



MASTER OF SCIENCE  
IN ENGINEERING

**Hes·SO**

Haute Ecole Spécialisée  
de Suisse occidentale  
Fachhochschule Westschweiz  
University of Applied Sciences and Arts  
Western Switzerland

Master of Science HES-SO in Engineering  
Av. de Provence 6  
CH-1007 Lausanne

# Master of Science HES-SO in Engineering

Orientation : Computer-Science (CS)

## RAGFISH - Fishing the right answer for retrieval augmented generation

Author :

THOMAS DAGIER

Under the direction of :

Prof. Dr. Jean Hennebert

Lausanne, HES-SO//Master, June 01, 2024



# Table des matières

<b>Résumé</b>	<b>viii</b>
<b>Liste des acronymes</b>	<b>ix</b>
<b>Liste des illustrations</b>	<b>x</b>
<b>Introduction</b>	<b>1</b>
<b>1 Cahier des charges</b>	<b>2</b>
1.1 Mise en contexte . . . . .	2
1.2 Methodologie et Objectifs . . . . .	3
1.3 Planification . . . . .	4
1.3.a Phase 1 : Mise en place du projet (2 semaines) . . . . .	4
1.3.b Phase 2 : Approfondissement (3 semaines) . . . . .	5
1.3.c Phase 3 : Sprints Itératifs (3 semaines par sprint) . . . . .	5
1.3.d Revues et Ajustements . . . . .	6
1.3.e Gantt prévisionnel . . . . .	6
<b>2 Les Large Language Models (LLM)</b>	<b>7</b>
2.1 Description Générale . . . . .	7
2.2 Fonctionnement détaillé . . . . .	8
2.2.a La tokenisation . . . . .	8
2.2.b L'embedding . . . . .	10
2.2.c La génération de texte . . . . .	11
2.2.d L'entraînement des modèles . . . . .	13
2.3 Etude comparative . . . . .	15
2.3.a GPT-4 . . . . .	15
2.3.b Gemini . . . . .	16
2.3.c LLaMA-2 . . . . .	16
2.3.d Mistral . . . . .	17
2.3.e Quelques autres modèles . . . . .	18
2.3.f Plusieurs axes de comparaison . . . . .	18
2.4 Des défis à relever . . . . .	21

<b>3</b>	<b>Les Retrieval Augmented Generation Systems</b>	<b>23</b>
3.1	Description Générale . . . . .	23
3.2	Fonctionnement détaillé . . . . .	25
3.2.a	L'embedding des données . . . . .	25
3.2.b	Les bases de données vectorielles . . . . .	27
3.2.c	Le module de chat . . . . .	30
3.2.d	Le LLM . . . . .	31
3.3	Reflexions sur l'architecture . . . . .	32
3.3.a	Le LLM . . . . .	32
3.3.b	Le RAG System . . . . .	34
3.4	Des défis à relever . . . . .	35
<b>4</b>	<b>Réalisations</b>	<b>36</b>
4.1	Description générale . . . . .	36
4.1.a	Les besoins . . . . .	36
4.1.b	Les contraintes . . . . .	37
4.2	Description détaillée . . . . .	37
4.2.a	L'interface web . . . . .	39
4.2.b	Les embeddings et la base de données . . . . .	39
4.2.c	Le Large Language Model . . . . .	42
4.2.d	Le RAG . . . . .	43
4.2.e	Les problèmes rencontrés . . . . .	44
4.2.f	Les réponses aux questions ouvertes . . . . .	46
	<b>Conclusion</b>	<b>47</b>
<b>5</b>	<b>Annexes</b>	<b>48</b>
5.1	Chatbot . . . . .	48
5.2	Running the web service - linux / mac . . . . .	48
5.3	How to launch the tool with docker image on Linux . . . . .	49
5.4	How to deploy on Kubernetes . . . . .	49
5.4.a	Dockerisation on Gitlab Container Registry . . . . .	49
5.4.b	Deployment on K8S . . . . .	50
5.4.c	Deployment on K8S with Gitlab CI . . . . .	50
5.5	Using the service . . . . .	50
5.6	Ollama Server Deployment . . . . .	51
5.6.a	Script to deploy ollama on kubernetes . . . . .	51
5.6.b	Using kubectl . . . . .	51
5.6.c	Testing the deployment . . . . .	51
5.6.d	Monitoring the deployment . . . . .	51

5.6.e	Opened issues . . . . .	51
5.6.f	Documentation . . . . .	52
5.6.g	Deployment script . . . . .	52
<b>Références documentaires</b>		<b>55</b>

## RAGFISH - Fishing the right answer for retrieval augmented generation

<b>Type de projet</b>	Projet d'approfondissement
<b>Planification</b>	aucune planification
<b>Responsable</b>	Hennebert Jean
<b>Etablissement</b>	HEIA-FR
<b>Spécialisations concernée-s</b>	CS / Communication Systems CS / Cybersecurity CS / Embedded Systems CS / Software Engineering DS / Data Analytics DS / Data Engineering DS / Data Services TIC / Ingénierie logicielle TIC / Systèmes embarqués et mobiles
<b>Entreprise</b>	iCoSys
<b>Confidentiel</b>	non
<b>Descriptif</b>	<p>L'émergence des modèles de langage à grande échelle (LLM) a révolutionné le domaine de l'intelligence artificielle en offrant des capacités de compréhension et de génération de texte sans précédent. Plusieurs challenges comme la qualité des réponses générées montrent que les LLM peuvent encore être améliorés. En ce point, les systèmes de génération augmentée par récupération d'informations (RAG) se présentent comme une solution très prometteuse visant à perfectionner la pertinence et la précision des réponses.</p> <p>Bien qu'encourageant, les RAG posent toutefois plusieurs questions ouvertes. Premièrement, il s'agit de quantifier les limites de ces systèmes en estimant à quel point ils peuvent intégrer et utiliser efficacement les données externes ? Un autre défi qui impactera les coûts monétaires et énergétiques est celui du choix de l'architecture la plus adaptée selon le cas d'utilisation. En définitive, il convient de déterminer comment doit se comporter le système lorsque les connaissances du LLM entrent en interaction avec les données externes.</p>
<b>Travail demandé</b>	<p>Ce Projet d'Approfondissement envisage une approche de recherche empirique couplée à un développement pratique. L'objectif est d'approfondir la compréhension des systèmes LLM-RAG pour intégrer efficacement ces derniers dans des applications réelles. En se basant sur des études de cas spécifiques, nous chercherons à déterminer les configurations optimales pour différents scénarios d'utilisation. Une attention particulière sera également accordée à l'optimisation liée à la récupération et la génération d'informations externes.</p> <p><i>Les questions de recherche suivantes peuvent être formulées:</i></p> <ul style="list-style-type: none"><li>• <i>Comment préparer/nettoyer/filtrer les données et les sources à utiliser selon l'application ?</i></li><li>• <i>Quels sont les possibilités et impacts de formulation des prompts?</i></li><li>• <i>Comment réagit le système lorsque la réponse n'est pas présente dans la partie retrieval? Comment réagit le système lorsque l'information de la partie retrieval contredit la connaissance générale du LLM?</i></li><li>• <i>Est-il possible de mettre plus ou moins de poids sur la connaissance spécifique (partie RAG) vs connaissance générale (partie LLM)?</i></li><li>• <i>Quelles sont les limites du système en terme de RAG (nombre de documents) ?</i></li><li>• <i>Quelle architecture utiliser en fonction des besoins et des moyens à disposition ?</i></li></ul>

## RAGFISH - Fishing the right answer for retrieval augmented generation

### Connaissances préalables

### Mots-clés

### Compétences visées

- *Comment faire du Real-Time basé sur les données externes ?*
- *Comment induire la collecte d'infos et la génération de réponses avec le chat ?*
- *Quels seraient les gains monétaires et énergétiques d'une solution LLM-RAG ?*

Bases du machine learning, du deep learning et du NLP.

CS Natural Language Processing; DS AI; DS Analyse de données textuelles; DS Applications et Services

Gérer le projet	15%
Analyser et spécifier des produits / services	20%
Développer et réaliser	45%
Documenter (rapport)	20%

## Résumé

Les progrès fulgurants en termes de génération automatique de texte ont permis l'émergence d'une nouvelle source d'information : les Large Language Model (LLM). Capables d'assister les utilisateurs dans un panel de tâches très variées, ils deviennent rapidement des outils incontournables aux yeux de tous. Cependant, ils restent limités par leurs incapacités à accéder à des informations très récentes, trop spécifiques, voire sensibles comme le score du match de football de la veille, le règlement interne d'une école ou un document confidentiel d'entreprise. Les Retrieval Augmented Generation (RAG) Systems ont été introduits dans l'optique de proposer des assistants personnalisés, capables de résoudre ces problèmes en combinant l'utilisation traditionnelle de ces modèles à des sources de données externes et privées comme des PDF ou des sites web. À la Haute Ecole de l'Ingénierie et d'Architecture de Fribourg (HEIA-FR), les difficultés d'accès aux informations administratives chez les étudiants ont fait émerger l'idée d'un assistant virtuel combiné à un RAG afin de centraliser l'accès à l'information. Autour de ce cas d'utilisation, nous rentrons en détail dans le fonctionnement d'un tel système, évaluons les solutions actuelles et tentons de répondre à diverses questions ouvertes comme la capacité du modèle à répondre à une question même lorsque celui-ci ne connaît à priori pas la réponse. Dans un milieu en constante évolution, le monde de l'Intelligence Artificielle (IA) générative ouvre la porte à une multitude de projets innovants dont voici un Proof of Concept (PoC) pouvant donner lieu à de futures améliorations très prometteuses.



Candidat :

**Thomas Dagier**

Filière d'études : Computer Science

Professeur-e(s) responsable(s) :

**Dr Jean Hennebert**

Travail soumis à une convention de stage  
en entreprise : non

Travail soumis à un contrat de  
confidentialité : non



## Liste des acronymes

**API** Application Programming Interface.

**BPE** Byte Pair Encoding.

**GPU** Graphics Processing Unit.

**HEIA-FR** Haute Ecole de l'Ingénierie et d'Architecture de Fribourg.

**IA** Intelligence Artificielle.

**LLM** Large Language Model.

**NLP** Natural Language Processing.

**PoC** Proof of Concept.

**RAG** Retrieval Augmented Generation.

# Liste des illustrations

1.1	Gantt prévisionnel du projet . . . . .	6
2.1	Exemple de tokenisation avec GPT-3 . . . . .	9
2.2	Exemple de représentation vectorielle de tokens . . . . .	11
2.3	Exemple de génération de token avec GPT-3 . . . . .	12
2.4	Comparaison entre une recherche Greedy et Beam . . . . .	13
2.5	Schématisation du fonctionnement de Llama-2 . . . . .	16
2.6	Mistral AI comparés aux autres modèles du même rang . . . . .	17
2.7	Répartition des nouveaux LLM depuis 2019 . . . . .	19
2.8	HELM Leaderboard . . . . .	20
3.1	Schématisation du fonctionnement d'un RAG . . . . .	24
3.2	Classement des LLM par Elo sur Hugging Face . . . . .	33
4.1	Interface du Chatbot . . . . .	39
4.2	Différentes catégories de RAG . . . . .	43

## Référence des URL

URL01	<a href="https://openai.com">openai.com</a>
URL02	<a href="https://blent.ai">blent.ai</a>
URL03	<a href="https://huggingface.co">huggingface.co</a>
URL04	<a href="https://ai.meta.com">ai.meta.com</a>
URL05	<a href="https://mistral.ai">mistral.ai</a>
URL06	<a href="https://arxiv.org/abs/2307.06435">arxiv.org/abs/2307.06435</a>
URL07	<a href="https://crfm.stanford.edu/helm">crfm.stanford.edu/helm</a>
URL08	<a href="https://arxiv.org/abs/2312.10997">arxiv.org/abs/2312.10997</a>

# Introduction

L'avènement de ChatGPT vers la fin de l'année 2022 a marqué une véritable révolution dans l'utilisation des systèmes d' IA générative. Cette dernière s'est manifestée par la mise en œuvre quasi instantanée des LLM chez les géants Meta (Facebook), Google, Microsoft, etc. À l'aube d'une immense vague d'innovation, il n'a jamais été aussi facile d'utiliser ces outils qui ont bouleversé nos quotidiens. Si OpenAI a déclenché cet enthousiasme grandissant, ces modèles reposent en réalité sur une architecture de réseaux neuronaux conceptualisée il y a plusieurs années par Google : les Transformers.

Combinés avec des techniques d'entraînement avancées et une quantité de données colossale, les Transformers font des LLM, de puissants assistants capables de générer et traduire du texte, de rédiger du code ou encore de répondre à des questions dans une multitude de langues. Le très célèbre rapport "Data Age 2025", réalisé par l'International Data Corporation, précise que la quantité de données générées en 2025 est évaluée à 163 zettabits contre 33 en 2018. Cette croissance exponentielle des données disponibles est un des facteurs clés du succès de ces modèles, mais soulève toutefois des questions sur leur obsolescence. Les coûts faramineux liés à l'entraînement, à la consommation énergétique et à la qualité des réponses sont tout autant de limites technologiques qui se posent. Dans un contexte où même les LLM ne sont pas des outils miracles, l'idée des RAG Systems émerge pour les améliorer.

Ce projet réalisé dans le cadre du master MSE de la HES-SO vise à étudier en détail le fonctionnement de ces technologies afin de répondre à plusieurs questions ouvertes. Une étude menée en 2024 par Florence Meyer, responsable du service académique à la HEIA-FR, sur la facilité d'accès aux informations administratives pour les étudiants, a fait émerger le besoin d'un assistant virtuel dans le but de centraliser les sources d'informations en ligne jugées trop éparses. La mise en place de ce use-case aidera aussi à identifier dans quelle mesure les RAG peuvent aider à concevoir des systèmes d' IA personnalisés et adaptés à des besoins spécifiques.

L'intégralité du code et des données utilisées sont disponibles sur le repo<sup>1</sup> de ce projet.

---

<sup>1</sup>Disponible à l'adresse : <https://gitlab.forge.hefr.ch/thomas.dagierjo/pa-chatbot>

# Chapitre 1 :

## Cahier des charges

### 1.1. Mise en contexte

Avec l'essor des LLM, le domaine de l'intelligence artificielle a fait un grand bond en avant en offrant des capacités de compréhension et de génération de texte sans précédent. Dotés de facultés de traitement du langage naturel hors norme, ces modèles permettent d'accéder à une quantité d'informations quasi illimitée en un temps record et avec une simplicité déconcertante. Ils peuvent être perçus comme des sources d'informations massives, capables de répondre à une immense variété de questions et de problèmes. Mais est-ce vraiment la solution miracle ? Est-il possible de faire 100 % confiance à un LLM ? Comment être sûr que les réponses fournies sont correctes et à jour ? Que se passe-t-il si le modèle ne dispose pas des informations nécessaires pour répondre à une question ?

En règle générale, les LLM sont limités par les données sur lesquelles ils ont été entraînés à un instant T. Cela signifie qu'ils ne sont probablement pas capables de donner le score du derby Nantes-Rennes (pourtant soldé par un glorieux 0-3!) datant du 20 avril 2024. De la même manière, il y a peu de chance que ces modèles soient en mesure d'indiquer à un étudiant quand est son prochain cours et dans quelle salle. Ces différents cas de figure illustrent tous deux des problématiques récurrentes dans l'utilisation des LLM : il reste relativement compliqué d'accorder une totale confiance à ces modèles, notamment lorsqu'il s'agit de questions récentes ou très spécifiques.

Il est très fréquent d'être confronté à ce genre de problème lorsqu'on utilise un assistant virtuel. Par exemple, lorsqu'il précise dans sa réponse : "Je ne suis pas en mesure de répondre à cette question, car mes connaissances sont limitées au 20.10.2023", ou encore quand il source une information avec un document ou un site web qui n'existe en réalité pas. Ces situations sont problématiques et peuvent être source de confusion pour l'utilisateur. Dans un contexte où l'information accessible en ligne est de plus en plus abondante, il est donc essentiel de trouver des solutions pour garantir la fiabilité des réponses fournies par ces modèles qui deviennent omniprésents au quotidien.

Actuellement, les meilleurs LLM sont souvent ceux qui possèdent les corpus de données les plus vastes à entraîner. Ce processus demande des investissements colossaux, ce qui explique en partie pourquoi les modèles proposés par OpenAI, Google ou Meta sont généralement les plus performants. Cependant, il pourrait être intéressant de se demander si la norme future ne serait pas plutôt d'avoir des petits modèles, mais très spécialisés dans un domaine spécifique.

Imaginons un modèle dont la spécialité serait d'aider les étudiants de la HEIA-FR à trouver des informations sur les cours, les horaires, le règlement, etc. Ce modèle serait entraîné sur des données spécifiques à l'école, il n'y aurait pas besoin de lui apprendre à faire des dissertations à notre place, pas besoin non plus de lui donner un énorme corpus de données pour qu'il sache répondre à des questions sur la physique quantique. Plus petit, plus rapide et beaucoup moins coûteux, il ferait finalement exactement ce qu'on attend de lui.

Mais en réalité, cette solution n'est probablement pas optimale dans la mesure où le modèle ne serait jamais pas capable de répondre avec une qualité linguistique aussi élevée que les modèles actuels. Cependant, des mécanismes existent pour permettre aux LLM existants d'aller chercher des informations ailleurs que dans leurs données d'entraînement. C'est notamment ce que permettent les RAG qui fournissent un accès à des sources de données externes comme des PDF, des sites web, des bases de données, etc. pour combler les lacunes des LLM.

Florence Meyer, responsable du service académique à la HEIA-FR, a soulevé une problématique concernant l'accès aux informations pour les étudiants de l'école. De sorte à améliorer leur expérience, elle souhaite mettre en place un assistant virtuel capable de répondre à leurs questions et de les rediriger vers les services adéquats. En marge d'un projet essentiellement tourné vers la recherche et l'analyse des technologies de LLM et de RAG, le développement d'un assistant virtuel constitue un PoC visant à poser les bases d'une finalité plus ambitieuse. En prenant en compte le temps à disposition, il convient de formaliser toutes les étapes nécessaires à la réalisation de ce projet dans sa globalité et de définir les objectifs à atteindre dans le cadre de ce travail.

