

# **FONDEMENTS DES GRANDS MODÈLES DE LANGUE ET DES AGENTS CONVERSATIONNELS**

Adrien Guille, Maître de conférences  
Université Lumière Lyon 2

# GRAND MODÈLE DE LANGUE : EN RÉSUMÉ

- Réseau de neurones profond qui reçoit un texte
- Modélise la loi conditionnelle du prochain mot de ce texte
- Opère sur des représentations vectorielles de (morceaux de) mots et des représentations vectorielles de position
- Basé sur le bloc Transformer
  - Mécanisme d'attention multi-têtes
  - Réseau à propagation avant

# AGENT CONVERSATIONNEL : EN RÉSUMÉ

- Un grand modèle de langue pré-entraîné sur un vaste corpus tiré du Web
- Ajusté sur un petit corpus pour maintenir la discussion
- Ajusté de nouveau pour respecter certaines préférences
- Compléments possibles :
  - Filtrage des prompts
  - Augmentation du contexte (RAG)

# **TYPES D'USAGES D'UN AGENT CONVERSATIONNEL**

# AUGMENTER LES CAPACITÉS HUMAINES

- **But : offrir un support à la prise de décision**
  - L'agent peut aider le décisionnaire en lui apportant une information supplémentaire, par exemple en lui proposant une synthèse à partir de documents liés à la prise de décision
  - L'agent peut proposer des scénarios alternatifs
- **Bénéfices potentiels :**
  - Gain de temps / gain informationnel
  - Créativité accrue

# AUTOMATISER DES TÂCHES

- **But : remplacer l'humain**
  - L'agent peut générer une argumentation
  - L'agent peut analyser un tableau de données
  - L'agent peut écrire un programme informatique / créer un site web
  - L'agent peut enseigner une notion
  - L'agent peut évaluer une demande

