



ÉCOLE
CENTRALE LYON

Option Info

Projet Metier

ChatBot : Application du traitement des langues naturelles

Auteurs :

M. Mohamed KHATIB
M. Youssef KAROUMA
M. Daniel TIBI

Encadrants :

M. Alexandre SAIDI

Version du
21 avril 2024

Table des matières

1	Introduction	3
2	État de l'art	4
3	Approches élémentaires d'un chatBot	5
3.1	Embedding en NLP	5
3.1.1	Word2vec	6
3.1.2	Embedding avec BERT	9
3.2	1 ère approche Retrieve simple d'un chatBot	10
3.2.1	Création du corpus de texte initial à partir du pdf	10
3.2.2	Réponse à une nouvelle question posée	11
3.2.3	1 er résultat trouvé et limites de l'approche	12
3.3	Modèle Seq2seq	13
3.3.1	Formulation du problème	13
3.3.2	Architecture	13
3.3.3	Fonction de perte	13
3.3.4	Inférence : Greedy decoding et beam search	14
3.4	Modèle Seq2seq avec mécanisme d'attention	15
3.4.1	Limite du modèle seq2seq simple	15
3.4.2	Le mécanisme d'attention	15
3.4.3	Nouvelle architecture	16
4	Approche RAG	17
4.1	Architecture de l'approche dans l'état de l'art	18
4.2	Division des textes en segments : Splitting	19
4.3	Embedding	20
4.4	Stockage des embedding	22
4.5	Récupération de la réponse :	23
4.6	Génération des réponses :	23
4.6.1	Choix du LLM pour notre tâche	23
4.6.2	Quantification du Llama2 :	24
4.7	Différence entre notre ChatBot et celui dans l'état de l'art	24
4.7.1	Changement dans l'architecture générale	24
4.7.2	Indexation des segments	25
4.7.3	Récupération des segments et génération des réponses	26

5 Évaluation du ChatBot	27
5.1 Réponses correctes	27
5.2 Réponses fausses	29
5.2.1 Problème dans la question	29
5.2.2 Problème dans le loading	29
6 Interface Graphique	30
7 Conclusion	34
8 Annexes	34
8.1 Algorigramme de Beam Search	34

1 Introduction

À l'ère des avancées technologiques fulgurantes, les chatbots suscitent un engouement et rencontrent un succès remarquable. Leurs applications sont aisément comprises par les entreprises de tous les horizons, et lorsqu'ils sont correctement déployés, ces assistants virtuels peuvent avoir un impact significatif sur le retour sur investissement d'une entreprise. Cependant, l'utilisation des chatbots ne se limite pas au monde professionnel ; elle peut également s'étendre au milieu académique. C'est dans ce contexte que s'inscrit notre projet, qui vise à créer un chatbot de type Questions-Réponses (ou Question Answering Chatbot) capable d'interagir avec des étudiants ou des professeurs pour répondre à leurs questions, concernant le règlement de la scolarité.

2 État de l'art

Le livre *Building Chatbots with python* de Sumit Raj présente une base solide pour le développement de chatbots en utilisant Python, soulignant l'importance du traitement du langage naturel (NLP) et de l'apprentissage machine (ML). Il traite de l'évolution des chatbots, de leur cas d'utilisation commerciale, et fournit un guide détaillé pour construire des chatbots en utilisant des cadres comme Dialogflow et Rasa. Il explore également le déploiement de chatbots sur des plateformes comme Facebook Messenger et Slack. Ce document est crucial pour comprendre les aspects techniques et commerciaux du développement de chatbots, offrant des insights à la fois introductifs et avancés pour créer des solutions efficaces.

Building an Enterprise ChatBot se concentre sur la construction de chatbots pour des applications d'entreprise, en mettant en évidence l'intégration des chatbots avec des données d'entreprise protégées en utilisant des cadres open-source. Il aborde les considérations architecturales, les préoccupations de sécurité, et les défis d'intégration de données uniques aux environnements d'entreprise. L'accent mis sur l'utilisation d'outils open-source pour construire des chatbots d'entreprise évolutifs et sécurisés rend ce document inestimable pour les projets visant le secteur des affaires .

Deep Learning for Natural Language Processing est dédié à l'intersection des chatbots et du deep learning dans le domaine du NLP, ce document explore en profondeur l'utilisation de réseaux neuronaux pour améliorer la fonctionnalité des chatbots. Il couvre la création de réseaux neuronaux avec Python, offrant une approche pratique pour implémenter des fonctionnalités NLP avancées dans les chatbots. Cette ressource est essentielle pour les projets visant à tirer parti des dernières avancées en IA et ML pour des interactions chatbot plus sophistiquées

Bien que le livre *Machine Learning with TensorFlow* ne soit pas exclusivement axé sur les chatbots, il présente les capacités de TensorFlow dans l'apprentissage machine, avec un chapitre dédié à l'application de TensorFlow dans la construction de chatbots. Il discute de l'application des algorithmes ML pour améliorer les interactions des chatbots, ce qui en fait une ressource cruciale pour les développeurs cherchant à améliorer leurs chatbots avec des insights basés sur l'IA .

LLama 2 : Open Foundation and Fine-Tuned Chat Models représente une avancée significative dans le domaine des modèles de langage pré-entraînés, optimisés pour des cas d'utilisation dialogique. Ce document décrit la méthodo-

logie de pré-entraînement de LLama 2, qui a commencé avec une approche de pré-entraînement décrite dans Touvron et al. (2023), en utilisant un transformateur auto-régressif optimisé mais avec plusieurs améliorations pour améliorer la performance. L'architecture du modèle, les hyperparamètres, et les détails de l'entraînement sont discutés en détail, montrant une évolution significative par rapport à LLama 1, notamment une augmentation de la longueur du contexte et l'utilisation d'une attention par requête groupée (GQA) pour améliorer la scalabilité de l'inférence pour les modèles plus larges.

En intégrant ces éléments, cet état de l'art offre un guide complet pour le développement d'un projet de chatbot qui traite les questions des élèves à propos de leur scolarité, de sa conception à sa mise en œuvre, en soulignant les considérations technologiques nécessaires à prendre en compte dans le cas précis du ChatBot dédié à la scolarité des élèves.

3 **Approches élémentaires d'un chatBot**

Dans un 1 er temps, nous avons voulu commencer par des approches élémentaires du chatBot, afin de nous familiariser avec les concepts fondamentaux en NLP et LLMs.

En 1 er temps nous allons introduire la notion d'embedding, présenter quelques types d'embedding trouvés dans l'état de l'art, ainsi qu'une 1 ère approche "Retrieve" simple basée sur la "similarité".

3.1 ***Embedding en NLP***

L'embedding, dans le domaine du traitement du langage naturel (NLP), désigne la représentation vectorielle d'un mot ou d'un texte dans un espace vectoriel continu. Cette technique permet de transformer des données textuelles en vecteurs numériques, facilitant ainsi le traitement automatique par les algorithmes d'apprentissage automatique.

Ainsi, il permet de capturer la signification sémantique des mots et leur contexte dans un corpus de texte.

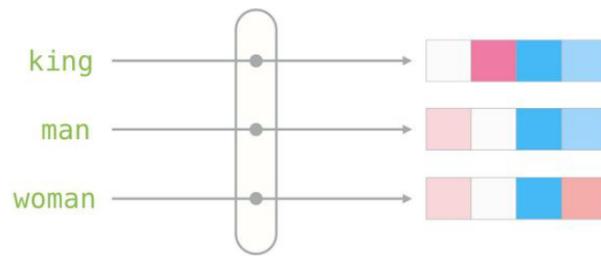


FIGURE 1 – Représentation vectorielle de 3 mots proches sémantiquement (king,man,woman)

Il existe plusieurs types d'embeddings dans l'état de l'art : **Word2vec** [1],**Glove**[2], Avec transformers (comme [?]) [3]...).

Nous allons présenter dans ce qui suit 2 types d'embedding de mots : **Word2vec** et l'embedding avec **BERT**.

3.1.1 Word2vec

Word2Vec est une méthode populaire d'embedding de mots en NLP, introduite par Mikolov et al. chez Google en 2013[1]. Son principe repose sur l'idée fondamentale selon laquelle les mots qui apparaissent dans des contextes similaires ont des significations similaires :

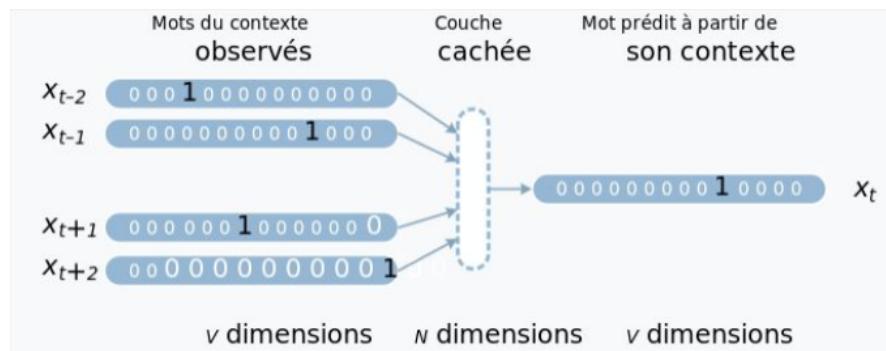
Exemple : roi-homme+femme=reine.

Word2Vec crée des embeddings de mots en se basant sur la distribution des mots dans un corpus de texte. Il existe deux architectures principales pour **Word2Vec** : Continuous Bag of Words (CBOW)[4] et Skip-gram[5].

1)Architecture CBOW

L'architecture Continuous Bag of Words (CBOW) est l'une des deux architectures principales utilisées dans **Word2Vec** pour créer des embeddings de mots en NLP.