## **1.2. Methodologie et Objectifs**

L'objectif premier de ce projet est d'explorer en détail la notion de RAG : cette nouvelle méthode visant à améliorer les performances et la fiabilité des LLM. Le travail à réaliser constitue une grande part de recherche et d'expérimentation nécessitant d'investiguer sur les technologies sous-jacentes, les stratégies de préparation des données, et les méthodes d'intégration dans un cas pratique. Les résultats de ces recherches et de ces implémentations aideront à répondre à certaines questions ouvertes telles que :

1. La sélection d'une architecture optimale au regard des critères de coût, d'efficacité, et de précision.
2. Les stratégies de préparation, nettoyage, et filtrage des données pour maximiser l'efficacité de l'utilisation des RAG.
3. L'étude des stratégies de formulation de prompts pour les LLM et leur impact.
4. La gestion des cas où les réponses fournies par le RAG ne figurent pas dans les données du LLM ou contredisent ses connaissances intégrées.
5. Les méthodes de structuration des données pour optimiser l'accessibilité.
6. La sélection des sources de données externes les plus adéquates et leur intégration dans le modèle.
7. L'exploration de la capacité du système à initier la collecte d'informations et la génération de réponses via le chat.
8. L'évaluation des avantages financiers et environnementaux d'une solution combinant LLM et RAG.
9. L'ajustement du poids accordé aux sources de données externes par rapport aux connaissances intégrées du LLM.

La méthodologie agile ainsi que des revues régulières avec les parties prenantes permettront d'ajuster les objectifs en fonction des découvertes, des progrès réalisés, du temps, des ressources disponibles et surtout des nouveautés émergentes dans le domaine de l' IA générative.

### **1.3. Planification**

#### **a) Phase 1 : Mise en place du projet (2 semaines)**

Semaine 1 : Mise en place du projet

- Définition des objectifs et des questions clés à aborder
- Premières lectures sur les LLM et les RAG
- Mise en place de la documentation.

Semaine 2 : Formalisation du cahier des charges, définition des objectifs et planification des tâches pour les sprints

- Réunion avec Florence Meyer pour parler du projet
- Cahier des charges et planification des tâches pour les sprints
- Réalisation d'un mini projet pilote pour tester les technologies et comprendre leur fonctionnement.

## **b) Phase 2 : Approfondissement (3 semaines)**

Semaine 3 : Exploration et étude du fonctionnement des LLM

- Comprendre comment fonctionne les LLM
- Tester différentes implémentations de LLM
- Faire une étude comparative entre les modèles.

Semaine 4 : Exploration et étude du fonctionnement des RAG

- Comprendre comment fonctionne les RAG
- Tester différentes implémentations de RAG
- Faire une étude comparative entre les systèmes.

Semaine 5 : Exploration et étude du fonctionnement des bases de données vectorielles

- Faire une étude comparative sur les bdd vectorielles existantes (algo, runtime, framework, embeddings ...)
- Spécifications de l'architecture du projet.

## **c) Phase 3 : Sprints Itératifs (3 semaines par sprint)**

Sprint 1 : Proposer un prototype de base (Semaines 6 à 8)

- Faire une API intégrant les documents envoyés par Florence.
- Faire un front-end pour exposer le chatbot.

Objectif: montrer un MVP et identifier les points d'amélioration.

Sprint 2 : Amélioration du prototype (Semaines 9 à 11)

- Répondre aux questions 2. 5. 6. ou 7. à travers des recherches.
- Modifier le prototype en conséquence.

Objectif: améliorer le prototype et comprendre comment gérer les données.

Sprint 3 : Optimisation et Évaluation (Semaines 12 à 14)

- Répondre aux questions 3. 4. 8. ou 9. à travers des recherches.
- Modifier le prototype en conséquence.
- Evaluer les performances du système selon différentes métriques.

Objectif: améliorer le prototype et comprendre comment interagir avec le LLM en optimisant le fonctionnement du RAG.

### d) Revues et Ajustements

Chaque sprint est clôturé par un bilan sur l'avancée pour évaluer les progrès, discuter des défis rencontrés et ajuster les objectifs des sprints suivants. Cela inclut également la mise à jour de la documentation. La planification est voulue flexible, permettant des ajustements en fonction de l'évolution des besoins du projet et des technologies tout en favorisant les phases de recherches mises au service du développement de l'assistant virtuel.

### e) Gantt prévisionnel

Tâches	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14
Mise en place du projet														
Formalisation du cahier des charges, définition des objectifs, et planification des tâches pour les sprints														
Exploration et étude du fonctionnement des LLM														
Exploration et étude du fonctionnement des RAG														
Exploration et étude du fonctionnement des bases de données vectorielles														
Proposer un prototype de base														
Amélioration du prototype														
Optimisation et Évaluation														

FIG. 1.1 : Gantt prévisionnel du projet

*Source : Thomas Dagier*



## Chapitre 2 :

# Les Large Language Models (LLM)

### 2.1. Description Générale

Les Large Language Models LLM, représentent une classe de systèmes d' IA dotés de capacités avancées dans le domaine du traitement automatique du langage naturel, aussi appelé Natural Language Processing (NLP). Ils se caractérisent par leur aptitude à analyser, comprendre et générer du texte, simulant ainsi les compétences linguistiques humaines. Ces modèles sont capables de telles prouesses du fait qu'ils sont entraînés sur de très vastes ensembles de données textuelles.

L'émergence des LLM est intrinsèquement liée aux progrès fulgurants de l' IA. Une étude menée par Deng Cai (1) sur les récents avancements des RAG évoque des avancées significatives dans le domaine de l'apprentissage automatique et de la data science qui sont à l'origine de ces évolutions. Notamment dues à la disponibilité croissante de puissantes infrastructures informatiques et des Graphics Processing Unit (GPU) de plus en plus performants, ces dernières ont surtout bénéficié de l'abondance de données accessibles en ligne. Coïncidant avec des percées majeures dans la conception d'architectures de réseaux de neurones profonds, les LLM ont réellement émergé à la fin des années 2010 grâce à des outils révolutionnaires tels que les transformers proposés par Vaswani et al. en 2017.

Cependant, il faut attendre le début des années 2020 pour les voir devenir des outils incontournables. Ils jouent, par exemple, un rôle essentiel dans l'automatisation de tâches, la traduction automatique ou encore la rédaction de contenu. Ils bénéficient de leurs immenses corpus de données utilisés pour les entraîner afin de produire un langage fluide et cohérent dans une multitude de langues.

Un exemple concret d'application et d'utilisation des LLM est celui des assistants virtuels (chatbots). Capables de répondre aux questions des utilisateurs avec une grande précision grâce à leur aptitude à comprendre le langage naturel, ils interagissent de manière efficace et personnalisée comme le fait ChatGPT-4 d'OpenAI.

## 2.2. Fonctionnement détaillé

Un LLM peut se voir comme une boîte noire qui prend en entrée un certain nombre de phrases et qui répond par d'autres phrases. En réalité, le fonctionnement peut être découpé en deux étapes clés : la tokenisation et la génération de texte.

### a) La tokenisation

Ce premier processus consiste à découper un texte écrit par l'utilisateur en unités discrètes appelées "tokens". Le LLM n'étant pas capable de comprendre le texte à proprement parler, il se base sur ces tokens pour traiter le texte. Basés sur des réseaux de neurones profonds, ce type de modèle n'est pas en mesure de traiter autre chose que des nombres. Il existe donc différentes stratégies permettant de transformer un texte en une séquence de tokens (de nombres) compréhensible par le modèle. Mais toute la complexité derrière cette tâche réside dans le fait que le sens des mots et des phrases doit être préservé à mesure que la dimensionnalité du texte réduit. En d'autres termes, même si le modèle transforme le texte en nombres, il doit être capable de comprendre le sens des phrases initiales pour générer une réponse pertinente.

La découpe du texte peut être faite de différentes manières, un token pouvant aussi bien représenter un mot, une sous-unité de mot ou même un seul caractère. Des procédés extrêmement simples peuvent être utilisés, comme la séparation des mots par des espaces. Ici, chaque mot est donc associé à un token. D'autres approches comme la décomposition en "subword tokens" permettent de gérer des structures grammaticales un peu plus complexes en ramenant les mots à leur forme canonique (aussi appelée "lemme"). On parle alors de lemmatisation, comme dans le cas du mot "manger" qui serait ramené à son lemme "mange" plutôt que sa racine "mang".

Bien qu'elles réduisent la taille du vocabulaire, ces méthodes ne permettent cependant pas de gérer avec une grande précision les structures de phrases plus complexes. Introduit en 1994 par Philip Gage dans un article intitulé "A New Algorithm for Data Compression", le Byte Pair Encoding (BPE) se démarque alors comme une technique de compression de données qui découpe le texte en groupes de caractères. Très majoritairement utilisée aujourd'hui, elle permet de réduire la dimensionnalité du texte tout en conservant le sens général des phrases dans lesquelles ces mots sont employés.

Cette solution a été adaptée pour le traitement du langage naturel afin de devenir sans conteste la méthode de tokenisation la plus performante. Dans le cadre des LLM, le BPE n'est alors plus utilisé uniquement pour compresser des données, mais aussi et surtout pour décomposer astucieusement le texte en tokens, facilitant ainsi le traitement linguistique opéré par la suite.

Cette technique commence par traiter chaque caractère comme un token, puis fusionne progressivement les paires de tokens les plus fréquemment adjacentes pour former de nouveaux tokens plus grands. Ce processus itératif se poursuit jusqu'à ce qu'un vocabulaire ciblé soit atteint. Le BPE réduit efficacement la taille du vocabulaire nécessaire en traitant à la fois des mots entiers fréquents et en décomposant des mots rares en sous-chaînes gérables par le LLM. Voici un exemple de tokenisation avec BPE :

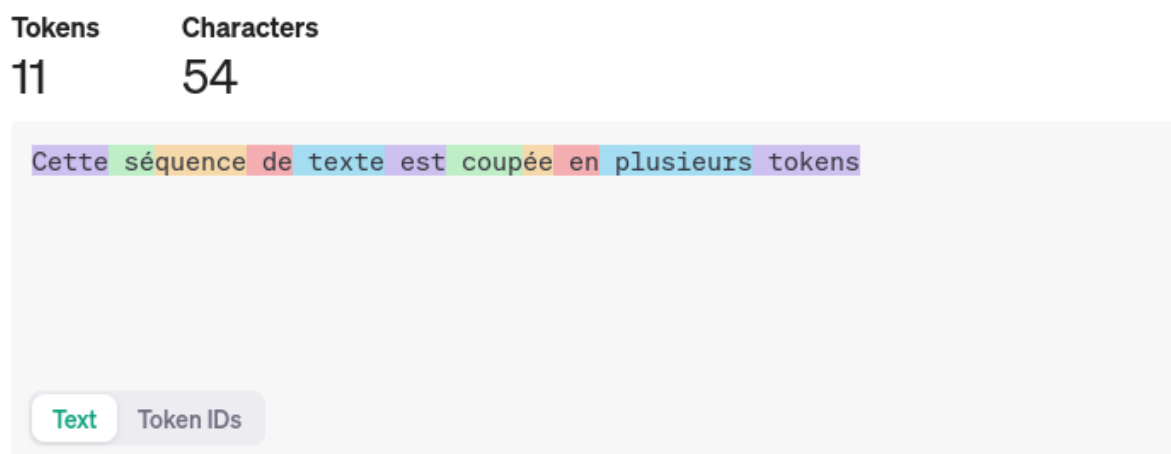


FIG. 2.1 : Exemple de tokenisation avec GPT-3

*Source : OpenAI, ref. URL01*

Cette image permet de constater que le texte est découpé en groupes de caractères de différentes tailles. Au premier abord, il semblerait que la séparation n'ait aucun sens, mais en réalité, c'est une méthode très efficace pour que le modèle puisse comprendre le sens global du texte. Cette image montre une technique de tokenisation propre à OpenAI et ses modèles GPT intitulée tiktoken<sup>1</sup>. C'est une variante de BPE, open source, pour laquelle un token généré correspond à environ quatre caractères et 100 tokens valent environ 75 mots.

D'autres dérivations de BPE existent, comme SentencePiece<sup>2</sup> qui combine BPE et Unigram pour offrir davantage de flexibilité dans la taille du vocabulaire. Cette méthode est connue pour permettre d'avoir une meilleure représentation des contextes, de réduire l'ambiguïté et de s'adapter à différentes langues. C'est notamment la méthode utilisée par les modèles Llama du groupe Meta AI. Basée sur des modèles de langues plus complexes, elle combine BPE et Unigram pour offrir plus de flexibilité au niveau de la taille du vocabulaire. À l'inverse, tiktoken possède l'avantage de gérer beaucoup plus efficacement les mots fréquents et rares tout en réduisant la taille du vocabulaire.

<sup>1</sup>Disponible à l'adresse : <https://github.com/openai/tiktoken>

<sup>2</sup>Disponible à l'adresse : <https://github.com/google/sentencepiece>

Des bibliothèques Python comme SpaCy<sup>3</sup> ou NLTK<sup>4</sup> proposent des outils pour réaliser ce genre de tâche, mais elles ne sont que très rarement implémentées dans les LLM à cause de leur complexité et de leur coût en ressources.

Comprendre comment fonctionne la tokenisation d'un LLM que l'on utilise est essentiel pour utiliser le modèle de manière efficace. D'après une conférence réalisée par Andrej Karpathy (2), ancien directeur de l'IA chez Tesla et spécialiste en Deep Learning chez OpenAI, la tokenisation explique à elle seule pourquoi les LLM ne sont pas toujours capables d'épeler des mots, ne sont pas aussi bons dans une autre langue que l'anglais, ou encore pourquoi ils ne sont pas capables de faire des calculs arithmétiques simples.

Si les techniques de tokenisation sont souvent des "boîtes noires" dans l'architecture des LLM, une étude approfondie menée par Mehdi Ali et Michael Fromm (3) montre que le choix du tokenizer peut avoir un impact significatif sur les performances du modèle, les coûts d'entraînement ainsi que l'inférence. En effet, une mauvaise tokenisation pourrait avoir comme effet de ne pas bien comprendre le sens des phrases. D'autre part, une mauvaise réduction du vocabulaire pourrait augmenter la taille du modèle et donc les coûts d'entraînement.

Un exemple fréquemment évoqué est celui de la tokenisation multilingue. Un modèle dont le tokenizer est centré sur l'anglais pourrait avoir des difficultés à traiter d'autres langues. Toujours selon cette étude, il semble que les tokenizers multilingues entraînés sur les cinq langues européennes les plus fréquentes nécessitent une augmentation de la taille du vocabulaire de trois fois par rapport à l'anglais. C'est donc un prix à payer pour permettre une meilleure polyvalence du modèle.

## **b) L'embedding**

Les mots maintenant transformés en tokens, il reste à les représenter de manière que le modèle puisse les manipuler. Les tokens sont représentés dans un espace latent, c'est-à-dire un espace de dimension réduite, comme des vecteurs. Ce sont précisément ces vecteurs qui sont utilisés par le modèle pour apprendre les relations entre les tokens.

Cette phase d'apprentissage est réalisée par des réseaux de neurones profonds qui sont des modèles mathématiques inspirés du fonctionnement du cerveau humain. Ce sont des unités de calcul interconnectées capables d'apprendre des représentations hiérarchiques de données. Voici un exemple de représentation vectorielle de tokens :

---

<sup>3</sup>Disponible à l'adresse : <https://spacy.io/>

<sup>4</sup>Disponible à l'adresse : <https://www.nltk.org/>

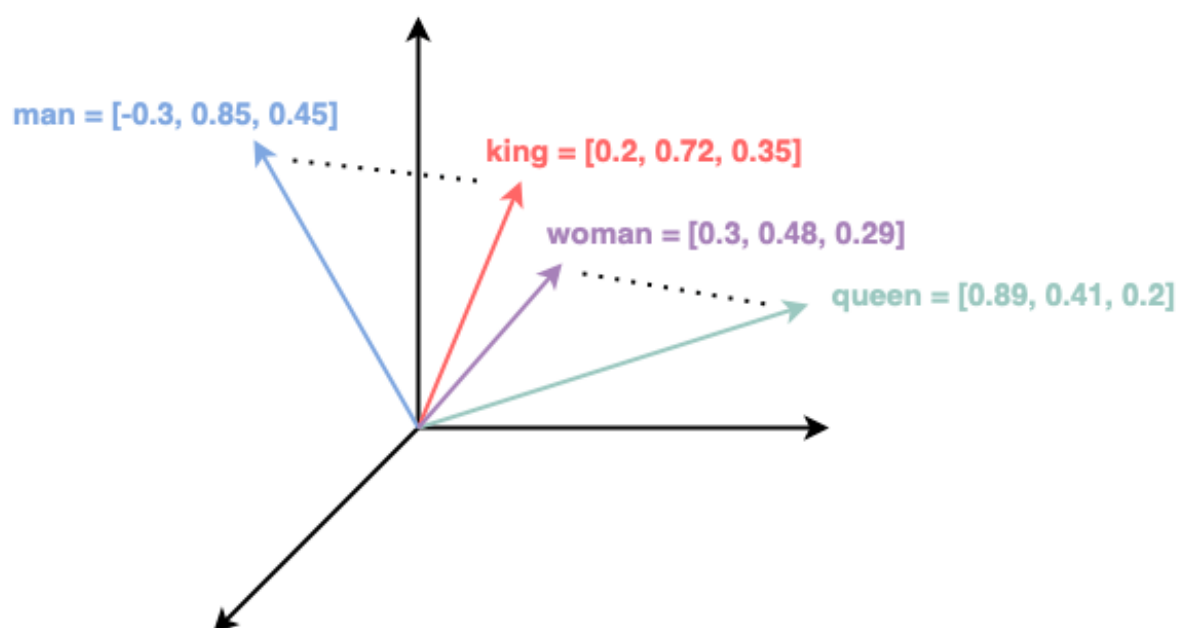


FIG. 2.2 : Exemple de représentation vectorielle de tokens

Source : *Blent.ai*, ref. URL02

Cette image montre comment les mots sont représentés dans un espace latent. L'embedding est intimement lié à la tokenisation, car c'est à partir des tokens que les vecteurs d'embeddings sont générés. Après une phase d'apprentissage permettant de capturer les relations sémantiques et syntaxiques entre les tokens, le modèle est capable de générer du texte cohérent avec les tokens qu'il a reçus en entrée.

