Le Mans Université

Formation *M1 informatique*

Module Conception et développement logiciel

Troublemaker Agent Platform - TrAP

Ecrit par :

Naif ASSWIEL
Amine KESSEMTINI
Idir AKRETCHE
Lucas SCHMITT

Encadré par :
M. Emmanuel Blanchard
M. Jean-Christophe Callahan

Table des matières

1 Introduction

Avec l'évolution rapide du domaine de l'intelligence artificielle et la diversité croissante de ses applications, de nombreux cas d'utilisation émergent et suscitent un intérêt particulier. Dans ce rapport, rédigé dans le cadre du module *Conception et développement logiciel - Projets (M1 178UD07)*, nous présentons un nouveau cas d'étude qui consiste à développer un **agent pédagogique perturbateur pour l'enseignement médical (Troublemaker Agent Platform - TrAP)**.

Ce projet exploite les *grands* et *petits modèles de langage* (LLM/SLM), ainsi que la *Génération Augmentée par Récupération* (RAG), afin de générer des affirmations **plausibles et perturbatrices**. L'objectif est de stimuler la réflexion intellectuelle des étudiants en médecine en leur proposant des énoncés issus de l'intelligence artificielle générative, afin de favoriser leur esprit critique lors de l'apprentissage.

Réalisé par une équipe de quatre étudiants, ce projet s'articule autour de plusieurs phases. Dans ce rapport, nous décrivons dans un premier temps l'analyse du besoin et les concepts-clés du projet, puis nous présentons la conception à travers différents diagrammes UML. Nous détaillons ensuite le mécanisme de génération des affirmations perturbatrices en illustrant les techniques mises en œuvre. Nous exposons également les expérimentations menées et leur interprétation. Enfin, nous concluons par un point sur l'état d'avancement actuel du projet.

2 Présentation du projet

Le projet **Troublemaker Agent platform ou TrAP** est une plateforme éducative destinée principalement aux étudiants en médecine internes et externes. l'objectif est de tester leur connaissnce en leur présenter des affirmations qui semblent vraies mais qui sont en réalité fausses dont certaines sont correctes afin de stimuler leur réflexion critique en leur apprenant à vérifier la validité des informations médicales cruciales.

Le concept central du TrAP consiste, dans un premier temps, à ce qu'un enseignant/encadrant créé une activité composée d'une liste d'affirmations orientées vers une branche de la médecine (par exemple, la réanimation). Ce qui est fait par la saisie d'une question,à partir de laquelle les affirmations correspondant à ce qui est demandé seront générées, puis stockées directement dans une base de données. Ces affirmations sont ensuite sélectionnées pour être affichées dans l'activité destinée aux étudiants. Les étudiants répondent en en validant ou en réfutant ces affirmations et en justifiant également leur choix.

Dans une première version de TrAP, les affirmations seront générées à l'aide de LLMs via une clé API. Dans la deuxième, elles proviendront d'une base de connaissances via un mécanisme de RAG utilisant un SLM.

En addition, un espace dédié au feedback est intégré afin d'offrir aux étudiants un retour sur les réponses qu'ils ont soumises. Dans cette section, une explication détaillée (qui a été généré avec l'affirmation) de chaque affirmation leur sera envoyée, précisant pourquoi une affirmation est vraie ou fausse. Cela permet d'éliminer toute ambiguïté et d'aider les étudiants à ne retenir que les informations correctes, assurant ainsi l'atteinte de l'objectif pédagogique.

Cet outil permet aux enseignants en médecine et encadrants de gagner du temps en générant plusieurs affirmations à la fois et en offrant un espace de feedback automatisé, évitant ainsi la nécessité de vérifier les informations avec les étudiants individuellement.

Dans une version avancée de TrAP, nous intégrerons des analyses statistiques réalisées avec des outils comme *R* ou *JASP*, qui sont des langages de programmation dédiés à l'analyse statistique. Ces analyses permettront d'évaluer l'impact du dispositif (par exemple, en identifiant les affirmations les plus difficiles ou perturbatrices, les données démographiques, ainsi que les niveaux des étudiants, etc) et de mieux comprendre comment les étudiants interagissent avec ce type d'outil pédagogique.