# **UN GRAND MODÈLE DE LANGUE : GPT-3**

# MODÈLE DE LANGUE

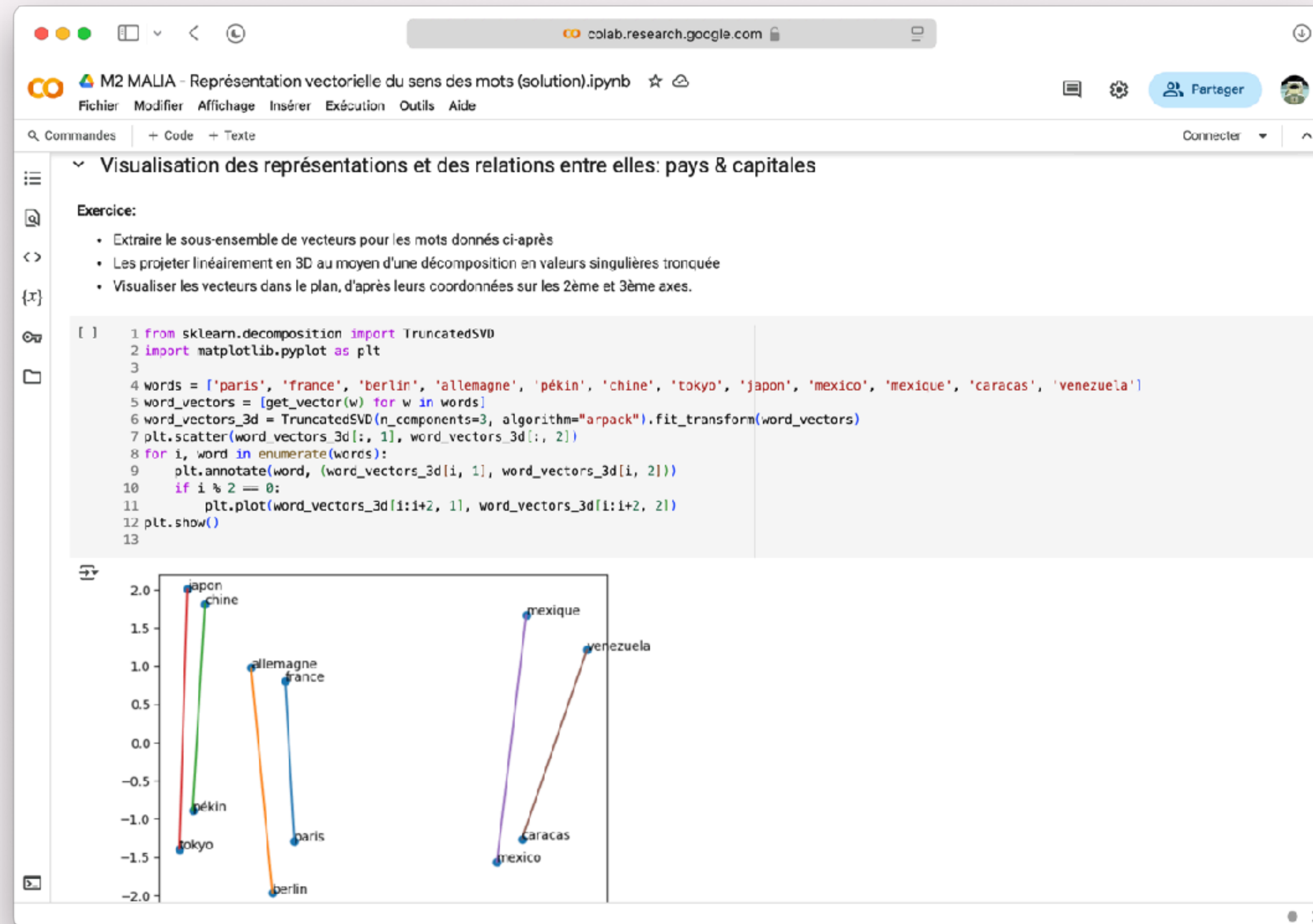
- On modélise la loi conditionnelle du prochain token sachant les tokens précédents
  - $P(T_{n+1} = t_{n+1} \mid t_1, t_2, \dots, t_n)$
- On voit le texte comme un processus stochastique auto-régressif
  - Pour générer du texte, on échantillonne les mots un à un
    - Échantillonnage top k
    - Recherche en faisceau (plus coûteux et déterministe)
      - Plus pertinent dans certains cas, e.g. en traduction



