Rapport technique de stage
03 juin 2024 - 30 août 2024

Nathan Cerisara

Stage dans le département de R&D d'Alcatel-Lucent Enterprise au pôle API à Illkirch-Graffenstaden

Alcatel-Lucent Enterprise
260 Rue Léon Foucault, 67400 Illkirch-Graffenstaden

Tuteur(s) de stage: Emmanuel Helbert (avec Sébastien Warichet)

Table des matières

Table des figures

Glossaire:

- *ALE*: Alcatel Lucent Enterprise.
- *Rainbow*: Application de messagerie développée par ALE.
- *Bulle*: Environnement privé dans Rainbow regroupant des utilisateurs autour d'une conversation, d'un espace de communication vocal et d'un espace de partage de documents.
- *RBI*: Instance Rainbow
- *NER*: Reconnaissance d'entités nommées.
- *Embedding*: Projection représentative d'une donnée dans un espace vectoriel.
- *CPU*: Processeur d'un ordinateur, Unité Centrale de Calcul, (Central Processing Unit).
- *GPU*: Processeur graphique d'un ordinateur (Graphics Processing Unit).
- *WebApp*: Application Web / Page Web connectée à un serveur.
- *Bot Rainbow*: Application qui se connecte à un serveur Rainbow avec les identifiants d'un utilisateur, et qui peut lire les messages et envoyer des messages. Avec plus d'autorisations pour l'application, il est possible d'effectuer des tâches comme créer ou supprimer des utilisateurs, des bulles, etc...
- *API*: *interface de programmation d'application* ou *interface de programmation applicative* (en anglais: *application programming interface*)
- *SDK*: ensemble d'outils fourni avec une plateforme matérielle, un système d'exploitation ou un langage de programmation. (en anglais: *Sotfware Development Kit*)

Note: Pour la version pdf de ce document, les textes en **gras** sont des liens cliquables, en plus des quelques liens avec un affichage plus classique.

1 - État de l'art

1.1 Introduction à la Recherche Sémantique

La recherche sémantique est une évolution de la recherche textuelle classique, visant à dépasser les limitations de celle-ci en utilisant des techniques qui comprennent la signification contextuelle des termes. Contrairement à la recherche par mots-clés qui se base essentiellement sur des correspondances littérales, la recherche sémantique s'appuie sur la compréhension des intentions et des relations sémantiques entre les mots et les phrases[1]. Cette capacité est particulièrement critique dans le traitement des messages provenant de messageries, où le langage est souvent non structuré, abrégé, et contextuel[2].

1.2 Les Modèles d'Embeddings : De Word2Vec à BERT

Les embeddings de mots ont révolutionné la manière dont les ordinateurs comprennent le langage humain. Word2Vec, introduit par Mikolov et al. en 2013[3], a été l'un des premiers modèles à représenter les mots dans un espace vectoriel, capturant ainsi des relations sémantiques entre eux. Cependant, Word2Vec et ses successeurs immédiats, comme GloVe[4]avaient une limitation majeure : ils ne tenaient pas compte du contexte dans lequel un mot apparaît, ce qui est crucial dans la recherche sémantique.

L'introduction de BERT (Bidirectional Encoder Representations from Transformers) par Devlin et al. en 2018[5] a marqué une avancée significative dans ce domaine. Contrairement aux modèles précédents, BERT est capable de comprendre le contexte de chaque mot dans une phrase grâce à son architecture bidirectionnelle. Cela signifie que BERT analyse non seulement les mots précédant un mot cible, mais aussi ceux qui suivent, offrant ainsi une représentation contextuelle riche[6]. Cette capacité rend BERT particulièrement adapté pour des tâches telles que la recherche sémantique dans des ensembles de messages, où le sens des mots peut varier considérablement en fonction du contexte.

1.3 Application de BERT à la Recherche Sémantique dans les Messageries

[1] Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to Information Retrieval
[2] Deerwester, S., et al. (1990). Indexing by latent semantic analysis.
[3] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space
[4] Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global Vectors for Word Representation.
[5] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.
[6] Vaswani, A., et al. (2017). Attention is All You Need.

L'application de BERT à la recherche sémantique dans des ensembles de messages présente plusieurs avantages. Premièrement, la richesse des embeddings de BERT permet de capturer non seulement les synonymes et les relations entre les mots, mais aussi les nuances subtiles du langage utilisé dans les messages[7]. Par exemple, une recherche sur le terme "problème" pourrait identifier non seulement les messages contenant ce mot exact, mais aussi ceux évoquant des notions de difficulté, d'obstacle, ou de défi, même si ces termes ne sont pas explicitement mentionnés[8].

Deuxièmement, les modèles basés sur BERT peuvent être pré-entraînés sur de grandes quantités de données textuelles générales puis affinés sur des corpus spécifiques à une messagerie particulière, améliorant ainsi leur précision[9]. Plusieurs études récentes ont montré l'efficacité de BERT et de ses variantes, telles que RoBERTa[10] et DistilBERT[11], dans des tâches de recherche d'information et d'extraction de texte, ce qui justifie leur utilisation dans le contexte de la recherche sémantique sur les messages.

1.4 Limites et Défis Actuels

Malgré les avancées apportées par BERT, il reste plusieurs défis à relever. Par exemple, le volume et la diversité des messages dans une messagerie peuvent entraîner des difficultés liées à la scalabilité du modèle et à la gestion de la complexité computationnelle[12]. De plus, les nuances culturelles, les abréviations, et les fautes de frappe courantes dans les messages peuvent affecter la précision des modèles sémantiques, nécessitant des techniques d'affinage supplémentaires [13].

Un autre défi réside dans la nature dynamique du langage dans les messageries, où de nouveaux termes et expressions émergent constamment. Les modèles comme BERT doivent être régulièrement mis à jour ou réentraînés pour rester pertinents face à ces évolutions[14].

---

[7] Peters, M. E., et al. (2018). Deep contextualized word representations.
[8] Radford, A., et al. (2019). Language Models are Unsupervised Multitask Learners.
[9] Howard, J., & Ruder, S. (2018). Universal Language Model Fine-tuning for Text Classification.
[10] Liu, Y., et al. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach.
[11] Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter.
[12] Strubell, E., Ganesh, A., & McCallum, A. (2019). Energy and Policy Considerations for Deep Learning in NLP.
[13] Dodge, J., et al. (2020). Fine-tuning Pretrained Language Models: Weight Initializations, Data Orders, and Early Stopping.
[14] Yin, W., Hay, J., & Roth, D. (2020). Benchmarking Zero-shot Text Classification: Datasets, Evaluation and Entailment Approach.

2 - Travail effectué

2.1 - Explication du sujet et des problèmes à résoudre

Dans l'application Rainbow, il y a un endroit où l'on peut faire des recherches de messages, de bulles ou d'utilisateurs. Exemple dans la **figure 1** ci-dessous.

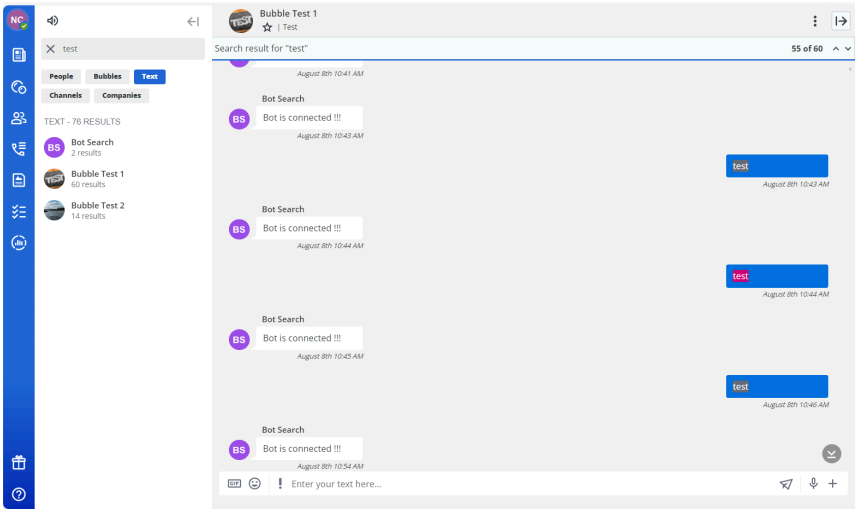


Figure 1: Exemple de l'interface de recherche déjà existante dans Rainbow
Barre de recherche en haut à gauche, et on peut naviguer avec les flèches sur les différentes occurences des résultats dans la partie de droite.

Mais il y a un problème avec la fonctionnalité de recherche existante: pour retrouver un message que l'on a envoyé dans une des conversations / bulles, l'algorithme recherche juste simplement si le texte entier apparaît dans les messages, et ne renvoie que ces messages qui ont été trouvés. Cet algorithme peut très bien fonctionner si l'on ne recherche qu'un mot clé, ou le nom d'une personne, mais devient très vite inefficace dès que l'on effectue des recherches plus complexes, comme par exemple rechercher un message dont on se souvient vaguement, mais dont l'on ne sait plus les mots précis, ou bien un message où l'on a fait une faute de frappe. De plus, dans Rainbow, les conversations s'accumulent rapidement, et peuvent contenir chacune plusieurs milliers de messages, avec certains de plusieurs années, ce qui rajoute encore plus de difficultés à la recherche de messages dans l'application.

Le sujet de mon stage a donc été de proposer une solution pour améliorer la recherche de messages dans les conversations en développant une approche sémantique fondée sur les modèles d'embeddings. L'objectif était de dépasser la simple recherche par mots-clés pour prendre en compte le sens global des messages.

Des contraintes spécifiques ont été respectées tout au long du projet. Il a notamment été impératif que l'ensemble du système fonctionne en local sur un ordinateur portable, tel que celui fourni pour ce stage, et que seuls des composants open-source soient utilisés. Cette approche assure plusieurs avantages : elle garantit la protection des données confidentielles en évitant les fuites, elle permet une utilisation en entreprise sans complications liées aux licences, et elle assure la compatibilité avec des environnements ayant des contraintes matérielles limitées.

2.2 - Présentation générale de mon travail et de l'architecture du projet

Durant ces 3 mois de stages, je me suis construit tout un environnement de travail et de tests pour avancer efficacement sur ce projet. Le cœur du projet est programmé en Python, mais j'ai aussi développé quelques pages web locales liées à un serveur websocket Python pour avoir une interface pour afficher des benchmarks, intéragir avec une démo, ou bien travailler sur des fichiers de configurations et les optimiser. Et finalement, j'ai aussi programmé un petit programme en C# pour faire le lien entre mes scripts python et l'API Rainbow vers la fin de mon stage pour avoir une démo plus proche d'un cas d'utilisation réel dans l'application. Les détails et descriptions de chaque fichiers et sous-dossiers du projet sont disponibles dans l'**annexe 1**. Tout le projet est disponible sur mon compte github personnel (https://github.com/nath54/SemanticSearch-ALE/).

Avant le stage, j'avais déjà préparé mon travail[15], donc j'avais déjà imaginé la structure moteurs - algorithmes modulable et générique que j'ai utilisé et affiné durant mon stage. Au tout début de mon stage, j'ai d'abord défini les classes de bases définissant les éléments à manipuler (utilisateurs, messages, ...). Je ne savais pas à l'avance sur quels types de données j'allais travailler ni si j'allais pouvoir travailler directement sur le code du serveur / de l'application Rainbow. Finalement, je n'ai eu accès qu'à une sandbox Rainbow mais sans avoir accès à un vrai environnement de travail pour développer dedans, donc je me suis construit mon propre environnement de travail reproduisant les mêmes modèles de données, avec une application web pour simuler un environnement Rainbow, puis j'y ai intégré mon moteur de recherche.

Ensuite j'ai optimisé au maximum les performances et la précision des modèles de calculs d'embeddings, puis j'ai travaillé sur des algorithmes et des concepts autour pour voir si on ne pouvait pas encore améliorer les performances et la précision, comme de la découpe de conversations et la reconnaissance d'entités nommées (NER).

[15] https://github.com/nath54/SemanticSearch-DiscordBot

Puis, en août, pour se rapprocher un peu plus d'un vrai environnement d'utilisation, on a essayé de voir ce que je pouvais faire pour intégrer le plus possible mon projet à l'application Rainbow, et la seule chose possible était de développer un bot dans une sandbox Rainbow (un environnement isolé pour faire des tests sans impacts sur la vraie application). J'ai utilisé l'API SDK en C# de Rainbow pour cela.

2.3 - Détails sur l'environnement de travail et le parseur de donnée

2.3.1 - Structure de données utilisées

Une représentation classique de Conversation - Utilisateur - Messages a donc été adoptée, semblable à ce que l'on peut attendre de toute messagerie. Pour une meilleure compréhension, Rainbow utilise un système de salons de messages, appelés "bulles". Chaque utilisateur a la possibilité de créer ses propres bulles et d'y inviter les personnes souhaitées. Un utilisateur peut posséder plusieurs bulles et envoyer des messages dans chacune d'elles.

Les classes utilisées pour représenter ces différents éléments sont les suivantes :

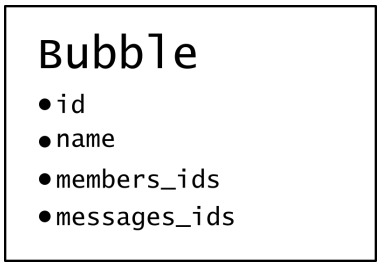


Figure 2: Représentation d'une bulle

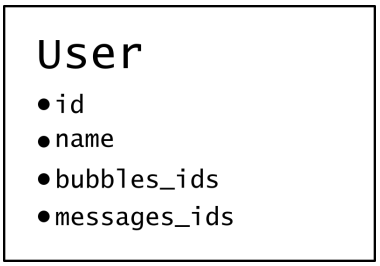


Figure 3: Représentation d'un utilisateur

Figure 4: Représentation d'un message

Une classe intitulée "Instance Rainbow (RBI)" a donc été définie, représentant un serveur Rainbow comprenant ses bulles, ses utilisateurs et ses messages.

