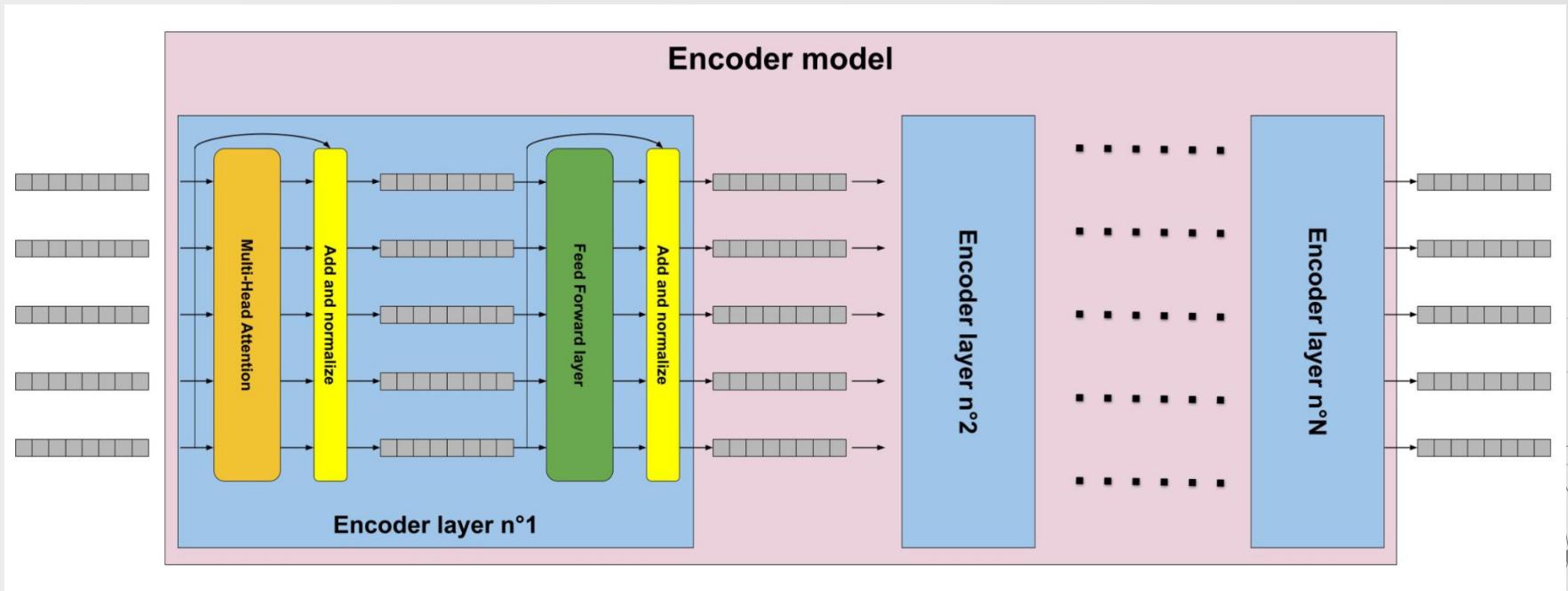
- Les **encoder-decoder** sont utilisés pour des tâches sequence-to-sequence, text summarization, translation, où il y a une relation très forte entre input/output.
- Pour gérer des inputs/outputs de différentes longueurs.
- Principe : l'input passe à travers l'encodeur qui génère un **embedding context-aware** de toute la séquence d'entrée, puis le décodeur génère l'output.
- **T5 (Text-to-Text Transfer Transformer)** de Google
- BART (Bidirectional and Auto-Regressive Transformers)



# Encoder-Decoder

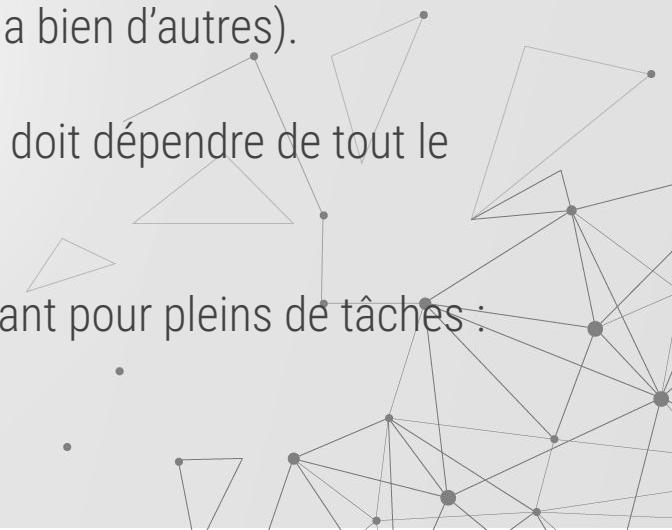
```
from transformers import T5Tokenizer, T5ForConditionalGeneration  
  
# Load the pre-trained T5 model and tokenizer  
model_name = "t5-small"  
tokenizer = T5Tokenizer.from_pretrained(model_name)  
model = T5ForConditionalGeneration.from_pretrained(model_name)
```