# REPRÉSENTATION VECTORIELLE DES TOKENS / POSITIONS

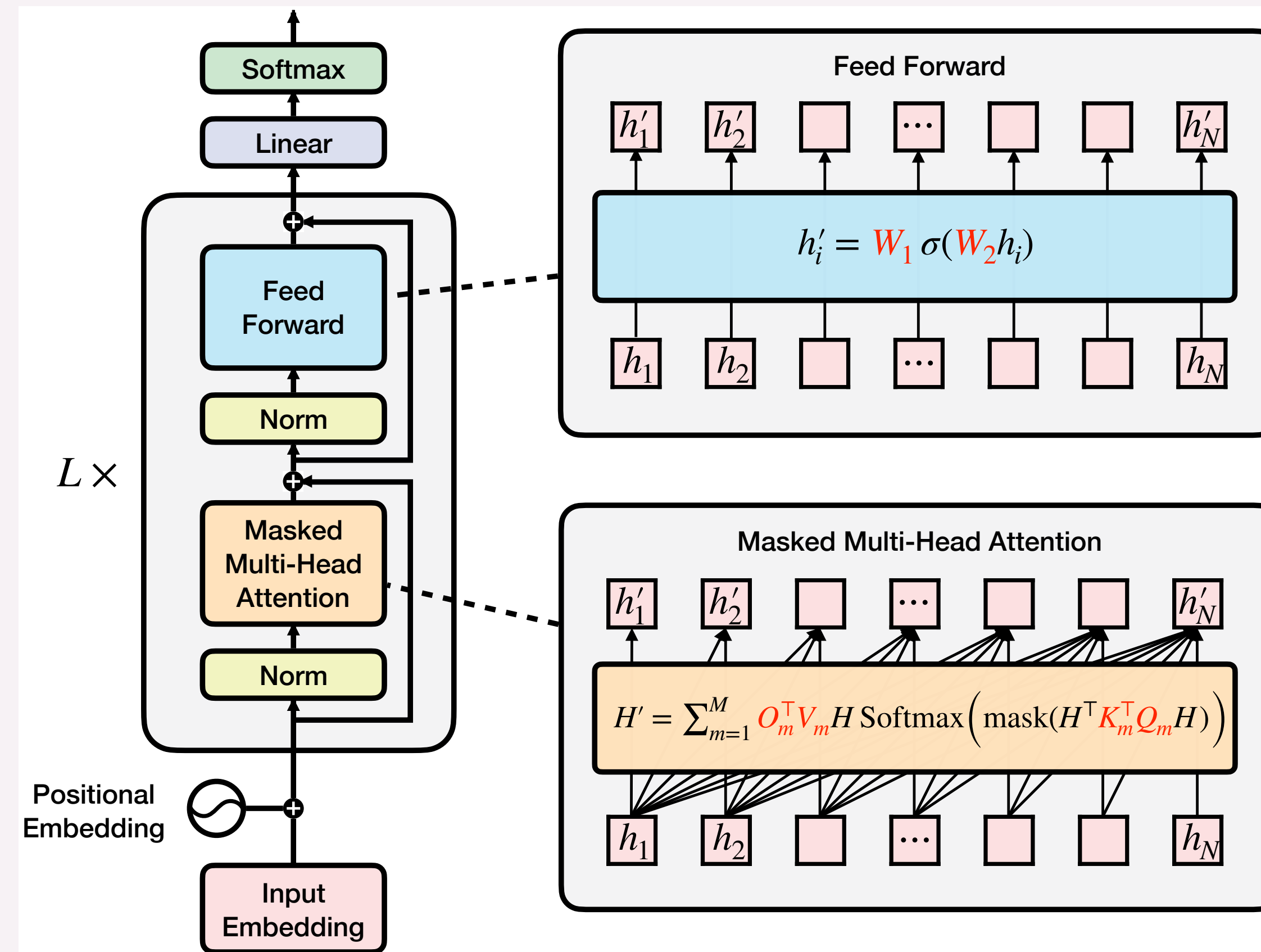
- Les réseaux de neurones opèrent sur des vecteurs
  - Alors on apprend un espace vectoriel dans lequel représenter les tokens
  - De sorte notamment que les distances mesurées dans cet espace reflètent la proximité sémantique entre les tokens
    - Mais capturent aussi d'autres régularités de la langue, au niveau syntaxique, sémantique, conceptuel, *etc.*
  - On apprend également des représentations vectorielles des positions de sorte que le modèle puisse estimer :
    - La position absolue des tokens dans la séquences
    - Les positions relatives des tokens les uns par rapport aux autres

# REPRÉSENTATION VECTORIELLE DES TOKENS / POSITIONS



# CONTEXTUALISATION DES REPRÉSENTATIONS

- Architecture du décodeur du Transformer



Source : An Overview of Large Language Models for Statisticians, Wi et al. 2025

# CONTEXTUALISATION DES REPRÉSENTATIONS

- Tête d'auto-attention (masquée)

- Entrée :  $H \in \mathbb{R}^{d \times n}$ , représentations des tokens/positions
- Paramètres :  $Q, K, V \in \mathbb{R}^{d' \times d}$
- Projections en requêtes  $QH$ , clés  $KH$  et valeurs  $VH$
- Calcul de l'attention  $A = \text{softmax} \left( (QH)^\top KH \right)$ 
  - Le triangle supérieur droit de cette matrice est « masquée » dans le cas du décodeur
- Calcul des représentations contextualisées  $H' = (VH)A$