### c) La génération de texte

La génération de texte n'est en réalité ni plus ni moins qu'un assemblage de tokens qui ont, eux-mêmes, été appris par le modèle. Lorsque le LLM répond à une question, il ne fait rien d'autre que de générer des tokens selon un processus itératif où le choix du token suivant est conditionné par la probabilité de s'intégrer au mieux dans le sens de la phrase.

L'ensemble du texte donné en entrée du modèle est appelé contexte. Ce dernier, décomposé en tokens, est passé à travers plusieurs couches de neurones qui permettent de capturer les relations entre les tokens et de comprendre, d'une certaine manière, le sens du texte. De cette manière, le modèle est capable de générer des tokens qui s'intègrent bien dans le contexte initial. Par exemple, si le modèle reçoit en entrée "Quelle est la capitale de la Suisse", il est capable de générer "Berne" en sortie.

Le modèle est capable de faire cela grâce à cette même étape de capture des relations entre les tokens qui a été faite pendant la phase d'entraînement. En supposant que le corpus de texte qui lui a été donné d'apprendre est suffisamment grand pour qu'il puisse avoir appris à un moment où à un autre que les mots "capitale" et "Suisse" sont fortement liés au mot "Berne" dans l'espace latent, il prédira que le mot "Berne" est le plus probable de répondre à la question. Bien évidemment, le modèle ne répond pas tel quel, il est entraîné à répondre de manière plus complexe, en générant des phrases entières.

En réalité, les vecteurs d'embedding sont passés à travers des couches de neurones qui permettent de générer des logits. Ces derniers sont ensuite transformés en probabilités par une fonction softmax. Le choix du token le plus adapté est alors fait en fonction de la probabilité conditionnelle la plus élevée associée aux vecteurs d'embedding. Cette technique est appelée échantillonnage stochastique :

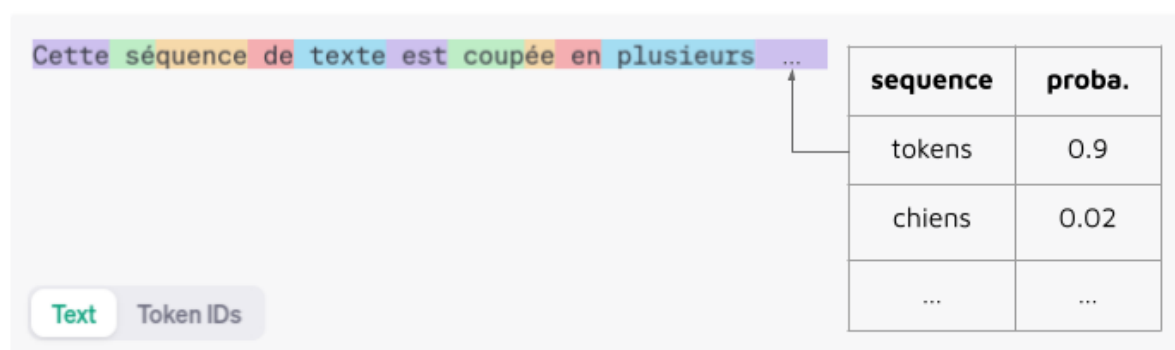


FIG. 2.3 : Exemple de génération de token avec GPT-3

*Source : OpenAI, ref. URL01*

Cette simple étape d'échantillonnage n'est cependant pas assez fiable pour être utilisée telle quelle. En effet, il n'est pas du tout garanti que le token le plus probable soit le plus adapté. C'est pourquoi d'autres techniques sont utilisées pour améliorer la qualité de la génération de texte. On note principalement trois méthodes qui sont le Greedy Sampling, le Beam Search et le Random Sampling. Ce sont tous des mécanismes d'attention qui aident à pondérer l'importance des tokens en fonction du contexte complet. D'après (4) qui référence plusieurs heuristiques, le Greedy Sampling, le plus simple, choisit le token le plus probable à chaque étape sans explorer d'autres possibilités. Sur le plan computationnel, c'est une méthode très efficace, mais cela peut conduire à des résultats peu diversifiés, voire déterministes. D'autre part, le Beam Search (ou recherche par faisceaux) est une amélioration du Greedy Sampling qui garde en mémoire les  $k$  tokens les plus probables à chaque étape. Voici une comparaison entre ces deux méthodes :

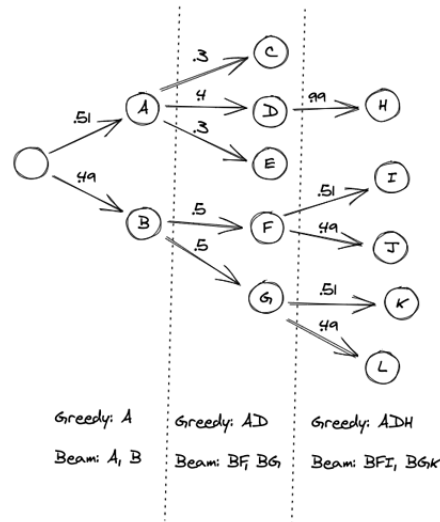


FIG. 2.4 : Comparaison entre une recherche Greedy et Beam

Source : HuggingFace, ref. URL03

Cette image permet de mettre en avant les différences entre les deux solutions, autant du point de vue de la complexité algorithmique que de la qualité de la séquence produite. Le Beam Search est, en effet, plus complexe, mais permet d'obtenir des résultats plus diversifiés et plus cohérents.

Une étude réalisée sur les différentes méthodes de génération de texte neuronal (5) évoque aussi une troisième méthode, le Random Sampling, qui consiste à choisir un token de manière aléatoire en fonction de sa probabilité. C'est une méthode qui permet d'obtenir des résultats très diversifiés, mais qui peut être difficile à contrôler. C'est pourquoi, il est souvent utilisé en combinaison avec d'autres méthodes pour obtenir des résultats plus cohérents comme le Top-k Sampling ou le Nucleus Sampling.

#### d) L'entraînement des modèles

Pour comprendre la manière dont les tokens sont déterminés pour la génération de texte, il a été fait mention plus tôt d'une phase d'entraînement du modèle. Cette dernière permet de capturer les relations entre les tokens, qui, eux-mêmes, réduisent la dimensionnalité de l'immense corpus de texte sur lequel le modèle doit être entraîné. Les ressources nécessaires pour réaliser cette tâche sont donc plus que considérables. Un entraînement de LLM se fait généralement en deux étapes : le pré-entraînement et le fine-tuning.

Lors du pré-entraînement, le modèle ingère de vastes ensembles de données textuelles non labellisées, souvent non structurées, et s'entraîne à capturer la structure et les schémas du langage. On parle alors d'apprentissage auto-supervisé. C'est précisément cette étape qui est la plus coûteuse en ressources.

Dans cette phase souvent décrite comme un apprentissage par transformer, le modèle est exposé à des exemples de données pour lesquels les poids des connexions entre les neurones sont ajustés afin de minimiser l'erreur de prédiction. Cela revient à capturer les relations sémantiques et syntaxiques entre les tokens, qui permettront, par la suite, de choisir les tokens suivants en fonction des logits.

Les corpus de textes comprennent généralement des articles en ligne, des livres ou des extraits de conversations, dans le but d'acquérir une compréhension approfondie des relations syntaxiques et sémantiques du langage. Le but est de représenter de la façon la plus fidèle possible la structure du langage naturel en capturant tout type de structure grammaticale, traditionnellement dans plusieurs langues. Pour la plupart des LLM connus du grand public, il ne semble pas possible de savoir dans les détails sur quoi ces derniers sont entraînés. Il est cependant estimé que les modèles les plus performants soient entraînés sur des corpus de plusieurs centaines de milliards de tokens, comme GPT-3 qui a été entraîné sur 570 Go de texte brut, soit environ 45 To de données tokenisées.

À la fin du pré-entraînement, le modèle est en théorie capable d'être utilisé comme une auto-complétion de texte. Pour qu'il puisse être utilisé dans des tâches plus complexes, il est alors nécessaire de le fine-tuner sur des tâches spécifiques à l'aide de données labellisées. Cette phase de fine tuning supervisée vise à adapter les poids du modèle pour une tâche particulière, telle que la traduction automatique, la génération de résumés ou la classification de texte.

En d'autres termes, quand un utilisateur pose une question à un LLM, il faut que le modèle soit capable de reprendre les mots de la question pour générer une réponse qui se rapproche au mieux de ce qu'un humain pourrait faire. Pour cela, il est nécessaire de lui fournir des exemples de questions et de réponses de sorte qu'il puisse apprendre à générer des réponses pertinentes.

Selon le LLM, la façon de labelliser les données peut changer, mais l'objectif et la démarche restent toujours les mêmes. Pour rentrer dans les détails architecturaux, le modèle entraîné peut être vu comme une superposition de couches qui possèdent chacune des rôles précis. Les couches supérieures du modèle sont généralement dédiées à l'apprentissage de tâches spécifiques comme la traduction ou la réponse aux questions, tandis que les couches inférieures, pré-entraînées sur des données non labellisées, conservent leur structure initiale. Cette approche permet d'exploiter les connaissances linguistiques générales acquises lors du pré-entraînement, tout en adaptant le modèle aux particularités des tâches que l'on souhaite le voir réaliser, comme de la traduction automatique ou de la génération de texte.



Contrairement au fine tuning supervisé traditionnel, qui se concentre sur l'optimisation des performances objectives en fonction de métriques prédéfinies, le fine tuning basé sur les préférences humaines vise à ajuster le modèle pour produire des résultats qui correspondent donc aux préférences subjectives des utilisateurs.

Cette approche implique nécessairement de solliciter des retours humains sur les sorties générées par le modèle et d'utiliser ces informations pour mettre à jour les poids du réseau. Par exemple, dans le domaine de la génération de texte, les utilisateurs peuvent fournir des commentaires sur la qualité, la pertinence ou la cohérence des phrases générées, ce qui permet d'ajuster progressivement le comportement du modèle pour mieux répondre aux attentes des utilisateurs. C'est aussi à cette étape qu'on intervient pour interdire certains cas d'utilisation ou pour corriger des biais.

## 2.3. Etude comparative

Il peut s'avérer très judicieux de comparer les modèles les plus connus et les plus actuels afin d'identifier des métriques de comparaison pertinentes et intemporelles, même si l'analyse ne compte que les modèles sortis avant Avril 2024.

### a) GPT-4

Développé par OpenAI et inauguré en mars 2023, GPT-4 se présente comme un jalon remarquable dans le domaine de l'intelligence artificielle conversationnelle, s'établissant rapidement comme une référence grâce à sa notoriété auprès du grand public. Cette prouesse technologique repose sur une architecture de type transformer, dotée de 1,5 trillion de paramètres qui témoigne de sa capacité à modéliser le langage naturel.

Entraîné sur des immenses corpus de données analogues à WebText, GPT-4 se distingue par ses performances inégalées sur divers benchmarks comme (6), attestant de ses compétences avancées en compréhension et en génération de texte. Accessible via Application Programming Interface (API) d'OpenAI, il offre un support multilingue très poussé et permet de customiser les prompts afin d'améliorer significativement ses performances sur des tâches spécifiques grâce à une quantité folle de 128 000 tokens de contexte.

Les différentes techniques de fine-tuning utilisées telles que le Reinforcement Learning from Human Feedback (RLHF) ou la maîtrise du contrôle de température et de max\_tokens affirment la position de GPT-4 comme leader dans l'évolution des systèmes de compréhension et de génération du langage naturel. La grande taille de ce modèle peut toutefois poser de gros défis en termes de déploiement, à tel point qu'il n'est disponible que via API.

## b) Gemini

Lancé par Google en réponse directe aux avancées réalisées par OpenAI avec ChatGPT, Gemini se positionne comme un acteur significatif dans l'espace des LLM avec une architecture sophistiquée dotée de 1.6 trillion de paramètres, entraîné, lui aussi, sur un corpus similaire à WebText. L'un des objectifs principaux de l'entraînement de Gemini réside dans l'exécution de tâches nécessitant un raisonnement très approfondi, ce qui lui permet de se distinguer par sa capacité à générer des explications scientifiquement précises et détaillées.

Sa conception est particulièrement axée sur l'apport de réponses dans des domaines nécessitant une expertise spécifique, marquant ainsi une évolution notable dans l'habileté des modèles de langage à traiter et à fournir des informations complexes. L'accessibilité de Gemini se fait via API mais n'est pas disponible en Suisse pour l'instant.

Les possibilités de personnalisation et d'accès restent davantage restreints, en raison de la nature propriétaire de la technologie. Si l'une de ses forces est son aspect multimodal, il est souvent mentionné que le multilingue peut être limité comparé aux autres solutions propriétaires.

## c) LLaMA-2

En juillet 2023, Meta AI propose LLaMA-2 qui vient se positionner comme une innovation majeure dans le paysage des LLM. Avec des modèles allant de sept à 70 milliards de paramètres, ce dernier incarne une architecture avancée et entraînée sur un corpus comprenant divers contenus extraits d'Internet. Voici un graphique montrant son fonctionnement général :

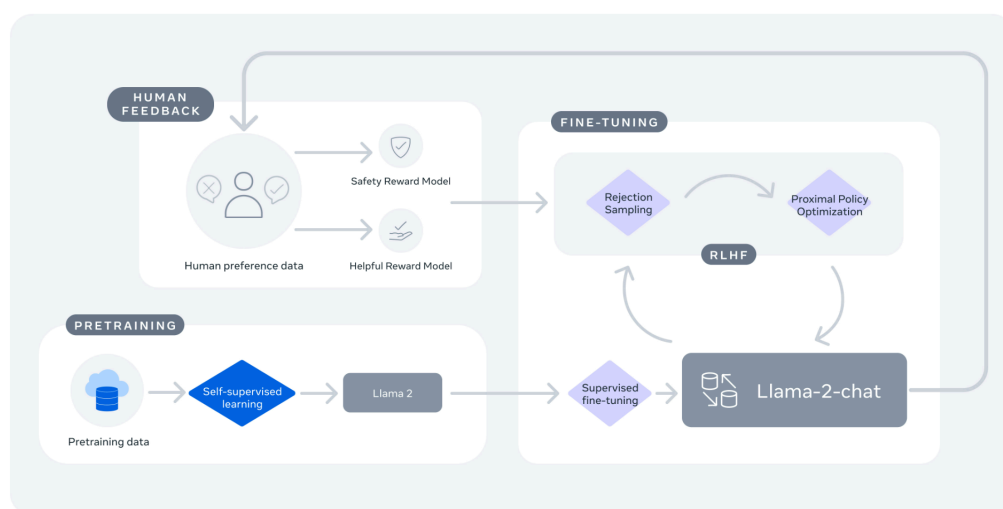


FIG. 2.5 : Schématisation du fonctionnement de Llama-2

Source : Meta, ref. URL04

Ce graphique aide à visualiser la manière dont le modèle fonctionne, en particulier comment il traite les données et comment fonctionne son entraînement. L'objectif central derrière LLaMA-2 est d'accorder une attention particulière à la modélisation linguistique, soutenue par des capacités multimodales très avancées.

Celles-ci permettent à LLaMA-2 de traiter et de générer du texte en association avec d'autres modalités sensorielles, étendant ainsi son application au-delà des simples tâches textuelles pour inclure des interactions impliquant des données visuelles ou auditives.

Offrant des optimisations pour réduire les exigences en matière de puissance de calcul, Meta AI propose ce modèle de manière open source avec une limite fixée à 4 000 tokens de contexte. Surtout utilisée pour le fine-tuning, cette version open source est appréciée pour sa polyvalence et sa capacité à s'adapter à diverses applications.

#### d) Mistral

Mistral AI se présente comme une initiative innovante dans le domaine des modèles d'intelligence artificielle, en mettant l'accent sur le développement de modèles plus petits et efficaces, tels que Mistral 7B et Mixtral, des modèles de Sparse Mixture of Experts. Cette orientation vers l'efficacité ne compromet pas la performance, au contraire, Mistral AI affiche une compétitivité vraiment remarquable avec les modèles de pointe en offrant des capacités d'inférence rapide grâce à des techniques d'attention par requêtes groupées et par fenêtre glissante.

Cependant, c'est surtout pour ses compétences avancées en matière de codage que Mistral AI se distingue aux yeux du grand public. Sous licence Apache 2.0, disponible sur HuggingFace et déployable sur des plateformes cloud, c'est le modèle qui offre la plus grande flexibilité en termes d'intégration pour les utilisateurs. Voici un graphique des scores proposés par Mistral AI :

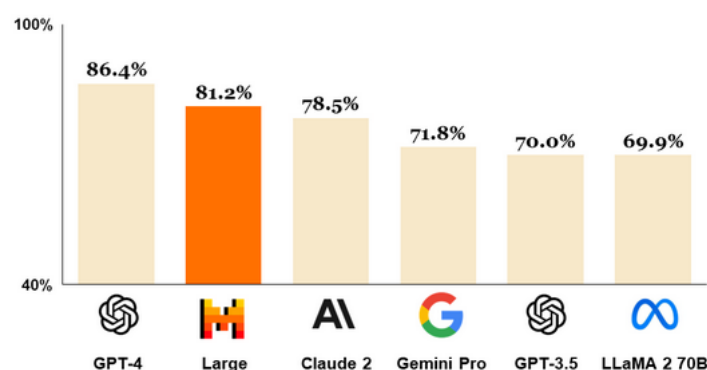


FIG. 2.6 : Mistral AI comparés aux autres modèles du même rang

Source : Mistral AI, ref. URL05

D'après ce graphique, il semblerait que Mistral AI se positionne comme un modèle très performant comparé à ses concurrents directs accessibles via API. Avec une fenêtre de contexte étendue à 8 000 tokens, ces modèles se prêtent bien à des applications nécessitant une compréhension approfondie, telles que les applications de chat ou la compréhension et la génération de code. Mistral AI dispose de modèles de plus petite taille, comme Mistral 7B ou 8x7B qui permettent de réduire les coûts de calcul afin de faciliter le déploiement sur des appareils avec des ressources limitées.

### **e) Quelques autres modèles**

D'autres modèles peuvent présenter un certain intérêt comme Flan-UL2, BLOOM ou Anthropic (Claude 3). Développé par Google Research, Flan-UL2 est un modèle de 20 milliards de paramètres qui s'appuie sur une méthode d'entraînement innovante, la Mixture-of-Denoisers (MoD), le rendant universellement efficace sur une variété de tâches de traitement du langage naturel (NLP). Bien que dépourvu de capacités multimodales, il excelle dans le traitement et la génération de texte en plusieurs langues, mais son accessibilité n'est pas précisée.