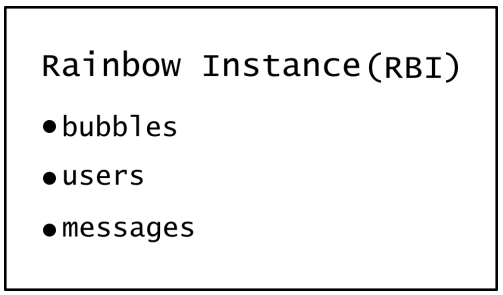


Figure 5: Représentation d'une Instance Rainbow (RBI)

La structure de serveur implémentée permet de travailler sur plusieurs RBIs simultanément.

2.3.2 - Exportations de bulles Rainbow et parseur de données

Il y a une fonctionnalité disponible directement sur Rainbow qui permet d'exporter sous un format textuel toute une bulle entière avec tous ses messages (voir **figure 6** juste ci-dessous).

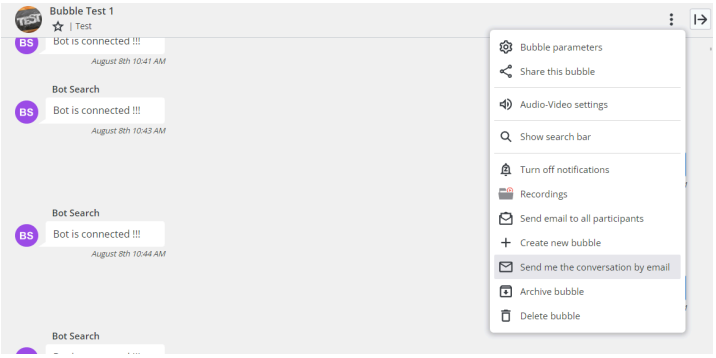


Figure 6: Exportation d'une bulle dans l'application Rainbow
Possibilité d'exporter une conversation par email via un bouton dans un menu déroulant dans l'interface.

Les bulles ainsi exportées ont le format suivant:

```
Prénom Nom Jour, Mois Numéro-du-jour, Année Heure
Contenu du message
```

Un exemple de données de tests générées avec **ChatGPT**[16] avec ce format est disponible à **l'annexe 2**.

Un parseur en Python a été développé afin d'extraire les structures de données présentées précédemment à partir de ces données exportées.

[16] https://platform.openai.com/playground/chat?models=gpt-4o

Étant donné le nombre limité de bulles à exporter sans données sensibles ou confidentielles, et pour disposer de suffisamment de données sur lesquelles travailler et tester les programmes, des données supplémentaires ont été générées à l'aide de LLMs tels que ChatGPT ou **Gemini**[17]. Ces données ont ensuite été utilisées pour créer des benchmarks, permettant ainsi de poursuivre le travail sans perdre de temps à chercher de bonnes données déjà existantes et utilisables.

2.4 - Détails sur le moteur de recherche

2.4.1 - Principe du moteur de recherche

Une architecture de moteur de recherche très modulable et simple d'utilisation a donc été implémentée. L'idée consiste à travailler avec plusieurs sous-algorithmes et à rassembler leurs différents résultats pour obtenir un résultat final.

On va donner en entrée au moteur de recherche le texte recherché, la RBI dans laquelle on va faire la recherche, ainsi que des potentiels paramètres de recherche (filtres, date, ...). Avec les paramètres de la recherche, on va extraire de la RBI une liste de messages que l'on va ensuite donner aux différents algorithmes, qui va retourner un score correspondant à chaque message, et on va mixer les scores de tous les algorithmes avec différents coefficients correspondants pour chaque algorithme. Le score final obtenu représente une distance de chaque message au texte de la recherche, on renvoie donc les messages triés par scores croissants à la fin.

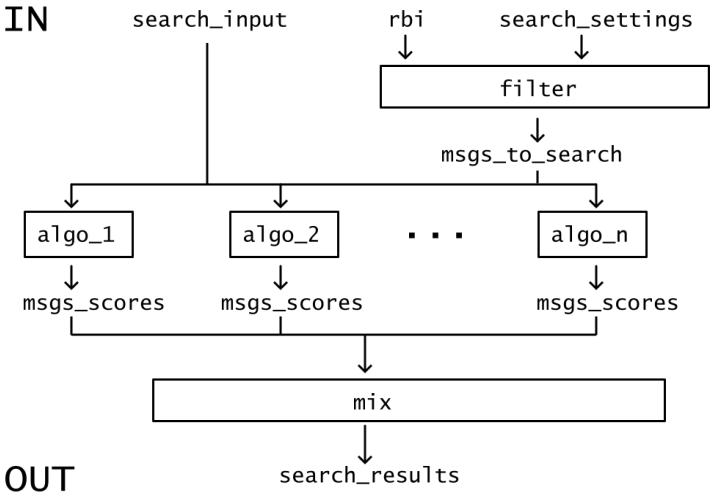


Figure 7: Moteur de recherche

[17] https://gemini.google.com/app

Les moteurs de recherches (comme tous les autres types de moteurs) fonctionnnent grâce à un fichier de configuration, qui permet de préciser les valeurs de certains paramètres et des algorithmes que l'on souhaite utiliser avec (voir exemple en **annexe 4**). De plus, il est aussi possible de voir/modifier/créer des configurations grâce à une interface web présentée dans la partie : 2.10.2 - Création et modifications des configurations des différents moteurs.

La configuration d'un moteur de recherche possède des paramètres tels que le nombre de résultats de recherches ou bien la taille des bouts de messages qui sont envoyés dans les algorithmes (quand un message est trop long, il est découpé de la meilleure façon possible pour ne pas couper les phrases et les mots). Le paramètre le plus important est la liste des algorithmes de la configuration, on peut combiner autants d'algorithmes que souhaité, et chaque algorithme a ses propres paramètres, mais ils en partagent tous un en commun: leur coefficient. Un moteur va exécuter au fur et à mesure les différents algorithmes, et va ensuite combiner les résultats en un seul en pondérant par les coefficients de chaque algorithme.

2.4.3 - Algorithme de recherche sémantique

La structure d'un algorithme de recherche est ensuite plutôt basique, on va appliquer de potentiels pré-traitements sur le texte des messages et de la recherche, comme une traduction, ou bien un remplacement d'un mot par sa définition. Et ensuite on va appliquer le cœur de l'algorithme, qui va souvent s'agir d'une comparaison / fonction de distance. Chaque algorithme renvoie donc au final un score / distance pour chaque message par rapport au texte recherché.

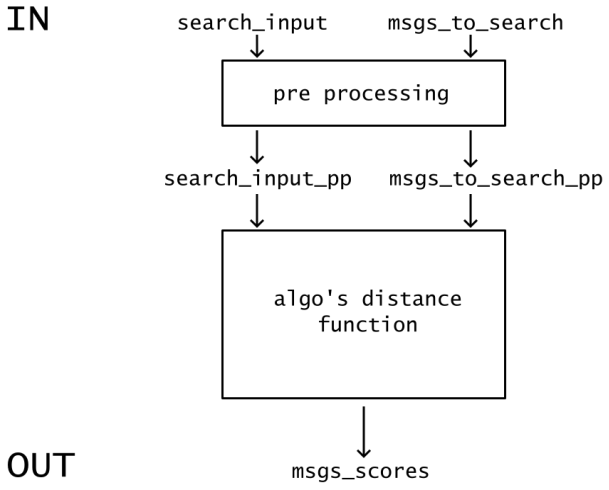


Figure 8: Algorithme de recherche

L'algorithme principal de recherche sémantique a ensuite été implémenté. Le principe est basique: on regroupe mes messages en différents batchs que l'on va donner en entrée à un modèle d'embedding, ce modèle va ensuite renvoyer un vecteur pour chaque message ainsi que pour le texte recherché. Ce vecteur correspond à une représentation sémantique du texte et permet ensuite de calculer directement une distance entre le texte recherché et les différents messages sur lesquels la recherche est effectuée. (Par exemple: un message qui parle de nourriture aura une distance plus proche avec un autre message qui parle de nourriture qu'avec un message qui parle d'autre chose).

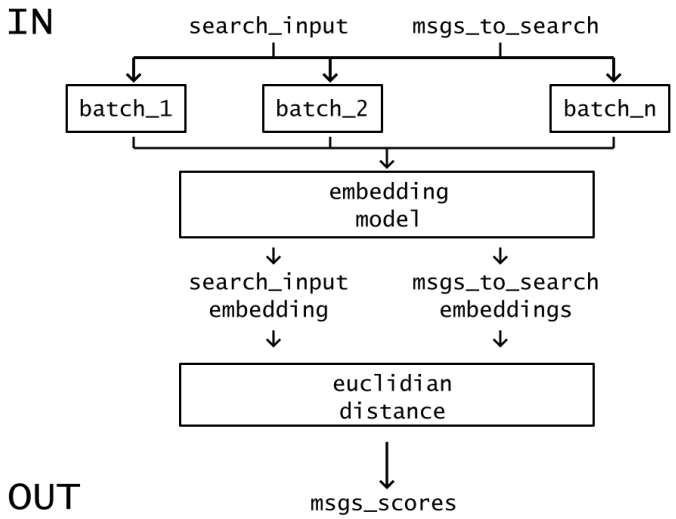


Figure 9: Calcul de distance sémantique avec modèle d'embedding

Lors des tests et de la comparaison de différents modèles d'embedding, réalisés vers juin - juillet 2024 (précision apportée en raison de la rapidité d'évolution dans ce domaine et de la publication fréquente de nouveaux modèles plus performants), le modèle offrant le meilleur rapport performance/taille parmi ceux disponibles était le modèle **all-MiniLM-L6-v2**[18]. C'est un modèle de 22 millions de paramètres avec l'architecture Transformers. Il a été entraîné par un FineTuning depuis le modèle **MiniLM-L12-H384-uncased**[19] par Microsoft, qui est une famille de petits modèles entraînés par distillation (apprendre à reproduire les sorties d'un plus gros modèle).

[18] https://huggingface.co/optimum/all-MiniLM-L6-v2

[19] https://huggingface.co/microsoft/MiniLM-L12-H384-uncased

2.4.4 - Amélioration des modèles d'embeddings avec de la traduction

Au cours des différents tests et benchmarks réalisés, il a été remarqué que, bien que certains modèles testés soient censés être multilingues, les résultats en anglais se sont avérés meilleurs qu'en français. Cela peut s'expliquer par le fait qu'il existe davantage de données de meilleure qualité en anglais pour l'entraînement de ces modèles.

	Moyenne	Evaluation Sémantique 1 en Anglais	Evaluation Sémantique 1 en Francais	Evaluation Sémantique 2 en Anglais	Evaluation Sémantique 2 en Francais	Test Sémantique en Français pour NER	Test d'usage dans vraie bulle extraite de Rainbow (Fr)
All-MiniLM-L6-v2 with NER Engine and Translation	0.8862	1	1	0.9167	0.9167	0.7146	0.7694
All-MiniLM-L6-v2 with NER Engine	0.7053	1	0.7456	0.9167	0.5387	0.4677	0.5632

Figure 10: Benchmarks résultats de recherche avec et sans traduction
On observe de meilleurs résultats pour tous les benchmarks en Français avec le modèle qui utilise de la traduction

Pour la traduction, un modèle basé sur l'architecture Transformer, reconnue comme étant l'une des meilleures en NLP actuellement, a été utilisé. La famille de modèles **opus-mt**[20], qui propose plusieurs versions pour la traduction d'une langue à une autre, a été privilégiée, comme le modèle **opus-mt-fr-en**[21] pour la traduction du français vers l'anglais. Ces modèles reposent sur l'architecture **MarianMT**[22] et ont été entraînés sur le dataset **Opus**[23].

La traduction a eu un réel impact sur la précision des résultats de recherche, mais avec un certain coût sur la performance (la traduction d'un message peut prendre en moyenne plus ou moins 5 secondes selon la longueur du message sur un ordinateur portable milieu de gamme). En utilisant un modèle d'embedding avec de meilleurs résultats pour le Français, on peut éviter d'avoir besoin de passer par l'étape de traduction.

[20] https://github.com/Helsinki-NLP/Opus-MT
[21] https://huggingface.co/Helsinki-NLP/opus-mt-fr-en
[22] https://huggingface.co/docs/transformers/model_doc/marian
[23] https://github.com/Helsinki-NLP/OPUS-MT-train

2.4.5 - Algorithmes Syntaxiques

En complément de l'algorithme de recherche sémantique, des algorithmes de recherche syntaxiques basiques ont également été testés, comme la comparaison du nombre de mots ou de suites de mots en commun entre deux textes, ou le calcul des **distances de Levenshtein**[24] entre les messages. Ces algorithmes sont particulièrement efficaces pour les recherches utilisant des mots clés précis. Ainsi, avec un système de détection du type de recherche, qui sélectionne ensuite les algorithmes appropriés en fonction de la nature de la recherche (longue phrase ou un ou plusieurs mots clés), il est possible d'obtenir de très bons résultats.

2.4.6 - Algorithmes de recherche utilisant de la NER

Note: Vous pouvez retrouver une définition et explication plus en détails du concept de NER ici: 2.5.1 - Qu'est-ce que la NER ? (définition générale).

Une difficultée qui avait été soulevée sur ce sujet de stage était le fait que les modèles d'embeddings n'ont pas forcément de très bonnes représentations vectorielles quand on utilise des noms propres ou bien quand on utilise un jargon spécifique à un domaine avec pleins d'abréviations et de mots clés.

Une solution pour essayer de contourner ce problème est d'utiliser à côté du modèle d'embedding un algorithme qui va d'abord récupérer les entités nommées du texte de la recherche et de tous les messages grâce à un moteur de NER, puis va comparer le nombre d'entités nommées en communs entre le texte de la recherche et un message (=*set similarity distance*, **figure 11**).

