1 Motivation

Ce projet est né d'un double intérêt. D'une part, j'ai l'opportunité de le présenter dans le cadre du processus de recrutement de l'**Intelligence Lab** de l'ECE, d'autre part, sachant pas mal de théories suivies en autodidacte, j'avais une forte envie d'explorer le RAG (Retrieval-Augmented Generation) lors d'un petit projet.

Le projet a duré à peu près une semaine (par hasard la semaine la plus chargé de mon emploi du temps haha), j'ai ainsi commencé petit à petit mais sans but précis juste de *voir* que ça fonctionnait, mais je n'avais pas de *use-case* intéressant donc c'était vite ennuyant.

C'est après avoir participé au forum carrière de l'ECE que j'ai trouvé le use case parfait : l'analyse et la sélection de CVs. J'ai réalisé à quel point le tri et la recherche d'informations dans une pile de CVs sont des tâches répétitives et chronophages pour les recruteurs. Un système RAG pouvait précisément résoudre ce problème en permettant de "questionner" directement une base de CVs en langage naturel. (Ca existe déjà mais c'est intéressant de le coder à la main !)

2 Fonctionnement

Split : Avant toutes choses on veut découper nos CVs en plusieurs partie un peu comme des sortes de *tokens* pour ensuite les transformer en vecteur. Pour garder une part de cohérence entre les différents splits, on va les superposer (overlap). Mais attention, plus on découpe plus on perd en contexte mais ça va plus vite, et inversement. On va voir dans ce projet que c'est toute une question de compromis si j'ai envie de laisser ce RAG sur ma machine (j'ai un petit ordi portable seulement calculant sur CPU)

Embedding : il faut savoir qu'en langage naturel on vectorise tout pour pouvoir les manipuler comme on le souhaite dans les réseaux de neurones. Je vais alors découper les CVs en *chunks* puis ensuite les former sous forme de vecteur. Dans mon cas j'ai utiliser principalement all-mpnet-base-v2 et paraphrase-multilingual-MiniLM-L12-v2. Ainsi je vais avoir un vecteur de cette forme par exemple :

$$\mathrm{V} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{382} \\ v_{383} \\ v_{384} \end{pmatrix} = \begin{pmatrix} 0.0537 \\ -0.1290 \\ 0.0114 \\ \vdots \\ -0.0882 \\ 0.0451 \\ -0.0039 \end{pmatrix} \tag{1}$$

FAISS : La partie la plus importante à mon sens d'un RAG, c'est la recherche d'informations dans un document. Et c'est particulièrement cette partie qui va s'en charger. Le FAISS (Facebook AI Semantic Search) c'est une lib créée par Meta (idée d'un chercheur français à Meta !) qui va permettre de faire correspondre de manière ultra rapide deux types de vecteurs très proches. La rapidité et l'invention de FAISS se fait dans une stockage de vecteur qui est alors indexé. Sans trop rentrer dans les détails (je ne suis pas non plus expert), la similarité va se faire grâce à l'embedding dans l'espace et on va chercher la similarité de cosinus. Une photo qui explique bien ce schéma est la suivante :

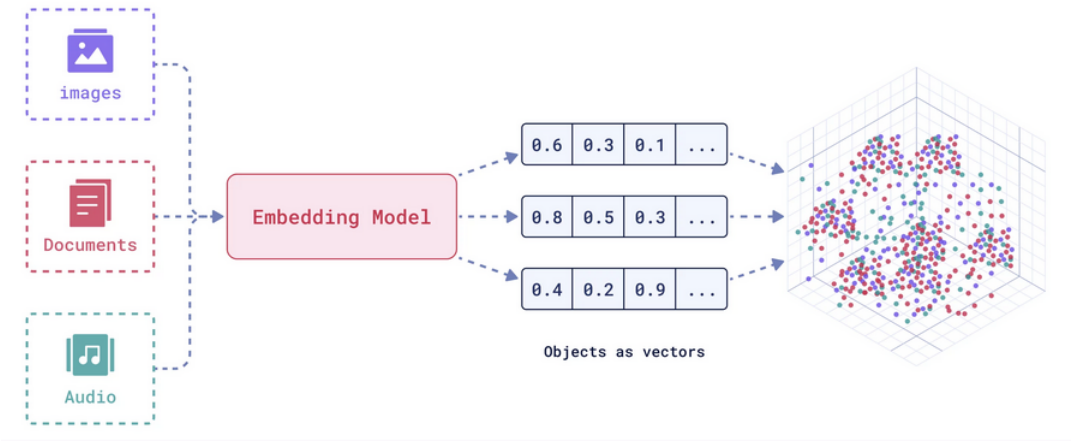

Figure 1: Processus pour embedding, on voit que la recherche de similarité se fait dans l'espace d'où la similarité de cosinus. Photo prise sur ce site

Modèle de langage : Cette partie va seulement permettre à générer de réponse cohérente sous forme de langage naturel. Dans mon cas vu la puissance de mon PC j'utilise principalement 2 familles de modèles : **phi3** et **mistral** et bien évidemment les plus petites versions. J'utilise Ollama pour permettre d'avoir ces modèles sur ma machine.