BLOOM, développé par le BigScience Workshop, est un modèle massif de 176 milliards de paramètres entraîné sur un corpus web multilingue qui se distingue par ses compétences d'optimisation du texte en fonction du style, du ton ou de la lisibilité. Modèle open source, il dispose d'une fenêtre de contexte de 2 000 tokens.

Enfin, Claude 3, proposé par Anthropic se démarque par son orientation vers la sécurité et l'alignement des valeurs dans l'IA, offrant des capacités multilingues, de traitement de la vision et de facilité de direction. Certains benchmarks comme (6) le place devant Llama-2 en termes de performance bien qu'il ne soit pas possible de tester ce modèle actuellement.

### **f) Plusieurs axes de comparaison**

Après avoir vu plusieurs modèles aux architectures et objectifs variés, il semble intéressant de les comparer pour mieux comprendre leurs différences. Au premier abord, les modèles comme GPT-4, Gemini et LLaMA-2 se distinguent par leur taille et leur capacité à traiter des tâches complexes, tandis que Mistral se démarque par son efficacité et sa flexibilité. Les modèles comme Flan-UL2, BLOOM et Anthropic (Claude 3) offrent des fonctionnalités spécifiques qui les rendent attrayants pour des applications particulières. On distingue déjà une première séparation marquée par la différence entre les modèles open-source et les modèles propriétaires.

Souvent optimisés pour une utilisation en production, les modèles propriétaires sont rarement autorisés à être inspectés, modifiés ou personnalisés. D'après (7), ils ne sont pas toujours disponibles gratuitement et ne permettent pas aux utilisateurs de contrôler les données qui sont utilisées pour l'entraînement.

Ces derniers doivent donc faire confiance au propriétaire du modèle pour garantir un engagement envers la protection de la vie privée des données et l'utilisation responsable de l'IA. Les modèles propriétaires sont fréquemment beaucoup plus grands du fait qu'ils bénéficient de plus de ressources pour leur développement. Cela ne garantit cependant pas toujours des performances supérieures.

On retrouve dans cette catégorie les modèles d'OpenAI, Google Gemini ou Claude 3. Avec le nombre estimé d'utilisateurs actifs de ChatGPT dépassant les 180 millions et un nombre récent de mises à jour époustouflantes, il est facile d'oublier qu'il existe en réalité de nombreux autres LLM. En fait, il y a même plus de modèles open source qui ont été publiés dernièrement que de modèles propriétaires.

Voici un graphique qui montre le nombre de modèles open source et propriétaires introduits au fil des années :

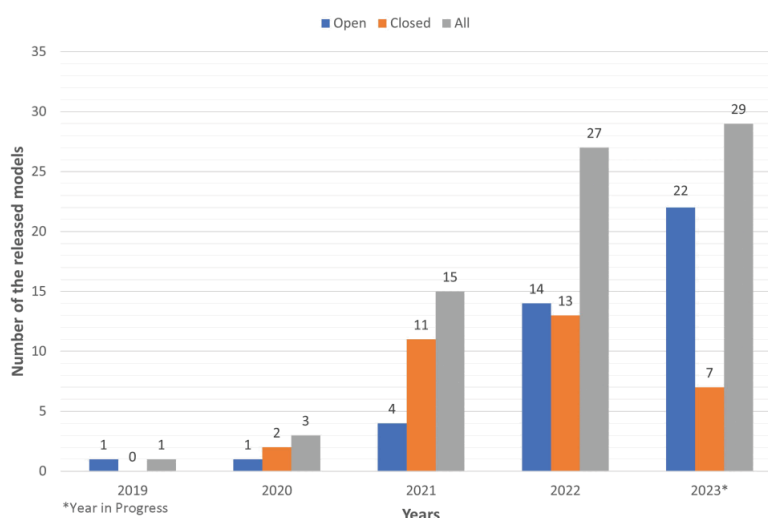


FIG. 2.7 : Répartition des nouveaux LLM depuis 2019

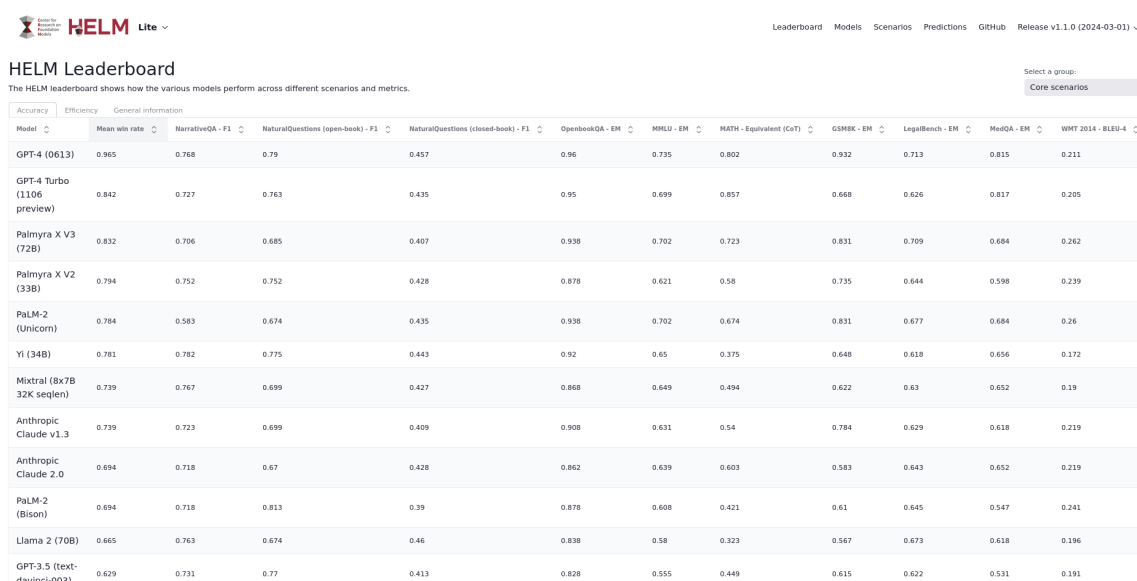
Source : *A Comprehensive Overview of LLM*, ref. URL06

Ce graphique montre que le nombre de modèles open source introduits a augmenté de manière significative ces dernières années, ce qui suggère une tendance croissante vers l'ouverture et la collaboration dans le domaine des LLM. Mis à disposition du public, ils peuvent être modifiés et personnalisés, mais ne sont pas toujours aussi optimisés et performants que les modèles propriétaires.

On retrouve dans cette catégorie des modèles comme LLaMA-2, Mistral ou Flan-UL2. L'aspect open-source de ces modèles permet une plus grande transparence et une plus grande confiance dans les résultats produits, bien qu'ils puissent être plus difficiles à déployer et à maintenir, car ils nécessitent souvent des ressources supplémentaires pour être utilisés efficacement.

Les modèles open-source sont généralement plus scalables et plus accessibles, ce qui les rend attrayants pour les chercheurs et les développeurs qui souhaitent personnaliser les modèles pour des tâches spécifiques. Des outils comme CodeGen pour les langages de programmation ou BloombergGPT pour la finance montrent une tendance vers le développement de modèles plus spécialisés pour une performance améliorée dans des domaines spécifiques et soutenus par la nature open-source de ces modèles. HuggingFace propose plusieurs benchmarks, dont (8) pour comparer les performances des modèles open-source.

Du point de vue des performances, il est difficile de départager les modèles. Les benchmarks comme (7) ou (8) montrent que les performances des modèles varient en fonction des tâches et des métriques utilisées :



The HELM leaderboard shows how the various models perform across different scenarios and metrics.

Model	Mean win rate	NarrativeQA - F1	NaturalQuestions (open-book) - F1	NaturalQuestions (closed-book) - F1	OpenbookQA - EM	NRRLU - EM	MATH - Equivalent (CoT)	GSMBK - EM	LegalBench - EM	MedQA - EM	WMT 2014 - BLEU-4
GPT-4 (0613)	0.965	0.768	0.79	0.457	0.96	0.735	0.802	0.932	0.713	0.815	0.211
GPT-4 Turbo (1106 preview)	0.842	0.727	0.763	0.435	0.95	0.699	0.857	0.668	0.626	0.817	0.205
Palmira X V3 (72B)	0.832	0.706	0.685	0.407	0.938	0.702	0.723	0.831	0.709	0.684	0.262
Palmira X V2 (33B)	0.794	0.752	0.752	0.428	0.878	0.621	0.58	0.735	0.644	0.598	0.239
PaLM-2 (Unicom)	0.784	0.583	0.674	0.435	0.938	0.702	0.674	0.831	0.677	0.684	0.26
Yi (34B)	0.781	0.782	0.775	0.443	0.92	0.65	0.375	0.648	0.618	0.656	0.172
Mixtral (8x7B 32K seqlen)	0.739	0.767	0.699	0.427	0.868	0.649	0.494	0.622	0.63	0.652	0.19
Anthropic Claude v1.3	0.739	0.723	0.699	0.409	0.908	0.631	0.54	0.784	0.629	0.618	0.219
Anthropic Claude 2.0	0.694	0.718	0.67	0.428	0.862	0.639	0.603	0.583	0.643	0.652	0.219
PaLM-2 (Bison)	0.694	0.718	0.813	0.39	0.878	0.608	0.421	0.61	0.645	0.547	0.241
Llama 2 (70B)	0.665	0.763	0.674	0.46	0.838	0.58	0.323	0.567	0.673	0.618	0.196
GPT-3.5 (text-davinci-003)	0.629	0.731	0.77	0.413	0.828	0.555	0.449	0.615	0.622	0.531	0.191

FIG. 2.8 : HELM Leaderboard

Source : *Stanford.edu*, ref. URL07

Avec cet extrait du classement des LLM réalisé le 01 Mars 2024, il paraît clair que les modèles propriétaires comme GPT-4 ou Gemini sont souvent en tête des classements, mais les modèles open-source comme LLaMA-2 ou Mistral ne sont pas loin derrière. Les performances des modèles dépendent de nombreux facteurs, notamment la taille du modèle, la qualité des données d'entraînement, les hyperparamètres utilisés, etc.

L'utilisation de mécanismes d'attention, de fonctions d'activation (par exemple, ReLU, GeLU, variantes de GLU) et de techniques de normalisation des couches est fréquemment retrouvée dans les architectures des modèles les plus performants.

Le HELM leaderboard (7) (pour Holistic Evaluation of Language Models) est un bon exemple pour avoir une idée générale des performances des modèles. Il montre que les modèles propriétaires comme GPT-4 sont généralement en tête des classements, mais que les modèles open-source comme LLaMA-2 ou Mixtral ne sont pas loin derrière.

On retrouve des métriques comme l'Exact Match (EM) pour évaluer la précision des réponses, le F1 pour évaluer la qualité des réponses, le BLEU-4 pour évaluer la qualité des traductions, le AI2 Reasoning Challenge (ARC) pour évaluer la capacité de raisonnement des modèles, etc. Ces métriques permettent de comparer les performances des modèles sur une variété de tâches et de domaines, ce qui donne une idée générale de leurs capacités.

## 2.4. Des défis à relever

Cette section sur les LLM permet de mettre la lumière sur la multitude de modèles disponibles et leurs caractéristiques spécifiques. Des solutions propriétaires qui garantissent des performances de pointe à des modèles open-source qui offrent une plus grande transparence et une plus grande flexibilité, il existe une variété de choix pour répondre aux besoins des utilisateurs.

Les performances des modèles varient en fonction des tâches et des métriques utilisées, mais il est clair que les LLM ont révolutionné le domaine du traitement automatique du langage naturel en offrant des capacités de compréhension et de génération de texte sans précédent.

Cependant, malgré leur puissance et leur polyvalence, ces modèles sont confrontés à plusieurs défis critiques qui limitent leur fiabilité et leur applicabilité dans des scénarios du monde réel.

Entraînés sur d'énormes corpus de texte collectés sur Internet, les LLM sont inévitablement exposés à des biais présents dans ces données d'entraînement. D'après (7), il est important de comprendre comment les biais se propagent dans les modèles de langage et comment ils peuvent être atténués pour garantir des résultats fiables.

De plus, (7) précise que les LLM ont tendance à générer du texte qui ne présente parfois pas de sens. Ces cas d'utilisation sont souvent provoqués par le fait que les informations apprises par le modèle ne sont plus forcément à jour ou simplement qu'il ne connaisse pas la réponse mais essaye d'y répondre quand même en fournissant de fausses informations.

Dans une toute autre mesure, (7) évoque aussi des considérations éthiques qui englobent une gamme de problématiques, allant de la transparence de l'entraînement des modèles et de leur fonctionnement à l'impact sociétal de leur déploiement. Les questions relatives à la vie privée des données, au consentement implicite dans les données d'entraînement et à l'autonomie dans la prise de décision soulignent la complexité de ces enjeux.

On parle généralement de considérations réfléchies et mesurées dans le développement des LLM puisque le choix des données d'entraînement peut avoir un impact majeur sur le comportement du modèle qui peut s'avérer discriminatoire dans des conditions extrêmes.

Lorsque les résultats sont incorrects, beaucoup parlent d'un phénomène d'hallucination associé aux données générales utilisées pour l'entraînement. Ce terme fait référence à la tendance des LLM à générer des informations fausses ou non vérifiées présentées comme des faits. Les réponses à des questions posées sont présentées de manière convaincante, même en l'absence de bases factuelles solides. Elles peuvent compromettre la fiabilité des applications basées sur des LLM, notamment dans des domaines sensibles tels que la santé, le droit et l'information, où la précision des données est cruciale.

Toutes ces préoccupations ont conduit à l'exploration de solutions plus avancées telles que les RAG. Ces modèles hybrides, qui proposent des capacités de génération augmentée de texte grâce à une récupération d'informations externes, offrent une voie prometteuse pour surmonter les limites des LLM traditionnels et améliorer leur précision et leur fiabilité.

C'est aussi une manière très efficace d'utiliser les LLM pour des tâches beaucoup plus spécifiques comme l'apprentissage d'une documentation spécifique à une entreprise. La section suivante se penchera sur l'analyse des RAG en explorant leur fonctionnement et en essayant de répondre à toutes les questions qui peuvent se poser à leur sujet.



## Chapitre 3 :

# Les Retrieval Augmented Generation Systems

### 3.1. Description Générale

Si les LLM offrent effectivement des capacités de compréhension et de génération de texte sans précédent, il n'en reste pas moins que certaines questions restent en suspens. La multiplication des données disponibles ainsi que la facilité d'accès en ligne font que les modèles peuvent être très rapidement rendus obsolètes. Il est souvent question de déterminer de quelle manière maintenir à jour les données utilisées. Un fine-tuning plus régulier pourrait aider, mais cela demanderait des ressources très importantes, ce n'est clairement pas une solution viable.

Plusieurs questions peuvent donc se poser quant à la fiabilité des réponses générées : est-ce que les informations sont encore fiables au moment où la question est posée ? Que se passe-t-il si le modèle ne connaît tout simplement pas la réponse ? Dans quelles mesures est-il capable de produire des fausses réponses ? Ce genre de problématiques sont habituellement corrélées avec l'effet d'hallucination qui décrit la manière dont les LLM mettent en avant d'inexactes informations pour tenter de répondre à tout prix.

En outre, il est fréquemment fait mention de limites au niveau de la taille du contexte des LLM. Si elle suffit généralement à tenir une discussion, elle peut être une forte limitation dans certains cas. Lorsqu'on demande au modèle de résumer un livre, par exemple, l'ensemble de son contenu est copié-collé dans le contexte. Selon la taille du contexte, il est assez facile de dépasser la limite des tokens autorisés. Dans ce cas, le modèle pourrait répondre sans tenir compte de l'ensemble des informations ou même refuser de répondre. C'est notamment le comportement de ChatGPT qui notifie un dépassement de la taille du contexte et coupe la discussion.

La partie précédente a montré que le contexte est généralement limité entre 2 000 et 8 000 tokens en fonction des modèles. Pour la plupart des tokenisers actuels, cela veut dire que les utilisateurs sont limités à environ 12 pages de texte, ce qui peut s'avérer très contraignant. À travers cet ensemble de problèmes, les constats effectués montrent que les LLM ne devraient pas uniquement se baser sur leurs connaissances générales pour répondre à des questions. C'est dans ce contexte que le groupe Meta fait émerger l'idée des RAG.

L'objectif de ces derniers est de mieux contextualiser les questions posées en allant chercher des informations vers des sources de données externes. Une étude sur l'utilisation des RAG menée par Patrick Lewis en 2021 et supportée par le laboratoire de recherche en IA de Facebook (9) fait une description du fonctionnement des RAG qui pourrait être schématisée par la figure suivante :

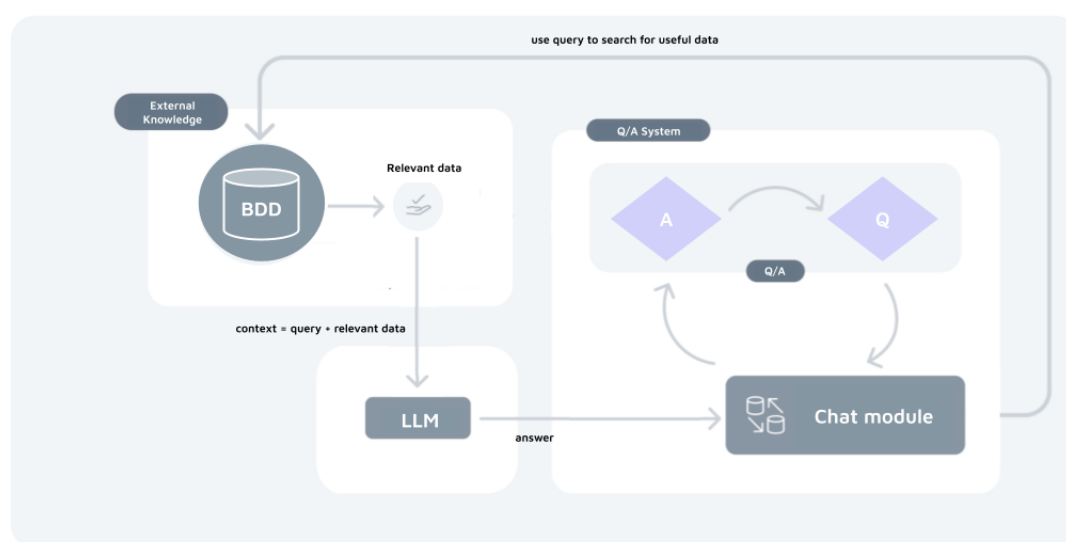


FIG. 3.1 : Schématisation du fonctionnement d'un RAG

*Source : Thomas Dagier*

Ce schéma met en avant une architecture d'un RAG divisée en trois modules : le chat, qui sert d'interface utilisateur, le LLM qui est le modèle de génération de texte et la base de données qui contient les informations externes sur lesquelles s'appuyer avant de passer par le LLM. L'idée est de pouvoir ajouter des informations contextuelles à la question posée pour garantir des réponses plus précises.