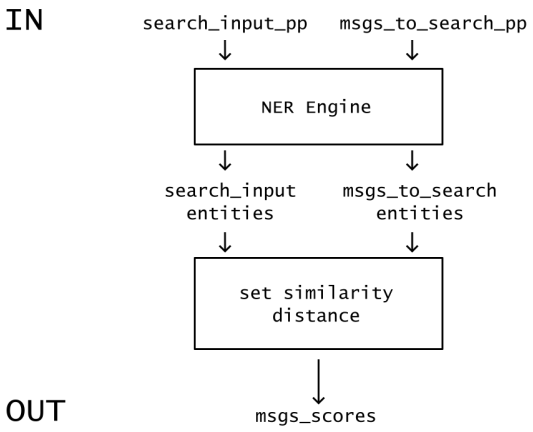


Figure 11: Calcul de distance avec NER pour algorithme de recherche

[24] https://fr.wikipedia.org/wiki/Distance_de_Levenshtein

2.4.7 - Détecter une date dans la recherche

Une autre idée que l'on a eu était de détecter les dates dans le texte de la recherche pour pouvoir filtrer une recherche sur une date précise. Par exemple, si on recherche "réunion sur ... mardi dernier", on détecte qu'il y a "mardi dernier" dans la recherche, on calcule à quelle date cela correspond, puis on filtre les messages de la recherche sur cette date. Pour cela, on m'avait proposé le module duckling de Facebook, mais je n'ai pas réussi à l'installer correctement, et je n'ai pas réussi à trouver et installer non plus des modules similaires sur l'ordinateur portable de travail qu'ALE m'avait prêté pour le stage. J'avais donc commencé à essayer de recoder une librairie en python légère pour faire cela basée sur des règles syntaxiques (code), mais je me suis vite rendu compte que cela ne marcherait pas vraiment pour s'adapter flexiblement à tous types de façon d'écrire les dates possibles dans le monde.

En utilisant un LLM génératif comme ChatGPT ou **LLama3.1**[25], on peut facilement extraire ce genre d'information, mais cela coûte extrêmement cher en performances ou bien en coût d'api externe. Mais avec des petits LLMs comme **SmolLM-135M**[26] cela pourraît bien être disponible rapidement.

Mais sinon, en fait, détecter des dates dans des textes de recherche pose d'autres difficultées comme le fait de détecter si l'expression de la date détectée correspond à un filtre sur une date pour la recherche de message, ou bien si cela correspond au sens sémantique du message à rechercher. Donc finalement, le plus simple à faire dans ce cas là est de gérer les filtres sur la date de la recherche directement sur l'interface de recherche dans l'application.

2.5 - Détails sur la Reconnaissance d'entité nommée (NER)

2.5.1 - Qu'est-ce que la NER ? (définition générale)

La Reconnaissance d'Entités Nommées (NER) est une technique essentielle en traitement automatique du langage naturel (NLP), dont l'objectif est de détecter et de classifier automatiquement des éléments spécifiques dans un texte, comme les noms de personnes, les lieux, les dates, les organisations, ou d'autres entités distinctes.

L'idée fondamentale derrière la NER est de donner un sens plus concret à des segments de texte en les reliant à des catégories prédéfinies. Par exemple, dans une

[25] https://ollama.com/library/llama3.1

[26] https://huggingface.co/HuggingFaceTB/SmolLM-135M

phrase telle que "Marie travaille chez Google à Paris", la NER permet d'identifier "Marie" comme une personne, "Google" comme une organisation, et "Paris" comme un lieu.

Cette classification joue un rôle crucial dans de nombreuses applications, telles que la recherche d'information, l'extraction de faits, ou la structuration de grandes quantités de données textuelles, car elle permet de transformer un texte brut en un ensemble de données plus structurées et facilement analysables. En somme, la NER agit comme un filtre intelligent qui, en distinguant les entités clés d'un texte, en améliore la compréhension et l'exploitation.

2.5.2 - Dictionnaires d'entités nommées

Une autre approche testée pour résoudre le problème évoqué précédemment (certains noms/abréviations ne sont pas bien compris par le modèle d'embedding, 1er paragraphe de: 2.4.6 - Algorithmes de recherche utilisant de la NER) consiste à remplacer le mot, l'abréviation ou l'expression problématique par une définition utilisant des mots simples, afin que le modèle d'embedding puisse mieux les comprendre.

Le principe est tout simple: on dispose de dictionnaires de définition pour différentes expressions (exemple en **annexe 3**) et pendant la phase de prétraitement de textes / messages lors d'un algorithme de recherche (voir 2.4.3 - Algorithme de recherche sémantique), les expressions présentes dans le dictionnaire sont détectées puis remplacées par leur définition.

Cela soulève également un autre problème : lorsque une expression ou un nom peut avoir plusieurs définitions possibles, comment déterminer laquelle est présente dans un texte donné ? Cette question n'a pas été traitée, et les expressions ont été remplacées dès qu'elles étaient identifiées, car cela aurait nécessité un travail considérable s'éloignant du sujet principal. Malgré cela, en appliquant le principe de base expliqué ci-dessus, une amélioration notable des résultats a été observée, avec ou sans remplacement de NER, dans un benchmark spécifique à ce problème (voir **figure 12** ci-dessous).

	Test Sémantique en Français qui se concentre sur la NER
all-MiniLM-L6-v2 with NER replacement	0.7751
all-MiniLM-L6-v2	0.5003

Figure 12: Benchmarks résultats de recherche avec et sans dictionnaires de NER

2.5.3 - Algorithme de NER syntaxique simple

Pour le principe de l'algorithme présenté dans la partie (2.4.6 - Algorithmes de recherche utilisant de la NER), on peut détecter certaines entités (surtout les noms propres ou abréviations) avec des règles syntaxiques simples, comme le fait qu'un nom propre commence par une majuscule, ou bien que certaines abréviations sont écrites toutes en majuscules, ou quand on a certain caractères, comme "//" on peut facilement détecter des liens web ou chemins de fichiers.

Un algorithme de détection de NER simple a donc été implémenté, basé sur ce type de règles (bien que peu de règles aient été implémentées, il est assez facile d'en ajouter d'autres). Ce procédé fonctionne plutôt bien lorsque le texte est correctement formaté, mais présente des limites lorsque aucun format n'est respecté. Une autre idée, qui n'a pas été mise en œuvre, consisterait à travailler avec des dictionnaires de noms propres et de noms communs, permettant ainsi de détecter des noms propres ou des mots absents du dictionnaire de noms communs. Malgré ces limitations, les premiers résultats obtenus dans ce domaine sont encourageants.

2.5.4 - Tests avec les modèles de NER de la librairie SpaCy

L'algorithme de la librairie Python **Spacy**[27] a également été testé directement, mais les résultats n'étaient pas nécessairement meilleurs que ceux de l'algorithme syntaxique sur le benchmark évalué (voir les figures ci-dessous). Il semble plutôt que les deux approches se complètent, chaque méthode présentant ses propres forces et faiblesses. Cependant, étant donné que les algorithmes de NER de Spacy nécessitent un temps d'exécution considérablement plus long (voir la **figure 13** ci-dessous), l'algorithme syntaxique a été privilégié pour l'algorithme présenté dans la section 2.4.6 - Algorithmes de recherche utilisant de la NER.

	Test de NER 1	
	score	vitesse
SimpleSyntaxicNER	0.7049	0.001 sec
SpaCy_LG_NER	0.6858	1.3032 sec
SpaCy_SM_NER	0.6815	0.351 sec

Figure 13: Comparaison entre algorithme syntaxique et SpaCy pour la NER

[27] https://pypi.org/project/spacy/

2.6 - Détails sur le moteur de découpe des conversations

2.6.1 - Principe de base de la découpe de conversation

Une idée pour optimiser le coût d'une recherche dans pleins de bulles différentes avec chacune beaucoup de messages était de détecter les différentes conversations dans chacunes des bulles, puis de les isoler et de calculer une représentation de la conversation (vectorielle, textuelle ou autre). Ensuite, lors de la phase de filtrage et de récupération des messages avec lesquels faire la recherche, on peut d'abord filtrer sur les conversations si elles paraissent intéressantes ou non.

L'extension à des représentations de conversation et à un filtre sur les conversations n'a pas été réalisée jusqu'à ce point, mais il est théoriquement possible d'obtenir des résultats intéressants tout en réduisant le temps de recherche.

Pour les différents algorithmes de clustering de conversation, une matrice de distance entre tous les messages a d'abord été calculée en utilisant les divers algorithmes de calcul de distance implémentés pour la recherche de message. L'algorithme a ensuite été appliqué sur cette matrice de distance entre tous les messages.

2.6.2 - Tests avec un algorithme de clustering de type Fusion

Pour la découpe de conversation, un algorithme de clustering classique de type Fusion a d'abord été testé. Le principe consiste à attribuer initialement chaque message à sa conversation respective, puis, tant qu'il est possible d'effectuer des ajustements, pour chaque message, on recherche une conversation acceptable plus proche de celle à laquelle il appartient déjà (à l'exception des messages seuls dans leur conversation).

Finalement, cette approche n'a pas montré une performance satisfaisante (voir **figure 16**), ce qui a conduit à l'exploration d'une autre méthode offrant des avantages supplémentaires dans le contexte d'une messagerie.

2.6.3 - Algorithme de clustering séquentiel

Cette autre méthode est de faire un clustering de manière séquentielle sur l'ordre des messages dans la bulle. Le principe est de prendre des messages dans l'ordre, et de voir s'il y a déjà une conversation dans laquelle le message peut appartenir, et si il n'y en a pas, on crée une nouvelle conversation dans laquelle on rajoute ce message.

Cette méthode a plusieurs avantages, déjà, elle suit le fait que les messages sont envoyés dans un certain ordre et non un ordre au hasard, et cela ne sert pas forcément à grand chose de faire des clusters aléatoires que l'on bouge ensuite vu qu'un message ne peut appartenir qu'aux conversations qui existent déjà avant qu'il ne soit envoyé, ou bien qu'il en crée une nouvelle. De plus, cela permet de facilement ajouter des nouveaux messages dans les conversations existantes sans avoir à tout recalculer comme on pourrait avoir envie de le faire pour un algorithme de type k-means par exemple.

Et finalement, cette méthode a plutôt bien l'air de fonctionner si l'on en suit les résultats des benchmarks (voir **figure 14** ci-dessous).

	Moyenne	ConvSeg Test - ChatGPT - En - 1	ConvSeg Test - ChatGPT - En - 2	ConvSeg Test - ChatGPT - Fr - 3	ConvSeg Test - ChatGPT - Fr - 4	ConvSeg Test - Fr - 5
Clustering Sequentiel	0.7188	0.63	0.8098	0.5685	0.7153	0.8706
Simple By Time Difference	0.5653	0.6784	0.5647	0.2312	0.7913	0.5607
Clustering Fusion	0.5065	0.5115	0.4979	0.4312	0.4928	0.5993

Figure 14: Comparaison entre algorithmes de découpe de conversations
Les colonnes représentent différents benchmarks, et les lignes les algorithmes testés sur chaque benchmark. On observe que l'algorithme de clustering séquentiel présenté juste avant a les meilleurs résultats.

2.7 - Détails sur les différentes optimisations de mon code

2.7.1 - Système de Profiling de code

Il peut être utile d'analyser le temps d'exécution d'une portion de code. Pour ce faire, des outils de profiling de code peuvent être employés pour évaluer les performances d'un code. Une librairie Python légère de profiling de code a donc été développée, permettant de calculer le temps d'exécution d'une portion de code et de regrouper hiérarchiquement les portions entre elles. Par exemple, une fonction appelant plusieurs sous-fonctions peut être analysée pour obtenir le temps global de la fonction ainsi que le temps des sous-fonctions, ces dernières étant indiquées comme appartenant à la fonction principale.

Une page web permettant de visualiser les données récoltées par la librairie de profiling a ensuite été développée. Cette page s'est révélée très utile pour analyser les aspects chronophages du code, notamment lors des essais de différentes méthodes de mise en cache de certaines données.

2.7.2 - Cache d'embedding

Selon le modèle et la machine utilisés, le calcul des embeddings peut varier en durée. Avec un GPU, il est possible de paralléliser les calculs en regroupant les messages en batchs, tandis que sur CPU, cette approche s'avère inefficace. Afin de réduire les temps de calcul, un cache d'embeddings a été implémenté. L'idée consiste à calculer un embedding pour un texte une seule fois, à le sauvegarder en cache, puis à charger cette information en cache lors des calculs ultérieurs pour le même texte.

Plusieurs méthodes ont été testées pour la gestion du cache d'embeddings. Initialement, un simple dictionnaire JSON était utilisé pour enregistrer les vecteurs convertis en listes de flottants. Cependant, dès que le nombre d'embeddings dans le cache a augmenté, la lecture et l'écriture du fichier JSON sont devenues trop lentes. Une tentative d'utilisation d'une base de données SQLite s'est également révélée peu efficace.

La solution retenue consiste à calculer un hash pour le texte d'un embedding et à enregistrer chaque embedding dans un fichier unique portant le nom de ce hash. De plus, une classe "MessageEmbedding" a été développée pour contenir l'embedding ainsi que d'autres informations associées, et cette classe est convertie directement en format binaire à l'aide de la bibliothèque **pickle**[28] de Python. Cette approche présente l'avantage de ne pas nécessiter le chargement de tous les embeddings en mémoire, permettant un accès direct et rapide à l'unique embedding requis.

[28] https://docs.python.org/3/library/pickle.html

2.7.3 - Cache de traduction

De même que pour le calcul d'embeddings, la traduction d'un message prend d'autant plus de temps que les modèles de traduction sont souvent plus lourds que les modèles d'embeddings. Un système de cache d'embeddings pour la traduction a donc été mis en place, permettant de ne traduire un message qu'une seule fois et de réutiliser le message du cache pour les traductions ultérieures.

Cependant, étant donné qu'une chaîne de caractères est nettement moins volumineuse qu'un vecteur de flottants en 384 dimensions, un dictionnaire de la forme `{''texte à traduire'' -> ''texte traduit''}` a pu être utilisé pour le cache de traduction. Pour environ quelques milliers de messages traduits, le fichier de cache JSON de traduction occupe environ 1 Mo d'espace de stockage, ce qui garantit toujours un chargement et une écriture rapides.