Gradio : J'ai mis sur Gradio pour que ça soit plus sympas visuellement pour un prototype. Gradio fonctionne ainsi localement, l'interface peut encore être amélioré, mais ce n'est pas le but dans ce projet !

3 Problèmes rencontrés

Mise à part la rapidité des réponses, le modèle hallucinait pas mal au début quand j'avais pas mis de template. Dès que j'ai mis un contexte ça a complètement amélioré les résultats. Un autre problème venait du fait que deux personnes s'appelle pareil dans ma base de données, ainsi il confondait ces deux personnes. Pour pallier ce problème, je pense qu'il faudrait améliorer le modèle de génération ou alors diminuer le nombre de chunks mais qui entrainerai un ralentissement du modèle.

4 Tests et validations

J'ai fais tous mes tests avec 3 CVs (un anglais, deux français avec le même prénom), et j'ai posé la même questions lors de tous les tests.

J'ai essayé plusieurs combinaisons de paramètres pour avoir le meilleur compromis entre rapidité et pertinence des résultats. Mes paramètres étaient principalement les suivant : `chunk_size`, `chunk_overlap`, et le modèle LLM. On aurait pu rajouter tous les modèles mais l'expérimentation allait être trop longue. Voici le tableau final :

LLM	Chunk Size	Overlap	Pertinence	Temps (s)
phi3:mini	1000	200	×	50
mistral	1000	200	✓	140
mistral	500	100	×	40
phi3:mini	500	100	×	30
mistral	700	200	×	110

Table 1: Comparatif de la rapidité et de la pertinence des réponses

5 Limitations

Puissance : Une des limitations les plus flagrantes reste la puissance de mon PC (je calcule seulement sur CPU). Pour que le modèle soit plus pertinent une des solutions serait de passer par un cloud et d'avoir le modèle qui tourne déjà dessus.

Rapidité: La rapidité des réponses a aussi été un facteur très important, j'ai passé beaucoup de temps à réduire le temps d'attente mais quand je le réduis à 20 secondes le modèle hallucine complètement. Je devais alors faire un compromis entre la lenteur et la pertinence. Pour la démo j'ai quand même préférer la pertinence des réponses qui était pour moi plus impréssionantes.

6 Pour aller plus loin

J'ai aussi fait un prétraitement des données, c'est à dire que j'ai passé tous les CVs dans un autre modèles fine-tuné et il devait me donner un format JSON avec toutes les informations du CV. Et ensuite ce format JSON serait vectorisé pour passé au prochain modèle. Mais la plus grande difficulté est que chaque CVs ajoute au moins 2 min 30s au prétraitement. Voici en schéma la pipeline complète qui devrait fonctionner avec un meilleur modèle :

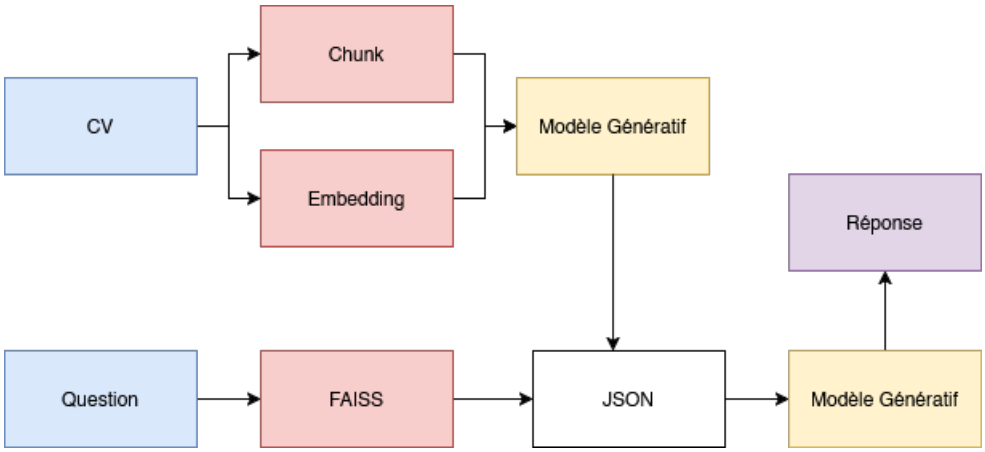

Figure 2: Pipeline assistant CV

L'implémentation d'un cloud est aussi une idée pour aller plus loin en production pour améliorer la rapidité et la précision du modèle utilisé (plus gros modèle implémenté).

7 Conclusion

Pour conclure, mon prototype est fonctionnel et permet de différencier au moins 3 CVs entre eux ! C'était un projet super intéressant qui m'a fait pratiquer ce que j'avais appris en théorie grâce à des vidéos YouTube. J'espère alors faire partie de l'intelligence lab pour travailler sur des projets intéressant comme celui-ci !

Le GitHub est disponible ici : `https://github.com/axelbrons/MiniRAG`