Le modèle apprend à prédire un mot cible (dans un corpus de texte) à partir de son contexte de mots environnants (n mots à droite et/ou à gauche). Voici comment fonctionne en détails cette architecture :(illustrée dans la figure ci-dessous) :

**FIGURE 2 – Architecture CBOW**

- **Collecte de données :** Pour entraîner le modèle CBOW, un corpus de texte est nécessaire. Ce corpus est utilisé pour apprendre les relations entre les mots et leurs contextes.
- **Construction du vocabulaire :** Le modèle CBOW construit un vocabulaire à partir du corpus de texte en identifiant tous les mots uniques et en leur assignant un index. Cette étape permet de créer une représentation numérique des mots du corpus. **Exemple : pour la phrase "le chat mange le poisson", le vocabulaire construit sera :(1 :"le" ,2 :"chat", 3 :"mange", 4 :"poisson") .**
- **Création des exemples d'entraînement :** Pour chaque mot dans le corpus, CBOW crée un exemple d'entraînement en utilisant les mots voisins comme contexte. Par exemple, pour le mot "chat" avec le contexte "le gros", l'exemple d'entraînement pourrait être ("le", "gros") => "chat".
- **Transformation des mots en vecteurs one-hot :** Chaque mot du contexte et du mot cible est transformé en un vecteur one-hot, où tous les éléments sont à zéro, sauf celui correspondant au mot, qui est à un.
- **Construction de la couche d'entrée :** Les vecteurs one-hot des mots du contexte sont concaténés pour former un seul vecteur d'entrée. Ce vecteur d'entrée est passé à travers une couche d'entrée, qui est une matrice de poids partagée par tous les mots du vocabulaire.
- **Calcul de la prédition :** La multiplication du vecteur d'entrée par la couche d'entrée donne un vecteur dense, qui est ensuite passé à travers une fonction d'activation (généralement softmax) pour calculer la prédition du modèle, qui est la **probabilité conditionnelle du mot cible donné le contexte**.
- **Entraînement du modèle :** Le modèle CBOW est entraîné en ajustant itérativement les poids de la couche d'entrée pour minimiser une fonction

de perte, qui mesure la différence entre les prédictions du modèle et les vraies observations.(une **cross-entropy loss**).

- **Obtention des embeddings de mots :** Une fois que le modèle est entraîné, les poids de la couche d'entrée sont utilisés comme embeddings de mots. Ces embeddings capturent les relations sémantiques et syntaxiques entre les mots dans le corpus de texte, ce qui permet de représenter chaque mot sous forme d'un vecteur dense dans un espace vectoriel.

A la fin de l'entraînement, on récupère la matrice de poids de la couche cachée : chaque colonne va donc représenter l'embedding (vecteur) de chaque mot du vocabulaire du corpus.

2)Architecture Skip-Gram

L'architecture Skip-Gram possède à peu près la même structure que CBOW(c-à-d qu'elle comporte juste une couche cachée, une entrée sous forme de vecteurs one-hot), sauf que l'output cette fois-ci est le contexte environnant du mot considéré (c-à-d qu'on prédit le contexte d'un mot dans un corpus(n mots à gauche et/ou à droite) à partir de ce mot-là).

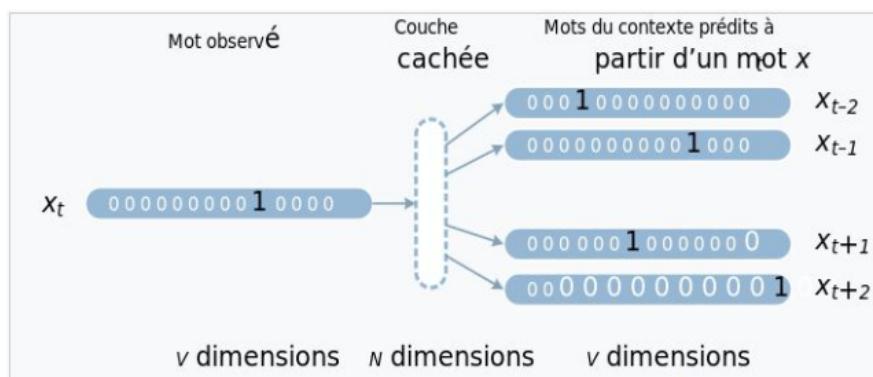


FIGURE 3 – Architecture Skip-Gram

3)Combinaison des 2 architectures CBOW et Skip-Gram

Afin d'exploiter la capacité des 2 architectures, c-à-d à la fois apprendre des embeddings qui capturent le mot à partir du contexte et le contexte à partir du mot, la meilleure chose est de combiner les 2 méthodes. Pour cela, on peut faire cela soit en faisant la moyenne position par position(**mean-pooling**), ou en prenant la valeur maximale à chaque position de l'embedding(**max-pooling**).

Une autre approche consisterait aussi à entraîner un réseau de neurones à part afin de connaître les poids de pondération de chaque vecteur (à savoir celui en provenance du modèle CBOW et de Skip-Gram).

3.1.2 Embedding avec BERT

BERT est un encodeur qui a été créé par Google[3], et fortement utilisé dans des taches NLP(tel que la classification de textes, analyse de sentiments, résumé de textes...).

Son architecture fondamentale essemble à ceci :

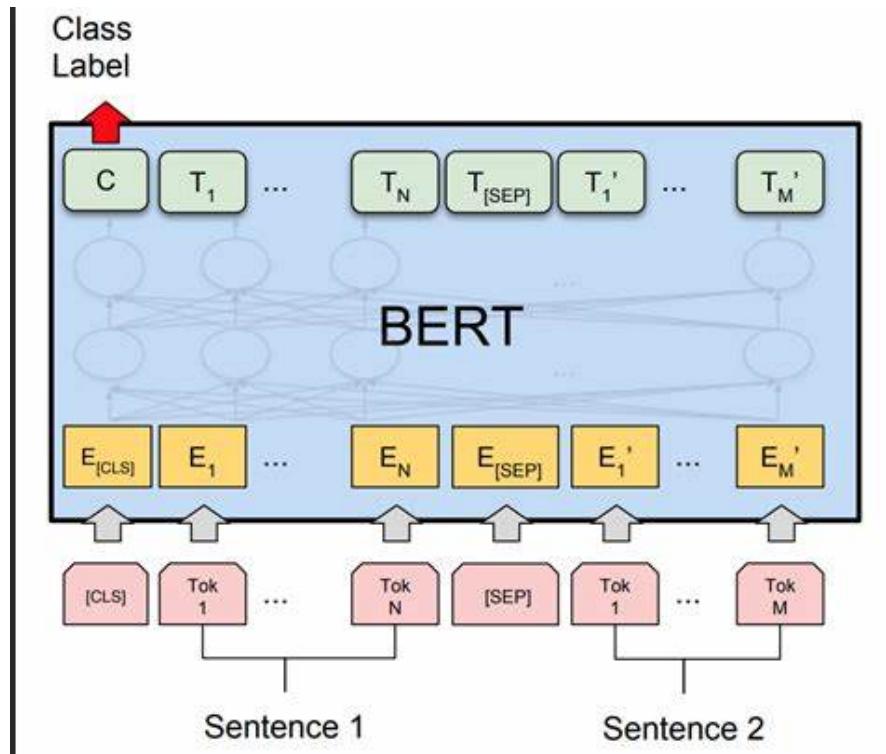


FIGURE 4 – Architecture de BERT

L'input est une séquence à 2 phrases (sentence 1 et sentence 2), séparées par un token <SEP>, et ayant au début un token <CLS> qui sera utilisé ensuite dans la classification.(à savoir classifier si les 2 phrases se suivent dans le corpus ou pas).

1) Pré-entraînement de BERT

BERT a été pré-entraîné sur 2 tâches principales :

- **MLM(Masked language model)** : Elle consiste à sélectionner systématiquement un pourcentage de tokens d'entrée (généralement 15 pourcent de WordPiece tokens de chaque séquence), les remplacer par un token <MASK>. On récupère les derniers embeddings(du hidden layer) des tokens masqués, on les rentre dans des couches softmax, et on prédit

le mot du vocabulaire correspondant. Ceci nous permet d'apprendre à représenter bi-directionnellement les tokens d'une séquence.

- **NSP(Next sentence prediction)** : Cette tache consiste à prédire pour la totalité des 2 séquences s'ils sont « IsNext » ou pas. Cette tache pourrait inclure par exemple la réponse à des questions (QA) (dans un chatBot) ou tout autre chose qui nécessite la compréhension de la relation sémantique entre 2 phrases.

Typiquement, lors du pré-entraînement, à chaque fois qu'on choisit un couple de phrases A, B, ces 2 phrases sont 50 pourcent du temps successifs dans le corpus (donc IsNext), et 50 pourcent du temps ils ne sont pas successifs (donc IsNotNext). Pour cela, on récupère l'embedding final du token <CLS>, et on effectue une classification là-dessus.

Typiquement pour notre cas, la tache de **MLM(masked language model)** paraît la plus adaptée. En effet, elle permet d'apprendre systématiquement des embeddings de tokens aléatoirement masqués dans un corpus, ce qui permet d'avoir des embeddings qui captent implicitement le contexte des mots(ou tokens).

3.2 1 ère approche Retrieve simple d'un chatBot

Dans un 1er temps, on a réfléchi à commencer par une approche simple pour répondre aux questions potentiellement posées par les étudiants. A partir de notre corpus de texte, (qui est dans ce cas le pdf de la scolarité), on a créé **une dataset csv** contenant dans chaque ligne **une paire question-réponse** à poser sur le contenu de chaque idée.

3.2.1 Crédation du corpus de texte initial à partir du pdf

Avant tout, nous devions créer un corpus de texte à la main(à savoir un google docs), dans lequel on retourne à chaque fois à la ligne en finissant une nouvelle idée(pour laquelle correspond une seule paire question-réponse). On laisse aussi une alinéa régulière au début de chaque idée, titre ou sous-titre de paragraphe.