2.7.4 - Utilisations de poids plus optimisés

Pour la partie calcul d'embedding, la librairie **Transformers**[29] avec **PyTorch**[30] de **Hugging-Face**[31] a d'abord été utilisée, offrant de bons résultats, notamment pour des aspects de recherche nécessitant personnalisation, expérimentation et contrôle complet. Cependant, pour une utilisation des modèles axée uniquement sur l'inférence, il existe des librairies plus optimisées qui se concentrent sur la vitesse d'exécution. Par défaut, lors de l'utilisation d'un modèle, les poids des modèles d'embedding sont en notation binaire flottante sur 16 bits (fp16) pour chaque paramètre. Des méthodes de quantification des poids des paramètres des réseaux de neurones permettent de les convertir en notation binaire sur 4 bits, réduisant ainsi le poids des modèles et permettant une exécution plus rapide.

C'est pourquoi la librairie **Optimum**[32], basée sur **ONNX**[33], a été choisie. Cette approche a permis d'obtenir un temps d'exécution plus rapide pour le calcul d'embedding.

[29] https://huggingface.co/docs/transformers/index
[30] https://pytorch.org/
[31] https://huggingface.co/
[32] https://huggingface.co/docs/optimum/onnxruntime/usage_guides/models
[33] https://onnx.ai/

2.8 - Détails sur le système de benchmarks

2.8.1 - Fichiers de benchmarks

Un système de benchmarks a donc été implémenté pour évaluer les performances des différentes configurations de moteurs/algorithmes. Étant donné qu'il existe trois types de moteurs distincts : moteur de recherche, moteur de découpe de conversations et moteur de NER, trois types de benchmarks ont été définis pour chacune de ces tâches.

Concernant la structure d'un fichier de benchmark pour évaluer la recherche, il a été décidé de travailler directement avec les RBIs préparées, ce qui évite d'inclure de grandes quantités de données de messages, bulles ou utilisateurs dans le fichier de benchmark. Il suffit simplement de préciser la RBI à utiliser. Le fichier contient uniquement une liste de recherches à effectuer, où chaque recherche se décompose en : le texte de la recherche, l'identifiant de l'utilisateur effectuant la recherche, et l'identifiant du message attendu ou la liste des identifiants des messages attendus (voir **annexe 5**).

Sinon, pour les deux autres types de fichiers de benchmarks, ils contiennent toutes les informations dont on a besoin directement dedans.

Pour la structure d'un fichier de benchmark pour évaluer la découpe de conversation, on a donc une liste de messages, puis la conversation à laquelle appartient chaque message, et aussi la liste des ids des messages pour chaque conversation. Avoir l'information dans les deux sens directement dans le fichier de benchmark permet de ne pas avoir à la recalculer quand on en a besoin et ça évite d'alourdir le code.

Pour la structure d'un fichier de benchmark destiné à évaluer la NER, une liste de tests est utilisée, chacun comprenant un texte et les entités annotées de ce texte. Ces entités sont présentées dans un format standard (début de la chaîne de caractères, chaîne de caractères, type de l'entité pour les modèles de NER plus complexes).

2.8.2 - Script python pour évaluer les benchmarks

Le script Python utilisé pour les benchmarks est relativement simple. Il permet de spécifier des paramètres pour exclure certains types de benchmarks. Pour chaque type de benchmark, le script commence par récupérer la liste des

fichiers de benchmarks. Ensuite, pour chaque configuration de moteur, il exécute les tests, calcule un score pour chacun, et enregistre les résultats dans un fichier JSON.

Grâce à l'enregistrement systématique des résultats de tous les benchmarks, le script permet de détecter facilement les benchmarks déjà réalisés. Ainsi, il évite de répéter tous les benchmarks à chaque exécution, se concentrant uniquement sur ceux qui ont été modifiés ou qui n'ont pas encore été effectués pour chaque configuration de moteur.

2.8.3 - Calcul de scores pour les benchmarks

Pour le calcul du score du benchmark de recherche, lorsqu'un seul message était attendu pour une recherche, la formule suivante a été utilisée, avec p représentant la position du message attendu dans les résultats :

$$\frac{1}{p+1}$$

Formule 1: Score d'une recherche en fonction de la position du message dans les résultats

Cela marche bien pour illustrer un score car c'est une fonction décroissante en p, qui commence à 1 et qui tend vers 0, avec une dérivée croissante, donc le score diminue de moins en moins au fur et à mesure que p augmente.

Et si plusieurs résultats étaient attendus, c'est la même formule qui est utilisée, mais au lieu d'utiliser la position, on utilise une pénalité qui indique le nombre de messages que l'on voit avant de voir tous les messages attendus, et la pénalité est réduite à chaque fois que l'on voit un message attendu.

Pour le calcul du score du benchmark de découpe de conversation, deux scores sont calculés : un **score F1**[34] pour les endroits où des changements de discussion apparaissent dans la liste des messages, et un **score d'édition**[35] adapté aux conversations (représentant le nombre d'opérations nécessaires pour passer de l'ensemble des conversations renvoyées par le moteur à l'ensemble attendu par le benchmark). Le score final correspond à la moyenne de ces deux scores.

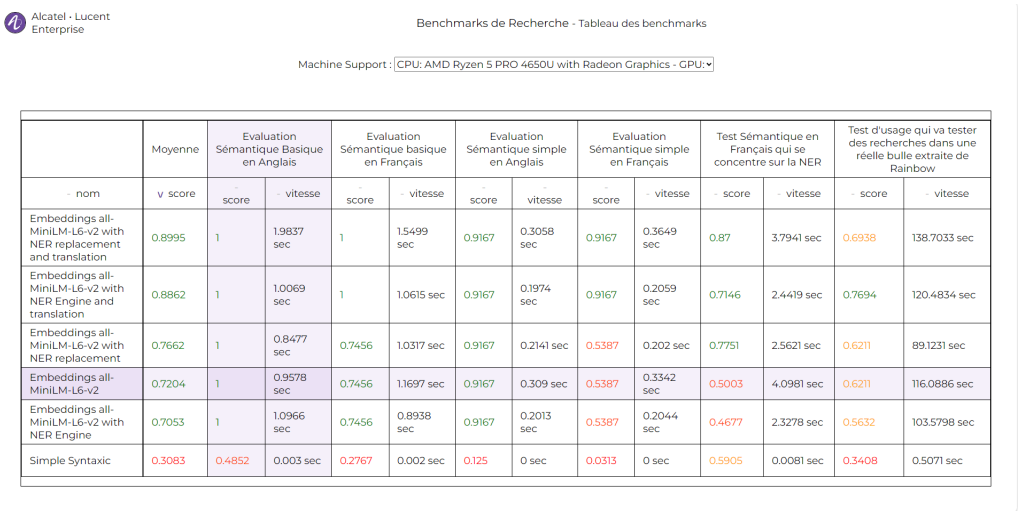
Pour le calcul du score du benchmark de NER, un score F1 est calculé pour l'ensemble des entités détectées.

[34] https://en.wikipedia.org/wiki/F-score

[35] https://en.wikipedia.org/wiki/Edit_distance

2.8.4 - Interface Web pour visualiser les résultats des benchmarks

Afin de visualiser de manière simple les résultats des différents benchmarks, une page web a été développée. Cette page affiche un tableau avec les configurations testées en lignes et les benchmarks en colonnes. Le score moyen pour chaque configuration est présenté en premier lieu, suivi du score et du temps d'exécution pour chaque benchmark. Le temps d'exécution est calculé sur une seule exécution du benchmark, ce qui le rend moins fiable et précis, mais il fournit un bon ordre de grandeur.



	Moyenne	Evaluation Sémantique Basique en Anglais		Evaluation Sémantique basique en Français		Evaluation Sémantique simple en Anglais		Evaluation Sémantique simple en Français		Test Sémantique en Français qui se concentre sur la NER		Test d'usage qui va tester des recherches dans une réelle bulle extraite de Rainbow	
nom	score	score	vitesse	score	vitesse	score	vitesse	score	vitesse	score	vitesse	score	vitesse
Embeddings all-MiniLM-L6-v2 with NER replacement and translation	0.8995	1	1.9837 sec	1	1.5499 sec	0.9167	0.3058 sec	0.9167	0.3649 sec	0.87	3.7941 sec	0.6938	138.7033 sec
Embeddings all-MiniLM-L6-v2 with NER Engine and translation	0.8862	1	1.0069 sec	1	1.0615 sec	0.9167	0.1974 sec	0.9167	0.2059 sec	0.7146	2.4419 sec	0.7694	120.4834 sec
Embeddings all-MiniLM-L6-v2 with NER replacement	0.7662	1	0.8477 sec	0.7456	1.0317 sec	0.9167	0.2141 sec	0.5387	0.202 sec	0.7751	2.5621 sec	0.6211	89.1231 sec
Embeddings all-MiniLM-L6-v2	0.7204	1	0.9578 sec	0.7456	1.1697 sec	0.9167	0.309 sec	0.5387	0.3342 sec	0.5003	4.0981 sec	0.6211	116.0886 sec
Embeddings all-MiniLM-L6-v2 with NER Engine	0.7053	1	1.0966 sec	0.7456	0.8938 sec	0.9167	0.2013 sec	0.5387	0.2044 sec	0.4677	2.3278 sec	0.5632	103.5798 sec
Simple Syntaxic	0.3083	0.4852	0.003 sec	0.2767	0.002 sec	0.125	0 sec	0.0313	0 sec	0.5905	0.0081 sec	0.3408	0.5071 sec

Figure 15: Page de benchmark

On peut trier les différentes configurations sur le score moyen décroissant (par défaut), mais aussi sur le score et la vitesse de chaque benchmark que ce soit un tri croissant ou décroissant.

On peut aussi voir les détails du résultat d'une configuration avec un benchmark en cliquant sur la bonne case du tableau pour chaque type de benchmark différent.

Pour un benchmark de recherche, on va afficher la position/pénalité de chaque recherche testée, et on peut voir les messages résultats en cliquant sur la bonne case.

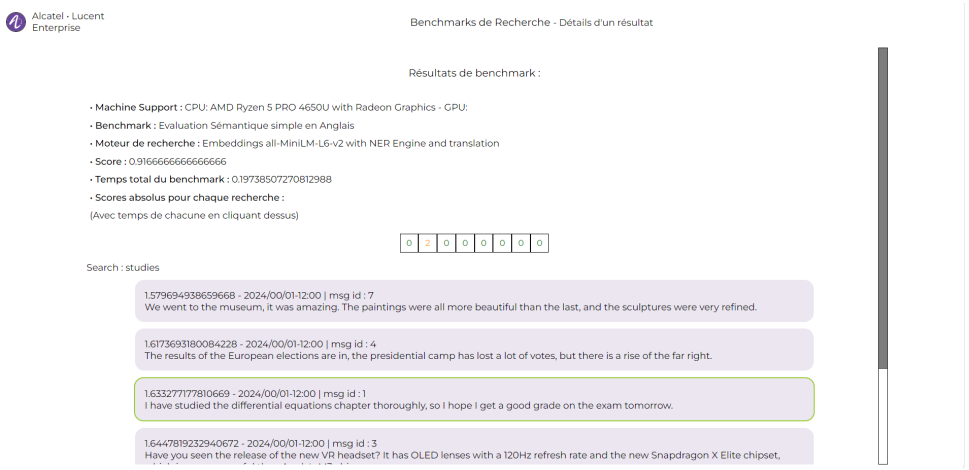


Figure 16: Page détail d'un benchmark de recherche

Pour un benchmark de découpe des conversations, on peut voir les messages colorés d'une couleur différente pour chaque conversation résultats attendus et résultats obtenus côtes à côtes.

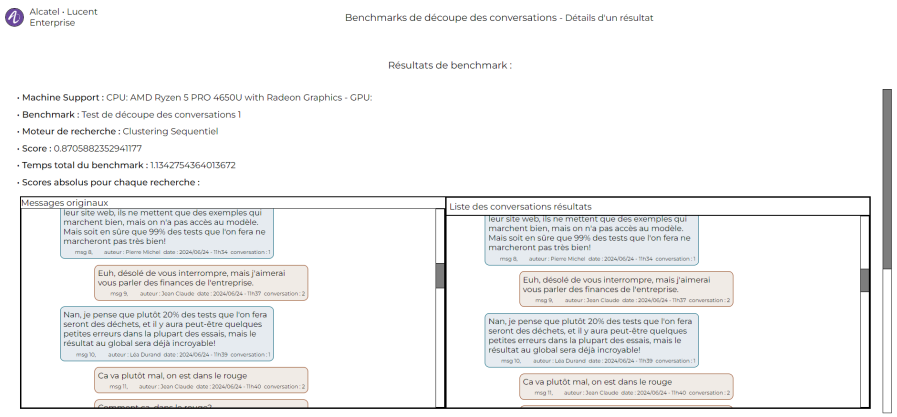


Figure 17: Page détail d'un benchmark de découpe de conversations

Et pour un benchmark de NER, on affiche chaque texte testé en soulignant les entitées attendues et en surlignant les entités obtenues.

2.9 - Détails sur l'application Web pour la démo

2.9.1 - Serveur python

Afin de faciliter la réalisation d'une démo et de créer un environnement de travail permettant l'expérimentation sur le projet, une page web a été développée et connectée via une liaison websocket à un serveur Python. Dans un souci de

flexibilité maximale, le serveur Python est conçu pour être multi-client, permettant ainsi de traiter plusieurs RBI simultanément.

Le serveur websocket est plutôt basique, il écoute les requêtes des clients, pour les petites requêtes comme des transmissions de données, il renvoie directement les données, et sinon, pour les plus grosses requêtes, comme faire une recherche, faire une découpe de conversation, ou bien importer une nouvelle bulle dans une RBI, il transmet la requête à un module qui gère les tâches plus lourdes.