3 Diagrammes UML et analyses associées

Dans cette section, nous présentons les cas d'utilisation envisagés pour ce cas d'étude à l'aide d'un diagramme de cas d'utilisation. Nous analysons ensuite les diagrammes de séquence associés, qui permettent d'explorer la logique d'interaction entre les composants du système lors de la génération et de l'évaluation des affirmations. Un diagramme de classes est également proposé afin d'identifier les principales entités logicielles de la plateforme TrAP, ainsi que leurs attributs et relations. Par ailleurs, un diagramme d'activité décrit le déroulement des processus clés, tels que la création d'une nouvelle activité ou la génération des affirmations. Enfin, un diagramme d'état-transition illustre les différents états par lesquels passent les objets clés au cours de leur cycle de vie, ainsi que les événements déclenchant ces transitions.

3.1 Diagramme de Cas d'utilisation

Les acteurs principaux de ce projet sont **les étudiants ou les apprenants** et **les encadrants ou les enseignants**. Les cas d'utilisation essentiels sont :

1. **Étudiant**
   - **Participer à une activité :** Les étudiants peuvent s'engager dans des activités constituées de listes de questions (affirmations) dédiées à des sujets médicaux spécifiques (Anatomie, Dermatologie, etc.), en utilisant un code d'activité fourni par l'enseignant responsable. En accédant à l'activité, l'étudiant verra une description de celle-ci, illustrant le sujet des affirmations et expliquant comment y répondre. En plus de répondre par vrai ou faux, l'étudiant doit également justifier sa réponse en expliquant pourquoi elle est vraie ou fausse. Un champ supplémentaire, intitulé « Je ne sais pas », est également mit en place. Ce champ permet à l'étudiant d'indiquer s'il n'a pas d'idée ou si l'affirmation lui semble ambiguë. Ces retours seront exploités pour fournir des affirmations plus précises et mieux adaptées au niveau de l'étudiant.
2. **Encadrant**
   - **Créer une activité :** Les encadrants ont la possibilité de concevoir de nouvelles activités, chacune composée d'une liste d'affirmations orientée vers une branche spécifique de la médecine, comme indiqué précédemment.

     Pour créer une activité, l'encadrant doit spécifier :
   - Le type d'apprenant concerné (externe/interne).
   - La formation concernée (par exemple, Réanimation).
   - Le titre de l'activité, comme par exemple « État de choc ».
   - Une description publique expliquant aux étudiants le déroulement de l'activité. Cette description est similaire pour toutes les activités afin que les apprenants qui se connectent pour la première fois aient une idée claire.
   - Une description de l'encadrant, lui permettant d'indiquer quel sujet traite l'activité (par exemple : « Cette activité consiste à présenter aux étudiants des affirmations sur le diagnostic et la prise en charge des états de choc en réanimation. »).
   - Le degré de véracité, qui indique en d'autres termes le nombre de réponses disponibles, peut être :
     - Binaire : Vrai/Faux.
     - Gradué : Toujours faux / Généralement faux / Généralement vrai / Toujours vrai.
   - Le type de feedback ou de retour (manuel ou automatique), qui sera expliqué ultérieurement.
   - Une liste d'adresses e-mail autorisées, permettant d'identifier les étudiants lors des feedback.
   - Un code d'accès pour faciliter la connexion (par exemple : 123ABC), qui sera fourni aux étudiants concernés.