## 3.2. Fonctionnement détaillé

### a) L'embedding des données

Pour aider les LLM à répondre, il est donc indispensable d'ajouter au contexte des informations pertinentes provenant de documents qui ne sont pas publics ou qui seraient trop spécifiques pour être intégrés dans les connaissances générales des LLM.

Il n'est pas difficile d'imaginer que les informations mises à disposition des étudiants de la HEIA-FR n'ont probablement pas été utilisées pour entraîner un LLM. Même si c'était le cas, ces données sont susceptibles de changer régulièrement. Un LLM seul ne suffit pas pour venir en aide aux étudiants. Par contre, couplé à un RAG, il serait capable de mettre en forme une réponse sans compter uniquement sur ses connaissances générales. Sachant que le contexte est très limité, l'enjeu est de trouver un moyen de stocker des données de manière efficace, accessibles rapidement et qui permette d'identifier les informations les plus pertinentes à retourner au LLM.

Comme pour ces derniers, il faut faire appel à des techniques d'embeddings. Il est possible de se baser sur des embeddings pré-entraînés ou de les générer à la volée (ce qui est souvent le cas pour les RAG car les données sont assez spécifiques). Plusieurs techniques d'embeddings peuvent être utilisées pour générer des représentations numériques des documents. BERT (pour Bidirectional Encoder Representations from Transformers) est de loin l'algorithme le plus utilisé dans le contexte des RAG du fait qu'il permette de générer des embeddings contextuels en traitant le texte dans les deux directions simultanément. Cette caractéristique le rend particulièrement efficace pour comprendre le contexte des mots dans une phrase.

Une variante de BERT, RoBERTa (pour Robustly Optimized BERT approach) est un modèle avec des hyperparamètres optimisés qui lui permettent d'obtenir de meilleures performances sur une variété de tâches de traitement du langage naturel. Plus coûteux en ressources que BERT, qui l'est déjà beaucoup, il nécessite un grand corpus de texte pour être entraîné. Toujours dans le même style que BERT, Sentence-BERT (SBERT) permet de générer des embeddings en réduisant la dimensionnalité afin d'accélérer la recherche d'informations bien qu'elle soit moins précise.

Moins utilisé, mais très efficace, Dense Passage Retrieval (DPR) est spécifiquement conçu pour améliorer la récupération d'informations dans les systèmes de questions-réponses. Grâce à des embeddings denses, DPR index et récupère des passages avec une très grande rapidité, mais peut s'avérer moins flexible que les autres solutions dans des domaines de NLP différents.

Bien que BM25 ne soit pas un algorithme d'embedding au sens traditionnel, il peut tout de même être intéressant de le mentionner en combinaison avec Elasticsearch puisqu'il reste très performant sur de la recherche par mots-clés et très simple à mettre en place. Basé sur des embeddings denses, il reste néanmoins moins performant que DRP pour comprendre le contexte ou la sémantique profonde.

Dans un tout autre registre, les modèles GPT-3 et suivants montrent de très bons résultats pour générer des embeddings de texte grâce à leur conception leur permettant de comprendre et générer des réponses basées sur une vaste gamme de contextes et de styles de texte. Extrêmement polyvalent, les coûts d'inférence et d'enregistrement des données sont cependant très élevés.

Parmi toutes ces possibilités, les techniques intrinsèques de génération d'embeddings semblent cependant toutes s'inspirer d'une manière ou d'une autre du BPE et du clustering de caractères. L'idée est de regrouper des caractères entre eux, puis d'en faire des regroupements cohérents pour finalement faire de l'overlap entre les séquences de texte pour conserver un minimum de contexte entre deux blocs successifs.

Plutôt que de réimplémenter ces techniques, il est possible de se tourner vers des frameworks d'embedding qui offrent des solutions prêtes à l'emploi. OpenAI propose divers modèles d'embedding, comme Ada, qui s'appuient sur des architectures de transformation avancées inspirées de GPT et d'autres modèles transformer. Sa facilité d'utilisation, sa polyvalence et le soutien d'une infrastructure cloud puissante en font un choix populaire pour les applications de NLP bien que les utilisateurs dépendent des API d'OpenAI, ce qui peut impliquer des coûts ou des limitations d'utilisation.

Instructor-xl est un autre framework d'embedding qui se concentre sur des capacités d'attention à grande échelle et des architectures transformer optimisées pour des tâches d'embedding spécifiques. Bien que les détails spécifiques sur "Instructor-xl" soient moins communs dans la littérature publique, les modèles tendent à être optimisés pour des tâches spécifiques ou des performances accrues sur certains types de données. Son utilisation peut cependant nécessiter une familiarisation avec l'architecture pour une utilisation optimale.

Bien plus répandu, Multilingual-e5-large est un modèle conçu pour générer des embeddings de texte efficaces sur plusieurs langues, basé sur des architectures transformer telles que celles utilisées dans BERT ou ses variantes multilingues. Ce type de modèle est essentiel dans les applications nécessitant un support multilingue, offrant une compréhension et une représentation transversales des langues. Parfois moins précis pour des tâches spécifiques dans une langue unique, il reste cependant la méthode la plus accessible pour un public global.

Peu importe la méthode ou le framework utilisé, un point crucial à prendre en compte pour générer des embeddings de qualité est la qualité des données en entrée. Il est essentiel de nettoyer les données pour éliminer les erreurs, les doublons et les informations inutiles, de même que les données bruitées ou mal formatées. Souvent chronophage, ce processus est relativement subjectif et doit être testé pas à pas en fonction des besoins spécifiques de chaque application.

La technique d'embedding qui est la plus connue et la plus utilisée dans le domaine du NLP est sans doute celle de Word2Vec. Elle permet de générer des embeddings de mots en se basant sur la distribution des mots dans un corpus de texte. Les mots qui apparaissent fréquemment ensemble sont considérés comme similaires et sont donc représentés par des vecteurs proches dans l'espace vectoriel. Cette technique est très efficace pour capturer les relations sémantiques entre les mots et est généralement utilisée comme point de départ pour des tâches plus complexes. Cependant, elle n'est pas très évoquée ici, car elle n'est pas particulièrement intégrée dans les frameworks d'embedding modernes qui privilégient des approches plus avancées.

En plus du choix du framework, il est aussi très important de prendre en considération certaines politiques de sécurité et de confidentialité des données. Les données sensibles ou privées doivent être protégées et stockées uniquement lorsque cela est absolument nécessaire, puisqu'elles seront retournées telles quelles par le RAG si elles sont stockées dans la base de données vectorielle.

## **b) Les bases de données vectorielles**

Les vecteurs d'embeddings sont des données numériques qui peuvent être stockées dans des bases de données vectorielles. Ces dernières sont des outils permettant de manipuler efficacement des vecteurs d'embeddings tout en établissant des relations sémantiques entre eux. C'est un peu l'équivalent de l'espace latent d'un LLM mais pour des données structurées et stockées de manière persistante.

Le choix de cet outil est crucial pour répondre aux besoins du RAG qui sont la rapidité de recherche et la pertinence des résultats retournés. Une étude paru en 2023 sur l'idée d'augmenter le contexte des LLM réalisée par Bowen Peng (10) a montré que la qualité des réponses générées est de moins en moins bonne à mesure que la taille du contexte augmente. Cela s'explique par le fait que les modèles ont parfois du mal à se concentrer sur les informations qui aident réellement à répondre à la question.

Dans la question : “Bonjour, je souhaiterais savoir quelle est la capitale de la Suisse, car un ami me l’a demandé avant-hier alors que nous étions au bar...”, on retrouve beaucoup d’informations inutiles qui viennent perturber l’échantillonnage stochastique (comme le fait que l’utilisateur était au bar) et qui peuvent mener à des réponses moins précises. Le RAG venant nécessairement augmenter la taille du contexte, il est indispensable de trouver l’équilibre entre quantité et qualité lorsque l’on utilise la base de données vectorielle.

Une base de données vectorielle peut être décomposée en une phase d’indexation et une phase de recherche. L’indexation consiste à stocker les vecteurs d’embeddings de manière à ce qu’ils soient facilement accessibles lors de la recherche. Cette étape qui intervient à la suite de la génération des embeddings est cruciale pour garantir des temps de réponse rapides. Plusieurs algorithmes d’indexation peuvent être utilisés pour organiser les vecteurs dans l’espace vectoriel, tels que les KD-Trees, les arbres de recherche HNSW ou les index GIST.

L’utilisation des KD-Trees (arbres k dimensionnels) est particulièrement répandue pour l’indexation de données vectorielles qui concerne des documents texte. Ces arbres permettent de partitionner l’espace en régions pour une recherche efficace du plus proche voisin ou des recherches par intervalle. Bien que les KD-Trees soient plus efficaces avec des données de faible dimensionnalité, ils peuvent être utilisés conjointement avec des techniques de réduction de dimensionnalité quand les données sont complexes.

Plusieurs frameworks et systèmes exploitent les KD-Trees pour l’indexation et la récupération efficaces de données spatiales. Scikit-learn, par exemple, est une bibliothèque Python populaire qui fournit des implémentations de KD-Trees. FLANN (pour Fast Library for Approximate Nearest Neighbors) et ANN (Approximate Nearest Neighbor) sont d’autres bibliothèques qui utilisent les KD-Trees pour l’indexation de données de haute dimension.

Bien qu’il ne soit pas une base de données vectorielle au sens strict, Elasticsearch peut utiliser les KD-Trees (en réalité, les BKD-Trees) pour indexer et rechercher des vecteurs de grande dimension à travers son type “dense\_vector”. Une autre possibilité open-source, PostgreSQL, utilise les R-Trees pour l’indexation de données spatiales. Bien que n’étant pas directement une base de données vectorielle, cela montre l’adaptabilité des structures d’arbres dans les tâches d’indexation.

Enfin, Faiss (pour Facebook AI Similarity Search) qui utilise principalement des méthodes basées sur la quantification pour indexer plutôt que des KD-Trees, fournit un ensemble d’outils pour une indexation efficace et un clustering de vecteurs denses.

Les KD-Trees sont des structures très intéressantes pour comprendre comment sont indexées les données dans une base de données vectorielle. Cependant, il est important de noter qu'à mesure que la dimensionnalité des données augmente, l'efficacité des KD-Trees diminue en raison du "curse of dimensionality". Dans de tels cas, d'autres structures de données ou méthodes d'indexation, telles que celles utilisées dans Faiss, peuvent offrir des solutions plus efficaces.

La phase de recherche quant à elle consiste à interroger la base de données pour récupérer les vecteurs les plus pertinents en fonction de la question posée. Le choix des vecteurs d'embeddings à retourner dépend de la similarité entre les vecteurs stockés et ceux générés à partir du contexte venant du module de chat. Plusieurs métriques peuvent être utilisées pour évaluer la distance entre les vecteurs, telles que la similarité cosinus, la distance de Manhattan (L1) ou la distance euclidienne (L2).

Dans le cadre du projet, il n'est clairement pas intéressant de réimplémenter tout cela. Il est donc préférable de se tourner vers des bases de données vectorielles qui offrent des solutions prêtes à l'emploi. ChromaDB, par exemple, utilise des techniques avancées de partitionnement de l'espace vectoriel pour une indexation efficace de grandes bases de données. Cet outil est optimisé pour des performances élevées avec un accent particulier sur le traitement en parallèle et la distribution des données. Il supporte une large variété de types de données vectorielles et vise une intégration haute performance. C'est de loin la solution la plus performante pour des bases de données de grande taille et pour une intégration rapide dans un projet.

Pgvector, une extension de PostgreSQL, repose sur des approches linéaires et des index GIST pour l'indexation de données vectorielles. Plus adaptée pour des applications basées sur PostgreSQL, elle offre une bonne intégration avec des recherches vectorielles directement depuis une base de données existante. La performance de Pgvector est substantielle et dépend de la configuration du serveur de base de données. Sa force est sa faculté à stocker des vecteurs numériques comme extension de types de données PostgreSQL, ce qui facilite l'intégration avec des applications existantes.

Qdrant, quant à lui, supporte plusieurs algorithmes, y compris les KD-Trees et HNSW pour l'indexation efficace de grandes bases de données vectorielles. C'est une solution conçue pour gérer efficacement des millions de vecteurs et offrir des temps de réponse rapides même avec de grandes bases de données. Cependant, la qualité des réponses dépendra beaucoup de l'implémentation qui peut s'avérer plus complexe que les autres solutions. Dans un autre registre, Redis possède une extension (Redisearch) supportant les vecteurs et possède un runtime extrêmement rapide en raison de son stockage en mémoire, idéal pour des opérations à faible latence. Limitée par la mémoire disponible, elle est très rapide pour des ensembles de données de taille modérée.

Il est important de noter que les bases de données vectorielles peuvent être exécutées localement, comme ChromaDB ou FAISS, mais aussi dans le Cloud, comme Pinecone ou Weaviate. Ces dernières offrent des solutions managées pour la gestion des données vectorielles, permettant de se concentrer sur le développement des applications plutôt que sur l'infrastructure sous-jacente.

En théorie, il est intéressant de noter que rien n'oblige à utiliser une base de données vectorielle pour construire un RAG. Cependant, il est clairement la norme dans le domaine du NLP vu la rapidité et l'efficacité de ces outils. Évidemment, cette technique ne garantit pas à 100% que le LLM réponde toujours bien, mais elle augmente considérablement les chances et permet de répondre à des questions qui n'auraient pas pu être répondues autrement.

### **c) Le module de chat**

Le module de chat est relativement équivalent à un chatbot classique. Il prend en input la question posée par l'utilisateur. Ces données que l'on appelle "contexte" subissent un pré-traitement pour être formatées de manière à être utilisées pour extraire des informations pertinentes de la base de données vectorielle. Une fois la phase de pré-traitement terminée, les données sont tokenisées puis transformées en embeddings pour être utilisées par le RAG.

Ce passage des tokens vers les embeddings, souvent appelé "incorporation", est réalisé à partir de modèles pré-entraînés comme Word2Vec ou GloVe, mais peut aussi être fait à l'aide de modèles de langage plus avancés comme BERT ou GPT évoqués plus tôt. Avec ces embeddings, le système peut identifier avec une haute précision les intentions derrière la question de l'utilisateur et extraire des informations pertinentes de la base de données vectorielle. Pour identifier quels tokens sont pertinents par rapport à la question posée, le système utilise des modèles de traitement du langage naturel (NLP) comme BERT ou ses variantes.

Ensuite, le système prépare une requête de recherche optimisée pour interroger la base de données vectorielle et trouver des documents ou des passages pertinents. Cette requête peut dépendre des spécificités de l'algorithme de recherche ou de la base de données utilisée mais retourne, dans tous les cas, des passages de texte qui sont ensuite récupérés par le module de chat pour être passés au LLM.

Pour garder un maximum de cohérence entre la question posée et les données extraites en lien, le module de chat doit être capable de formater correctement les données pour les passer au LLM. Cela peut impliquer de reformuler la question, de supprimer des informations inutiles ou de préciser au LLM comment traiter les données.



La plupart des modèles de langage communiquent avec des requêtes possédant des flags spécifiques pour indiquer comment traiter ces dernières. On parle alors de “system prompt” lorsque l’on veut indiquer au modèle une manière particulière de se comporter. Sinon, on parle de “user prompt” pour indiquer au modèle de se comporter de manière standard et de répondre aux questions posées.

Un exemple de system prompt pourrait être : “Tu es un assistant qui fait ceci, je vais te poser des questions sur cela, tu auras des passages à disposition pour t’aider à répondre, cite-les de cette manière, etc.”.

La requête est maintenant complète et peut être passée au LLM pour générer une réponse. Étant donné que le projet est un chatbot, il est nécessaire de conserver un historique des questions posées et des réponses générées pour garantir une conversation cohérente. Généralement, cela est gardé en mémoire vive et concaténé à chaque nouvelle question posée dans le contexte envoyé au LLM. Il faut donc réfléchir à toute une stratégie permettant de n’utiliser que la dernière question pour aller chercher des informations dans la base de données vectorielle et, dans le même temps, conserver l’historique des questions posées pour tout envoyer au LLM.

#### d) Le LLM

Le LLM est le cœur du système. C’est lui qui va générer les réponses à partir des informations contextuelles fournies par le module de chat. Ces dernières, envoyées de manière textuelle, sont généralement accompagnées de différents paramètres qui permettent de contrôler la qualité de la réponse. On retrouve notamment le “stop sequence” qui indique au modèle quand arrêter de générer du texte, la “température” qui contrôle le niveau de créativité ou d’imprévisibilité dans les réponses ou encore le “max tokens” qui détermine le nombre maximum de tokens que le modèle peut générer en réponse.

D’autres paramètres moins fréquents peuvent être utilisés pour affiner les interactions avec le modèle et obtenir des réponses qui correspondent mieux aux intentions ou aux besoins spécifiques de l’utilisateur. On peut citer le “presence penalty” qui réduit la probabilité de répéter des tokens qui sont déjà apparus dans la réponse, le “frequency penalty” qui réduit la probabilité de répéter des tokens qui sont apparus plusieurs fois dans la réponse, le “best of” qui permet de générer plusieurs réponses à un prompt et de sélectionner la meilleure selon certains critères ou encore le “beam search” qui explore plusieurs chemins de génération de texte en parallèle pour trouver une réponse plus optimale.

Il est important de démarrer chaque nouvelle conversation avec le LLM par une requête “system prompt” afin de rappeler au modèle comment il doit se comporter. Dans un second temps, toutes les requêtes futures peuvent être envoyées avec un “user prompt” pour indiquer au modèle de répondre de manière standard.