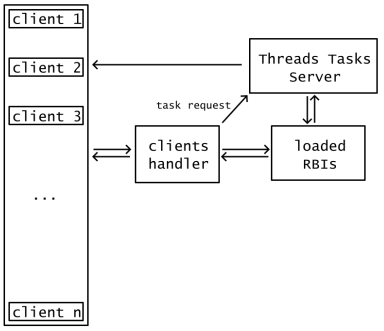


Figure 18: Architecture du serveur python

Le module chargé de la gestion des tâches plus lourdes a été nommé *Threads Tasks Server*. Ce nom reflète sa capacité à gérer plusieurs types de tâches et à offrir une architecture multi-thread permettant l'exécution simultanée de ces tâches. De plus, ce module est équipé d'une file d'attente pour traiter les demandes de tâches excédant les capacités disponibles.

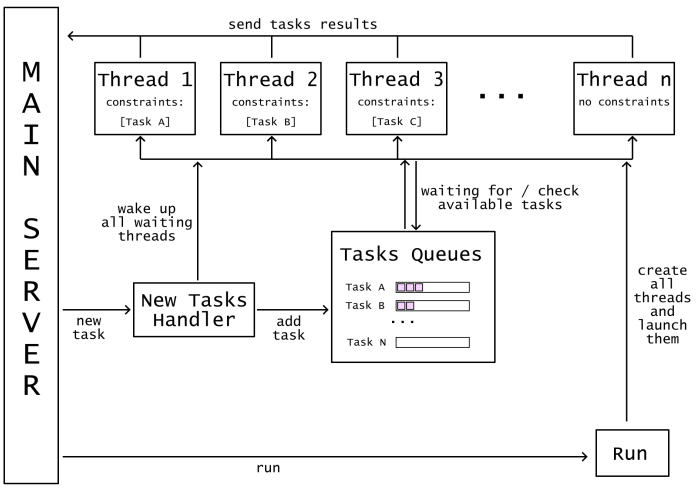


Figure 19: Architecture du module Threads Tasks Server

Chacun des threads peuvent être contraints à ne réaliser que certains types de tâches, cela permet de pouvoir assurer le fait d'avoir au moins un thread qui gère chaque type de tâche à tout instant. Un thread va donc attendre qu'une tâche

soit disponible, va l'éxécuter en envoyant la requête vers le bon moteur qui va s'occuper de la tâche, puis va renvoyer directement les résultats au bon client. Puis il va regarder s'il reste des tâches à faire avant d'attendre qu'on le réveille à nouveau.

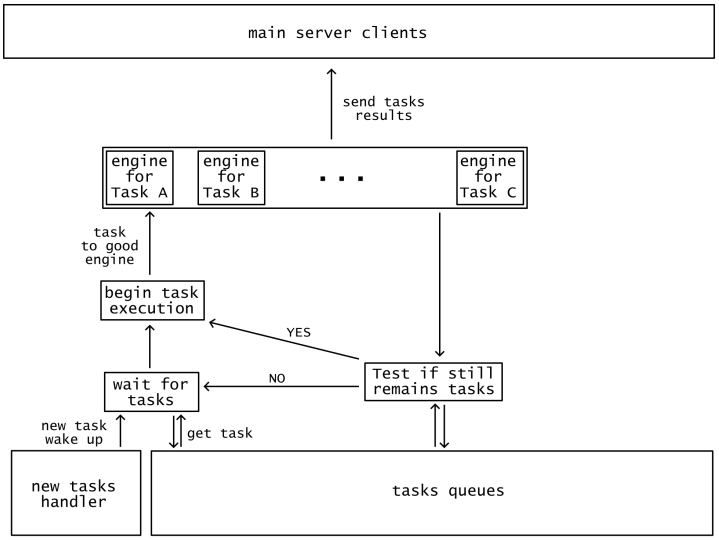


Figure 20: Détail d'un thread dans le module Threads Tasks Server

2.9.2 - Recherche d'un message dans une bulle

Il y a donc dans la démo web une page pour faire des recherches, avec un affichage tout à fait classique, avec une barre de recherche en haut, puis un affichage des résultats en dessous sous forme de liste. Il y a juste une particularité à noter, c'est qu'il faut préciser avec quel utilisateur on effectue la recherche, car sur cette démo web, il n'y a pas de connexion, on ne représente pas un utilisateur. Et ensuite, le moteur de recherche va bien filtrer la recherche uniquement sur les messages dont l'utilisateur a accès. Ce choix se fait sur le menu déroulant à droite de la barre de recherche.

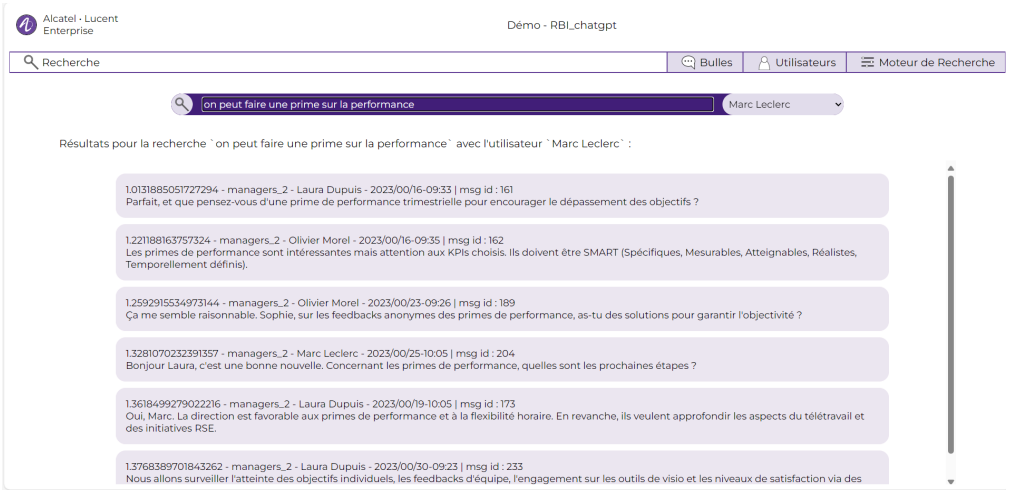


Figure 21: Page de recherche de messages

2.9.3 - Visualisation des bulles et découpe des conversations

On peut aussi explorer les bulles, c'est utile pour préparer des benchmarks, analyser les données ou bien vérifier qu'il n'y a pas de problèmes. Cela se fait par une autre page dans l'application web, accessible par le menu de navigation en haut.

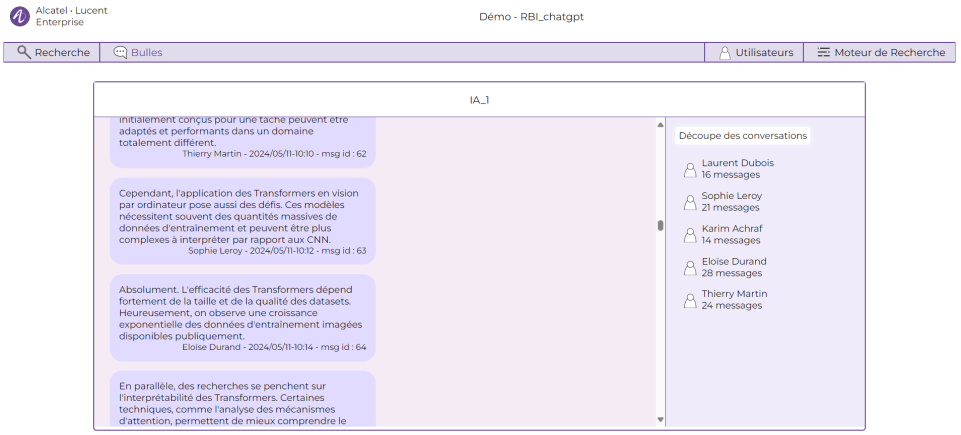


Figure 22: Visualisation d'une bulle

Et il est aussi possible de visualiser une découpe de conversation, en appuyant sur le bouton "découper des conversations" au-dessus de la liste des utilisateurs, cela envoie une requête au serveur, et le serveur renvoie ensuite les résultats quand il a fini.

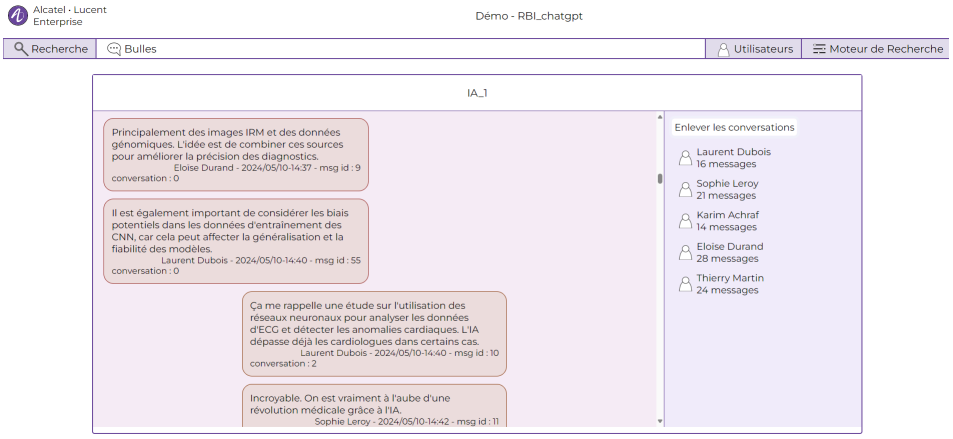


Figure 23: Découpe de conversation dans une bulle

2.10 - Détails sur l'application web pour la création / modification de configurations et pour l'optimisation des hyper-paramètres

2.10.1 - Contexte de cette application Web

Au début, les configurations étaient manipulées et les valeurs modifiées manuellement, selon l'intuition. Cette approche fonctionnait bien dans un premier temps, mais à mesure que les configurations devenaient de plus en plus complexes, il est devenu pertinent de développer une application pour identifier directement les meilleures valeurs des hyper-paramètres des modèles au sein des configurations. Une page web, similaire à celle de la démo, a donc été créée, se connectant au serveur Python via websocket. Cette page permet également de créer, modifier et supprimer des configurations directement depuis l'interface web.

Afin de permettre une telle manipulation des configurations, un système de types a été implémenté, similaire à la définition de classes en programmation orientée objet (voir **annexe 8**). Autour de ces types modélisés, un ensemble complet de fonctions a été développé pour les utiliser, ainsi que pour récupérer et modifier des valeurs dans les configurations en respectant ces types, etc...

2.10.2 - Création et modifications des configurations

Il y a donc une page dans cette application web qui permet de créer, de modifier ou bien de supprimer les différentes configurations pour les trois types de tâches sur lesquelles j'ai développé des moteurs (recherche, NER, découpe de conversation) (exemple de configuration en **annexe 4**).

Grâce au modèle de types énoncé dans la partie précédente, on peut donc facilement créer un modèle de base pour une configuration donnée, avoir les valeurs par défauts, et pouvoir vérifier si les données que rentre l'utilisateur sont valides ou non. On peut donc s'assurer que toutes les configurations qui sont enregistrées sur le disque dur sont valides.

La page web communique avec le serveur python pour récupérer la liste des configurations, pour enregistrer les modifications effectuées à une configuration ou bien pour supprimer une configuration. Ces opérations étant rapides et légères, elles sont effectuées directement et ne passent pas par le système de moteur de

thread présenté plus tôt. En revanche, pour éviter les problèmes de synchronisation et de concurrence liés à la programmation asynchrone, toutes les opérations touchant à la lecture et l'écriture de fichier sont protégées par des mutex.

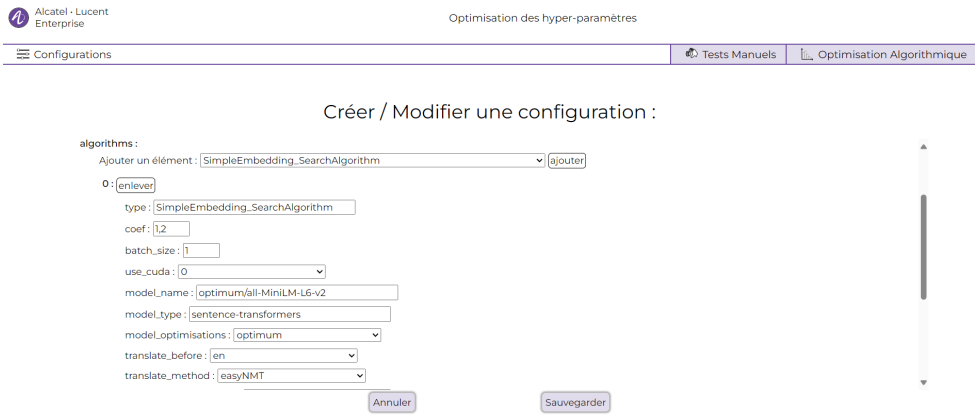


Figure 24: Création / Modification d'une configuration d'un moteur de recherche

2.10.3 - Tests des hyper-paramètres à la main

Pour comprendre et chercher rapidement des valeurs d'hyper-paramètres, il y a une page web pour tester rapidement des valeurs à la main. Quand l'utilisateur clique sur le bouton "Tester", une requête est envoyée au serveur python qui va faire le benchmark et renvoyer les résultats à la page, qui les affiche.