# Encoder-only



# Encoder-only

- Le rôle d'un modèle **encoder-only** est d'extraire des **embeddings** (token ou sentence).
- **BERT : Bidirectional Encoder Representations from Transformers** est le plus connu.
- **RoBERTa** : une variation de BERT plus robuste (et il y en a bien d'autres).
- Ils sont **bidirectionnels** parce que l'embedding du token doit dépendre de tout le contexte.
- L'**embedding contextualisé** de chaque token est important pour pleins de tâches : text classification, **NER**, **POS tagging** etc..



# Encoder-only

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification, pipeline

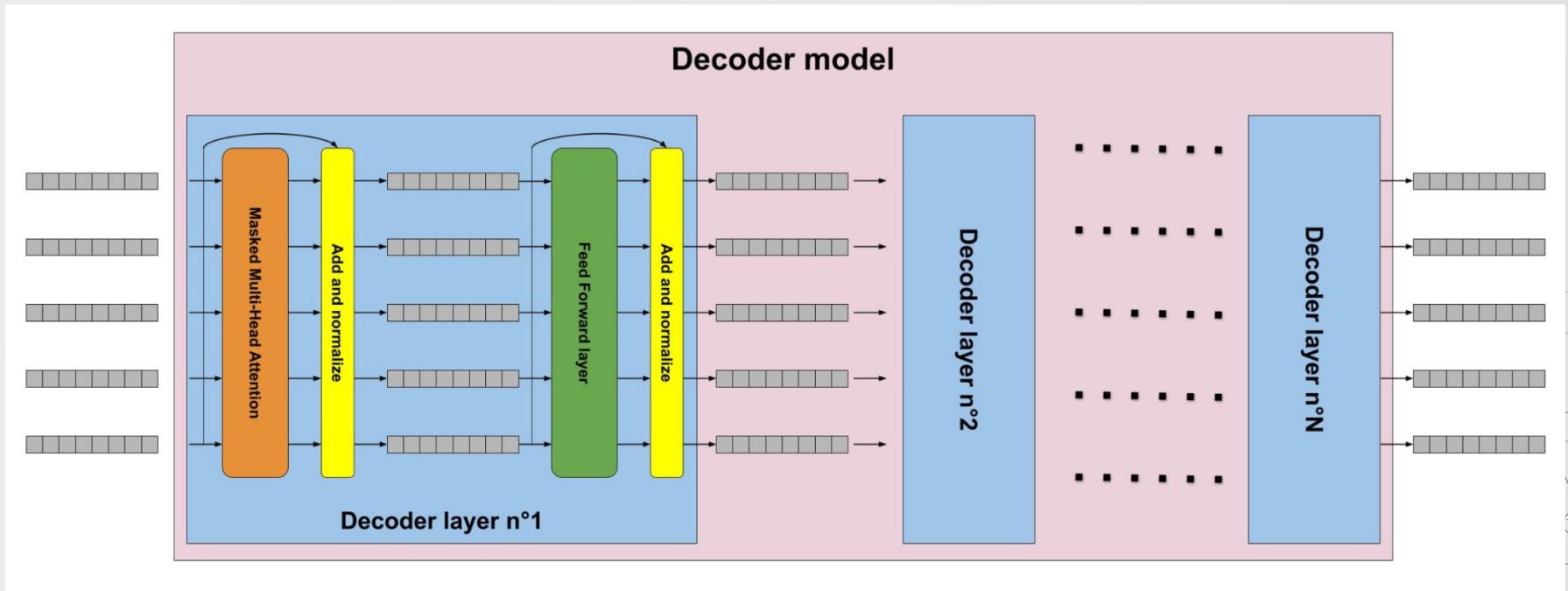
# Load the tokenizer and model
model_name = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name)

# Create a pipeline for text classification
classifier = pipeline("sentiment-analysis", model=model, tokenizer=tokenizer)