### 3.3. Reflexions sur l'architecture

Maintenant que les LLM et les RAG ont été présentés dans les détails, il serait intéressant de se poser les premières questions sur les choix architecturaux à adopter pour le projet.

#### a) Le LLM

Le premier choix très important est de déterminer quel LLM il serait judicieux d'utiliser. Il est clair que les modèles de la famille d'OpenAI ou d'Anthropic sont les plus performants pour générer du texte. Cependant, ce sont des solutions propriétaires qui peuvent s'avérer très coûteuses en fonction de l'utilisation qui en est faite. Il est donc important de bien réfléchir à l'impact financier que cela pourrait avoir sur le projet. D'autre part, des modèles open-source comme ceux proposés Mistral ou Meta peuvent tout autant répondre aux besoins du projet à condition de pouvoir héberger les modèles localement. Dans ce cas, on risque d'être limité en termes de performances, mais cela peut être un compromis acceptable.

Le gros problème qui se pose actuellement avec le choix des LLM à utiliser, c'est qu'il y en a de plus en plus et qu'ils sont de plus en plus performants. De nouveaux modèles sortent chaque jour et rendent les précédents souvent obsolètes. Pour ce projet, il est donc judicieux de choisir une architecture permettant de facilement changer de modèle. Pour des solutions propriétaires, on passe fréquemment par des API privées qui nécessitent une clé d'accès. Pour des solutions open source, on peut passer par des librairies Python comme HuggingFace ou Llama qui permettent de charger un type de modèle spécifique rapidement. Dans tous les cas, l'objectif reste le même, éviter toute dépendance à un LLM spécifique.

Dans une première phase exploratoire, il serait intéressant de comparer les performances de différents modèles open-source. Beaucoup de benchmarks et de métriques peuvent donner une idée de la performance d'un modèle sur une tâche spécifique. Un papier sur les récents avancements dans les RAG (1) propose un rapide état de l'art des LLM dans lequel il est fait mention de la complexité à trouver le bon modèle pour une tâche donnée.

Souvent, les benchmarks sont des questions de compréhension de texte ou de génération de texte qui ne sont pas toujours représentatives de la performance d'un modèle dans un contexte réel. Par exemple, les modèles Gemini de Google ont des scores très élevés en compréhension massive et multitâche du langage (MMLU). C'est un benchmark très prisé qui mesure la qualité d'un modèle multimodal. Cependant, il semblerait qu'en pratique ChatGPT-4 soit meilleur, mais cela reste presque impossible à prouver. La théorie la plus probable est que les tests aient fuité dans l'entraînement de Gemini mais ce n'est ni prouvé ni vérifiable, car c'est basé sur des ressentis, certes généralisés, mais pas suffisamment fiables pour être pris en compte.

En réalité, et même si beaucoup de métriques existent, il paraîtrait que rien ne soit aussi fiable qu'un classement réalisé sur les ressentis humains. C'est d'ailleurs ce que propose le ranking by elo de Hugging Face (11) dont voici un extrait daté du 15 Mars 2023 :

Model	Arena Elo	MT-bench	MMLU	License
<a href="#">GPT-4-Turbo</a>	1233	9.32		Proprietary
<a href="#">GPT-4-0314</a>	1191	8.96	86.4	Proprietary
<a href="#">GPT-4-0613</a>	1157	9.18		Proprietary
<a href="#">Claude-1</a>	1151	7.9	77	Proprietary
<a href="#">Claude-2.0</a>	1130	8.06	78.5	Proprietary
<a href="#">Claude-2.1</a>	1120	8.18		Proprietary
<a href="#">GPT-3.5-Turbo-0613</a>	1116	8.39		Proprietary
<a href="#">Mixtral-8x7b-Instruc</a>	1116	8.3	70.6	Apache 2.0
<a href="#">Claude-Instant-1</a>	1110	7.85	73.4	Proprietary
<a href="#">Tulu-2-DPO-70B</a>	1110	7.89		AI2 ImpACT Low-risk
<a href="#">Yi-34B-Chat</a>	1109		73.5	Yi License
<a href="#">Gemini Pro</a>	1106		71.8	Proprietary
<a href="#">GPT-3.5-Turbo-0314</a>	1105	7.94	70	Proprietary
<a href="#">WizardLM-70B-v1.0</a>	1102	7.71	63.7	Llama 2 Community
<a href="#">Vicuna-33B</a>	1096	7.12	59.2	Non-commercial
<a href="#">Starling-LM-7B-alpha</a>	1088	8.09	63.9	CC-BY-NC-4.0
<a href="#">OpenChat-3.5</a>	1077	7.81	64.3	Apache-2.0
<a href="#">GPT-3.5-Turbo-1106</a>	1077	8.32		Proprietary
<a href="#">pplx-70b-online</a>	1075			Proprietary
<a href="#">Llama-2-70b-chat</a>	1074	6.86	63	Llama 2 Community
<a href="#">OpenHermes-2.5-Mistr</a>	1072			Apache-2.0
<a href="#">Dolphin-2.2.1-Mistra</a>	1072			Apache-2.0

FIG. 3.2 : Classement des LLM par Elo sur Hugging Face

Source : *HuggingFace*, ref. URL03

Cette image montre le classement des meilleurs LLM par Elo. On remarque que les modèles GPT-4 sont clairement en tête, suivis de près par les modèles Claude. Mais ce qui est très intéressant, c'est de regarder les lignes marquées en jaune qui représentent les modèles open-source de Mistral. Mixtral-8x7B se positionne même en huitième position devant GPT-3.5, les modèles Llama de Meta ou encore Gemini de Google. Cela montre que les modèles open-source peuvent être tout aussi performants, voire meilleurs que les modèles propriétaires.

Cependant, ce classement ne tient pas compte de la taille des modèles. Il se trouve que la taille des modèles, donc leur nombre de paramètres, est un facteur très fortement lié à la performance d'un modèle et à la quantité de ressources nécessaires pour le faire tourner. L'immense avantage de Mistral est de proposer des modèles open-source avec des performances plus que remarquables tout en étant beaucoup plus petits que les modèles propriétaires. Par exemple, un modèle comme Llama-2 avec 70 milliards de paramètres nécessite au minimum deux cartes graphiques Nvidia 4080 Ti pour tourner.

Pour avoir des modèles aussi petits, mais aussi performants, Mistral fait des compromis sur le risque d'hallucination, comme le montrent les benchmarks BBQ et Bold qui mesurent un taux d'hallucination plus élevé pour les modèles 7B que pour les modèles 70B. Cela n'est cependant pas un problème puisque le RAG va permettre de compenser ce défaut.

Mistral propose essentiellement deux types de modèles qui peuvent être intéressants pour le projet : les modèles 7B et les Mixture of Experts. Ces derniers, dont Mixtral-8x7B peuvent être vus comme une combinaison de plusieurs modèles 7B qui sont spécialisés dans des tâches spécifiques. Cela permet d'avoir des performances très élevées sur une grande variété de tâches et c'est très probablement de cette manière que fonctionne GPT-4. Avec cette technique, au moment de la génération de texte, seulement quelques sous-modèles sont activés pour répondre à la question posée. En termes de ressources, cela revient à profiter des performances d'un modèle de 56 milliards de paramètres tout en ne nécessitant la puissance de calcul que d'un modèle de 14 milliards de paramètres.

En termes de performance, un énorme avantage des modèles de Mistral est l'extrême rapidité de génération de texte. Même si le modèle Mixtral-8x7B semble très performant, les ressources disponibles pour le projet ne permettent pas de déployer un modèle de plus de sept milliards de paramètres. Il peut donc être intéressant de se tourner vers un modèle plus léger comme OpenHermes-2.5 qui est un modèle 7B amélioré par la communauté Hugging Face. Il est très performant et peut être déployé sur un laptop ou un serveur sans problème.

## **b) Le RAG System**

Au regard des premières recherches effectuées, il semble que ChromaDB et Faiss soient les deux solutions de base de données vectorielles les plus adaptées au projet. Le choix déterminant va dépendre des performances en pratique. Il est important de mesurer la rapidité d'inférence des deux solutions en fonction de la quantité de documents à vectoriser. A priori, le choix de Faiss semble plus cohérent pour le projet, puisqu'il faut être à la fois rapide et performant.

ChromaDB qui est aussi une bonne solution, paraît plus adapté lorsque l'on possède des bases de données externes, ce qui ne sera pas le cas ici. Tout comme les LLM, il est très important de ne pas dépendre d'une solution spécifique, étant donné que les outils disponibles et les techniques d'embedding évoluent très rapidement. Pour s'adapter aux différents outils disponibles, il est nécessaire d'utiliser un framework qui intègre des alternatives comme Langchain. C'est de loin le framework le plus intéressant pour intégrer des LLM dans une application et faire du RAG. Comparé à Pinecone, par exemple, il intègre un ensemble de modèles d'embeddings pré-entraînés comme BERT, GPT, etc. Il permet aussi d'importer des LLM avec différentes librairies Python comme Ollama.

### **3.4. Des défis à relever**

À travers la prochaine section consacrée au travail réalisé, il sera question de répondre à plusieurs questions évoquées dans le cahier des charges. Premièrement, il peut être intéressant de s'attarder sur l'étude du comportement des RAG de manière générale.

Par exemple, lorsque le système ne connaît pas la réponse exacte à une question, comment va-t-il réagir ? Va-t-il donner, inventer une réponse à partir de ses connaissances générales, va-t-il dire qu'il ne sait pas ? Cela peut être très important pour la crédibilité du système. De plus, la qualité des documents à vectoriser peut avoir un impact important sur la pertinence des réponses. La question du contexte peut aussi être cruciale. Faut-il privilégier un contexte plus large ou plus précis ?

En réalité, certaines questions ne peuvent être répondues qu'en pratique, c'est pourquoi il est important de tester plusieurs architectures et de réaliser plusieurs itérations pour trouver la solution la plus adaptée au projet.

## Chapitre 4 :

# Réalisations

### 4.1. Description générale

Le but du projet est de mettre à disposition des étudiants un assistant virtuel capable de répondre à des questions précises sur le contenu des documents de la HEIA-FR. Dans le cadre de ce travail, l'objectif premier, est essentiellement de mettre en place une pipeline fonctionnelle qui servirait de base pour des développements futurs. Au vu du temps et des ressources disponibles, il est important de pouvoir dissocier le travail réalisable du travail complet.

#### a) Les besoins

Florence Meyer exprime un besoin fondamental de séparer les dépendances entre le service académique et le service informatique, en veillant à ce que l'application soit accessible même pour les utilisateurs sans compétences techniques. L'interface doit être simple et intuitive, facilitant la saisie des questions et l'obtention de réponses. Sur le plan technique, l'application doit être fiable, dotée de mécanismes de monitoring et de redéploiement automatique pour assurer une gestion autonome en cas de problème serveur. La gestion des documents sources est cruciale, permettant l'ajout, la vérification, l'indexation et le stockage sécurisé de divers formats (PDF, sites web, textes, JSON, CSV, etc.) dans une base de données vectorielle.

La transformation des documents Excel en texte brut ou CSV pour une meilleure indexation est également envisagée. Florence Meyer souhaite aussi développer des interfaces web et mobiles similaires en termes de fonctionnalité et de graphisme, adaptées aux contraintes spécifiques de chaque plateforme. La sélection des technologies devra être validée par des benchmarks rigoureux pour garantir des performances optimales, tant en termes de temps de réponse que de qualité.

Les réponses fournies doivent être pertinentes, bien citées, et non paraphrasées, avec un historique des interactions pour chaque utilisateur, permettant une gestion multi-utilisateurs avec différents rôles (étudiant, professeur, administrateur, etc.).

Un système de retour utilisateur simple (pouce en l'air, pouce en bas, étoiles) permettra de mesurer la satisfaction et d'améliorer continuellement les réponses du modèle. L'architecture doit rester flexible, permettant de modifier le choix des technologies sans affecter l'application. Enfin, les réponses doivent être de qualité et fournies rapidement, avec des citations précises des sources documentaires.

## **b) Les contraintes**

Le projet est soumis à des contraintes de temps et de ressources, limitant l'étendue des fonctionnalités à implémenter. La qualité des documents sources (horaires, fichiers Excel, sites web) pose un défi majeur, nécessitant une vérification rigoureuse pour garantir des réponses pertinentes et à jour. La séparation des dépendances entre les services académiques et informatiques doit être maintenue pour une gestion autonome par les utilisateurs non techniques. De plus, il est crucial de privilégier l'utilisation de services open-source, excluant l'utilisation des API de ChatGPT-4 ou de Gemini en production. L'application doit être conçue de manière à permettre une gestion flexible et indépendante des technologies utilisées (modèles LLM, algorithmes d'embedding, bases de données vectorielles), sans impacter son comportement global.

## **4.2. Description détaillée**

En supposant que le temps et les ressources mises à disposition du projet permettent de réaliser l'ensemble de l'application, il faudrait prévoir plusieurs interfaces (web et mobile) permettant aux étudiants d'interagir avec l'assistant virtuel. Il faudrait aussi prévoir une interface de monitoring, mise à disposition du service académique, pour gérer l'indexation des documents à utiliser par le RAG. Cette interface permettrait aussi de gérer le système de feedback à utiliser lorsque l'application n'est pas en mesure de répondre de manière précise, sourcée et pertinente à une question posée. Dans les cas où cette situation arriverait, il faudrait retourner cette question dans l'interface de monitoring afin qu'un expert puisse y répondre et que la réponse puisse être ajoutée à la base de données pour les prochaines fois.

En plus des interfaces et d'un point de vue plus technique, il faudrait se pencher sur le backend et l'architecture globale du projet. Une pipeline de déploiement et de redéploiement automatique serait nécessaire pour garantir une disponibilité continue de l'application. Cette dernière serait aussi utilisée pour mettre à jour l'application lorsque des documents sont indexés ou supprimés par le service académique.

Les choix technologiques cruciaux devraient être faits en fonction des performances et des benchmarks réalisés sur les différentes solutions disponibles. Il faudrait donc prévoir du temps à investir pour réaliser des batteries de tests et des benchmarks de qualité afin de s'assurer que les choix technologiques permettent d'atteindre les objectifs fixés.

Ne disposant pas de suffisamment de temps et de ressources pour réaliser l'ensemble de l'application, l'application proposée est essentiellement un prototype fonctionnel qui permet de répondre aux besoins de base exprimés par Florence Meyer. Cette dernière est alors composée d'un frontend web très simple, un backend qui gère le RAG et expose une API contenant toutes les fonctionnalités nécessaires pour répondre aux questions posées par les utilisateurs. Le choix des technologies et de l'architecture à déployer est optimisé pour la performance et la simplicité, sans nécessairement prendre en compte la scalabilité de l'application. L'objectif premier est essentiellement de garantir un temps de réponse très court et une qualité de réponse optimale.

Les choix technologiques, qui concernent essentiellement le choix de l'algorithme d'embedding et celui du LLM sont effectués en mesurant de manière relativement subjective la qualité des réponses. À partir d'une liste de quelques questions type, les réponses générées par les différentes combinaisons d'algorithmes, de LLM et de base de données vectorielles sont comparées pour déterminer laquelle offre le meilleur rapport qualité/performance.

Dans une version plus aboutie de l'application, il serait nécessaire de fixer la priorité sur un système de monitoring pour la gestion des documents. Les choix technologiques et ceux liés au déploiement automatique de l'application devraient être revus dans un second temps pour garantir une architecture optimale, mais ne constituent pas une réelle priorité pour le moment.

Enfin, et à mesure que le travail se poursuivrait, il serait nécessaire de se pencher sur la réalisation plus poussée des interfaces web et mobiles afin d'exposer l'application à un plus grand nombre d'utilisateurs.



### a) L'interface web

Dans le cadre de la réalisation propre à ce projet, le frontend réalisé pour interagir avec l'assistant virtuel est une application web très simple. Cette dernière est composée d'une page unique contenant un champ de texte pour saisir la question à poser à l'assistant virtuel et un bouton pour envoyer la question. Lorsque la question est envoyée, une requête est faite au backend pour obtenir la réponse. Cette dernière est ensuite affichée à l'utilisateur, accompagnée des sources utilisées pour générer la réponse. Voici un aperçu de l'interface réalisée en HTML/CSS/JS :

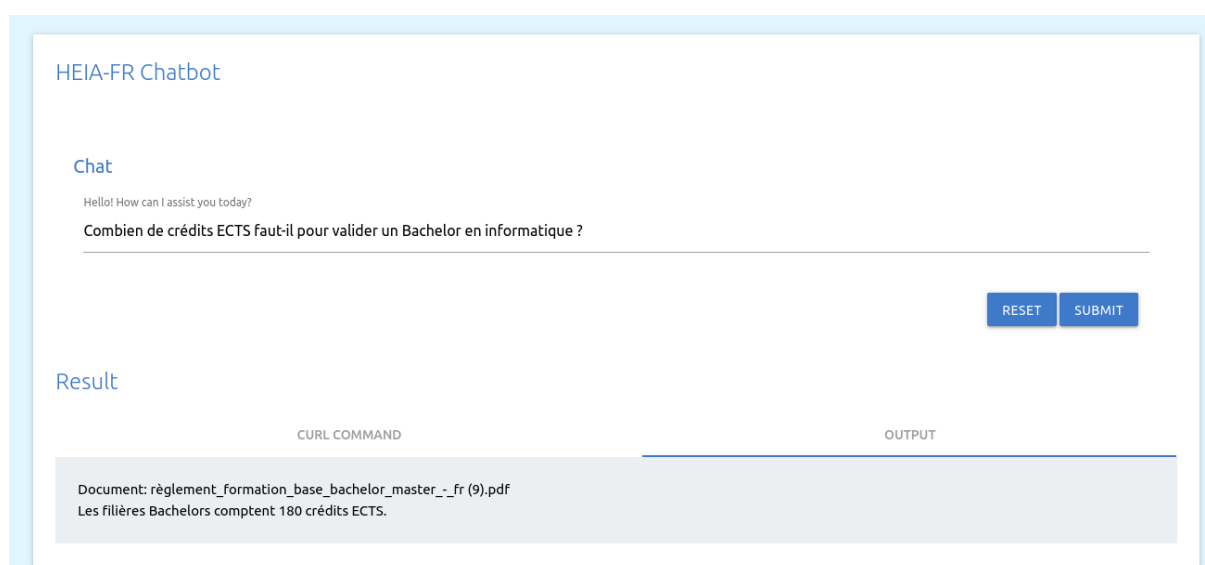


FIG. 4.1 : Interface du Chatbot

*Source : Thomas Dagier*

Cette image illustre une interaction très simple avec le chatbot et mets en situation un étudiant qui aurait des difficultés à se connecter à IS-Academia, un site web de l'école permettant de consulter les horaires des cours...