# ESTIMATION DES PARAMÈTRES

- 175 milliards de paramètres à estimer pour GPT-3
  - Représentations des tokens en entrée (environ 30 000 tokens)
  - Représentations des positions en entrée
  - Pour chaque bloc encodeur
    - Les matrices  $Q_m, K_m, V_m, O_m$  pour chaque tête d'auto-attention
    - Les matrices  $W_1, W_2$  du réseau à propagation avant
  - Le classifieur linéaire en sortie



# ESTIMATION DES PARAMÈTRES

- Apprentissage auto-supervisé

- Jeu de données (300 milliards de tokens)

- Textes tirés du Web, filtrés et dédoublés + Wikipedia (en)

- Objectif

- Prédiction des mots un à un en démasquant les textes au fur et à mesure

- Maximisation de la log-probabilité des mots employés par les auteurs des textes

$$\pi^* = \operatorname{argmax}_{\pi} \sum_{k=1}^{N_{pages}} \sum_{i=1}^{N_{tokens_k}} \log P_{\pi}(T_i = t_i | t_0, \dots, t_{i-1}), \text{ avec } t_0 \text{ un token spécial « <start> »}$$

# UTILISATION DU MODÈLE DE LANGUE PRÉ-ENTRAÎNÉ

- **Complétion du texte reçu en entrée**
  - Zero-shot : on exprime un besoin par une instruction
  - One-shot : on donne un exemple d'instruction et un exemple de réponse, puis l'instruction qui exprime le besoin
  - Few-shot : même principe avec plusieurs exemples
- **Très mauvaise performance en zero-shot**
  - Performance bien supérieure en few-shot