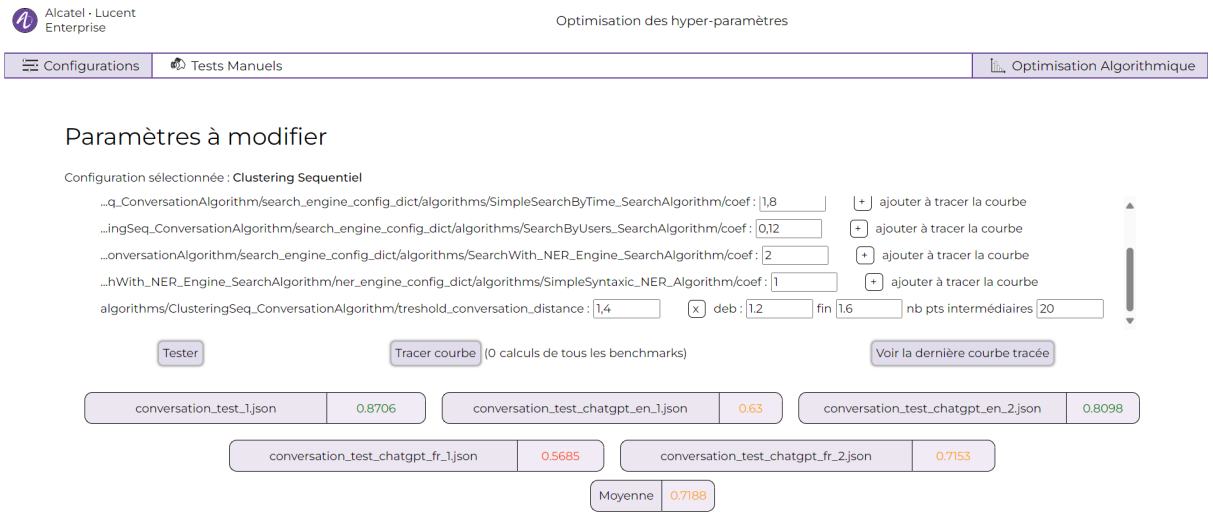


Figure 25: Tests des hyper-paramètres à la main

Sur cette page web, il y a aussi une autre fonctionnalité qui est pratique pour trouver rapidement de bonnes valeurs qui fonctionnent bien pour les

hyper-paramètres, c'est le fait de pouvoir tracer une courbe d'évolution du scores des benchmarks en fonction des valeurs prises par un paramètre sur un intervalle donné. Pour faire cela, il faut cliquer sur le bouton "+" avec le texte "ajouter à la courbe" sur la ligne de l'hyper-paramètre que l'on veut tester, puis de renseigner l'intervalle sur lequel on veut le tester, ainsi que le nombre de points dans l'intervalle.

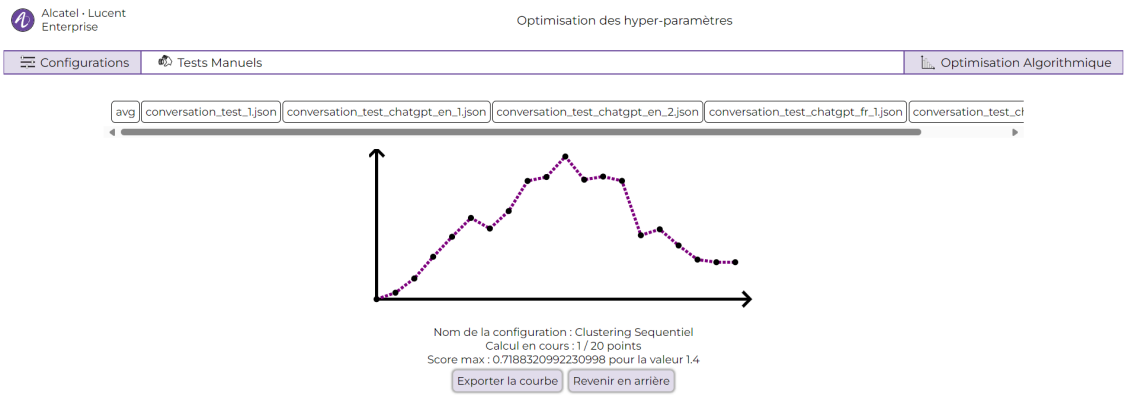


Figure 26: Tracé d'une évolution de valeurs d'un hyper-paramètre
Ici, on peut voir l'évolution du score moyen des benchmarks pour chaque valeur d'un hyper-paramètre

2.10.4 - Optimisation Algorithmique des Hyper-Paramètres

Enfin, il y a une dernière page dans cette webapp qui permet d'utiliser un algorithme d'optimisation des hyper-paramètres pour ne pas avoir à le faire à la main. Il est possible de sélectionner les hyper-paramètres à optimiser, indiquer les valeurs que ces paramètres peuvent prendre, sélectionner comment le score est construit avec tel ou tel benchmark et en spécifiant les coefficients des benchmarks pour le calcul du score à optimiser.

Plusieurs algorithmes sont disponibles:

Un premier tout simple, appelé *Random Search, va juste* tester des valeurs aléatoires dans l'espace des valeurs possibles. L'idée, c'est que comme en général ces espaces ne sont pas si grands, avec un grand nombre de tests, on devrait atteindre certaines zones de cet espace où il y a les meilleures valeurs possibles.

Un autre, appelé *Grid Search*, va explorer sous la forme d'une grille en *n* dimensions l'espace des valeurs possibles pour les hyper-paramètres. En séléctionnant l'algorithme *Recursive Grid Search*, il est possible de faire une recherche récursive, dans le sens où après avoir fait une première exploration en grille, l'algorithme va refaire une exploration de grille autour du meilleur point

trouvé. Cette approche s'apparente un peu à un algorithme de force brute dans l'idée de tester un petit sous-ensemble de toutes les possibilités possibles.

Un autre algorithme, appelé *Gaussian Exploration*, se décompose en deux phases: une phase d'initialisation et une phase d'exploration. Lors de la phase d'initialisation, l'algorithme va tester plusieurs valeurs aléatoires dans l'espace des valeurs possibles. Ensuite, grâce à ces premières valeurs initiales, l'algorithme va essayer de modéliser l'espace du résultat avec une somme moyennée de fonctions gaussiennes de la forme:

$$\text{minimal\_value} + (\text{bell\_center\_function\_value} - \text{minimal\_value}) \cdot \exp(-\frac{||x - \text{bell\_center}||^2}{2.0 * \sigma^2})$$

Formule 2: Expression d'une fonction Gaussienne utilisée pour la modélisation d'un espace

avec

$$\sigma^2 = \left(\frac{1}{2\pi}\right) \cdot \left(\frac{\text{space\_volume} * (\text{average\_value} - \text{minimal\_value})}{\text{bell\_center\_function\_value} - \text{minimal\_value}}\right)^{\frac{2}{n}}$$

Formule 3: Expression de sigma carré de la formule 2

Chaque fonction gaussienne est liée à un point déjà calculé, *bell_center_function_value* est la valeur du point calculé et *bell_center* l'endroit du point calculé, ensuite, plus généralement, *space_volume* est le volume de l'espace des possibilités, *minimal_value* est la valeur minimale des points calculés et *average_value* est la valeur moyenne des points calculés. L'idée derrière ces fonctions gaussiennes était d'avoir une fonction en forme de cloche avec une valeur précise en un point, une valeur moyenne donnée et une valeur minimale donnée, d'où la formule un peu compliquée pour σ^2 qui provient d'une résolution d'équation pour obtenir l'effet souhaité.

Ensuite, la phase d'exploration va jouer entre explorer des nouvelles zones peu explorées et explorer les zones qui ont des bonnes valeurs. Et la modélisation est mise à jour après chaque nouveau point calculé.

Malheureusement, les premiers tests n'ont pas été très concluants et cet algorithme n'avait pas l'air beaucoup plus efficace que l'exploration aléatoire. Mais il est sans doute possible qu'en utilisant mieux cette approche, on devrait pouvoir avoir de bons résultats intéressants.

Un dernier algorithme, appelé *Linear Individual Searchs*, va reprendre la méthode naïve d'optimisation des hyper-paramètres en esseyant de chercher la meilleure valeur possible, mais en ne prenant en compte que la meilleure valeur trouvée à chaque moment pour les autres hyper-paramètres. Théoriquement, cela fonctionne bien tant que les hyper-paramètres ne sont pas trop interdépendants

les uns des autres. Ce qui n'avait pas l'air d'être le cas ici, donc au final, c'est cet algorithme qui fonctionne le mieux lors de mes quelques tests.

Ces algorithmes d'optimisation automatique des paramètres optimisables des configurations sont intrinsèquement liés à l'ensemble des benchmarks sélectionnés pour l'optimisation. Une bonne pratique est donc de préparer un ensemble de benchmarks "d'entraînement", et en garder un autre "de test". Ces benchmarks doivent alors bien représenter tous les types de situations représentatives auquelles les moteurs devront répondre.

2.11 - Intégration dans Rainbow

2.11.1 - Bot avec SDK Rainbow dans une Sandbox

La seule manière d'intégrer au mieux le travail effectué durant le stage dans l'application Rainbow a été de créer un bot dédié. Ce bot se connecte au serveur Python, permettant ainsi d'effectuer des recherches via le moteur présenté précédemment et de renvoyer les résultats sous forme de message.

Cela a été réalisé par un script en C# qui utilise l'API SDK de Rainbow et qui se connecte via une connexion socket au serveur python. Une version modifiée du serveur python a aussi été développée.

Le petit défaut, c'est que cela ne permet pas de concevoir une interface adaptée pour effectuer une recherche. Actuellement, l'utilisateur doit soumettre une commande sous la forme : "`\search texte à rechercher`". Le bot renvoie alors les résultats sous la forme d'un message : "`résultat de la recherche : 1) message 1. 2) Message 2. …`". Par ailleurs, l'application Rainbow manque d'une fonctionnalité particulièrement utile dans ce type de situation, que l'on trouve par exemple dans l'application Discord, une plateforme de messagerie instantanée et de VoIP populaire auprès du grand public. Cette fonctionnalité permet de créer un lien cliquable dans un message, redirigeant directement vers ce message au sein de l'application.

Annexes

Table des annexes

Documents annexes

Annexe 1: Architecture / fichiers du projet

Voici l'architecture globale de mon projet. N'y sont représentés que les fichiers / dossiers les plus importants accompagnés d'une explication / description.

+ **ALE-SemanticSearch**
| **readme.md** *(document expliquant tout ce qu'il y a à savoir concernant ce projet d'un point de vue utilisateur/développeur)*
| **requirements.txt** *(liste des bibliothèques python nécessaires à installer pour pouvoir exécuter les différents scripts python)*
+---**cache**
| ... *(données cachées ici pour accélérer certains aspects de mes programmes)*
+---**Configs_ConversationEngines**
| ... *(fichiers de configurations pour mon moteur de découpe des conversations)*
+---**Configs_NER_Engines**
| ... *(fichiers de configurations pour mon moteur de reconnaissance d'entités nommées)*
+---**Configs_SearchEngines**
| ... *(fichiers de configurations pour mon moteur de recherche)*
+---**DataToImport**
| ... *(données de conversations sous le format d'export standard de Rainbow)*
+---**SavedImported_RBI**
| ... *(données de conversations importées sauvegardées sous un format simple à importer)*
+---**models**
... *(dossier où les différents modèles d'embedding ou de traduction utilisés seront téléchargés en local)*
+---**profiling_results** *(Dossier qui contient les donées de profiling de mes scripts python + une page web pour pouvoir les visualiser / les analyser)*
+---**results**
| ... *(Dossier où sont sauvegardées les données de profiling des différentes éxécutions de certains de mes programmes python)*
+---**web_visualisation** *(Page web pour visualiser et analyser le temps d'exécution de différentes parties de certains de mes programmes python)*
| **index.html**
+---**css**
| ... *(styles css)*
+---**js**
| ... *(scripts js)*
+---**PythonScripts** *(Dossier où il y a tous les différents programmes python de ce projet)*
| **bubble.py** *(Contient la classe Bulle définissant une bulle d'un environnement Rainbow)*
| **config.json** *(Fichier de configuration pour le serveur websocket python et les scripts sous-jacents, contient les configurations websocket ainsi que tous les chemins vers les données à sauvegarder ou à charger)*
| **config.py** *(Contient une classe représentant le fichier de configuration ci-dessus)*
| **config_socket_api.json** *(Fichier de configuration, mais pour le serveur socket auquel on va connecter le client C# qui va intéragir avec l'API SDK Rainbow)*
| **config_socket_api.py** *(Contient une classe représentant le fichier de configuration ci-dessus)*
| **conversations_algorithms.py** *(Contient les différents algorithmes de découpe de conversations)*
| **conversations_engine.py** *(Contient le moteur de découpe de conversations)*
| **embeddings_cache.py** *(Contient un module qui va gérer les caches des embeddings des messages, pour ne pas avoir à les recalculer à chaque fois)*
| **embedding_calculator.py** *(Contient une interface générique pour calculer des embeddings depuis du texte)*
| **global_variables.py** *(Module simple à utiliser pour gérer des variables globales facilement et proprement dans un projet complexe en python)*
| **language_translation.py** *(Contient un module qui va gérer la traduction)*
| **lib.py** (*Librairies de fonctions génériques non spéciales)*
| **lib_date_recognition.py** *(Module de Reconnaissance de dates dans du texte basé sur des règles syntaxiques)*