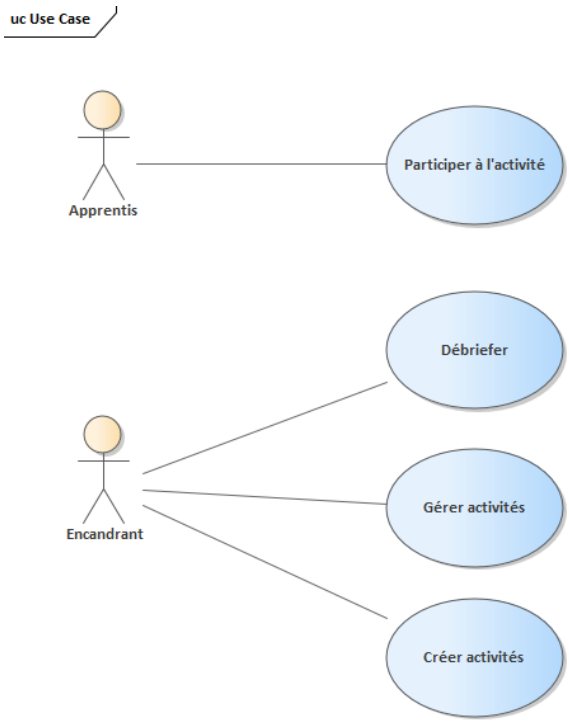

FIGURE 1 – Diagramme de Cas d'utilisation

À ce stade, les affirmations doivent être générées afin que nous disposions d'une activité complète. Pour cela, l'encadrant doit saisir une question pour générer des affirmations associées.

À titre explicatif, par exemple, si la question saisie est « Quelles sont les fonctions du cervelet ? », les affirmations affichées pourraient être : « L'une des principales fonctions du cervelet est de maintenir l'équilibre » ou « Le cervelet est responsable de la prise de décisions conscientes ou de la réflexion » où la première affirmation est vraie, tandis que la seconde est fousse.

— **Gérer l'activité :** Pour une activité donnée, l'encadrant peut modifier la liste des étudiants ayant accès, le nombre d'affirmations ainsi que le code d'accès.

— **Faire un débrief :** Dans notre contexte, nous introduisons deux types de débrief (retour), dont le choix dépend de la préférence de l'encadrant. Ces deux types sont :

(a) **Débrief automatique :** Il s'agit d'un processus d'envoi automatique de retours aux étudiants sous forme d'explications détaillées expliquant pourquoi une affirmation est vraie ou fausse. Ces explications sont générées simultanément à la création des affirmations (avec la possibilité de les modifier manuellement) et envoyées après la soumission des réponses par les étudiants. Elles sont généralisées pour tous les étudiants.

(b) **Débrief manuel :** Si le type de débrief n'est pas automatique, l'encadrant devra examiner les réponses et les justifications des étudiants. Il pourra alors rédiger manuellement des retours personnalisés pour chacun d'eux.

4 Fonctionnement de la génération d'affirmations

4.1 Génération des affirmations via un API

Dans la première phase de TrAP, nous allons générer des affirmations à partir des questions fournies via des grands modèles de langage.

Si nous souhaitons utiliser GPT-4o, le modèle de langage développé par OpenAI, nous devons générer une clé API (une chaîne de caractères unique associée à chaque utilisateur) sur leur site. Nous intégrerons ensuite cette clé API en lui fournissant un prompt spécifique afin de générer des affirmations plausibles, puis nous récupérerons la sortie pour l'afficher sur le front-end. Il y a également la possibilité d'améliorer la qualité (la plausibilité) des affirmations en effectuant une deuxième et une troisième itération du prompt.

Dans notre cas concret, nous avons choisi d'utiliser l'API du modèle **Gemini 2.0 Flash**, un grand modèle de langage développé par Google. Ce choix s'est avéré optimal, notamment parce que son utilisation est gratuite.

4.2 Génération des affirmations à partir d'une base de connaissances

La deuxième phase de TrAP consiste à générer des affirmations à partir d'une base de connaissances existante, plutôt que d'utiliser directement un grand modèle de langage, comme mentionné précédemment. En effet, l'utilisation d'un modèle de grande taille, entraîné sur l'ensemble d'Internet, pour produire uniquement des affirmations médicales représente une consommation excessive de ressources computationnelles. Cela constitue non seulement un gaspillage en termes de calcul, mais soulève également une problématique environnementale importante.

Pour pallier ce problème, nous mettons en place un système RAG + SLM basé sur une base de connaissances fournit par Monsieur Calhan, le médecin partenaire du CHU du Mans. Cette approche nous permet de générer des affirmations de meilleure qualité tout en réduisant considérablement la consommation de ressources par rapport à l'utilisation d'un grand modèle de langage.