Cette fois-ci, une fois que les idées sont bien structurées dans le nouveau document, on appelle une API GPT qui parcourt toutes les idées, et crée pour chacune une paire de question-réponse sur cette idée.

Principes

Article 1 – Admission à l'Ecole Centrale de Lyon

Les étudiants admis à l'Ecole Centrale de Lyon dans la formation ingénieur pour préparer le diplôme national d'ingénieur de l'Ecole Centrale de Lyon sont appelés « élèves ».

L'admission en 1ère année de la formation ingénieur s'effectue par la voie du concours commun « Centrale-Supélec » et par la voie de l'admission sur titre, soit « CASTing », admission sur titres d'ingénieurs, commun au Groupe des Ecoles Centrales, soit « ENISE ». Le nombre de places offertes au concours et à l'admission sur titre est fixé par le ministre chargé de l'enseignement supérieur, après avis du Conseil d'Administration. Les élèves admis par ces voies sont appelés « élèves admis sur concours ou sur titres ».



Question,Answer,num_article

quel est le sujet du paragraphe,le sujet du paragraphe est ladmission à lecole centrale de lyon,Article 1

comment sont appellés les étudiants admis à lecole centrale de lyon dans la formation ingénieur ,les étudiants admis à lecole ce

quelles sont les deux voies dadmission en 1ère année de la formation ingénieur ,ladmission en 1ère année de la formation ingénier

quels sont les types dadmissions proposés dans le cadre des formations double diplômantes ,des admissions sur titres sont propo

quels types détudiants peuvent être accueillis en 3e année à lecole centrale de lyon pour suivre une formation non conduisant a

FIGURE 5 – Transformation corpus vers questions-réponses

3.2.2 Réponse à une nouvelle question posée

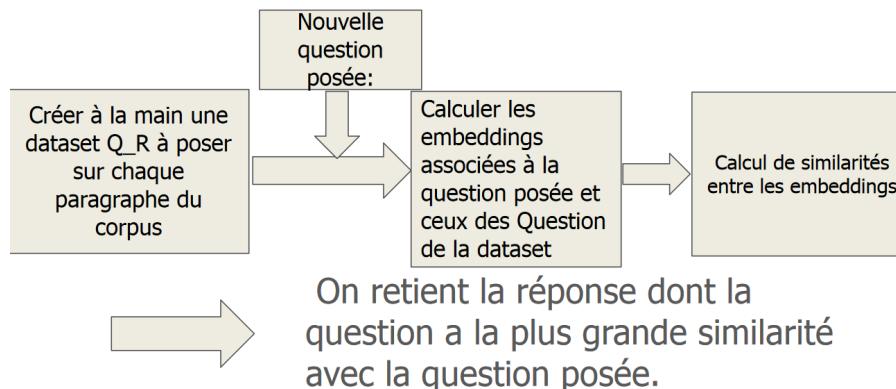
Pour une nouvelle question posée, on calcule la similarité de cette question avec les différentes questions, relève la question ayant la meilleure similarité , et on récupère la réponse correspondante à cette question.

1) Embedding et similarité utilisée

Pour des raisons de simplicité, de rapidité d'inférence et de ressources de calcul limitées, nous avons choisi les embeddings obtenus par word2vec. Typiquement, pour une phrase(question posée), on relève la moyenne de tous les embeddings de ses mots.

La similarité utilisée est la **cosin**, c-à-d le produit scalaire de 2 vecteurs sur le produit de leur norme.

On peut résumer la totalité de l'approche dans la figure suivante :

**FIGURE 6 – Résumé de l'approche Retrieve**

3.2.3 1 er résultat trouvé et limites de l'approche

Bien que cette approche soit simple, et n'avait pour but que de nous familiariser avec les principales notions en NLP, on a voulu tester pour une nouvelle question posée :

```

>>>Comment valider la 1 ère année ?
We strongly recommend passing in an `attention_mask` since your input_ids may be padded. See https://huggingface.co
/datasets/transfomers/troubleshooting#incorrect-output-when-padding-tokens-arent-masked.
Similarity: 0.5199403464794159 - un semestre detude dans une universite a letranger est valide si leleve obtient 30
credits ets ou leur equivalent au quatrième semestre valide 2 semestres a 30 credits ets et effectue son travail
de fin detudes aux cinquième et sixième semestres ou aux quatrième et cinquième semestres valide un double diplome
dans une universite etrangere partenariale de lecole centrale de lyon ou obtient 30 credits ets ou equivalents lors
d'une année de cesure
  
```

FIGURE 7 – Résultat obtenu pour une question posée

On remarque donc que la meilleure réponse trouvée n'avait rien avoir avec la question posée.

En effet, cette approche ne capture pas le contexte des idées dans le corpus (elle traite les questions (et donc les idées) séparément).

En plus, l'utilisation d'un modèle CBOW pré-entraîné ne permet pas forcément de couvrir le vocabulaire énorme des questions potentielles à poser. Enfin, cette approche dépend de la richesse de la dataset, ne crée pas une nouvelle réponse mais juste récupère ce qui existe.

3.3 Modèle Seq2seq

3.3.1 Formulation du problème

Formellement, on dispose d'une séquence d'entrée x_1, x_2, \dots, x_m (la question) et l'on souhaite obtenir en sortie une séquence y_1, y_2, \dots, y_n (la réponse). A noter que les deux séquences ne sont pas nécessairement de même taille. L'objectif est d'apprendre une fonction $p(y|x, \theta)$ de paramètres θ et de trouver l'argument qui maximise cette fonction $\arg \max_{y'} p(y'|x, \theta)$

3.3.2 Architecture

Voici un schéma de l'architecture du modèle simple seq2seq :

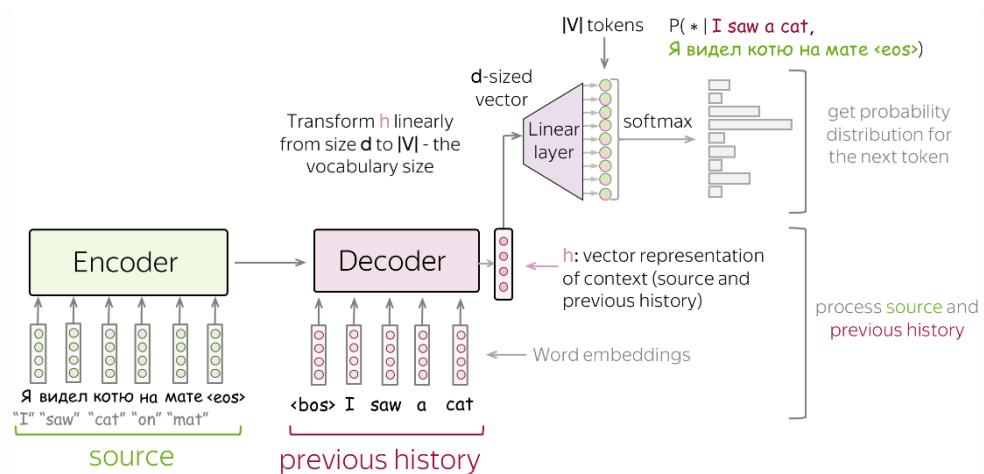


FIGURE 8 – Modèle simple

L'architecture Encodeur-Decodeur se décompose en 2 éléments :

1. Un **encodeur** constitué d'un RNN de type LSTM ou GRU qui lit la séquence d'entrée et la compresse dans un vecteur appelé **état caché**.
2. Un **décodeur** constitué d'un RNN de type LSTM ou GRU qui génère une séquence de sortie à partir de la séquence source et du vecteur d'état caché.

3.3.3 Fonction de perte

A partir de la séquence source et du vecteur d'état caché de l'encodeur, le décodeur produit à chaque instant un vecteur de contexte de dimension d et

le convertit via une couche linéaire en un vecteur dont la taille correspond au nombre de mot de vocabulaire du corpus. Une couche softmax permet de convertir les valeur de ce vecteur en une distribution de probabilité. Il s'agit donc d'un problème de classification puisque l'on essaie de concentrer la distribution de probabilité sur le sur le prochain mot à générer. La fonction de cout est donc la **cross entropie catégorielle**.

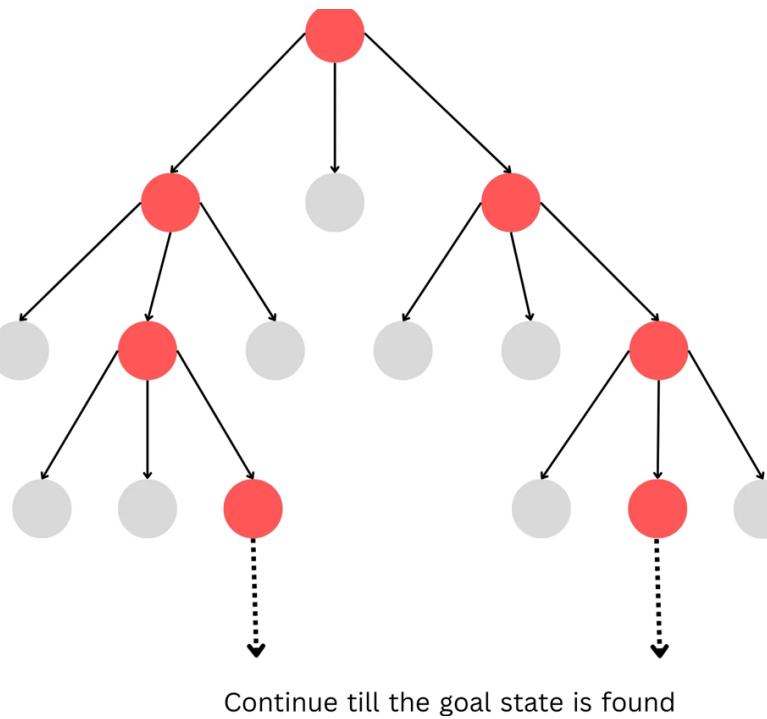
3.3.4 Inférence : Greedy decoding et beam search