| **lib_embedding.py** *(Contient la classe MessageEmbedding ainsi que quelques fonctions pour calculer des distances entre embeddings)*
| **lib_hp_optimization.py** *(Contient des fonctions pour faire de l'optimisation d'hyper-paramètres)*
| **lib_main_server.py** *(Contient la classe abstraite MainServer)*
| **lib_number_converter.py** *(Contient un module de conversion de nombres écrits textuellement en format numérique, pour le script lib_date_recognition.py, ex: "Quatre cent vingt-trois" -> 423)*
| **lib_types.py** *(Contient une représentation des types et architectures de configurations pour l'optimisation des hyper-paramètres)*
| **main_convert_data_to_rbi.py** *(Script principal à éxécuter / Point d'entrée pour importer les données du format d'exportation de base de bulles rainbow en données structurées facilement utilisables pour ce projet)*
| **main_server_for_api.py** *(Script principal à éxécuter / Point d'entrée pour le serveur pour le bot / l'API Rainbow)*
| **main_server_for_webapp.py** *(Script principal à éxécuter / Point d'entrée pour la webapp démo et l'optimisation des hyper-paramètres)*
| **main_tests_benchmarks.py** *(Script principal à éxécuter / Point d'entrée pour lancer les benchmarks sur les différentes configurations de mes moteurs (recherche, conversations et NER))*
| **message.py** (*Contient la classe Message définissant un message d'un environnement Rainbow)*
| **ner_algorithms.py** *(Contient les différents algorithmes de reconnaissance d'entités nommées)*
| **ner_engine.py** *(Contient le moteur de reconnaissance d'entités nommées)*
| **profiling.py** *(Contient un module pour faire du profiling de code et voir / analyser facilement le temps d'éxecution de mes différents portions de codes python -> affichage des résultats via une page web statique dans le dossier profiling_results)*
| **rainbow_instance.py** *(Contient la classe RainbowInstance définissant une instance Rainbow (RBI) / un environnement Rainbow)*
| **search_algorithm.py** *(Contient les différents algorithmes de recherche)*
| **search_engine.py** *(Contient le moteur de recherche)*
| **threads_tasks_server.py** *(Contient le serveur multi-tâches qui va éxécuter sur plusieurs threads les tâches demandées avec un système de file d'attente)*
| **user.py** *(Contient la classe User définissant un utilisateur d'un environnement Rainbow)*
+---**Rainbow_App** *(Dossier contenant la partie C#)*
| **api_config.json** *(Fichier de configuration pour l'application C#)*
| **MainRainbowAppSDKApi.cs** *(Programme principal C# client socket au serveur python et utilisant l'API SDK Rainbow)*
+---**TestsBenchmarks**
| **...** *(Fichiers de benchmarks pour les différentes tâches: search, NER et découpe de conversation)*
+---**WebApp** *(Dossier contenant la WebApp principale de mon projet, avec une démo, des benchmarks, et la page d'optimisation des hyper-paramètres. Les pages de la démo et de l'optimisation des hyper-paramètres nécessitent que le serveur python des webapps soit lancé)*
| **benchmarks.html** *(page d'accueil des benchmarks, redirigeant vers les trois différentes pages de benchmark juste ci-dessous)*
| **benchmarks_auto_ner.html** *(Page de benchmark de la tâche de NER)*
| **benchmarks_conversation_cutting.html** *(page de benchmark de la tâche de découpe des conversations)*
| **benchmarks_semantic_search.html** *(page de benchmark de la tâche de recherche)*
| **demo.html** *(page web de la démo pour la recherche)*
| **explications.html** *(page web qui contient des explications générales sur comment fonctionne ce projet)*
| **hyper_parameters_optimisation.html** *(page web qui contient l'application d'optimisation des hyper-paramètres)*
| **index.html** *(page d'accueil redirigeant vers les benchmarks, la démo, ou bien les explications)*
+---**css** *(Dossier avec les styles CSS)*
| ...
+---**data_benchmarks** *(Dossier où les scripts python sauvegardes les résultats des benchmarks)*
| **benchmark_results.js** *(Script js contenant les données des deux fichiers json ci-dessous dans des variables, pour pouvoir y accéder facilement depuis les autres scripts js, fichier généré dynamiquement par les scripts python)*
| **benchmark_results.json** *(Données brutes json contenant les résultats des benchmarks testés)*
| **data_benchmarks.json** *(Données brutes json contenant les infos sur les benchmarks testés)*
+---**js** *(Dossier avec les scripts js)*
| ...

Annexe 2: Exemple de données sous le format d'export d'une bulle Rainbow depuis l'application.

Cet exemple de discussion a été généré par ChatGPT et a servi pour faire des tests pour évaluer différentes configurations de moteurs de recherche.

```
…
Sophie Leroy Tuesday, June 11, 2024 10:06 AM
Avez-vous des exemples précis d'applications des Transformers en vision par
ordinateur ?

Eloïse Durand Tuesday, June 11, 2024 10:08 AM
Oui, des modèles comme DETR (Detectron Transformer) ont obtenu des performances
remarquables dans la détection d'objets, surpassant même les approches basées sur
CNN dans certains cas.

Thierry Martin Tuesday, June 11, 2024 10:10 AM
C'est fascinant de voir comment des modèles initialement conçus pour une tâche
peuvent être adaptés et performants dans un domaine totalement différent.

Sophie Leroy Tuesday, June 11, 2024 10:12 AM
Cependant, l'application des Transformers en vision par ordinateur pose aussi des
défis. Ces modèles nécessitent souvent des quantités massives de données
d'entraînement et peuvent être plus complexes à interpréter par rapport aux CNN.

Eloïse Durand Tuesday, June 11, 2024 10:14 AM
Absolument. L'efficacité des Transformers dépend fortement de la taille et de la
qualité des datasets. Heureusement, on observe une croissance exponentielle des
données d'entraînement imagées disponibles publiquement.

Thierry Martin Tuesday, June 11, 2024 10:16 AM
En parallèle, des recherches se penchent sur l'interprétabilité des Transformers.
Certaines techniques, comme l'analyse des mécanismes d'attention, permettent de
mieux comprendre le raisonnement derrière les prédictions du modèle.

Laurent Dubois Tuesday, June 11, 2024 10:18 AM
C'est une piste intéressante. L'interprétabilité des modèles d'IA est cruciale, en
particulier dans des domaines sensibles comme la vision par ordinateur médicale.

Sophie Leroy Wednesday, June 12, 2024 1:15 PM
On a beaucoup parlé de vision par ordinateur et de CNN. Je suis curieuse, est-ce
que quelqu'un a des connaissances sur l'utilisation des Réseaux Adversaires
Génératifs (GAN) pour la génération de données synthétiques ?

Eloïse Durand Wednesday, June 12, 2024 1:17 PM
Oui, les GAN sont un domaine de recherche très actif. Ils permettent de générer
des données réalistes qui peuvent être utilisées pour augmenter la taille et la
diversité des datasets d'entraînement.
…
```

Annexe 3: Exemple de dictionnaire de définition de NER

Ceci est un exemple de dictionnaire NER - définition que l'on peut utiliser dans la phase de pré-traitement lors d'un algorithme de recherche pour remplacer des expressions par leur définition. Ce dictionnaire a été constuit autour d'une discussion et un ensemble de recherches précises.

```
{
    "EDT": "emplois du temps",
    "JIRA": "système de suivi de bugs, de gestion des incidents et de gestion de projets",
    "LinkedIn Learning": "online learning platform that provides video courses taught by industry
experts",
    "Coursera": "entreprise numérique proposant des formations en ligne ouvertes à tous",
    "KPI": "indicateur clé de performance",
    "KPIs": "indicateurs clés de performance",
    "RSE": "responsabilité sociétale des entreprises",
    "GAN": "réseau adversaire génératif",
    "GANs": "réseaux adversaires génératifs",
    "SurveyMonkey": "site de sondage en ligne gratuit avec sondages personalisables et série de
programmes statistiques",
    "ViT": "Vision Transformer",
    "DETR": "Detection Transformer, Detectron"
}
```

Annexe 4: Exemple de configuration de moteur de recherche

Ceci est un exemple de fichier de configuration qui représente un moteur de recherche.

```
{
    "config_name": "Embeddings all-MiniLM-L6-v2 with NER Engine and translation",
    "max_message_length": 200,
    "nb_search_results": 6,

    "algorithms": [

        {
            "type": "SimpleEmbedding_SearchAlgorithm",
            "coef": 1.2,
            "model_name": "optimum/all-MiniLM-L6-v2",
            "model_type": "sentence-transformers",
            "batch_size": 1,
            "use_cuda": 0,
            "model_optimisations": "optimum",
            "translate_before": "en",
            "translate_method": "easyNMT"
        },

        {
            "type": "SearchWith_NER_Engine_SearchAlgorithm",
```

```
            "coef": 2.0,
            "ner_engine_config_dict":

                {
                    "config_name": "SimpleSyntaxicNER",

                    "algorithms": [
                        {
                            "type": "SimpleSyntaxic_NER_Algorithm",
                            "coef": 1.0
                        }
                    ]
                }
        }
    ]
}
```

Annexe 5: Exemple de benchmark de recherche

```
{
    "type": "semantic_search",
    "title": "Evaluation Sémantique basique en Français",
    "description": "Test de recherche sémantique basique avec messages et des recherches en français
sémantiquement différents, avec des recherches exprimant un message sémantique proche du message
recherché. Le dataset a été généré par Gemini/ChatGPT et préparé/modifié/traité/controllé par Nathan
Cerisara.",
    "rbi_path": "RBI_base_semantic_fr",
    "searchs": [

        {
            "search_input": "Le repas était excellent, je vous remercie.",
            "user_id": 0,
            "awaited_result_message": 0
        },

        {
            "search_input": "J'espère que vous me rembourserez l'argent que vous me devez.",
            "user_id": 3,
            "awaited_result_message": 3
        },

        {
            "search_input": "J'ai beaucoup révisé pour ces exams, c'était compliqué.",
            "user_id": 8,
            "awaited_result_message": 8
        },

        {
            "search_input": "L'art contemporain, c'est différent de l'art classique quand même.",
            "user_id": 23,
            "awaited_result_message": 23
        },
```

```
        {
            "search_input": "Est-ce que vous pensez que l'on pourra vivre dans l'espace plus tard?",
            "user_id": 27,
            "awaited_result_message": 27
        }

    ]
}
```

Annexe 6: Exemple de benchmark de NER

```
{
    "type": "NER",
    "title": "Test de NER 1",
    "description": "On va avoir des messages, et le but, va être de détecter les entités pour chaque message.",
    "tests": [

        {
            "text": "Bonjour, j'ai appelé Jean Claude hier, on a pu discuter du développement de Rainbow hier, on a été bien productif, mais il y a pierre paul dupont qui est venu nous embêter à la fin.",
            "entities": [
                [21, "Jean Claude", "PERS"],
                [76, "Rainbow", "PROD"],
                [127, "pierre paul dupont", "PERS"]
            ]
        },

        ...
    ]
}
```