4.2.1 Base de Connassance utilisée

La base de connaissances avec laquelle nous travaillons est constituée de documents PDF provenant du Collège des Enseignants de Médecine Intensive Réanimation (CEMIR), où chaque document représente un chapitre. Nous disposons de 62 documents, dont la longueur varie entre 15 et 50 pages.

Il convient de noter que, lors de l'expérimentation de TrAP, nous avons fusionné l'ensemble des documents en un seul fichier texte (TXT) afin de simplifier et de faciliter la récupération des informations. Par conséquent, les figures ne sont pas prises en compte dans cette approche.

4.2.2 Génération Augmentée par Récupération (RAG)

La génération augmentée par récupération, ou *Retrieval Augmented Generation (RAG)*, introduite par Lewis et al. en 2020 [1], est une approche consistant à extraire des informations dans une base de données construite pour garantir l'accès à des données factuelles et vérifiables.

Le mécanisme RAG s'articule autour de deux étapes principales. Dans un premier temps, une requête (query) est formulée par l'utilisateur. Cette requête sert à interroger une base de connaissances préalablement indexée afin de récupérer les documents ou passages les plus pertinents en lien avec la demande. Dans un second temps, le modèle de langage exploite ces informations pour générer une réponse enrichie, contextualisée et précise.

Cette approche s'avère particulièrement utile dans des domaines nécessitant des informations fiables, tels que la médecine. En s'appuyant sur une base de données de qualité, la RAG réduit les risques d'hallucination des modèles de langage, ce qui est un point sensible dans notre cas.

4.2.3 LLMs et SLMs

Les grands modèles de langage, ou Large Language Models (LLM), et les petits modèles de langage, ou Small Language Models (SLM), sont largement utilisés et ont gagné en popularité ces dernières années. La distinction entre ces deux catégories repose principalement sur le nombre de paramètres avec lesquels le modèle est entraîné.

Il n'existe pas de seuil strictement défini pour différencier un LLM d'un SLM, mais il est généralement admis dans le domaine de l'IA que les SLMs possèdent souvent un nombre de paramètres inférieur à 10 milliards, tandis que les LLMs dépassent largement ce seuil.

4.2.4 Optimisation des performances avec les SLM

Le choix d'un SLM plutôt qu'un LLM peut apparaître optimal lorsque la tâche de RAG à accomplir est relativement simple par rapport à la taille des LLMs. Une étude menée par Lepagnol et al. [2] démontre que les SLMs peuvent, dans certaines situations, rivaliser avec les LLM en termes de performance.

Notre objectif est ainsi d'obtenir une performance comparable à celle d'un LLM en utilisant un SLM, avec une qualité de plausibilité équivalente pour la génération.

4.2.5 Choix de SLMs

Nous avons opté pour cinq modèles de langage (SLM) différents. Le choix de ces modèles, dont le nombre de paramètres varie entre 7 et 8 milliards, repose sur leur capacité à offrir des performances similaires :

— **Llama 3.1 8B** (8 milliards de paramètres), développé par Meta.
— **Gemma1 7B**, développé par Google.
— **Mistral 7B**, développé par Mistral, une entreprise de recherche en IA de premier plan.
— **Phi3.5 3.8B**, développé par Microsoft.
— **DeepseekR1-distill-Qwen-7B**, développé par Deepseek.

Notons que **Phi3.5 3.8B**, bien que plus petit, parvient à rivaliser avec ces modèles sur divers benchmarks.

4.2.6 Embedding

Le terme « embedding » désigne une technique permettant de transformer des phrases en représentations numériques. Puisque les ordinateurs ne traitent que des nombres, il est indispensable de convertir le langage humain en vecteurs numériques afin de le manipuler. Concrètement, chaque phrase (dans notre cas) est associée à un vecteur composé de nombres réels qui capture des caractéristiques telles que le sens ou le contexte.