La question est maintenant de savoir comment l'on determine l'argument qui maximise la fonction de notre problème : $\arg \max_y p(y|x) = \arg \max_y \prod_{t=1}^n p(y_t|y_{<t}, x)$

Il y a classiquement 2 types d'inférence possible :

1. Le **greedy decoding** qui consiste à générer à chaque instant le mot avec la plus forte probabilité. Cette méthode est intrinsèquement biaisée car le meilleur mot à un instant donné ne mène pas nécessairement à la meilleure séquence de sortie : $\arg \max_y \prod_{t=1}^n p(y_t|y_{<t}, x) \neq \prod_{t=1}^n \arg \max_y p(y_t|y_{<t}, x)$
2. Le **beam search** dont l'algorithme est le suivant :
 - On génère les k mots les plus probables avec k un paramètres spécifié par l'utilisateur.
 - A chaque étape, tous les successeur des k mots sont générés
 - Si l'une des séquences générées se termine, on la selectionne comme séquence de sortie
 - Sinon on selectionne les k meilleures séquences parmi celles générées et on réitère le processus

L'algorithme beam search peut être vu comme la génération d'un arbre où l'on selectionne à chaque niveau les k noeuds menants aux meilleurs chemin depuis la racine. Les autres branches sont élaguées.

**FIGURE 9 – Beam Search Tree**

Un algorigramme présentant les processus et décisions de l'algorithme beam search est présenté en annexe.

3.4 Modèle Seq2seq avec mécanisme d'attention

3.4.1 Limite du modèle seq2seq simple

Dans le modèle seq2seq simple, l'encodeur compresse toute la séquence d'entrée dans un seul vecteur. Si la séquence d'entrée est longue, l'encodeur aura des difficultés à compresser toutes l'informations dans un seul vecteur.

De plus, à chaque étape de génération de texte certaines parties de la séquence d'entrée sont plus pertinentes à prendre en compte par le décodeur que d'autres. Extraire l'information pertinente à partir d'un seul vecteur fixe n'est pas une tâche simple pour le décodeur.

3.4.2 Le mécanisme d'attention

Avec le mécanisme d'attention, l'état s_k de l'encodeur de chaque mot de la séquence d'entrée est prise en compte. Le mécanisme d'attention calcul la

pertinence de l'état s_k avec l'état actuel du décodeur h_t . Un score de pertinence est calculé ce qui permet de générer via une somme pondérée un vecteur de contexte prenant en compte la pertinence de chaque mot de la séquence d'entrée pour générer le prochain mot du décodeur. L'ensemble des étapes est représenté ci-dessous :

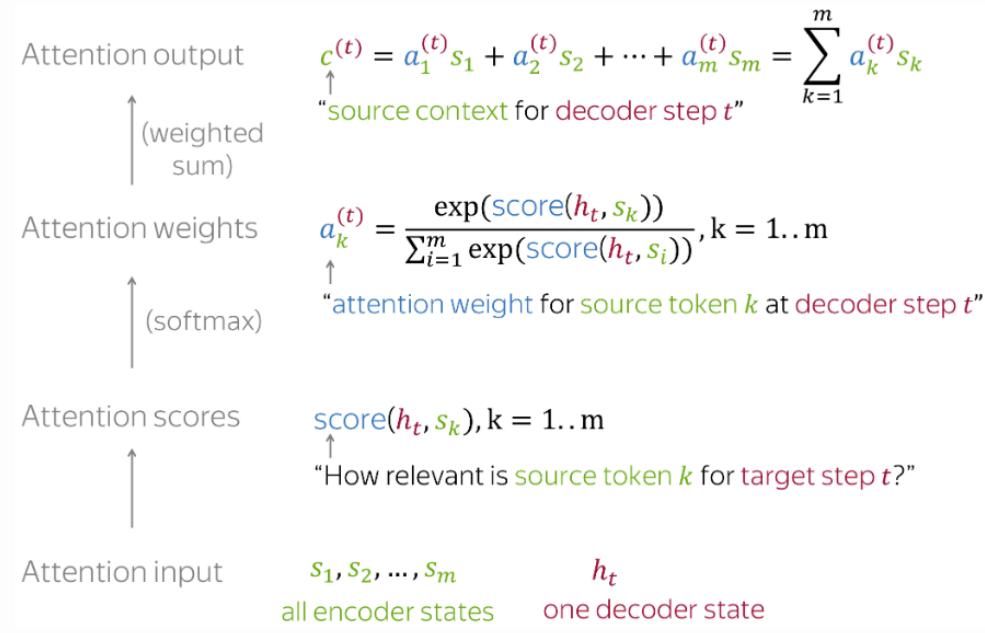


FIGURE 10 – Context vector

3.4.3 Nouvelle architecture

Voici un schéma de l'architecture du modèle seq2seq avec mécanisme d'attention :

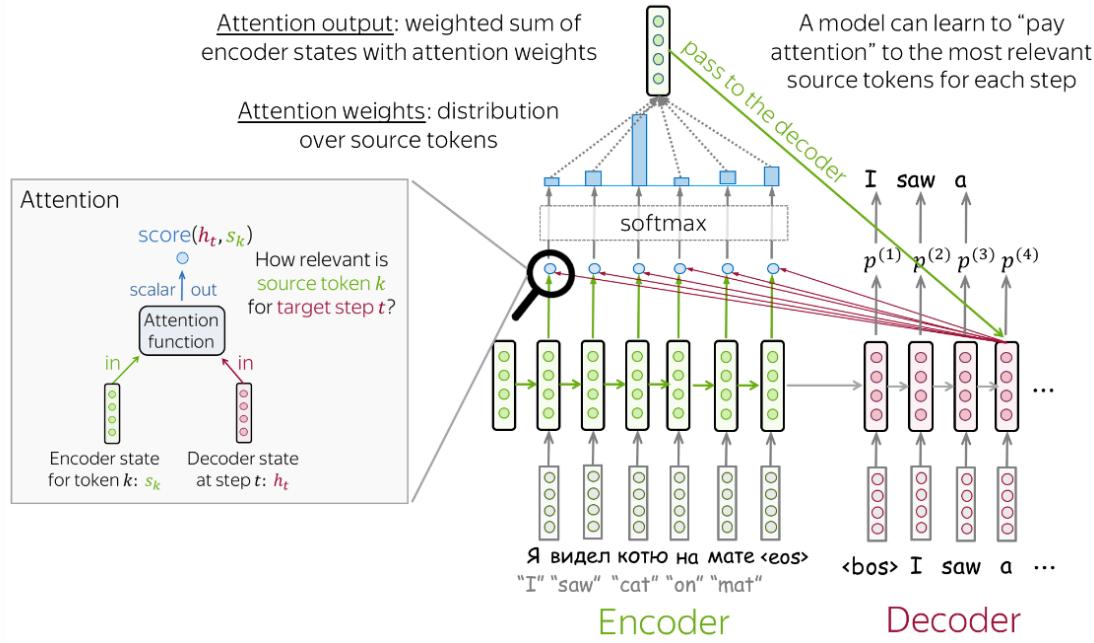


FIGURE 11 – seq2seq attention architecture

Lors de la génération d'un nouveau token, le mécanisme d'attention génère (via le processus décrit ci-dessus) une distribution de probabilité de la pertinence des mots de la séquence d'entrée. Un vecteur de contexte ainsi que l'ensemble des mots précédemment générés par le décodeur sont utilisés pour générer le nouveau mot.

4 Approche RAG

L'approche RAG, qui signifie Retrieval-Augmented Generation, est une technique utilisée dans le développement de chatbots et d'autres systèmes de traitement du langage naturel pour améliorer leur capacité à générer des réponses précises et informatives. Cette approche combine deux composantes principales : la récupération d'informations (retrieval) et la génération de texte, en tirant parti des avantages de chacune pour produire des réponses de haute qualité. Voici comment elle fonctionne en général :

Récupération d'informations (Retrieval) :

La première étape de l'approche RAG est de rechercher dans une large base de données ou un corpus de documents pour trouver des informations perti-

nentes par rapport à la requête de l'utilisateur. Cette étape utilise des algorithmes de recherche ou de récupération d'informations pour identifier les passages, articles, ou documents qui contiennent des éléments de réponse potentiels. L'idée est de trouver des sources d'information qui peuvent contenir la connaissance nécessaire pour répondre à la question posée.

Génération de texte :

La deuxième étape utilise les informations récupérées lors de la première étape comme contexte pour un modèle de génération de texte, souvent basé sur des réseaux de neurones profonds comme GPT (Generative Pre-trained Transformer). Ce modèle prend les passages sélectionnés et génère une réponse qui synthétise les informations trouvées, formulant une réponse cohérente et contextuellement appropriée à la requête de l'utilisateur.

4.1 Architecture de l'approche dans l'état de l'art

L'architecture d'un système de chatbot qui emploie l'approche Retrieval-Augmented Generation (RAG) se déroule en plusieurs étapes théoriques clés. Initialement, les documents sont chargés dans le système à partir de diverses sources et stockés dans une base de données. Ensuite, ces documents sont divisés en segments plus petits pour faciliter la récupération d'informations spécifiques. Chaque segment est encodé en un vecteur numérique via un processus d'embedding, qui permet de représenter le texte sous forme de données numériques qui peuvent être comparées en termes de similitude. Ces vecteurs sont stockés dans une base de données spécialisée, prête pour la récupération. Lorsqu'une requête est soumise, elle est également convertie en vecteur et comparée aux vecteurs de la base de données pour trouver les segments les plus pertinents. Ces segments sont ensuite utilisés comme contexte par un modèle de langage de grande taille (LLM) pour générer une réponse informative et contextuellement pertinente à la question de l'utilisateur. La figure ci-dessous résume le processus.