**UN AGENT CONVERSATIONNEL : CHATGPT**



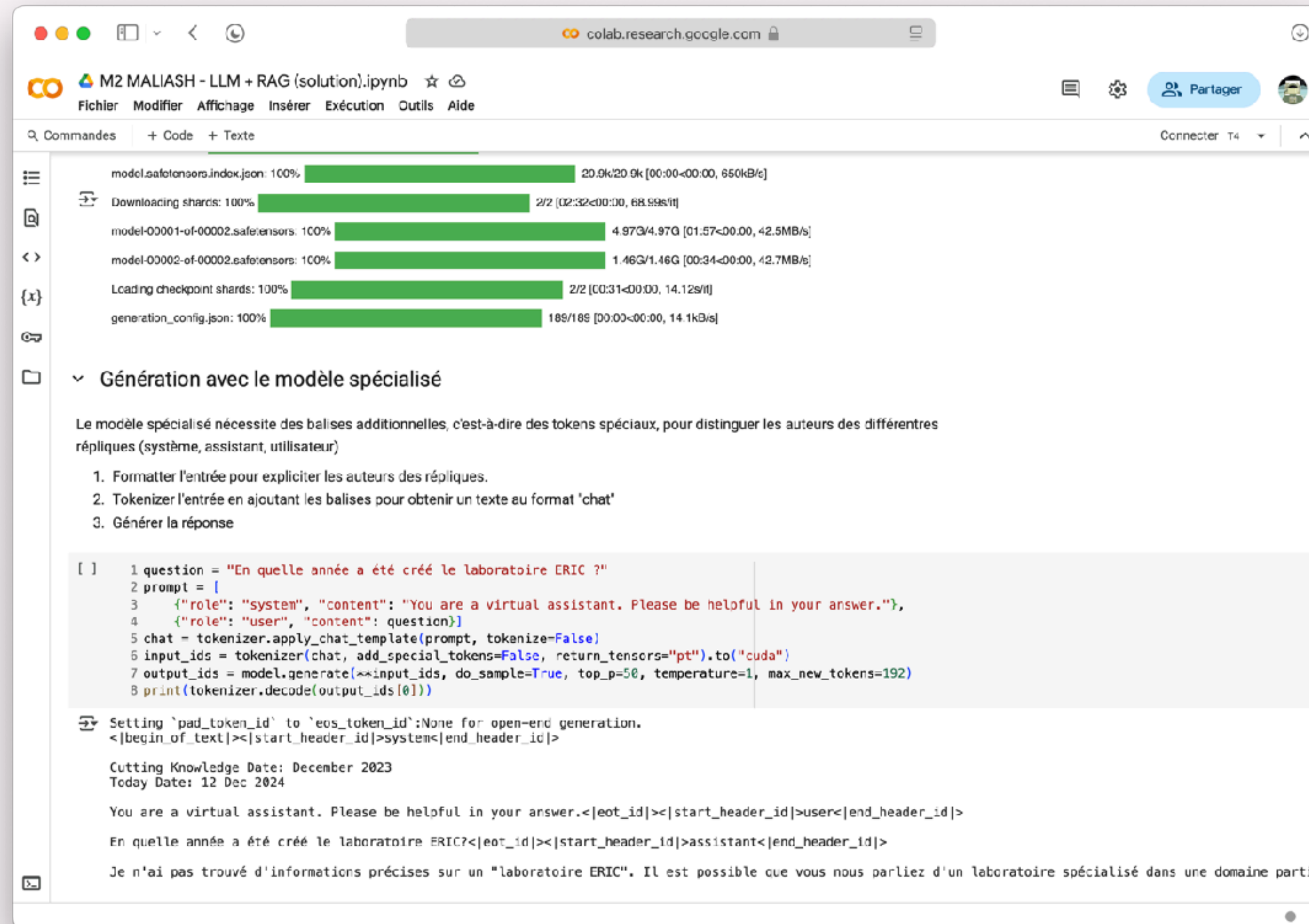
# APPRENDRE À RÉPONDRE AUX INSTRUCTIONS

- Démarche en plusieurs étapes
  - Étape 1 : Pré-entraînement auto-supervisé du modèle de langue
  - Étape 2 : Ajustement supervisé du modèle de langue sur un corpus instructions/réponses
  - Étape 3 : Apprentissage d'un modèle de récompense
  - Étape 4 : Réajustement du modèle de langue par apprentissage par renforcement

# AJUSTEMENT SUPERVISÉ DES PARAMÈTRES

- **Apprentissage supervisé classique**
  - Petit jeu de données rédigé en interne
    - Paires prompt + exemple de réponse
- Objectif
  - Prédiction des mots de la réponse un à un
  - Maximiser la log-probabilité des mots employés par les auteurs

# AJUSTEMENT SUPERVISÉ DES PARAMÈTRES



```
model.safetensors.index.json: 100% 20.9k/20.9k [00:00<00:00, 650kB/s]
Downloading shards: 100% 2/2 [02:32<00:00, 68.99s/it]
model-00001-of-00002.safetensors: 100% 4.97G/4.97G [01:57<00:00, 42.5MB/s]
model-00002-of-00002.safetensors: 100% 1.46G/1.46G [00:34<00:00, 42.7MB/s]
Loading checkpoint shards: 100% 2/2 [00:31<00:00, 14.12s/it]
generation_config.json: 100% 189/189 [00:00<00:00, 14.1kB/s]
```

### ▼ Génération avec le modèle spécialisé

Le modèle spécialisé nécessite des balises additionnelles, c'est-à-dire des tokens spéciaux, pour distinguer les auteurs des différentes répliques (système, assistant, utilisateur)

1. Formater l'entrée pour expliciter les auteurs des répliques.
2. Tokenizer l'entrée en ajoutant les balises pour obtenir un texte au format 'chat'
3. Générer la réponse

```
[ ] 1 question = "En quelle année a été créé le laboratoire ERIC ?"
    2 prompt = [
    3     {"role": "system", "content": "You are a virtual assistant. Please be helpful in your answer."},
    4     {"role": "user", "content": question}]
    5 chat = tokenizer.apply_chat_template(prompt, tokenize=False)
    6 input_ids = tokenizer(chat, add_special_tokens=False, return_tensors="pt").to("cuda")
    7 output_ids = model.generate(input_ids, do_sample=True, top_p=50, temperature=1, max_new_tokens=192)
    8 print(tokenizer.decode(output_ids[0]))
```