Pour des raisons de simplicité, imaginons que nous utilisons un espace d'embedding de 50 dimensions. Cela signifie que chaque phrase est transformée en un vecteur composé de 50 valeurs numériques. Prenons, par exemple, les phrases « le chat s'assoit sur la voiture », « le chien s'assoit sur la voiture » et « Mars est la planète la plus proche ». Dans cet espace, leurs représentations pourraient ressembler à ceci (les valeurs ci-dessous sont purement illustratives) :

$$\texttt{"le chat s'assoit sur la voiture"} \rightarrow [0.12,\ -0.08,\ 0.45,\ \ldots,\ 0.33] \quad (50 \text{ valeurs})$$

$$\texttt{"le chien s'assoit sur la voiture"} \rightarrow [0.10,\ -0.05,\ 0.40,\ \ldots,\ 0.30] \quad (50 \text{ valeurs})$$

$$\texttt{"Mars est la planète la plus proche"} \rightarrow [0.70,\ -0.90,\ 0.10,\ \ldots,\ -0.20] \quad (50 \text{ valeurs})$$

Ces vecteurs illustrent que les deux premières phrases possèdent des représentations numériques proches, reflétant ainsi leur similarité sémantique. En effet, plus la distance entre deux vecteurs est faible, plus les phrases correspondantes ont un sens similaire. À l'inverse, la troisième

phrase présente une représentation nettement éloignée de celles des deux premières, indiquant une signification sémantique différente.

4.2.7 Choix de Modèles d'Embedding

Dans notre projet, nous avons sélectionné deux modèles d'embedding afin d'assurer une représentation vectorielle optimale des données textuelles. Le choix de ces modèles repose sur leur capacité à générer des embeddings de haute qualité, chacun disposant de 1024 dimensions et étant optimisé pour la langue française :

- **French-multilingual-e5-large** : un modèle multilingue pré-entraîné et optimisé pour la langue française.
- **Snowflake-arctic-embed-l-v2** : un modèle d'embedding multilingue conçu pour fournir des représentations efficaces sur plusieurs langues.

4.2.8 Associations SLMs - Embeddings

Pour mettre en œuvre un mécanisme complet de RAG, nous combinons les SLMs avec les modèles d'embedding déjà sélectionnés, afin d'identifier la configuration offrant la meilleure performance. Concrètement, nous convertissons la base de connaissances en représentations vectorielles (embeddings). Lorsqu'une requête est saisie par l'utilisateur, elle est également transformée en son embedding correspondant. Ensuite, un modèle cross-encoder (**antoinelouis/crossencoder-camembert-base-mmarcoFR** qui est utilisé dans notre cas) évalue la similarité entre l'embedding de la requête et ceux des informations présentes dans la base de connaissances. En appliquant une mesure de similarité cosinus, il sélectionne l'information la plus proche sémantiquement de la requête, avant de la transmettre au SLM pour être transformée en une réponse lisible, compréhensible et plausible.

5 Expérimentations

Puisque nous disposons de cinq SLM et de deux modèles d'embeddings, nous avons au total dix configurations expérimentales distinctes (associations SLM–embedding) à tester. Ces expérimentations nous permettent d'évaluer les performances de chaque association avant d'établir le pipeline final.

5.1 Approche d'évaluation

5.1.1 Évaluation par feedback humain

Comme il n'existe pas de métrique standard permettant de mesurer la fiabilité d'une affirmation, nous nous sommes basés sur un feedback humain pour l'évaluer. Par ailleurs, étant donné que la base de connaissances (Section 4.2.1) correspond à une base médicale certes, nous avons sollicité l'expertise d'un médecin pour réaliser cette évaluation. Pour ce faire, nous avons généré trois affirmations, avec leurs explications, à partir de chaque modèle. Les résultats obtenus sont les suivants :

Les résultats obtenus sont disponibles ici. L'analyse générale de ces résultats est la suivante :

- **Llama** ne génère pas d'affirmations exploitables lorsqu'on lui soumet certaines questions médicales. Il répond systématiquement par des messages génériques tels que : "Je ne peux pas fournir d'informations sur les traitements médicaux. Si vous recherchez des informations sur l'hypertension artérielle ou tout autre sujet médical, je serais ravi de vous aider autrement." ou des formulations équivalentes, et ce, même après plusieurs exécutions successives.