### b) Les embeddings et la base de données

Comme évoqué précédemment, plusieurs alternatives ont été envisagées pour l'embedding des documents et la base de données vectorielle à utiliser. Si le choix du framework se porte naturellement sur Langchain qui permet clairement l'intégration la plus simple et les meilleures performances, plusieurs algorithmes d'embedding ont été testés pour déterminer lequel offrait les meilleures performances. Le premier test a été d'utiliser E5 (12). Cette solution inspirée de SBERT possède l'avantage d'être très rapide et de fournir des embeddings de qualité. Une alternative à E5 est d'utiliser BGE-M3 (13). Proposé par la Beijing Academy of Artificial Intelligence, ce modèle est plus récent et offre des performances légèrement supérieures à E5.

Cependant, des discussions en interne ont fait émerger l'idée d'utiliser GPT-3 (14) pour l'embedding des documents. En effet, la quantité de données à traiter reste relativement faible et il semblerait que la qualité des embeddings obtenus soit supérieure à celle des autres solutions testées. Étant donné qu'il n'est pas nécessaire de faire plusieurs fois un embedding sur un même document, le choix d'une solution payante, mais à très moindre coût paraît être une option viable au regard de la qualité des embeddings obtenus. Pour la version actuelle du projet, cela représente un coût total de 0.16\$.

Pour juger la qualité des embeddings, il serait en théorie nécessaire de procéder à diverses mesures de satisfaction qui prendraient en compte le temps de traitement, la taille des embeddings une fois stockés et, surtout, la qualité des réponses obtenues. Florence Meyer mets à disposition une liste de plusieurs questions types qui permettent de comparer la qualité des réponses en fonction des embeddings et des bases de données vectorielles utilisées.

Il est important de noter que les paramètres des algorithmes d'embeddings sont identiques pour les trois solutions testées. On distingue la taille des chunks qui est de 200 caractères et l'overlap qui est de 20 caractères. Cela signifie qu'un nombre (l'embedding) sera associé à un groupe de maximum 200 caractères et que chaque groupe se chevauche de maximum 20 caractères avec le précédent et le suivant. Ce choix permet de garantir qu'un embedding soit fait sur des groupes de mots suffisamment grands pour faire du sens sémantiquement tout en garantissant une continuité avec les groupes environnant de sorte à garder la cohérence du texte complet. Ensuite, l'algorithme rentre en jeu avec toute une série d'étapes qui permettent de couper intelligemment le texte sur des mots entiers, des phrases entières ou des paragraphes distincts et de supprimer les caractères spéciaux, la mise en forme, et les autres éléments qui ne seraient pas pertinents pour le chatbot.

Voici un exemple des réponses obtenues à partir de la question "Combien de crédits ECTS dois-je valider pour obtenir mon Bachelor en Informatique?" :

- E5: Les filières Bachelors comptent 180 crédits ECTS et s'organisent sur 6 semestres pour la forme d'études à plein temps et sur 8 semestres pour la forme d'études en emploi. La durée de la formation à temps partiel est variable. La durée maximale de la formation est fixée dans les règlements de filière. Elle ne peut pas dépasser 12 semestres.

Source: règlement\_formation\_base\_bachelor\_master\_fr.pdf

- BGE-M3:

3 Les filières Bachelors comptent 180 crédits ECTS et s'organisent sur 6 semestres pour la forme d'études à plein temps et sur 8 semestres pour la forme d'études en emploi.

Source: règlement\_formation\_base\_bachelor\_master\_fr.pdf

- GPT-3: Les filières Bachelors comptent 180 crédits ECTS.

Source: règlement\_formation\_base\_bachelor\_master\_fr.pdf

Avec cet exemple, on observe les différentes réponses obtenues avec les embeddings E5, BGE-M3 et GPT-3. Visiblement et à travers l'exemple précédent, il semblerait que E5 ait plus de difficultés à créer des groupes de mots pertinents. Si on regarde le document et l'extrait exact qui a permis de répondre, il se trouve que c'est l'entièreté du paragraphe qui a été utilisée pour répondre à la question. Cela signifie que l'algorithme est moins capable de créer des groupes de mots pertinents et a donc groupé l'ensemble du texte, ce qui peut poser un problème lorsque la question posée est plus longue ou plus complexe.

D'un autre côté, BGE-M3 paraît offrir des résultats plus intéressants, mais ouvre la discussion sur un autre problème. Cet algorithme semble avoir quelques lacunes pour identifier le texte pertinent de celui qui l'est moins. On constate que l'index de l'alinéa dans le document est resté dans la réponse. Cela peut poser des problèmes au niveau de la qualité des sources externes utilisées par le LLM.

Enfin, GPT-3 semble offrir les résultats les plus pertinents et les plus précis sur cet exemple et sur bien d'autres. En effet, la réponse est plus concise et le contenu du texte est plus cohérent avec la question posée. De manière générale, ceci peut s'expliquer par le fait que GPT-3 est un modèle de génération de texte qui a été entraîné sur une quantité de données bien plus importante que les autres algorithmes testés.

Ces choix technologiques concernant les embeddings vont de pair avec le choix de la base de données vectorielle à utiliser. La section précédente, toujours sous l'angle de la simplicité d'implémentation, faisait mention de ChromaDB et Faiss. Ces deux solutions ont été testées à partir de Langchain mais n'ont pas permis d'identifier clairement laquelle offrait les meilleures performances.

En théorie, ChromaDB est une solution plus adaptée pour des sources de données variées comme des bases de données, des documents, des sites web, etc. Faiss, quant à lui, est plus rapide, mais plus complexe à mettre en place. Toujours en théorie, Faiss pourrait être une solution plus adaptée au projet dans la mesure où la quantité de données à stocker ne sera jamais très importante. En pratique, le choix de Faiss est plus adapté à ce projet car il se combine très bien avec l'intégration du LLM.

### c) Le Large Language Model

Dans la section sur les LLM, le choix de l'open source a été privilégié pour des raisons de coût et de simplicité. Même si l'adoption de GPT-3 comme algorithme d'embedding va quelque peu à l'encontre de l'architecture envisagée pour le projet, ce n'est pas le cas du LLM qui se doit d'être open source. En effet, il est crucial de pouvoir supporter une quantité variable, parfois importante de requêtes et le choix d'une solution propriétaire comme GPT-4 d'OpenAI pourrait poser de gros problèmes en termes de coûts.

Le second problème qui se pose avec le LLM c'est le fait que des nouveaux modèles sortent régulièrement et qu'il serait, en pratique, très important d'avoir toujours le meilleur modèle open source disponible. Cela signifie que le modèle doit être facilement interchangeable. Langchain intègre une librairie, Ollama qui permet de faire tourner des modèles LLM sur GPU en simulant un serveur avec une API REST. Maintenu très régulièrement à jour, Ollama est une solution qui permet de garantir une certaine pérennité dans le choix du LLM à utiliser.

La complexité derrière cette solution réside dans le fait de proposer une architecture plus complexe que celle envisagée initialement. En effet, il est nécessaire de mettre en place un module supplémentaire : un serveur Ollama qui va servir de pont entre le backend de l'application et le LLM. Cette alternative permet néanmoins de garantir que le modèle soit déployé en utilisant le GPU et donc que le temps de réponse soit beaucoup plus court (de l'ordre de 5 secondes par requête contre 30 secondes pour un modèle qui n'utilise pas de carte graphique pour fonctionner). Voici un exemple de requêtes permettant de communiquer avec le LLM à travers le Ollama serveur :

```
curl https://ollama.kube.isc.heia-fr.ch/api/pull -d '{
  "name": "mistral"
}'

curl https://ollama.kube.isc.heia-fr.ch/api/generate -d '{
  "model": "mistral",
  "prompt": "Combien de crédits ECTS faut-il pour valider
            un Bachelor en informatique ?",
  "stream": false
}'
```

À travers ces deux commandes, on spécifie le modèle que l'on souhaite déployer sur le GPU et on envoie une requête pour générer une réponse à partir d'un prompt donné. Ces requêtes sont envoyées au serveur Ollama qui les transmet au LLM avant de renvoyer la réponse obtenue. Le modèle utilisé actuellement est donc Mistral puisqu'il est le modèle le plus pertinent à utiliser dans ce cas.

### d) Le RAG

L'ensemble de l'architecture RAG qui compose cette application est finalement relativement simple. Une question posée par l'utilisateur depuis l'interface web servira à extraire des informations pertinentes depuis la base de données vectorielle. La pertinence de ces dernières est évaluée à partir d'algorithmes d'embeddings qui permettent de mesurer la similarité entre la question posée et les passages des différents documents indexés. Ces informations sont utilisées pour compléter un user prompt qui est envoyé au LLM à travers une requête API vers le Ollama serveur.

Le comportement du LLM est fixé à partir d'un prompt system afin de garantir que le modèle se comporte comme attendu. Dès lors que la réponse est générée, elle est renvoyée à l'utilisateur avec les sources utilisées pour être affichées dans l'interface web initiale.

Au fur et à mesure que les RAG évoluent et deviennent une norme relative à la personnalisation des LLM, les différents papiers s'accordent à classifier les RAG selon leur degré de complexité architecturale. D'après (15), il y aurait en réalité trois catégories de RAG distinctes dont le Naive RAG, le Advanced RAG et le Modular RAG :

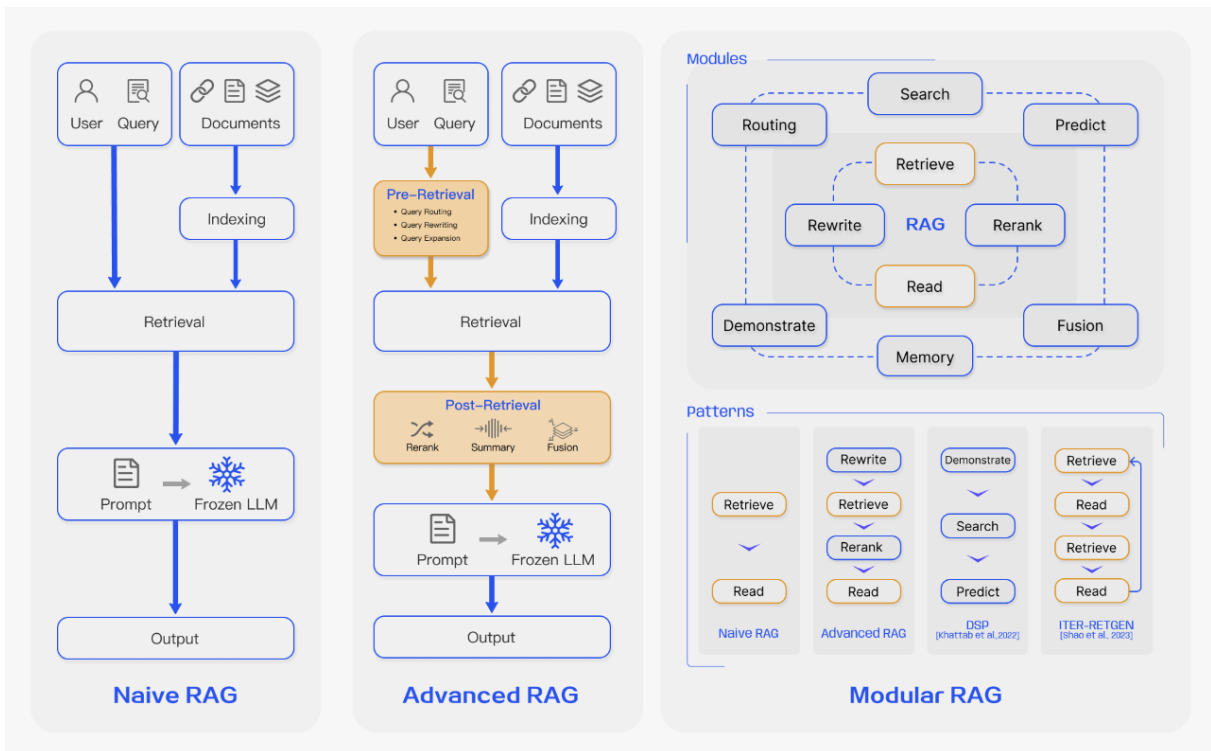


FIG. 4.2 : Différentes catégories de RAG

Source : *Retrieval-Augmented Generation for Large Language Models : A Survey*, ref. URL08

Ce schéma permet de rendre compte des différentes étapes qui composent chaque catégorie de RAG et met en lumière la notion de procédé de pre-retrieval et post-retrieval. Ces stratégies n'existant pas dans le Naive RAG répondent à une problématique du LLM qui ne se base plus que sur les données externes pour générer une réponse et non plus sur ses connaissances générales. On parle de Naive RAG lorsque le système ne fait que répéter ce qui lui est inscrit dans son user prompt sans chercher à générer une réponse plus complète.

En réalité, l'utilisation d'un système prompt est, déjà en soi, une forme de pre-retrieval qui permet de guider le modèle dans la génération de la réponse. Pour réaliser de l'Advanced RAG au sens propre, plusieurs techniques peuvent être mises en œuvre comme la réécriture de la question initiale pour matcher les résultats de la recherche sur plusieurs questions très similaires.

En matière de post-retrieval, il est possible de mettre en place des stratégies de reformulation ou de ré-indexation des chunks de texte pour éviter la surcharge d'informations (on parle alors de reclassement).

L'application développée dans le cadre de ce projet pourrait être vue entre le Naive RAG et l'Advanced RAG. En effet, le système de pre-retrieval est déjà en place et permet de guider le modèle dans la génération de la réponse. Cependant, il n'y a pas de stratégie de post-retrieval pour reformuler la réponse ou indexer à nouveau les chunks de texte.

### e) Les problèmes rencontrés

Au cours de ce projet, plusieurs problèmes ont été rencontrés. Le premier concerne la lenteur des réponses obtenues avec le LLM. En effet, le modèle est très gourmand en ressources et nécessite d'être déployé sur un GPU pour garantir des temps de réponse acceptables. Le déploiement du modèle sur le cluster de l'école a posé un problème en raison du manque de RAM GPU disponible. Il a donc été nécessaire de déployer intelligemment le modèle sur un serveur Kubernetes pour garantir des temps de réponse acceptables. C'est la raison principale pour laquelle le serveur Ollama a été mis en place afin de pouvoir être utilisé par plusieurs projets en même temps.

Un second problème, plus propre aux embeddings concerne la librairie PDFMiner et plus généralement la qualité des documents sources. En effet, les documents sources utilisés pour générer les embeddings ne sont pas toujours de qualité optimale. Certains documents PDF contiennent des emplois du temps ou des tableaux qui peuvent poser un problème à PDFMiner et donc à la génération des embeddings. Certains documents sont presque inexploitable et nécessitent un nettoyage et une préparation des données pour garantir des réponses de qualité. Le problème a été évoqué avec Florence Meyer à qui il a été recommandé de mettre en place des stratégies afin de remplacer les emplois du temps et les tableaux par des documents textes, CSV, ou JSON plus explicites.

Dans une version plus aboutie de l'application, il serait intéressant de rajouter des systèmes de nettoyage et de filtrage des données plus précis pour se rapprocher encore plus d'un Advanced RAG. Si une première étape serait de transformer les documents difficilement exploitables en fichier texte ou autre structure de données plus simple, il pourrait aussi s'avérer judicieux de réfléchir à d'autres techniques permettant, par exemple, de filtrer les données en amont de l'embedding pour ne garder que les informations les plus pertinentes.

De manière générale, la question de la lecture et de l'interprétation des tableaux est un sujet très ouvert et très complexe dans le domaine du NLP. La solution la plus simple est donc de faire une copie des documents problématiques et de les transformer en documents textes. Cela permet de garantir que les embeddings générés puissent réellement aider à répondre aux questions comme : "Dans quelle salle se déroule mon cours de mathématiques ?".

Heureusement, cela concerne une petite quantité de documents et il a été possible de trouver une alternative rapidement afin de tout de même bénéficier des informations présentes dans ces derniers.

Dès lors qu'une version stable et fonctionnelle de l'application a pu être déployée, il a été nécessaire de réaliser du prompt engineering pour améliorer la qualité des réponses. En effet, il est apparu que le modèle avait parfois du mal à générer des réponses pertinentes, qu'il paraphrasait les sources ou qu'il ne répondait pas correctement à la question posée.

Des stratégies de reformulation itérative ont donc été mises en place pour guider le modèle dans la génération de la réponse. Il se trouve que ces stratégies ont permis d'améliorer significativement la qualité des réponses obtenues.

Elles ouvrent même la discussion sur la possibilité de mettre davantage l'accent sur le prompt engineering et de se demander à quel point le système pourrait encore être amélioré uniquement en travaillant sur le system prompt. Typiquement, une question assez intéressante à se poser, c'est d'évaluer le comportement qu'aurait le LLM lorsqu'il n'a pas la réponse à une question. Est-ce qu'il devrait être capable de demander des précisions à l'utilisateur ? Est-ce qu'il devrait être en mesure de répondre n'importe quoi malgré tout ou plutôt dire explicitement qu'il ne connaît pas la réponse ?

Avec un system prompt quelque peu bien construit, il est assez simple de beaucoup influencer le comportement du modèle et de le guider dans la génération de la réponse. On peut, par exemple, influencer son comportement lorsqu'il ne connaît pas la réponse, quand il est nécessaire d'utiliser ses connaissances générales, comment éviter la paraphrase tout en citant au mieux les sources, etc. Voici l'exemple qui est utilisé pour le projet :

Tu as à ta disposition des informations et des réglementations sur la haute école d'ingénierie et d'architecture de Fribourg en Suisse. Si la question porte sur un ou plusieurs passages textuels, utilisez-les pour répondre à la question mais ne les paraphrase pas. Si la question ne porte sur aucun passage textuel, générez une réponse à partir de tes connaissances. Assure-toi de toujours fournir la réponse, la source de l'information et de mentionner dans quel fichier l'information a été trouvée en utilisant le format suivant:

Document: <document name>

<answer>

# Start of passages

{passages}

# End of passages

#### f) Les réponses aux questions ouvertes

La première question à laquelle on pourrait tenter de répondre, c'est d'évaluer les avantages et inconvénients d'une solution moins qualitative du point de vue de l'application déployée, mais qui fonctionnerait entièrement sur un ordinateur portable plutôt qu'un serveur Kubernetes avec des GPU. En pratique, l'application se veut portable et il est tout à fait possible de faire fonctionner le service en local.