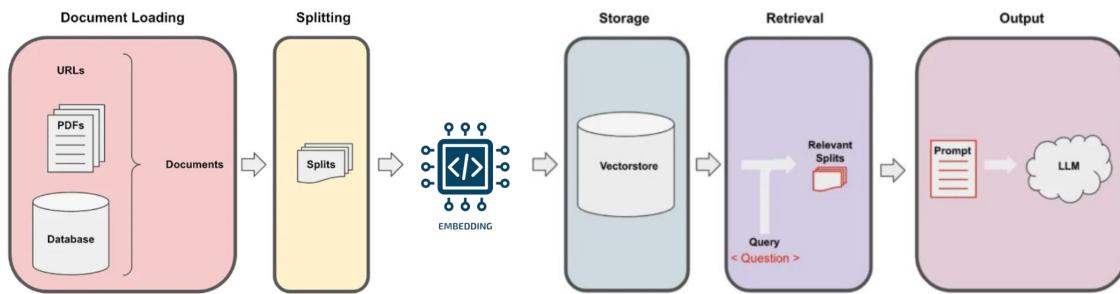


FIGURE 12 – Architecture RAG ChatBot dans l'état de l'art

4.2 Division des textes en segments : *Splitting*

La segmentation du texte en unités distinctes est essentielle au sein du processus, car il est impératif que chaque unité encapsule une information complète et autonome. Si les segments contiennent seulement des fragments d'information, cela pourrait aboutir à des réponses incohérentes et dénuées de sens pour les utilisateurs du ChatBot. Pour réussir la segmentation, il est nécessaire d'analyser la manière dont les informations sont structurées au sein des documents. Nous allons maintenant analyser une partie du règlement de scolarité.

A.2.2 Mobilités internationales

La CEU internationale examine les dossiers des étudiants candidats au recrutement dans le cadre des accords internationaux de l'établissement pour admission en qualité d'élèves à l'Ecole Centrale de Lyon pour une scolarité de deux ans en vue de l'obtention du diplôme d'ingénieur généraliste de l'Ecole Centrale de Lyon. Elle examine aussi les dossiers des étudiants candidats pour une scolarité en 3e année en vue de l'obtention du DESECL ou l'acquisition de crédits ECTS (European Credit Transfert System), ainsi que pour le certificat de formation à la recherche et l'admission pour un échange d'un semestre en S8. Dans tous les cas, ces candidats sont en priorité des étudiants proposés par les universités partenaires. Le Directeur de l'Ecole Centrale de Lyon fixe la liste des élèves admis sur titre sur proposition de la Commission des Echanges Universitaires.

La CEU internationale examine également les dossiers des élèves ingénieurs centraliens qui sont candidats à une formation double diplomante ou un échange dans une université étrangère suivant les critères présentés dans le dossier de demande. Elle statue sur l'autorisation à postuler dans un ou plusieurs établissements (à l'exclusion de tout autre) dans le cadre d'un échange. En cas de refus, les motifs de cette décision seront transmis aux élèves concernés. L'élève est finalement autorisé à partir lorsque l'établissement d'accueil émet un avis favorable pour l'accueillir, après validation de son Tronc Commun (Etendu le cas échéant) par le jury de Tronc Commun, et après validation du *learning agreement* par la Direction du Développement des Relations Internationales selon les modalités indiquées.

FIGURE 13 – Extrait du règlement de scolarité de l'école Centrale de Lyon

Nous remarquons depuis cet extrait que deux paragraphes différents contiennent deux informations différentes. De plus, dans un unique paragraphe, on peut rencontrer plusieurs informations. Ainsi, nous avons déterminé que notre méthode de segmentation tiendrait d'abord compte des paragraphes, c'est-à-dire que nous

évitons de créer des segments qui chevauchent deux paragraphes distincts. Étant donné la longueur parfois conséquente des paragraphes, nous avons adopté une approche qui se limite à des extraits de moins de 500 caractères. Nous examinons chaque paragraphe individuellement : si sa longueur est inférieure à 500 caractères, il est intégralement retenu comme segment. Si le paragraphe dépasse cette limite, nous le subdivisons en nous basant sur les phrases. Nous procédons en partant de l'avant dernière phrase et nous remontons le texte jusqu'à identifier la coupure où le segment résultant ne dépasse pas 500 caractères, en donnant la priorité à la fin des phrases puis aux espaces. On ne coupe jamais un mot en deux ! Si le paragraphe n'est pas entièrement retenu, nous pourrions perdre de l'information qui serait divisé entre deux segments. Pour éviter cela, nous créons un chauvechement entre les segments successifs d'un paragraphe. La figure ci-dessous explique le principe de chauvechement.

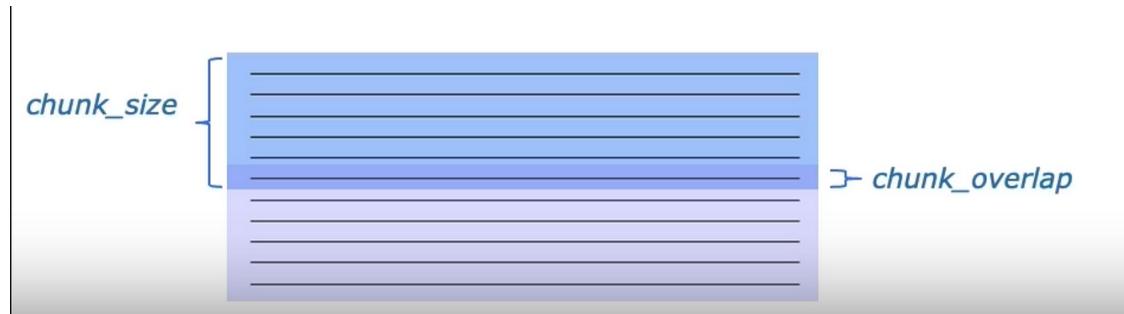


FIGURE 14 – Chauvechements entre segments

Comme montre la figure, nous avons le segment bleu et le segment violet. La dernière partie du segment bleu contient de l'information qui est nécessaire pour contextualiser la partie violète. L'overlap (chevauchement) est utilisé ainsi pour s'assurer qu'aucune information contextuelle importante n'est perdue entre les segments de texte, permettant une continuité dans l'analyse et la compréhension du texte intégral.

4.3 *Embedding*

Pour l'embedding, nous avons utilisé un embedding de type MiniLm décrit dans l'article [11]. C'est une approche avancée pour comprimer les modèles pré-entraînés de type Transformer, comme BERT et RoBERTa, en utilisant une technique appelée "distillation de la relation d'auto-attention multi-têtes" pour créer une version compressée. Cette technique repose sur deux principes clés :

la distillation des relations d'auto-attention et une stratégie de sélection des couches du modèle enseignant (teacher) pour l'entraînement du modèle étudiant (student).

Distillation des relations d'auto-attention :

La distillation des relations d'auto-attention consiste à transférer la connaissance des relations entre les paires de vecteurs de requêtes, clés et valeurs dans les modules d'auto-attention du modèle enseignant vers le modèle étudiant. Cette méthode permet de généraliser et de simplifier la distillation d'auto-attention en se concentrant uniquement sur ces relations, sans nécessiter que le modèle étudiant ait le même nombre de têtes d'attention que le modèle enseignant. Cette flexibilité offre un avantage significatif par rapport aux méthodes antérieures, qui nécessitaient une correspondance directe entre les nombres de têtes d'attention des modèles enseignant et étudiant. De plus, cette approche permet de capter des connaissances plus détaillées sur l'interaction entre les mots dans le texte, ce qui améliore les performances du modèle étudiant.

Stratégie de sélection des couches :

Au lieu de transférer les connaissances de toutes les couches du modèle enseignant vers le modèle étudiant, MiniLM utilise une stratégie de sélection des couches. Cette stratégie repose sur l'idée que transférer les connaissances d'une couche spécifique du modèle enseignant (souvent une couche intermédiaire supérieure pour les grands modèles) peut être plus efficace que de se concentrer uniquement sur la dernière couche ou d'essayer de transférer les connaissances de toutes les couches. Les auteurs de l'article ont expérimenté avec différents modèles enseignants et ont découvert que cette approche de sélection de couches conduit à de meilleures performances pour le modèle étudiant, tout en réduisant la complexité et le temps d'entraînement. La figure ci-dessous schématisse le processus.