Setting 'pad\_token\_id' to 'eos\_token\_id':None for open-end generation.  
<|begin\_of\_text|><|start\_header\_id|>system<|end\_header\_id|>

Cutting Knowledge Date: December 2023  
Today Date: 12 Dec 2024

You are a virtual assistant. Please be helpful in your answer.<|eot\_id|><|start\_header\_id|>user<|end\_header\_id|>

En quelle année a été créé le laboratoire ERIC?<|eot\_id|><|start\_header\_id|>assistant<|end\_header\_id|>

Je n'ai pas trouvé d'informations précises sur un "laboratoire ERIC". Il est possible que vous nous parliez d'un laboratoire spécialisé dans une domaine parti

# RÉAJUSTEMENT SELON LES PRÉFÉRENCES UTILISATEURS

- **Limitation de l'apprentissage supervisé**
  - Mesure d'erreur au niveau du token
  - Pénalise le modèle pour toute réponse différente de l'exemple
- **Passage à l'apprentissage par renforcement**
  - Mesure d'une récompense au niveau de la réponse
  - Permet au modèle d'explorer les réponses possibles

# RÉAJUSTEMENT SELON LES PRÉFÉRENCES UTILISATEURS

- Apprentissage d'un modèle de récompense
  - Créer un corpus de prompts
  - Générer  $K$  réponses à chaque prompt
  - Annoter les paires réponses
    - Comparer les  $\binom{K}{2}$  paires de réponses à un prompt ; désigner la meilleure réponse de chaque paire



# RÉAJUSTEMENT SELON LES PRÉFÉRENCES UTILISATEURS

- Apprendre un modèle de récompense

- Modèle Bradley-Terry

- Prompt  $x$ , réponse préférée  $y_+$  et autre réponse  $y_-$

- $$P(y_+ \succ y_-) = \frac{e^{r(x,y_+)}}{e^{r(x,y_+)} + e^{r(x,y_-)}} = \sigma \left( r(x, y_+) - r(x, y_-) \right)$$

- Ajuster un fork du modèle de langue par maximisation de la vraisemblance, en remplaçant la couche de classification par une couche de régression

# RÉAJUSTEMENT SELON LES PRÉFÉRENCES UTILISATEURS

- Apprentissage par renforcement pour réajuster le modèle
  - Réajuster les paramètres par maximisation de l'espérance de la récompense
  - Inclure une pénalisation pour s'aligner sur les préférences sans trop s'éloigner du modèle de départ ( $\pi_{ref}$ )

- $$\pi^* = \operatorname{argmax}_{\pi} \mathbb{E}_{x \sim D, y \sim \pi(y|x)} \left[ r(x, y) - \beta \frac{\pi(y|x)}{\pi_{ref}(y|x)} \right]$$

# **PERSONNALISER UN AGENT CONVERSATIONNEL :**

## **RAG**



# COMMENT PERSONNALISER UN ASSISTANT ?

- On pourrait ajuster le modèle, e.g. sur une base de connaissance privée
  - Nécessite des ressources matérielles importantes pour mettre en œuvre la descente de gradient stochastique
- On pourrait recourir à une procédure efficace, comme l'adaptation de rang faible (a.k.a LoRA)
  - Pour toute matrice  $W \in \mathbb{R}^{d' \times d}$  paramétrant le modèle et  $\Delta W \in \mathbb{R}^{d' \times d}$  l'ajustement à  $y$  apporter, avec la matrice résultant de l'ajustement  $W' = W + \Delta W$
  - On approche  $\Delta W$  par un produit de matrices de rang faible,  $\Delta W \approx A \times B$  avec  $A \in \mathbb{R}^{d' \times r}$ ,  $B \in \mathbb{R}^{r \times d}$  et  $r \ll d$