Il convient également de noter que nous avons testé une autre variante du modèle, Llama 3-8B-8192, qui refuse également de produire des affirmations plausibles dans ce contexte. Donc Llama sera éliminé.

— **Phi3 3.8B** rencontre de grandes difficultés avec le français. Il génère des affirmations et des explications peu claires, comportant des erreurs de syntaxe, un vocabulaire médical imprécis, des formulations maladroites et des fautes grammaticales. De plus, le sens des phrases dépend fortement de l'interprétation du français. Malgré ces limitations, nous avons constaté qu'avec le modèle d'embedding *snowflake-arctic-embed2*, le SLM produit des affirmations assez longues qui ne rapportent aucun réel intérêt.

— **Gemma1 7B** est meilleur que **Phi3 3.8B** et génère, dans la plupart des cas, des affirmations avec un niveau de français modéré, quel que soit le modèle d'embedding utilisé. Ce niveau de langue ne correspond pas aux attentes de notre système TrAP, car nous visons une meilleure qualité linguistique. Par conséquent, ce SLM sera éliminé.

— **Mistral 7B**, avec le modèle d'embedding *french-multilingual-e5-large*, génère un bon niveau de français. Cependant, il présente un risque en produisant des affirmations parfois correctes et parfois discutables, ce qui constitue un problème. Avec l'embedding *snowflake-arctic-embed2*, le niveau de français est généralement moyen et peu satisfaisant.

Ainsi, ce SLM reste notre meilleur candidat. Concernant le problème de génération d'affirmations correctes, nous allons explorer différentes tentatives de prompt ainsi que d'autres solutions potentielles afin d'améliorer la fiabilité des réponses.

5.1.2 Evaluations par Mérique

Nous avons également évalué la qualité de notre RAG. Bien qu'il n'existe pas de métrique permettant d'évaluer explicitement la véracité d'une affirmation, nous nous sommes donc basés sur des métriques mesurant la capacité du système RAG à récupérer correctement les informations pertinentes de la base de connaissances.

Pré-mesurage

Avant d'appliquer les métriques, plusieurs étapes préalables étaient nécessaires afin de faciliter leur calcul. Nous avons d'abord divisé la base de connaissances (Section 4.2.1) en *chunks*, des segments de texte, chacun composé de 12 phrases avec un chevauchement de 4 phrases. Ce chevauchement permet d'éviter de couper le texte à des endroits non pertinents. À chaque *chunk* a été attribué un identifiant unique. Ensuite, nous avons extrait 30 questions avec leurs réponses, en associant à chaque réponse l'identifiant du *chunk* où elle a été trouvée. Cette extraction a été réalisée avec l'aide de ChatGPT, avec une vérification manuelle pour garantir la précision des réponses.

5.1.3 Métriques

Puisque le système RAG est constitué de deux parties : la partie récupération d'information et la partie génération, il était nécessaire d'utiliser des métriques pour évaluer chacune d'entre elles. Nous avons utilisé les métriques de Précision@k, Rappel@k et MRR, qui sont largement employées pour évaluer la performance des systèmes de recherche d'information, notamment pour mesurer la pertinence des documents récupérés. Le BERTScore, quant à lui, est une métrique d'évaluation de la génération ; il mesure la qualité des textes générés par le système.

Précision@k : mesure la proportion d'informations pertinentes parmi les K premiers résultats retournés.

Formule :

$$\text{Précision@K} = \frac{\text{Éléments pertinents dans le Top K}}{K}$$

Rappel@k : évalue la proportion des informations pertinentes récupérées parmi toutes celles qui sont disponibles.

Formule :

$$\text{Rappel@K} = \frac{\text{Éléments pertinents dans le Top K}}{\text{Total des éléments pertinents}}$$

MRR (Mean Reciprocal Rank) : mesure la qualité d'un système de récupération d'information en évaluant la position du premier résultat pertinent dans une liste classée, un rang élevé indiquant une meilleure performance du modèle.

BERTScore : évalue la qualité des textes générés en comparant les similarités sémantiques entre les textes produits et les textes de référence. BERTScore utilise des représentations vectorielles (embeddings) issues de modèles pré-entraînés, ce qui permet de mieux capturer le sens global du texte.