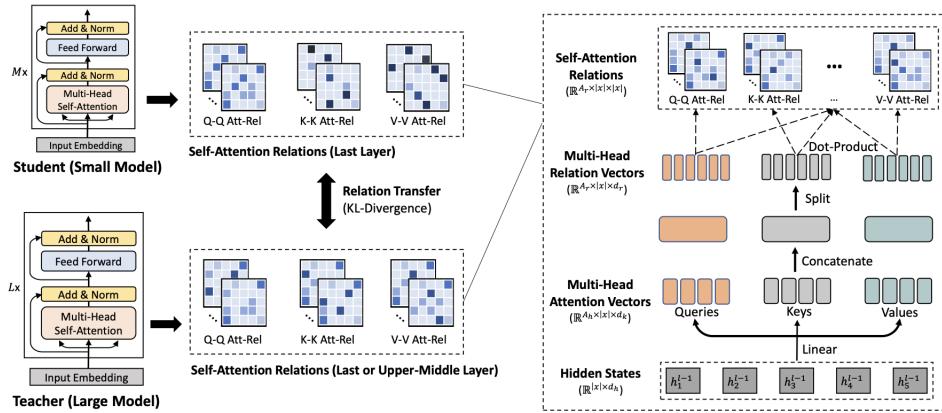
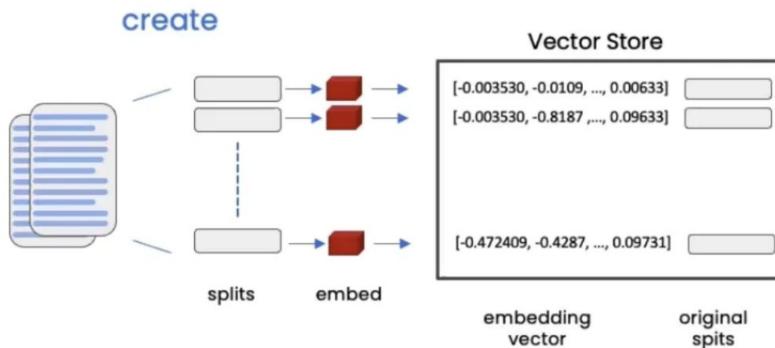


FIGURE 15 – Schéma montrant le processus de compression MiniLM

En résumé, MiniLM innove en introduisant une distillation d'auto-attention qui se concentre sur les relations entre les vecteurs de requêtes, clés et valeurs, et en appliquant une stratégie sélective pour le transfert des connaissances des couches du modèle enseignant. Cette méthode permet de produire des modèles étudiants qui sont non seulement plus petits et plus rapides que leurs modèles enseignants, mais aussi capables d'atteindre ou de surpasser leurs performances dans une variété de tâches de traitement du langage naturel, tout en offrant une plus grande flexibilité dans la conception du modèle étudiant.

4.4 Stockage des embedding

Le stockage des embedding est l'étape qui suit directement leur génération comme le montre la figure 12. On construit une nouvelle base de données contenant les segments et leur embedding associé pour qu'on puisse éviter de refaire la tâche d'embedding à chaque requête. Cela permet par la suite de manipuler et d'effectuer des opérations mathématiques sur les embeddings et récupérer facilement leurs segments associés. La figure ci-dessous résume ce processus.

**FIGURE 16 – Stockage des données**

4.5 Récupération de la réponse :

Pour récupérer la réponse, on applique le même type d'embedding sur la question posée par l'utilisateur, nous allons noté cet embedding x . Nous allons ensuite essayer de résoudre le problème d'optimisation qui minimise la distance euclidienne entre notre question et tous les embeddings du corpus. Soit le problème

$$k = \min_{y \in C} \|x - y\|$$

ou son équivalent quadratique

$$k = \min_{y \in C} \|x - y\|^2$$

où C représente l'ensemble des embeddings des segments du corpus, et k est un hyperparamètre qui représente le nombre de morceaux qu'on veut récupérer.

Dans notre cas, les questions liées à la scolarité sont généralement courtes, donc nous avons choisi $k = 2$.

4.6 Génération des réponses :

Après l'obtention des k proches morceaux de la question de l'utilisateur, ces morceaux ne seront pas cohérents et ne formeront pas un paragraphe ou une phrase compréhensible pour l'utilisateur. Ici intervient le rôle du LLM qui essayera de synthétiser ces k morceaux pour qu'ils donnent du sens à l'utilisateur.

4.6.1 Choix du LLM pour notre tâche

Pour ce ChatBot qui a pour objectif de répondre aux questions liées à la scolarité des élèves, nous avons opté pour un LLM qui soit efficace, open-source,

performant et personnalisable. L'article [10] nous montre que tous ces critères sont satisfaits par le modèle Llama2, et c'est pour ça que nous l'avons pris comme LLM qui effectuera cette tâche.

4.6.2 Quantification du Llama2 :

Le llama2 est un modèle costaud, et qui ne marchera pas simplement dans des appareils normaux, c'est pour cela que nous avons opté à sa version quantifiée sur HuggingFace. La quantification est le processus par lequel des valeurs continues et infinies sont mappées à un ensemble plus restreint de valeurs discrètes et finies. Dans le contexte des grands modèles de langage (LLMs), cela concerne la conversion des poids du modèle de types de données de haute précision vers des types de moindre précision.

4.7 Différence entre notre ChatBot et celui dans l'état de l'art

4.7.1 Changement dans l'architecture générale

Dans notre ChatBot utilisé, nous n'avons pas utilisé l'architecture générale célèbre dans l'état de l'art. Nous avons ajouté une étape entre les embeddings et leur stockage, nous l'avons appelé indexation comme mentionné dans l'article [12] (ou clustering comme dans l'article [13]). L'architecture globale de notre ChatBot peut se schématiser de la manière suivante :

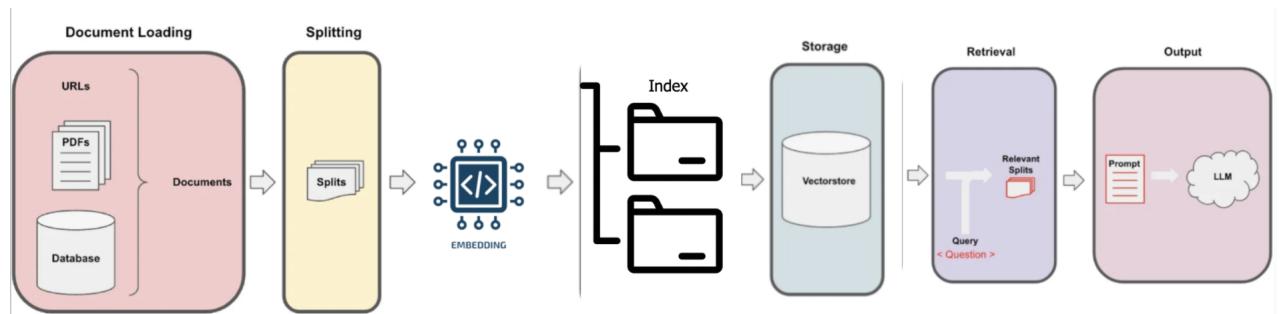


FIGURE 17 – Architecture globale de notre ChatBot

La figure montre que tous les autres étapes sont les mêmes que sur l'architecture d'un ChatBot dans l'état de l'art. La seule différence est l'ajout de l'étape d'indexation qui influencera à son rôle l'étape de stockage, car elle vient juste avant cette étape.

4.7.2 Indexation des segments

Le processus d'indexation des segments n'est qu'un processus de simple clustering avec l'algorithme k-means. Comme montré dans l'article [12], nous allons prendre $k = \sqrt{l}$ où l représente le nombre de segments total. L'idée derrière ce clustering est de minimiser la complexité du recherche des segments les plus proches à la question d'utilisateur. En supposant l'uniformité, comme nous avons \sqrt{l} clusters, donc moyennement nous aurons \sqrt{l} éléments dans chaque cluster. Et du coup si la complexité du problème de l'état de l'art :

$$k - \min_{y \in C} \|x - y\|^2$$

est $O(l * d)$ où d représente la dimension des vecteurs, alors le nouveau problème ($k = 2$)

$$\min_{c \in C_1} \|x - c\|^2$$

où C_1 représente l'ensemble des centroides du clustering est de complexité de $O(\sqrt{l} * d)$. Il convient ensuite de rechercher dans le cluster de centroïde le plus proche à la question. Cette étape est de complexité $O(\sqrt{l} * d)$, et ainsi, nous avons divisé la complexité globale par \sqrt{l} .

Pour rendre le problème de plus moins complexe, et si l est trop grand, ce qui n'est pas le cas de notre ChatBot. Nous pouvons faire un deuxième clustering sur l'erreur $y - c_y$ où y est un segment du corpus et c_y est son centroïde associé. Et pour qu'on puisse récupérer exactement un élément du corpus après les deux clustering sans avoir besoin de rechercher dans les clusters, l'article [12] nous a proposé une idée d'indexation produit. On divise nos vecteurs de dimension d en b sous vecteurs de dimension $\frac{d}{b}$, c'est à dire $y = (y^{(0)}, \dots, y^{(b-1)})$ et on fait b k-means, chacune sur une de ces sous vecteurs. On note le premier k-means fait par q^i processus de k-means associé au i^{me} vecteur q_i . Chacune avec $k = \sqrt{l}$. Nous obtenons ainsi \sqrt{l} centroides pour chaque sous vecteur.

On note ainsi le deuxième clustering par

$$q^f(y) = (q_0(y^{(0)} - c_y^{(0)}), \dots, q_b(y^{(b-1)} - c_y^{(b-1)})$$

On approche un vecteur du corpus y ainsi par

$$y \approx q(y) = q^i(y) + q^f(y)$$

Il est improbable que deux vecteurs auront la même approximation puisque deux vecteurs qui n'appartiennent pas au même clustering initial sont assez loins et le processus du deuxième clustering rend difficile que le clustering de

deux vecteurs soit exactement le même. Ainsi récupérer les valeurs $q^i(y)$ et $q^f(y)$ vaut la récupération de y .

Le problème maintenant peut se reformuler à la manière suivante : 1 . Ré-soudre le problème :

$$\min_{c \in C_1} \|x - c\|^2$$

qui est de complexité $O(\sqrt{l} * d)$. On note C^* le cluster de sa solution.

2 . Résoudre le problème

$$k - \min_{y \in C^*} \|x - q(y)\|^2$$

La différence entre cette approche à deux clustering successifs et celle avec un seul clustering est que les valeurs de $q(y)$ sont dans un ensemble fini. Le calcul de $\|x - y\|^2$ doit se refaire de chaque segment à un autre, car les composantes du segment varie sur \mathbf{R} tout entier, et donc deux vecteurs auront souvent toutes les composantes les unes différentes des autres, ainsi le calcul sera fait pour chaque morceau. Soit $\sqrt{l} * d$ calculs. Par contre, avec cette approche, nous pouvons voir

$$\|x - q(y)\|^2 = \sum_{i=0}^{b-1} \|x^{(i)} - q(y)^{(i)}\|^2$$

Les valeurs de $q(y)^{(i)}$ étant finies, nous pouvons calculer toutes les différences au préalable et puis faire la somme pour chaque nouveau vecteur et donc le problème sera de complexité $\sqrt{l} * b$ qui est très inférieure à la première.