# COMMENT PERSONNALISER UN ASSISTANT ?

- **Approche qui reste coûteuse**
  - Si on a plusieurs cas d'usage on doit calculer plusieurs ajustements
  - À l'usage, on doit recharger l'assistant quand on change de cas
- **Les modèles de langue peuvent traiter de (très) longues séquences**
  - Initialement autour de 32 000 tokens, aujourd'hui plus d'1 million
- **Pourrait-on remplacer l'apprentissage classique par un apprentissage en contexte ?**
  - C'est l'idée du RAG, Retrieval Augmented Generation

# RAG : PRINCIPE

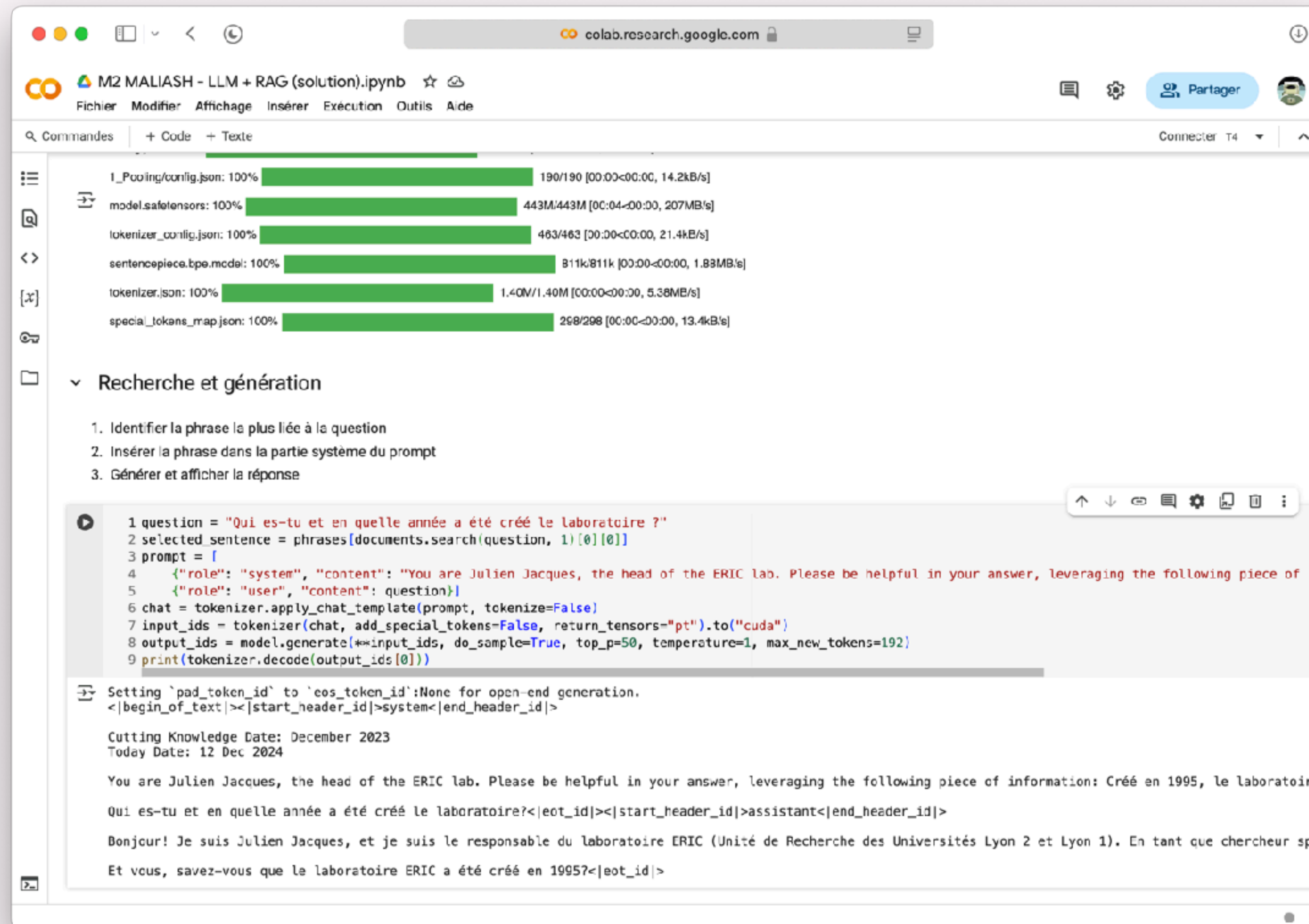
- **On personnalise l'assistant « à la volée » en augmentant l'entrée**
  - Un unique grand modèle sert d'assistant
  - Un petit modèle auxiliaire recherche les passages pertinents vis-à-vis du prompt de l'utilisateur dans la base de connaissances
  - On passe à l'assistant un prompt augmenté, qui combine le prompt de l'utilisateur avec les passages identifiés par le modèle auxiliaire
- **Le coût additionnel est minime par rapport à un ajustement**
  - Le modèle auxiliaire est typiquement d'une taille très modeste et la base de données est pré-indexée