# Example text
text = "I love using transformer models!"

# Classify the text
result = classifier(text)
print(result)
```

# Decoder-only

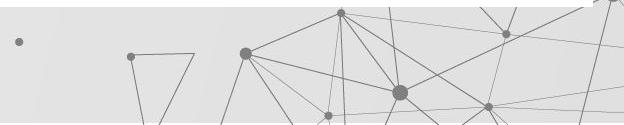
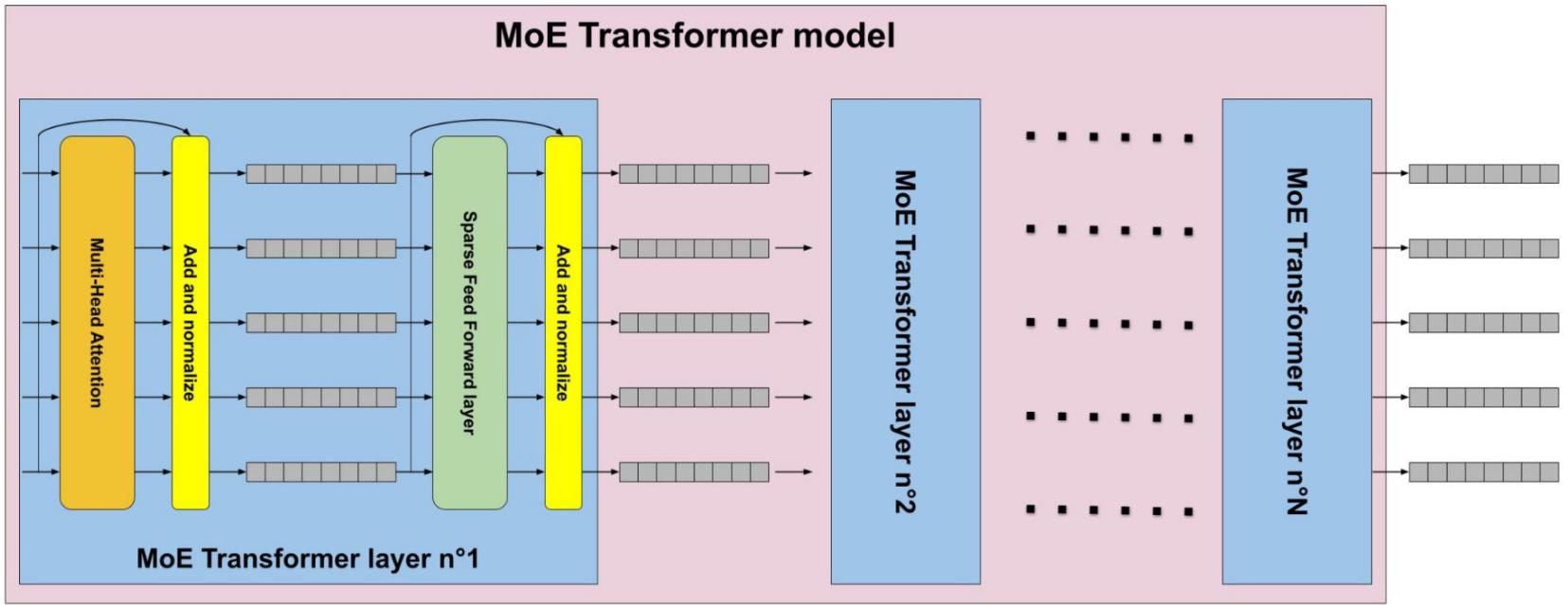


# Decoder-only

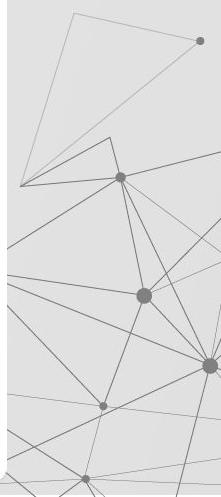
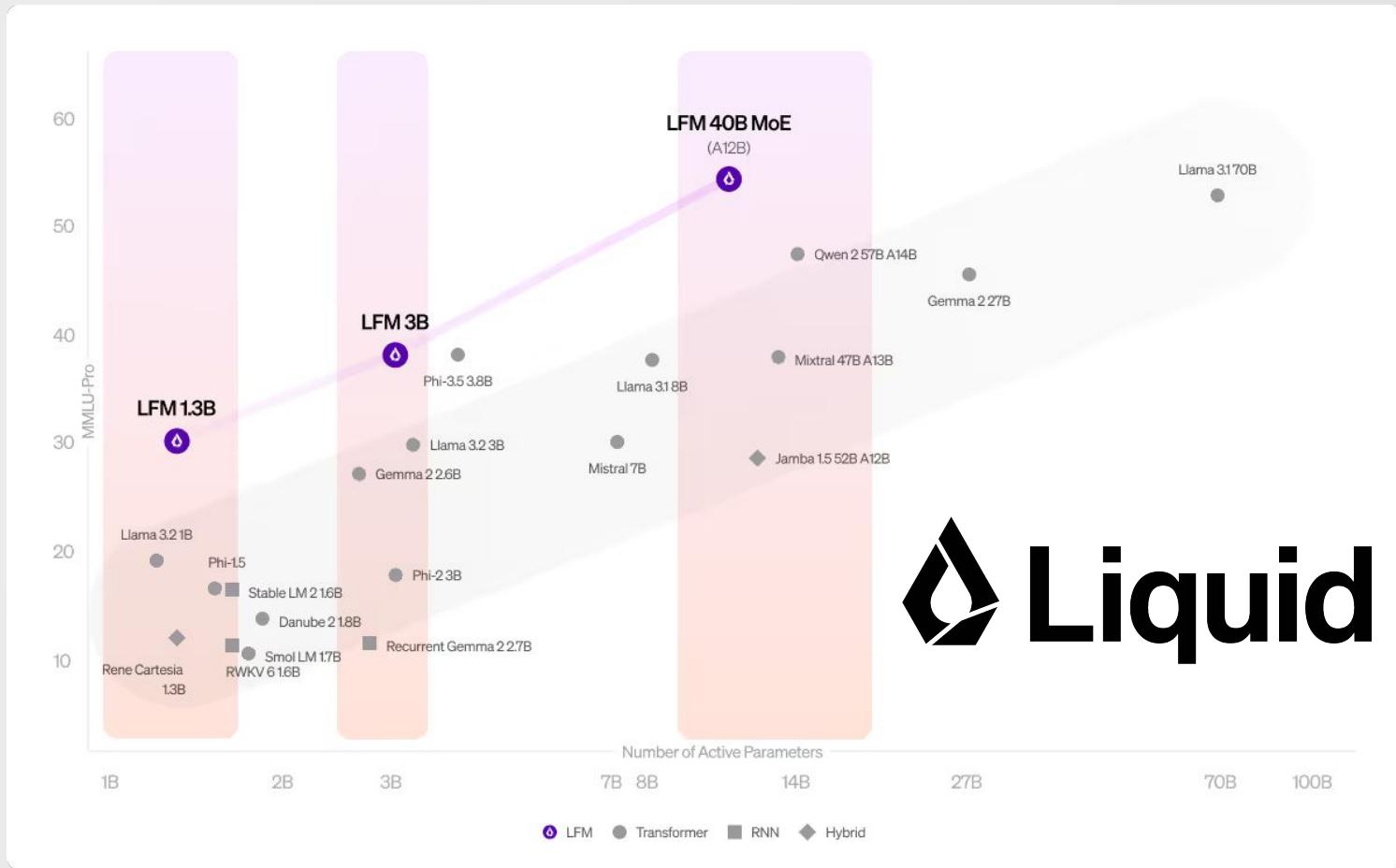
- Ce sont tous les modèles que nous avons vu pour l'instant (les GPTs)
- Modèles autorégressifs.
- Commence avec un prompt et prédit le token suivant.
- Particulièrement utile pour text-generation, summarization, language modeling, etc.
- **En théorie donc les LLMs sont des decoder-only !**
- **Un decoder-only fine-tuned pour Q&A reste decoder-only même s'il résout des tâches similaires à un encoder-decoder.**



# Mixture-of-experts



# Au-delà des transformers



# Pour aller plus loin

Quelques articles de recherche structurants :

- <https://arxiv.org/abs/1706.03762>, **Attention is all you need.**
- <https://arxiv.org/abs/2010.11929>, **An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.**

Quelques blogs pour comprendre plus en détail

- <https://jalammar.github.io/illustrated-gpt2/> excellent illustration de tous les détails de GPT-2.
- <https://towardsdatascience.com/how-gpt-works-a-metaphoric-explanation-of-key-value-query-in-attention-using-a-tale-of-potion-8c66ace1f470>
- <https://www.datacamp.com/tutorial/how-transformers-work>