Évidemment, et c'est ce qui motive l'utilisation du Ollama serveur, les performances sont beaucoup plus élevées lorsque les poids du LLM sont chargés dans la RAM GPU. Une utilisation relativement simple et peu efficace pourrait toutefois être faite sur un ordinateur portable ne possédant pas de GPU. Toujours en considérant que l'ensemble de l'application serait en local sur un ordinateur portable, la seule vraie limite serait la taille du LLM à charger sur le GPU. En effet, si le modèle était relativement lourd, l'utilisateur serait contraint d'utiliser un modèle plus petit comme Phi3 de Microsoft.

Une autre question qui s'est posée au moment d'intégrer les données externes au contexte du LLM, c'est de se demander si le choix d'un LLM avec plus de contexte aurait un impact sur la qualité des réponses. Relativement aux besoins de l'application, on pourrait estimer que ce n'est pas un critère déterminant, car on fait du RAG avec des chunks de petite taille. Augmenter le contexte pourrait aussi avoir un effet négatif sur la précision des réponses, alors qu'il serait plutôt recommandé de travailler avec des prompts courts, mais concis. De plus, les modèles avec beaucoup de contexte sont souvent plus lourds et plus longs à charger, ce qui pourrait poser des problèmes de performances sur le Ollama serveur. C'est donc un bon axe de réflexion, mais qui n'est pas primordial pour la suite.



## Conclusion

Réalisé dans le cadre du Master de la HES-SO, ce projet aura été l'occasion d'en apprendre davantage sur une technologie émergente et très prometteuse. Les RAG viennent répondre à un besoin crucial dans le domaine du traitement automatique du langage naturel NLP. La nécessité de pouvoir faire confiance aux LLM mais également la volonté de pouvoir personnaliser leurs utilisations font des RAG, une alternative très intéressante et à moindre coût.

A travers une étude poussée sur le fonctionnement des LLM et sur les différents modules qui composent le RAG System, beaucoup de notions abordées ont servi à réaliser un prototype de chatbot fonctionnel. Dans une version plus aboutie du projet, les étudiants de la HEIA-FR pourraient effectivement trouver réponses à leurs questions grâce au RAG mis en place.

Si ce projet se prête particulièrement bien à l'implémentation d'un RAG on pourrait tout de même se demander dans quel cas de figure l'utilisation d'un tel système serait vraiment nécessaire. Dans la majeure partie des cas, il se pourrait qu'un LLM relativement performant comme ChatGPT-4 suffise largement. En revanche, lorsqu'il y a besoin de traiter et d'analyser des documents en grande quantité, il est clair que le RAG est la solution la plus adaptée.

En somme, la vraie réponse à cette question serait avant tout de se questionner sur les ressources disponibles (argent, temps, données, infrastructure, etc.) et sur les besoins réels de l'utilisateur. Si le besoin est spécifique et que les ressources sont disponibles, alors il est clair que le RAG est une solution très intéressante. Dans le cas contraire, il est fort probable que l'utilisation d'un LLM suffise amplement.

Au-delà d'un chatbot relativement simple, l'architecture mise en place pour ce projet pourrait être très facilement étendue à d'autres applications sous réserve d'avoir assez de documents à traiter. De plus en plus, les systèmes d' IA génératives prennent une place très importante dans notre quotidien. Comprendre le fonctionnement des LLM et des RAG, c'est avant tout comprendre comment utiliser judicieusement les outils de demain et en tirer le meilleur parti.

## Chapitre 5 :

### Annexes

#### 5.1. Chatbot

This microservice, produced by Thomas Dagier for Master MSE PA aims to expose a chatbot for HEIA-FR students. The chatbot is available [here](#).

See report for further informations about the application and the technologies used.

The specification file is defined according to openapi v3 (OAS3).

#### 5.2. Running the web service - linux / mac

The web api is in the folder `/api_rest/`. I used uvicorn and fastapi for the development. You may use an own Python virtual environment in the `/api_rest/` folder installing the python modules from `/api_rest/requirements.txt`.

- Tools should work with python3. They were used with **python 3.7**, **python 3.8** and **python 3.9**.
- Clone the repository, create virtualenv, activate the virtual env, install required packages and test.

```
python3 -m venv venv
source ./venv/bin/activate
pip install --upgrade pip
pip install -r ./api_rest/requirements.txt
export OPENAI_API_KEY="YOUR_API_KEY"
uvicorn api_rest.main:rootapp --reload
```

The OpenAPI specifications are available under the route `/specification` and the Swagger interface to test the API under the route `/docs`. The showcase is under the route `/showcase`.

### 5.3. How to launch the tool with docker image on Linux

Make sure docker is installed. Follow instructions here to install docker.

```
docker build -t registry.forge.hefr.ch/thomas.dagierjo/pa-chatbot/  
pa-chatbot:latest .  
docker run -it -p 8000:8000 registry.forge.hefr.ch/thomas.dagierjo/pa-chatbot/  
pa-chatbot:latest
```

To quit the docker image, use `exit`

### 5.4. How to deploy on Kubernetes

Preliminary steps : you need to check that you are connected to the K8S iCoSys cluster with a `kubectl version --short`. For that you need to create a file in your path `~/.kube/config` with the cluster Kubeconfig file that you find under your cluster on <https://rancher.kube.isc.heia-fr.ch/>.

You need to create a Docker images that is then pushed on Gitlab Container Registry, our repository of images. Then scripts are used to let Kubernetes deploy a service from the pushed image stored on Gitlab Container Registry.

#### a) Dockerisation on Gitlab Container Registry

One important step is to build the Docker image with a proper tag. The parts of the tag will be used by the deployment scripts. In our case, the tag is : `registry.forge.hefr.ch/thomas.dagierjo/pa-chatbot/pa-chatbot:latest`

Where the first part defines the container registry (`registry.forge.hefr.ch`), the second part defines the namespace (`thomas.dagierjo/pa-chatbot`), the third part defines the image name (`pa-chatbot:latest`) and the last part the version number that will be incremented when new versions of the langid will be done (`dev`). So the building of the image is done with `docker build -t registry.forge.hefr.ch/thomas.dagierjo/pa-chatbot/pa-chatbot:latest ..`

You need to login to Gitlab Container Registry from Docker so that when you do a docker push, it is pushed on Gitlab Container Registry. The login is done with `docker login registry.forge.hefr.ch`. The push is then done with `docker push registry.forge.hefr.ch/thomas.dagierjo/pa-chatbot/pa-chatbot:latest` After that your image should be visible here.

## **b) Deployment on K8S**

To let K8S pull these images, you should create a secret in the cluster with your credentials to reach the image. I did it with `kubectl -n ragfish create secret docker-registry my-secret --docker-server=registry.forge.hefr.ch --docker-username=XXX --docker-password=XXX`.

The XXX docker-password must be changed with the token created on Gitlab Container Registry. You create it here. You need to check the `read_registry` box at least.

The first time you deploy on K8S, you need to `kubectl apply` the config files like that :

```
kubectl apply -f api_rest/deploy.yaml
```

After this, you may want to have a look in the Rancher web interface to make sure everything is correctly deployed and running.

For any issue, please refer to the following documentation.

## **c) Deployment on K8S with Gitlab CI**

A CI/CD pipeline is set up within Gitlab's project to facilitate the building of a Docker image and its deployment on Kubernetes (K8S). Our development workflow involves two environments : a dev environment for working on new and revised software code, and a staging environment for deploying software into production. You can access the configuration details here.

## **5.5. Using the service**

When the deployment is done, you can access the service here

## 5.6. Ollama Server Deployment

Done by Thomas Dagier & Sam Corpataux in May 2024.

### a) Script to deploy ollama on kubernetes

First, make sure you configured your kubectl to point to the right cluster. Config should be in ~/.kube/config or you can use `kubectl config use-context <context>` to switch context.

### b) Using kubectl

To deploy the application, run the following command :

```
kubectl apply -f deploy.yaml
```

### c) Testing the deployment

First, you'll need to pull a model :

```
curl https://ollama.kube.isc.heia-fr.ch/api/pull -d '{
  "name": "mistral"
}'
```

Then, you can generate text using the following command :

```
curl https://ollama.kube.isc.heia-fr.ch/api/generate -d '{
  "model": "mistral",
  "prompt": "Why is the sky blue?",
  "stream": false
}'
```

### d) Monitoring the deployment

To check the GPU RAM available, you can run the following command :

```
ssh checkgpu@icolab-gpu-6.isc.heia-fr.ch
ssh checkgpu@icolab-gpu-10.isc.heia-fr.ch
```

Warning : This will show how much RAM is not used, not how much is available for reservation. This is why, if the command shows 14000 Mib are available, you can't reserve 55\*256 Mib during the deployment.

### e) Opened issues

- ☒ Issue with nginx, we need to switch port manually and randomly

- ❑ Not enough GPU RAM (max tencent.com/vcuda-memory : 20 so  $20 \times 256 = 5120$ ) to hold the weights of Llama3 for example
- ❑ Not that fast (however, Phi3 is faster than Llama3)

## f) Documentation

- K8s ISC Cluster Documentation
- Ollama Helm Chart
- An example to serve Ollama on Kubernetes cluster

## g) Deployment script

```
# SERVICE
---
apiVersion: v1
kind: Service
metadata:
  name: ollama
  namespace: ollama-namespace
spec:
  type: ClusterIP
  selector:
    name: ollama
  ports:
    - protocol: TCP
      port: 80
      targetPort: 11434

# DEPLOY
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ollama
  namespace: ollama-namespace
spec:
  selector:
    matchLabels:
      name: ollama
  template:
    metadata:
```

```
    labels:
      name: ollama
  spec:
    affinity:
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
            - matchExpressions:
                - key: kubernetes.io/hostname
                  operator: In
                  values:
                    - icolab-gpu-10
    containers:
      - name: ollama
        image: ollama/ollama:latest
        ports:
          - name: http
            containerPort: 11434
            protocol: TCP
        resources:
          requests:
            tencent.com/vcuda-core: 10 # 10 % of CUDA cores
            tencent.com/vcuda-memory: 20 # 20 * 256MiB = 5120MiB
          limits:
            tencent.com/vcuda-core: 10
            tencent.com/vcuda-memory: 20

# INGRESS
---
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ollama-ingress
  annotations:
    nginx.ingress.kubernetes.io/use-regex: 'true'
    nginx.ingress.kubernetes.io/rewrite-target: /$1
    nginx.ingress.kubernetes.io/ssl-redirect: 'true'
    nginx.ingress.kubernetes.io/proxy-body-size: 20m
```

```
    nginx.ingress.kubernetes.io/proxy-connect-timeout: '3600'
    nginx.ingress.kubernetes.io/proxy-read-timeout: '3600'
    nginx.ingress.kubernetes.io/proxy-send-timeout: '3600'
  namespace: ollama-namespace
spec:
  ingressClassName: nginx
  tls:
    - hosts:
        - ollama.kube.isc.heia-fr.ch
      secretName: tls-secret
  rules:
    - host: ollama.kube.isc.heia-fr.ch
      http:
        paths:
          - path: /(.* )
            pathType: ImplementationSpecific
            backend:
              service:
                name: ollama
                port:
                  number: 11434
```



## Références documentaires

1. CAI, Deng, WANG, Yan, LIU, Lemao et SHI, Shuming. Recent Advances in Retrieval-Augmented Text Generation. In : *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval* [en ligne]. New York, NY, USA : Association for Computing Machinery, juillet 2022. pp. 3417-3419. [Consulté le 18 mars 2024]. SIGIR '22. ISBN 9781450387323. Disponible à l'adresse : <https://doi.org/10.1145/3477495.3532682>
2. KARPATHY, Andrej. *Let's build the GPT Tokenizer* [en ligne]. [Consulté le 18 mars 2024]. Disponible à l'adresse : <https://www.youtube.com/watch?v=zduSFxRajkE>
3. ALI, Mehdi, FROMM, Michael, THELLMANN, Klaudia, RUTMANN, Richard, LÜBBERING, Max, LEVELING, Johannes, KLUG, Katrin, EBERT, Jan, DOLL, Niclas, BUSCHHOFF, Jasper Schulze, JAIN, Charvi, WEBER, Alexander Arno, JURKSCHAT, Lena, ABDELWAHAB, Hammam, JOHN, Chelsea, SUAREZ, Pedro Ortiz, OSTENDORFF, Malte, WEINBACH, Samuel, SIFA, Rafet, KESSELHEIM, Stefan et FLORES-HERR, Nicolas. *Tokenizer Choice For LLM Training : Negligible or Crucial ?* [en ligne]. octobre 2023. arXiv. [Consulté le 18 mars 2024]. Disponible à l'adresse : <http://arxiv.org/abs/2310.08754>  
arXiv :2310.08754 [cs]
4. WILT, Christopher, THAYER, Jordan et RUMMLER, Wheeler. A Comparison of Greedy Search Algorithms. *Proceedings of the International Symposium on Combinatorial Search* [en ligne]. août 2010. Vol. 1, n° 1, pp. 129-136. [Consulté le 18 mars 2024]. DOI 10.1609/socs.v1i1.18182. Disponible à l'adresse : <https://ojs.aaai.org/index.php/SOCS/article/view/18182>
5. WELLECK, Sean, KULIKOV, Ilia, ROLLER, Stephen, DINAN, Emily, CHO, Kyunghyun et WESTON, Jason. *Neural Text Generation with Unlikelihood Training* [en ligne]. septembre 2019. arXiv. [Consulté le 18 mars 2024]. Disponible à l'adresse : <http://arxiv.org/abs/1908.04319>  
arXiv :1908.04319 [cs, stat]

6. *Holistic Evaluation of Language Models (HELM)* [en ligne]. [Consulté le 22 mars 2024]. Disponible à l'adresse : <https://crfm.stanford.edu/helm/latest/#/leaderboard>
7. NAVEED, Humza, KHAN, Asad Ullah, QIU, Shi, SAQIB, Muhammad, ANWAR, Saeed, USMAN, Muhammad, AKHTAR, Naveed, BARNES, Nick et MIAN, Ajmal. *A Comprehensive Overview of Large Language Models* [en ligne]. février 2024. arXiv. [Consulté le 22 mars 2024]. Disponible à l'adresse : <http://arxiv.org/abs/2307.06435>  
arXiv :2307.06435 [cs]
8. *Open LLM Leaderboard - a Hugging Face Space by HuggingFaceH4* [en ligne]. [Consulté le 22 mars 2024]. Disponible à l'adresse : [https://huggingface.co/spaces/HuggingFaceH4/open\\_llm\\_leaderboard](https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard)
9. LEWIS, Patrick, PEREZ, Ethan, PIKTUS, Aleksandra, PETRONI, Fabio, KARPUKHIN, Vladimir, GOYAL, Naman, KÜTTLER, Heinrich, LEWIS, Mike, YIH, Wen-tau, ROCKTÄSCHEL, Tim, RIEDEL, Sebastian et KIELA, Douwe. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks* [en ligne]. avril 2021. arXiv. [Consulté le 25 mars 2024]. Disponible à l'adresse : <http://arxiv.org/abs/2005.11401>  
arXiv :2005.11401 [cs]
10. PENG, Bowen, QUESNELLE, Jeffrey, FAN, Honglu et SHIPPOLE, Enrico. *YaRN : Efficient Context Window Extension of Large Language Models* [en ligne]. novembre 2023. arXiv. [Consulté le 29 mars 2024]. Disponible à l'adresse : <http://arxiv.org/abs/2309.00071>  
arXiv :2309.00071 [cs]
11. *LMSys Chatbot Arena Leaderboard - a Hugging Face Space by lmsys* [en ligne]. [Consulté le 8 avril 2024]. Disponible à l'adresse : <https://huggingface.co/spaces/lmsys/chatbot-arena-leaderboard>
12. NEELAKANTAN, Arvind, XU, Tao, PURI, Raul, RADFORD, Alec, HAN, Jesse Michael, TWOREK, Jerry, YUAN, Qiming, TEZAK, Nikolas, KIM, Jong Wook, HALLACY, Chris, HEIDECHE, Johannes, SHYAM, Pranav, POWER, Boris, NEKOUL, Tyna Eloundou, SASTRY, Girish, KRUEGER, Gretchen, SCHNURR, David, SUCH, Felipe Petroski, HSU, Kenny, THOMPSON, Madeleine, KHAN, Tabarak, SHERBAKOV, Toki, JANG, Joanne, WELINDER, Peter et WENG, Lilian. *Text and Code Embeddings by Contrastive Pre-Training* [en ligne]. janvier 2022. arXiv. [Consulté le 27 mai 2024]. Disponible à l'adresse : <http://arxiv.org/abs/2201.10005>  
arXiv :2201.10005 [cs]

13. CHEN, Jianlv, XIAO, Shitao, ZHANG, Peitian, LUO, Kun, LIAN, Defu et LIU, Zheng. *BGE M3-Embedding : Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation* [en ligne]. février 2024. arXiv. [Consulté le 27 mai 2024]. Disponible à l'adresse : <http://arxiv.org/abs/2402.03216>  
arXiv :2402.03216 [cs]
14. WANG, Liang, YANG, Nan, HUANG, Xiaolong, JIAO, Binxing, YANG, Linjun, JIANG, Daxin, MAJUMDER, Rangan et WEI, Furu. *Text Embeddings by Weakly-Supervised Contrastive Pre-training* [en ligne]. février 2024. arXiv. [Consulté le 27 mai 2024]. Disponible à l'adresse : <http://arxiv.org/abs/2212.03533>  
arXiv :2212.03533 [cs]
15. GAO, Yunfan, XIONG, Yun, GAO, Xinyu, JIA, Kangxiang, PAN, Jinliu, BI, Yuxi, DAI, Yi, SUN, Jiawei, WANG, Meng et WANG, Haofen. *Retrieval-Augmented Generation for Large Language Models : A Survey* [en ligne]. mars 2024. arXiv. [Consulté le 27 mai 2024]. Disponible à l'adresse : <http://arxiv.org/abs/2312.10997>  
arXiv :2312.10997 [cs]