4.7.3 Récupération des segments et génération des réponses

Pour faciliter la tâche de récupération des segments réponses, nous avons conçu une nouvelle approche de stockage, ça ressemble à celle de l'état de l'art mais elle convient aussi au processus d'indexation. Elle est schématisée dans la figure ci-dessous.

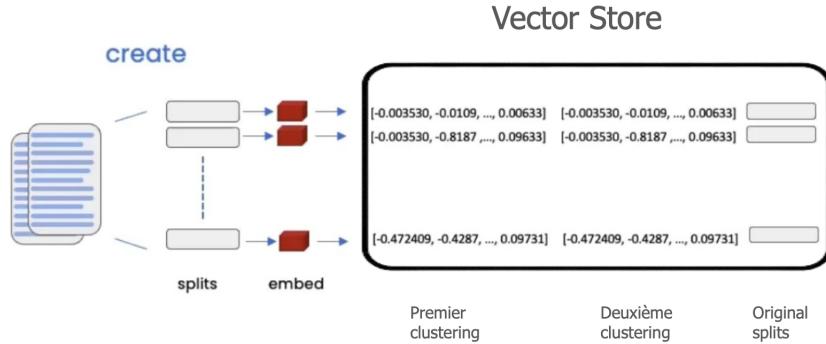


FIGURE 18 – Schématisation du stockage avec étape d'indexation

On récupère en un premier temps la valeur de $q^i(y)$ qu'on la met dans la première partie, on cherche ensuite la valeur de $q(y)$ que l'on met dans la seconde partie, et on récupère finalement le segment original. On travaille par la première partie pour résoudre le premier problème :

$$\min_{c \in C_1} \|x - c\|^2$$

On travaille avec la deuxième partie pour résoudre le problème final :

$$k - \min_{y \in C^*} \|x - q(y)\|^2$$

Et on récupère finalement les segment originaux solution du deuxième problème pour les donner au Llm pour qu'il puisse nous générer une réponse cohérente à la question.

5 Évaluation du ChatBot

Nous n'avons pas pu trouver de métrique pour mesurer la performance de notre ChatBot numériquement vu la contextualisation du LLM, qui peut donner une réponse correcte avec plusieurs façons différentes. Nous l'avons donc juste mis en test en lui donnant beaucoup de questions et analysons ces réponses fausses.

5.1 Réponses correctes

Dans cette phase d'évaluation, nous avons reçu beaucoup de réponses correctes, comme les montrent les figures ci-dessous.

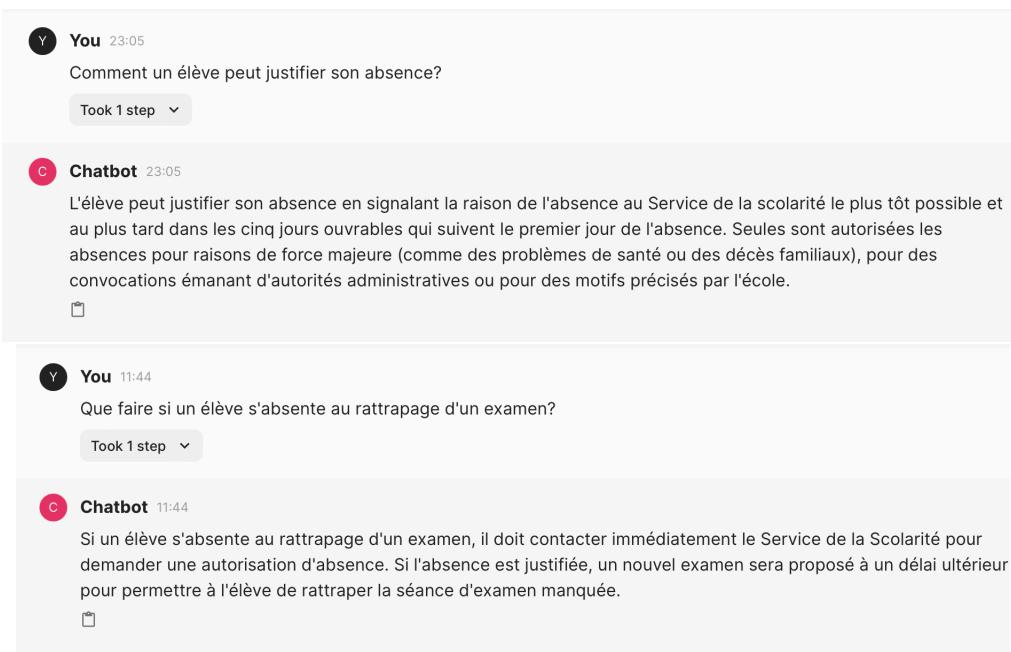


FIGURE 19 – Réponses correctes du ChatBot

Ces questions ont été posées et nous savions que leurs réponses se trouvent sur le règlement de scolarité. C'était des tests pour savoir si le processus de calcul de similarité adopté ainsi que l'étape d'indexation marchent très bien.

5.2 Réponses fausses

5.2.1 Problème dans la question

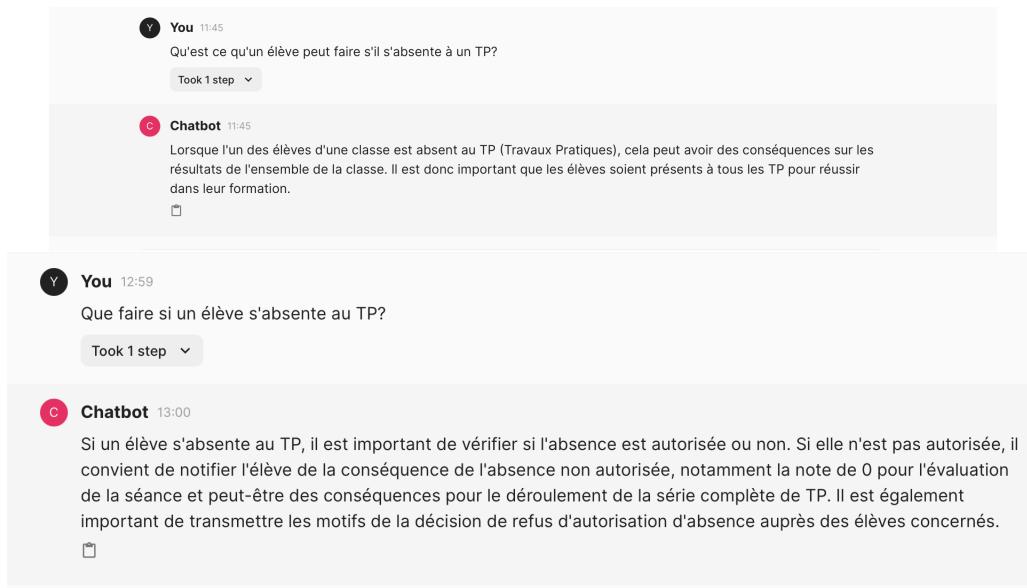


FIGURE 20 – Réponses différentes en reformulant la question

Comme nous remarquons, nous avons posé la question de deux manières différentes. La première nous a donné une réponse qui ne figure même pas sur le règlement de scolarité, et donc c'est le LLM qui a répondu, tandis que l'autre a donné exactement ce que nous attendions. Ceci peut être aussi résultat de calcul de la similarité.

5.2.2 Problème dans le loading

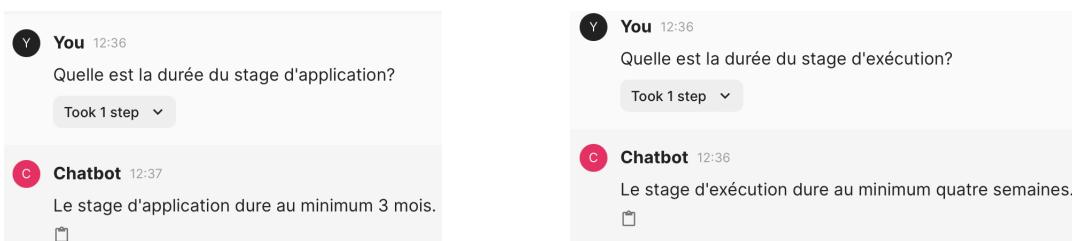


FIGURE 21 – Réponses correctes

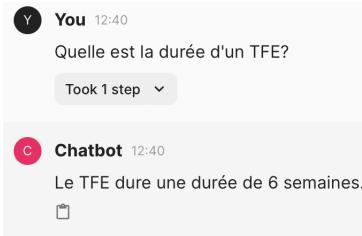


FIGURE 22 – Réponse fausse

Après analyse de ses résultats, nous avons trouvé que la durée du TFE n'a été mentionné que sur un tableau, contrairement à la durée du stage d'exécution ou d'application où il y avait des textes écrites qui les mentionnent. Nous avons supposé ainsi que notre loader PyPdf traite mal les tableaux. Il serait donc possible de prendre toutes les données tabulaires dans le règlement et de les traiter à part pour que ça puisse donner plus de précision.

Ceux sont pas les uniques problèmes rencontrés lors de la phase du test. Le test s'est fait sur un Macbook M1 Air. Le Llm parfois arrête sa génération sans avoir terminé la phrase, d'autres fois y avait des problèmes de RAM. Parfois le Llm commence a généré en anglais et d'autres fois il donne des réponses fausses que l'on a pas pu interpréter la cause, mais ceci était rarement le cas. Il travaillait généralement d'une manière correcte.