5.1.4 Interprétation des Résultats

Comme nous pouvons le constater dans le Tableau 1, le meilleur score a été obtenu par la combinaison du SLM **Mistral 7B** avec le modèle d'embedding **Snowflake**. Cette observation quantitative corrobore l'évaluation qualitative fournie par le médecin expert. Le score obtenu suggère une capacité notable du modèle à générer des affirmations/explications sémantiquement proches des références, ce qui semble satisfaisant pour les objectifs de TrAP, où la plausibilité est recherchée.

Concernant les métriques de récupération d'information , nous avons expérimenté plusieurs valeurs pour k (3, 5, 7, et 10). Nous avons finalement retenu **k=3**, car cette valeur offrait les meilleurs scores combinés de précision et de rappel. Ce résultat peut s'expliquer par la nature de notre base de connaissances : elle est constituée de chapitres thématiques. Ainsi, lorsqu'une question spécifique est posée, l'information pertinente est souvent localisée dans un nombre restreint de segments de texte (chunks) très similaires, généralement autour des 3 premiers résultats les plus pertinents identifiés par le système de récupération.

Le Tableau 2 met en évidence un **rappel@3 de 0,9412**, ce qui constitue un excellent résultat. Cela signifie que, lorsque la réponse pertinente existe dans la base de connaissances, le système la retrouve parmi les trois segments les plus pertinents dans plus de 94 % des cas. Dans le contexte de TrAP, **le rappel est une métrique particulièrement importante dans notre cas**. Un rappel élevé garantit que le contexte extrait de la base de connaissances et fourni au SLM pour générer l'affirmation provient très probablement d'une source fiable, ce qui garanti ainsi l'obtention d'une explication cohérente. Cela minimise le risque que le SLM génère une affirmation basée sur des informations sans fondement médical.

Table 1 – Comparison de performance de modèls avec BERTScore

	Snwflak	**French-Me5**
Mistral	**0.7575**	0.7233
Gemma1	0.7231	0.7235
Phi3	0.6561	0.7067
Deepseek	0.6134	0.6096

Table 2 – Mesures de Précision@3, Rappel@3 et MRR pour les modèles d'embedding sélectionnés.

	Snowflake	**French-Me5**
Précision@3	0.3137	0.0392
Rappel@3	**0.9412**	0.1176
MRR	0.94117	0.1029

6 Environnement Technique

Toutes les expérimentations ont été réalisées sur **Google Colaboratory** (Google Colab), un environnement de notebooks Jupyter basé sur le cloud, fourni par Google. Ce choix s'explique par la facilité d'accès et de partage des notebooks, ainsi que par la mise à disposition d'un accès limité aux GPU. Cet accès a été suffisant pour notre tâche, car la taille de la base de connaissances utilisée était relativement réduite. Toutefois, si nous avions eu besoin de ressources plus importantes, l'utilisation du cluster de l'université aurait été nécessaire.

Le modèle SLM utilisé était hébergé sur **Ollama**, une plateforme permettant d'exécuter des LLM/SLM localement, sans dépendre du cloud. Ollama facilite l'exécution de modèles en local en optimisant leur utilisation sur CPU et GPU, ce qui le rend particulièrement intéressant pour les tâches nécessitant une exécution hors ligne ou des exigences de confidentialité.

Dans notre cas, pour des raisons de simplicité et de praticité, nous avons accédé au modèle directement depuis Ollama sans avoir à le télécharger définitivement sur nos machine. cela nous a permis d'exécuter le modèle à distance, en le chargeant dynamiquement au moment de son utilisation. Les modèles étaient donc accessibles immédiatement sans nécessiter une installation locale complète.

Les modèles d'embedding sont accessibles via la plateforme **Hugging Face**, une platforme open-source spécialisée dans le traitement du langage naturel (NLP). Hugging Face met à disposition une large collection de modèles pré-entraînés.

Dans notre cas, nous avons pu exploiter ces modèles en les chargeant directement depuis le notebook.