Annexe 7: Exemple de benchmark de Découpe de Conversation

```
{
    "type": "conversation",
    "title": "Test de découpe des conversations 1",
    "description": "Il va y avoir une liste de messages, et le but est de découper cette liste de messages en plusieurs conversations.",
    "messages": [

        {
            "id": "0",
            "content": "Bonjour, salut tout le monde, comment ca va?",
            "author": "Jean Dupont",
            "date": "2024/06/24 - 11h23",
            "answer_message": ""
```

```
        },

        {
            "id": "1",
            "content": "Salut, ca va bien, et toi?",
            "author": "Léa Durand",
            "date": "2024/06/24 - 11h24",
            "answer_message": 0
        },

        {
            "id": "2",
            "content": "Coucou!",
            "author": "Pierre Michel",
            "date": "2024/06/24 - 11h24",
            "answer_message": 0
        },

        {
            "id": "3",
            "content": "Ca va bien, merci.",
            "author": "Jean Dupont",
            "date": "2024/06/24 - 11h26",
            "answer_message": 1
        },

        {
            "id": "4",
            "content": "Du coup, je voulais vous parler des nouvelles IA génératives, vous en pensez
quoi?",
            "author": "Jean Dupont",
            "date": "2024/06/24 - 11h26",
            "answer_message": ""
        },

        {
            "id": "5",
            "content": "C'est vraiment impressionnant, le développement de ces nouvelles technologies,
mon fils a réussi à mettre en place un système de génération de musique de bonnes qualités avec une
pipeline génération de paroles avec Gemma, génération de musique avec Udio, génération d'image de
Cover avec Stable-Diffusion, et quelques autres éléments. Et il génère environ une 20aine de musiques
par jour, qu'il publie ensuite sur Youtube. Il a pas mal de fans maintenant, ca commence à être une
petite star!",
            "author": "Léa Durand",
            "date": "2024/06/24 - 11h28",
            "answer_message": ""
        },

        {
            "id": "6",
            "content": "Perso, je trouve ca cool, mais c'est pas encore au point, par exemple, la
génération d'images ou de vidéos, on peut très facilement remarquer énormément de défauts. C'est
nul.",
            "author": "Pierre Michel",
            "date": "2024/06/24 - 11h29",
```

```
            "answer_message": ""
        },

        {
            "id": "7",
            "content": "Eh, les résultats sont incroyables quand même! Même la génération de vidéos est franchement bluffante, t'as pas vu les clips de démo de l'IA SORA?",
            "author": "Léa Durand",
            "date": "2024/06/24 - 11h30",
            "answer_message": ""
        },

        {
            "id": "8",
            "content": "Bof, c'est normal que dans leur communication et leur site web, ils ne mettent que des exemples qui marchent bien, mais on n'a pas accès au modèle. Mais soit en sûre que 99% des tests que l'on fera ne marcheront pas très bien!",
            "author": "Pierre Michel",
            "date": "2024/06/24 - 11h34",
            "answer_message": ""
        },

        {
            "id": "9",
            "content": "Euh, désolé de vous interrompre, mais j'aimerai vous parler des finances de l'entreprise.",
            "author": "Jean Claude",
            "date": "2024/06/24 - 11h37",
            "answer_message": ""
        },

        {
            "id": "10",
            "content": "Nan, je pense que plutôt 20% des tests que l'on fera seront des déchets, et il y aura peut-être quelques petites erreurs dans la plupart des essais, mais le résultat au global sera déjà incroyable!",
            "author": "Léa Durand",
            "date": "2024/06/24 - 11h39",
            "answer_message": ""
        },

        {
            "id": "11",
            "content": "Ca va plutôt mal, on est dans le rouge",
            "author": "Jean Claude",
            "date": "2024/06/24 - 11h40",
            "answer_message": ""
        },

        {
            "id": "12",
            "content": "Comment ça, dans le rouge?",
            "author": "Jean Dupont",
            "date": "2024/06/24 - 11h41",
            "answer_message": 11
```

```
        },

        {
            "id": "13",
            "content": "On a eu une nouvelle dépense s'élevant à environ 31 milles euros de coûts
d'API OpenAI qui n'avaient pas été prévues dans le budget du semestre derniers provenant de votre
secteur, Jean Dupont.",
            "author": "Jean Claude",
            "date": "2024/06/24 - 11h44",
            "answer_message": ""
        },

        {
            "id": "14",
            "content": "Comment ça qui n'ont pas été prévues ?! L'année dernière j'avais dit que
j'allais les rajouter dans nos pipelines de production!",
            "author": "Jean Dupont",
            "date": "2024/06/24 - 11h47",
            "answer_message": ""
        },

        {
            "id": "15",
            "content": "Eh, je viens de regarder, il y a aussi des modèles open-source de génération
de vidéo qui sont prometteurs!",
            "author": "Léa Durand",
            "date": "2024/06/24 - 11h50",
            "answer_message": ""
        },

        {
            "id": "16",
            "content": "Euh, et qu'est-ce qu'il faut comme config pour les faire tourner?",
            "author": "Pierre Michel",
            "date": "2024/06/24 - 11h41",
            "answer_message": ""
        },

        {
            "id": "17",
            "content": "Et bien apparemment ils n'ont pas pris en compte ce que tu as dit. D'ailleurs
c'est quoi cette histoire de modèles open-source, c'est du gratuit c'est ça? Ca permet de faire des
économies?",
            "author": "Jean Claude",
            "date": "2024/06/24 - 11h43",
            "answer_message": ""
        },

        {
            "id": "18",
            "content": "Mais comment ça ils n'ont pas pris en compte ce que j'ai dit, je les ai
prévenus pleins de fois !?",
            "author": "Jean Dupont",
            "date": "2024/06/24 - 11h45",
            "answer_message": ""
```

```
        },

        {
            "id": "19",
            "content": "Ca va encore retomber sur moi toute cette histoire...",
            "author": "Jean Dupont",
            "date": "2024/06/24 - 11h50",
            "answer_message": ""
        },

        {
            "id": "20",
            "content": "Ca dépend, ca peux vite couter cher en facture électrique et en investissement
serveurs les modèles open-source quand même!",
            "author": "Pierre Michel",
            "date": "2024/06/24 - 11h51",
            "answer_message": ""
        },

        {
            "id": "21",
            "content": "Bonjour, suite à un déficit budgétaire délicat, j'ai la tristesse de vous
annoncer que votre camarade Jean Dupont quitte ses fonctions ce soir.",
            "author": "Gérard Pont",
            "date": "2024/06/25 - 07h54",
            "answer_message": ""
        },

        {
            "id": "22",
            "content": "Mais quoi !? Ce n'est pas de ma faute si des personnes de cette entreprises
n'écoutent pas ce que je leur dit !",
            "author": "Jean Dupont",
            "date": "2024/06/25 - 08h03",
            "answer_message": ""
        },

        {
            "id": "23",
            "content": "Je n'ai pas à écouter vos excuses Mr. Dupont. Rangez votre bureau, rendez nous
les affaires que nous vous avons prêtées, et partez d'ici !",
            "author": "Gérard Pont",
            "date": "2024/06/25 - 08h05",
            "answer_message": ""
        },

        {
            "id": "24",
            "content": "Au revoir, Jean. Bonne chance.",
            "author": "Pierre Michel",
            "date": "2024/06/25 - 08h12",
            "answer_message": ""
        },

        {
```

```
            "id": "25",
            "content": "Sinon, maintenant que Mr. Dupont est parti, c'était une très bonne idée de sa
part d'utiliser un LLM pour aider à générer tous les documents de travail de cette entreprise, creusez
cette idée, et trouvez moi un LLM qui coûte pas cher.",
            "author": "Gérard Pont",
            "date": "2024/06/25 - 10h00",
            "answer_message": ""
        },

        {
            "id": "26",
            "content": "Monsieur, vous souhaitez utiliser des LLM en local ou avec des API?",
            "author": "Léa Durand",
            "date": "2024/06/25 - 10h02",
            "answer_message": ""
        },

        {
            "id": "27",
            "content": "Je n'en ai rien à faire, le plus important est qu'ils soient performants, tout
en étant le moins cher possible.",
            "author": "Gérard Pont",
            "date": "2024/06/25 - 10h05",
            "answer_message": ""
        },

        {
            "id": "28",
            "content": "Mr. Pont, qu'en est-il du budget de notre équipe / division pour le semestre
prochain?",
            "author": "Pierre Michel",
            "date": "2024/06/25 - 10h22",
            "answer_message": ""
        },

        {
            "id": "29",
            "content": "Vous avez le budget du semestre précédent augmenté de 5% pour couvrir les
nouvelles dépenses.",
            "author": "Gérard Pont",
            "date": "2024/06/25 - 10h31",
            "answer_message": ""
        },

        {
            "id": "30",
            "content": "Ah oui, ok, ca ne sera toujours pas assez pour tout couvrir monsieur.",
            "author": "Pierre Michel",
            "date": "2024/06/25 - 10h33",
            "answer_message": ""
        },

        {
            "id": "31",
```

```
            "content": "Débrouillez vous, optimisez vos dépenses ou parvenez moi une proposition de licenciement. C'est votre boulot Mr. Michel.",
            "author": "Gérard Pont",
            "date": "2024/06/25 - 10h35",
            "answer_message": ""
        }

    ],

    "conversation_colors": {
        "0":  0,
        "1":  0,
        "2":  0,
        "3":  0,
        "4":  1,
        "5":  1,
        "6":  1,
        "7":  1,
        "8":  1,
        "9":  2,
        "10": 1,
        "11": 2,
        "12": 2,
        "13": 2,
        "14": 2,
        "15": 1,
        "16": 1,
        "17": 1,
        "18": 1,
        "19": 2,
        "20": 1,
        "21": 3,
        "22": 3,
        "23": 3,
        "24": 3,
        "25": 4,
        "26": 4,
        "27": 4,
        "28": 5,
        "29": 5,
        "30": 5,
        "31": 5
    },

    "conversation_results": [
        ["0", "1", "2", "3"],
        ["4", "5", "6", "7", "8", "10", "15", "16", "17", "20"],
        ["9", "11", "12", "13", "14", "18", "19"],
        ["21", "22", "23", "24"],
        ["25", "26", "27"],
        ["28", "29", "30", "31"]
    ],

    "conversation_results_advanced": [
        ["0", "1", "2", "3"],
```

```
        ["4", "5", "6", "7", "8", "10", "15", "16", ["17", [67, 188]], "20"],
        ["9", "11", "12", "13", "14", ["17", [0, 66]], "18", "19"],
        ["21", "22", "23", "24"],
        ["25", "26", "27"],
        ["28", "29", "30", "31"]
    ]

}
```

Annexe 8: Modélisation des types des configurations en python

```python
# Classe abstraite des paramètres de la config pour les algorithmes et moteurs
# la clé correspond au nom du paramètre de la config, et la valeur est un tuple indiquant
#   (
#     0 = type de la valeur
#     1 = booléen indiquant si c'est une valeur à optimiser
#     2 = contraintes sur les valeurs prises
#           * Si c'est une liste, la valeur doit être dans cette liste
#           * Si c'est un dictionnaire avec la forme {"type": "interval", "min": a, "max": b}, la
valeur doit être entre ces deux valeurs
#           * Sinon, c'est un None / null, et pas de contraintes particulières. On s'en fout, le
paramètre peut prendre n'importe quelle valeur (sauf exception bien sûr, par exemple, pour les types
qui sont dans la liste type, et surtout les listes)
#     3 = valeur par défaut,
#     4 = si c'est une clé nécessaire dans une config ou non. Si elle vaut 0, cela veut dire qu'on
s'en fout si le paramètre n'est pas dans la config.
#   )
#
TYPES: dict[str, dict[str, tuple[str, int, Optional[Any], Optional[Any], int]]] = {

    "SimpleTimeDifferences_ConversationAlgorithm": {
        "type": ("string", 0, None, "SimpleTimeDifferencesAlgorithm", 1),
        "coef": ("number", 1, None, 1.0, 1),
        "treshold_value": ("number", 1, None, 60, 1),
        "treshold_type": ("string", 1, ["seconds", "minutes", "hours", "days"], "minutes", 1)
    },

    "ClusteringFusion_ConversationAlgorithm": {
        "type": ("string", 0, None, "ClusteringKmeansAlgorithm", 1),
        "coef": ("number", 1, None, 1.0, 1),
        "search_engine_config_dict": ("SearchEngine", 1, None, None, 1),
        "treshold_conversation_distance": ("number", 1, None, 1.4, 1)
    },

    "ClusteringSeq_ConversationAlgorithm": {
        "type": ("string", 0, None, "ClusteringSeqAlgorithm", 1),
        "coef": ("number", 1, None, 1.0, 1),
        "search_engine_config_dict": ("SearchEngine", 1, None, None, 1),
        "treshold_conversation_distance": ("number", 1, None, 1.4, 1)
    },

    "ConversationEngine": {
        "config_name": ("string", 0, None, "", 1),
        "algorithms": ("list|ConversationAlgorithm", 0, None, [], 1)
    },
```

```
"SimpleSyntaxic_NER_Algorithm": {
    "type": ("string", 0, None, "SimpleSyntaxicNER", 1),
    "coef": ("number", 1, None, 1.0, 1)
},

"SpaCy_SM_NER_Algorithm": {
    "type": ("string", 0, None, "SpaCy_SM_NER", 1),
    "coef": ("number", 1, None, 1.0, 1)
},

"SpaCy_LG_NER_Algorithm": {
    "type": ("string", 0, None, "SpaCy_LG_NER", 1),
    "coef": ("number", 1, None, 1.0, 1)
},

"NER_Engine": {
    "config_name": ("string", 0, None, "", 1),
    "algorithms": ("list|NER_Algorithm", 0, None, [], 1)
},



"SimpleEmbedding_SearchAlgorithm": {
    "type": ("string", 0, None, "", 1),
    "coef": ("number", 1, None, 1.0, 1),
    "batch_size": ("number", 0, None, 1, 1),
    "use_cuda": ("number", 0, [0, 1], 0, 0),
    "distance_function": ("string", 0, list(DISTANCES_FUNCTIONS.keys()), "euclidian", 0),
    "model_name": ("string", 0, None, "optimum/all-MiniLM-L6-v2", 1),
    "model_type": ("string", 0, None, "sentence-transformers", 1),
    "model_optimisations": ("string", 0, ["optimum", ""], "optimum", 0),
    "translate_before": ("string", 0, ["", "en", "fr", "es", "zh", "ja", "de", "es"], "en", 0),
    "translate_method": ("string", 0, ["easyNMT", ""], "", 0),
    "NER_text_replacement": ("number", 0, [0, 1], 0, 0)
},

"SimpleSyntaxic_SearchAlgorithm": {
    "type": ("string", 0, None, "", 1),
    "coef": ("number", 1, None, 1.0, 1),
    "translate_before": ("string", 0, ["", "en", "fr", "es", "zh", "ja", "de", "es"], "en", 0),
    "translate_method": ("string", 0, ["easyNMT", ""], "", 0),
    "NER_text_replacement": ("number", 0, [0, 1], 0, 0)
},

"SyntaxicFullSentenceLevenshtein_SearchAlgorithm": {
    "type": ("string", 0, None, "", 1),
    "coef": ("number", 1, None, 1.0, 1),
    "translate_before": ("string", 0, ["", "en", "fr", "es", "zh", "ja", "de", "es"], "en", 0),
    "translate_method": ("string", 0, ["easyNMT", ""], "", 0),
    "NER_text_replacement": ("number", 0, [0, 1], 0, 0)
},

"SyntaxicWordsLevenshtein_SearchAlgorithm": {
    "type": ("string", 0, None, "", 1),
    "coef": ("number", 1, None, 1.0, 1),
    "close_words_factor": ("number", 1, None, 0.4, 1),
    "translate_before": ("string", 0, ["", "en", "fr", "es", "zh", "ja", "de", "es"], "en", 0),
    "translate_method": ("string", 0, ["easyNMT", ""], "", 0),
    "NER_text_replacement": ("number", 0, [0, 1], 0, 0)
},

"SimpleDictJaccard_NER_SearchAlgorithm": {
    "type": ("string", 0, None, "", 1),
    "coef": ("number", 1, None, 1.0, 1),
```

```
        "translate_before": ("string", 0, ["", "en", "fr", "es", "zh", "ja", "de", "es"], "en", 0),
        "translate_method": ("string", 0, ["easyNMT", ""], "", 0),
        "NER_text_replacement": ("number", 0, [0, 1], 0, 0)
    },

    "SimpleSearchByTime_SearchAlgorithm": {
        "type": ("string", 0, None, "", 1),
        "coef": ("number", 1, None, 1.0, 1),
        "translate_before": ("string", 0, ["", "en", "fr", "es", "zh", "ja", "de", "es"], "en", 0),
        "translate_method": ("string", 0, ["easyNMT", ""], "", 0),
        "NER_text_replacement": ("number", 0, [0, 1], 0, 0)
    },

    "SearchByUsers_SearchAlgorithm": {
        "type": ("string", 0, None, "", 1),
        "coef": ("number", 1, None, 1.0, 1),
        "translate_before": ("string", 0, ["", "en", "fr", "es", "zh", "ja", "de", "es"], "en", 0),
        "translate_method": ("string", 0, ["easyNMT", ""], "", 0),
        "NER_text_replacement": ("number", 0, [0, 1], 0, 0)
    },

    "SearchWith_NER_Engine_SearchAlgorithm": {
        "type": ("string", 0, None, "", 1),
        "coef": ("number", 1, None, 1.0, 1),
        "translate_before": ("string", 0, ["", "en", "fr", "es", "zh", "ja", "de", "es"], "en", 0),
        "translate_method": ("string", 0, ["easyNMT", ""], "", 0),
        "NER_text_replacement": ("number", 0, [0, 1], 0, 0),
        "ner_engine_config_dict": ("NER_Engine", 0, None, None, 1)
    },

    "SearchEngine": {
        "config_name": ("string", 0, None, "", 1),
        "nb_threads": ("number", 0, None, 1, 0),
        "max_message_length": ("number", 0, None, 200, 1),
        "nb_search_results": ("number", 0, None, 30, 1),
        "distance_limit": ("number", 0, None, None, 0),
        "algorithms": ("list|SearchAlgorithm", 0, None, None, 1)
    }
}
```