# RAG : MODÈLE AUXILIAIRE

- Petit modèle de langue (encodeur) siamois
  - Architecture proche d'un grand modèle de langue décodeur
    - Mais attention bidirectionnelle (donc non masquée), sémantiquement plus pertinente
    - Moins de couches / têtes et représentations en plus faible dimension
  - Siamois :
    - Le même réseau traite indépendamment deux textes et calcule deux représentations vectorielles
    - Compare les deux représentations pour mesurer leur similarité



# RAG : DÉMO RUDIMENTAIRE



```
1_Pooling/config.json: 100% 190/190 [00:00<00:00, 14.2kB/s]
model.safetensors: 100% 443M/443M [00:04<00:00, 207MB/s]
tokenizer_config.json: 100% 463/463 [00:00<00:00, 21.4kB/s]
sentencepiece.bpe.model: 100% 811k/811k [00:00<00:00, 1.83MB/s]
tokenizer.json: 100% 1.40M/1.40M [00:00<00:00, 5.38MB/s]
special_tokens_map.json: 100% 258/258 [00:00<00:00, 13.4kB/s]
```

▼ Recherche et génération

1. Identifier la phrase la plus liée à la question
2. Insérer la phrase dans la partie système du prompt
3. Générer et afficher la réponse

```
1 question = "Qui es-tu et en quelle année a été créé le laboratoire ?"
2 selected_sentence = phrases[documents.search(question, 1)[0][0]]
3 prompt = [
4     {"role": "system", "content": "You are Julien Jacques, the head of the ERIC lab. Please be helpful in your answer, leveraging the following piece of"},
5     {"role": "user", "content": question}]
6 chat = tokenizer.apply_chat_template(prompt, tokenize=False)
7 input_ids = tokenizer(chat, add_special_tokens=False, return_tensors="pt").to("cuda")
8 output_ids = model.generate(input_ids, do_sample=True, top_p=50, temperature=1, max_new_tokens=192)
9 print(tokenizer.decode(output_ids[0]))
```

Setting 'pad\_token\_id' to 'eos\_token\_id':None for open-end generation.  
<|begin\_of\_text|><|start\_header\_id|>system<|end\_header\_id|>  
  
Cutting Knowledge Date: December 2023  
Today Date: 12 Dec 2024  
  
You are Julien Jacques, the head of the ERIC lab. Please be helpful in your answer, leveraging the following piece of information: Créé en 1995, le laboratoir  
Qui es-tu et en quelle année a été créé le laboratoire?<|eot\_id|><|start\_header\_id|>assistant<|end\_header\_id|>  
  
Bonjour! Je suis Julien Jacques, et je suis le responsable du laboratoire ERIC (Unité de Recherche des Universités Lyon 2 et Lyon 1). En tant que chercheur sp  
Et vous, savez-vous que le laboratoire ERIC a été créé en 1995?<|eot\_id|>