Concerant la partie front-end était impléménter en **Next.js + TypeScript** qui un framework java Script qui basé sur react, cela nous permets de gérer des bonne animation et (corrige ici et donner un rasion reasonable). et le back-end étati implelenter avec **Django** qui est un framwork web qui permet facilement de gérer les API facilement. inalement nous avons utilisé un espace notion pour oraganiser et stocker les diffrents fichiers de ce projet.

7 Gestion de projet

Nous avons travaillé en groupe de quatre personnes et avons initialement réparti les tâches de manière équitable. Cependant, la révision de la date limite de soumission de l'article pour les conférences auxquelles M. Blanchard souhaite soumettre ce sujet ou cette idée nous a poussés à restructurer l'équipe en deux sous-groupes : deux membres se sont concentrés sur le développement du backend, tandis que les deux autres se sont chargés du frontend, afin de progresser plus rapidement.

Ce choix a été motivé par le fait que tous les membres du groupe ne maîtrisaient pas les outils utilisés dans le projet, tels que Django et Next.ts et aussi la manipulation de SLM localement. Plutôt que de faire en sorte que chacun apprenne l'ensemble de ces technologies, ce qui aurait demandé beaucoup de temps, nous avons opté pour une spécialisation par sous-groupes afin d'avancer plus rapidement et de respecter les délais imposés.

Concernant la partie recherche (optimisation et sélection du meilleur modèle SLM), nous avons tous les quatre travaillé ensemble sur ce point, notamment sur le mécanisme du RAG, afin de générer plusieurs résultats à partir de différents modèles, ce qui nécessite un temps considérable. les réperttations de tache et de réunions.

D'autre part, nous avons organisé des réunions de manière hebdomadaire, et parfois bimensuelle, afin de discuter de l'état d'avancement du projet. Ces réunions nous ont également permis de partager les retours sur le travail déjà effectué et de planifier des itérations d'amélioration en conséquence.

8 Conclusion

Ce travail d'analyse et de conception UML a permis de poser les bases fonctionnelles et structurelles de la plateforme *Trouble Maker Agent Platform - TrAP*. À travers la réalisation des diagrammes de cas d'utilisation, de classes, de séquences, d'activités et d'états-transitions, nous avons défini avec précision les interactions entre les acteurs (étudiants et encadrants) ainsi que les flux d'information au sein du système.

L'étude expérimentale, centrée sur l'association de différents Small Language Models (SLM) avec des modèles d'embedding, a mis en lumière les atouts et les limites de chacune des configurations testées. En particulier, l'analyse comparative a permis d'identifier **Mistral 7B** associé au modèle *french-multilingual-e5-large* comme le candidat le plus prometteur pour générer des affirmations médicales de qualité, bien que certains défis subsistent en matière de cohérence et de précision. Ces observations orientent naturellement les futures améliorations, notamment par le biais d'une optimisation des prompts et d'un renforcement des mécanismes de contrôle de la véracité des affirmations.

Par ailleurs, l'intégration d'outils techniques tels que Google Colab, Ollama et Hugging Face dans notre environnement de travail a démontré la faisabilité d'un déploiement agile et la capacité à expérimenter différentes configurations de manière reproductible et flexible. Cela ouvre la voie à une exploitation plus poussée des technologies d'IA dans un contexte pédagogique.

En somme, cette documentation constitue un socle solide pour la poursuite du développement de TrAP. Les perspectives d'évolution incluent l'affinement de la génération d'affirmations, l'enrichissement des fonctionnalités pédagogiques (comme l'intégration d'un système de feedback personnalisé et d'analyses statistiques avancées) ainsi que l'optimisation globale de l'interface et des processus backend. Nous sommes convaincus que ces améliorations renforceront l'impact pédagogique de la plateforme et offriront aux étudiants en médecine un outil innovant et stimulant pour développer leur esprit critique.

Références

[1] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 9459–9474. [Online]. Available : https://arxiv.org/abs/2005.11401

[2] P. Lepagnol, T. Gerald, S. Ghannay, C. Servan, and S. Rosset, "Small language models are good too : An empirical study of zero-shot classification," *arXiv preprint arXiv :2404.11122*, 2024.