6 Interface Graphique

L'interface graphique a été codée d'une manière simple par Chainlit. C'est un framework Python asynchrone open-source développé par Literal AI, conçu pour permettre aux développeurs de créer rapidement et facilement des applications d'intelligence conversationnelle ou d'agent conversationnel à grande échelle. Il est destiné à la construction d'applications semblables à ChatGPT, d'assistants virtuels intégrés, et de co-pilotes logiciels. Chainlit offre également la possibilité de personnaliser l'interface utilisateur pour créer une expérience agentic unique selon les besoins spécifiques d'un projet . Nous avons codé une interface graphique basique avec chainlit comme vous pouvez le voir dans la figure ci dessous :

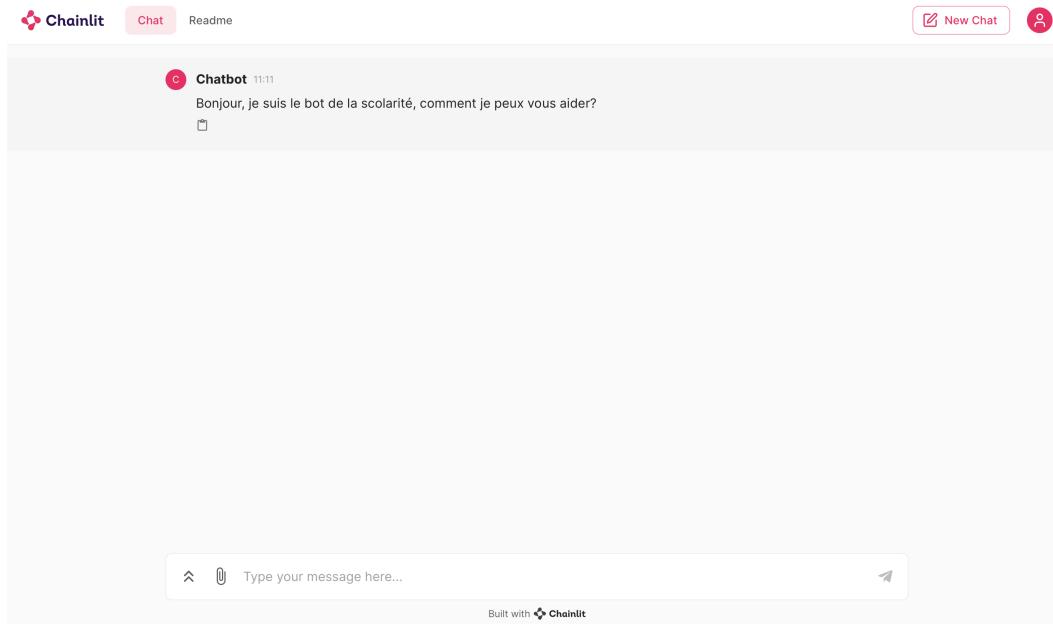


FIGURE 23 – Fenêtre initiale du ChatBot

La page est initialisé par un message de ChatBot qui dit : "Bonjour, je suis le bot de la scolarité, comment je peux vous aider ?". L'utilisateur a la possibilité de poser sa question tout en bas de la page, il l'envoie ou bien en cliquant sur Entrée sur le clavier ou bien sur le bouton send à droite en bas de la page.

Nous remarquons qu'il y a deux boutons à gauche en bas de la page. Celui qui est au milieu est pour joindre un fichier, notre ChatBot ne le supporte pas. Le premier retourne l'historique des discussions.

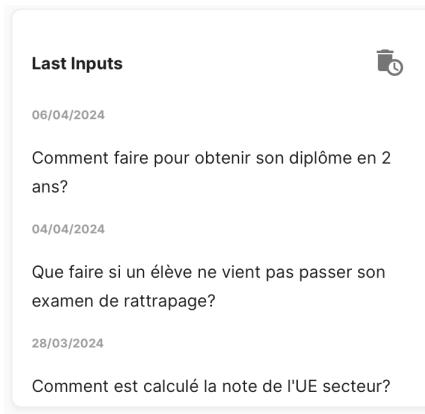


FIGURE 24 – Historique des discussions

Le ChatBot, en répondant, donne la possibilité à l'utilisateur de copier sa réponse sur le clipboard. Le boutton convenable se trouve à gauche directement en dessous de la réponse du ChatBot comme montre la figure ci-dessous

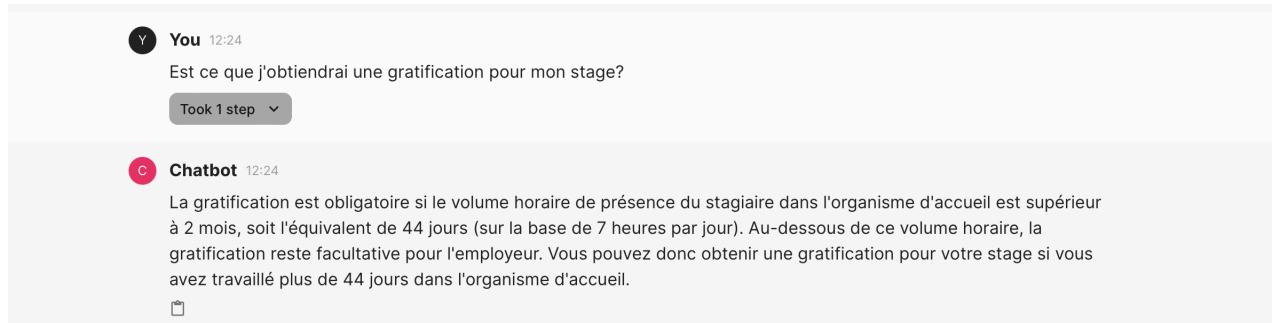


FIGURE 25 – Interface graphique du ChatBot

La figure montre aussi une partie sous la question de l'utilisateur où on trouve écrit "Took 1 step". Dans cette partie, on peut voir l'avancement de la réponse, en partant du Retrieve et de voir les segments réponses associés jusqu'à l'arrivée à Generate avec le Llm.

En haut à droite de la page, nous pouvons initialiser une nouvelle messagerie ou voir les paramètres de l'interface graphique.

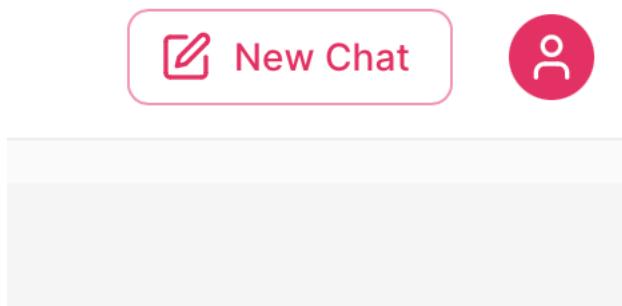


FIGURE 26 – Quelques options

En cliquant sur l'icône du monsieur, on accède au paramètres de l'interface.



FIGURE 27 – Paramètres de l’interface graphique

1. **Expand Messages** : Ce paramètre, actuellement désactivé (comme l’indique le bouton sur la position off), pourrait permettre à l’utilisateur d’afficher les messages de chat en entier si l’on suppose qu’ils sont normalement résumés ou tronqués.

2. **Hide Chain of Thought** : Cette option est également désactivée. Si elle était activée, elle pourrait masquer la "chaîne de pensée" ou le processus de réflexion interne d’un bot conversationnel.

3. **Dark Mode** : Comme les autres paramètres, celui-ci est désactivé. S’il était activé, cela changerait le thème de l’interface de l’utilisateur à un fond sombre avec un texte clair, ce qui est souvent préféré dans des conditions de faible luminosité ou par des utilisateurs qui souhaitent réduire la fatigue oculaire.

En résumé, la framework chainlit facilite le codage d’une interface graphique pour un ChatBot, vu que toutes les fonctionnalités sont prédefinies et que l’on a pas besoin de les créer dès le début.

7 Conclusion

Ce projet d'option avait pour objectif la conception d'un chatbot de type Question-Réponse destiné à répondre aux interrogations des professeurs et des étudiants concernant le règlement de scolarité. Cette initiative nous a permis, dans un premier temps, d'effectuer une recherche bibliographique approfondie sur le sujet, apportant ainsi une base solide à notre projet. Dans un second temps, nous avons pu explorer et mettre en œuvre diverses approches, allant du traitement de texte au deep learning. Après analyse, il est apparu que le modèle RAG se révèle actuellement être le plus fiable et performant sur nos données disponibles. Cependant, nous avons été confrontés à plusieurs défis, notamment le manque de données annotées, essentielles à la performance des modèles, ainsi que le besoin en ressources supplémentaires, notamment des corpus de textes en français pour l'entraînement.

En ce qui concerne les perspectives d'amélioration, il serait envisageable de travailler sur la structure même du règlement de scolarité afin de mieux identifier les sujets de questions et les paragraphes contenant les réponses potentielles, facilitant ainsi l'annotation automatique.

8 Annexes

8.1 Algorigramme de Beam Search

On peut considérer différentes structures de données telles que des tenseurs ou bien des tableaux pour manipuler et garder en mémoire les données du Beam Search. Notre choix s'est porté sur un arbre non binaire permettant d'interpréter plus intuitivement les différentes étapes et résultats de l'algorithme.

Chaque noeud de l'arbre a la structure suivante :

1. un champ **value** représentant un des mots du corpus
2. un champ **score** obtenu par multiplication des scores de prédiction depuis la racine. Il permet de quantifier la pertinence de la séquence formée entre la racine et le noeud. Note : L'utilisation du logarithme peut s'avérer utile puisque les scores calculés deviennent relativement faibles.
3. un champ **end** qui permet d'indiquer les noeuds marquant la fin d'une séquence
4. un champ **children** qui est une liste regroupant l'ensemble des noeuds enfants.

L'algorigramme est présenté ci-dessous :

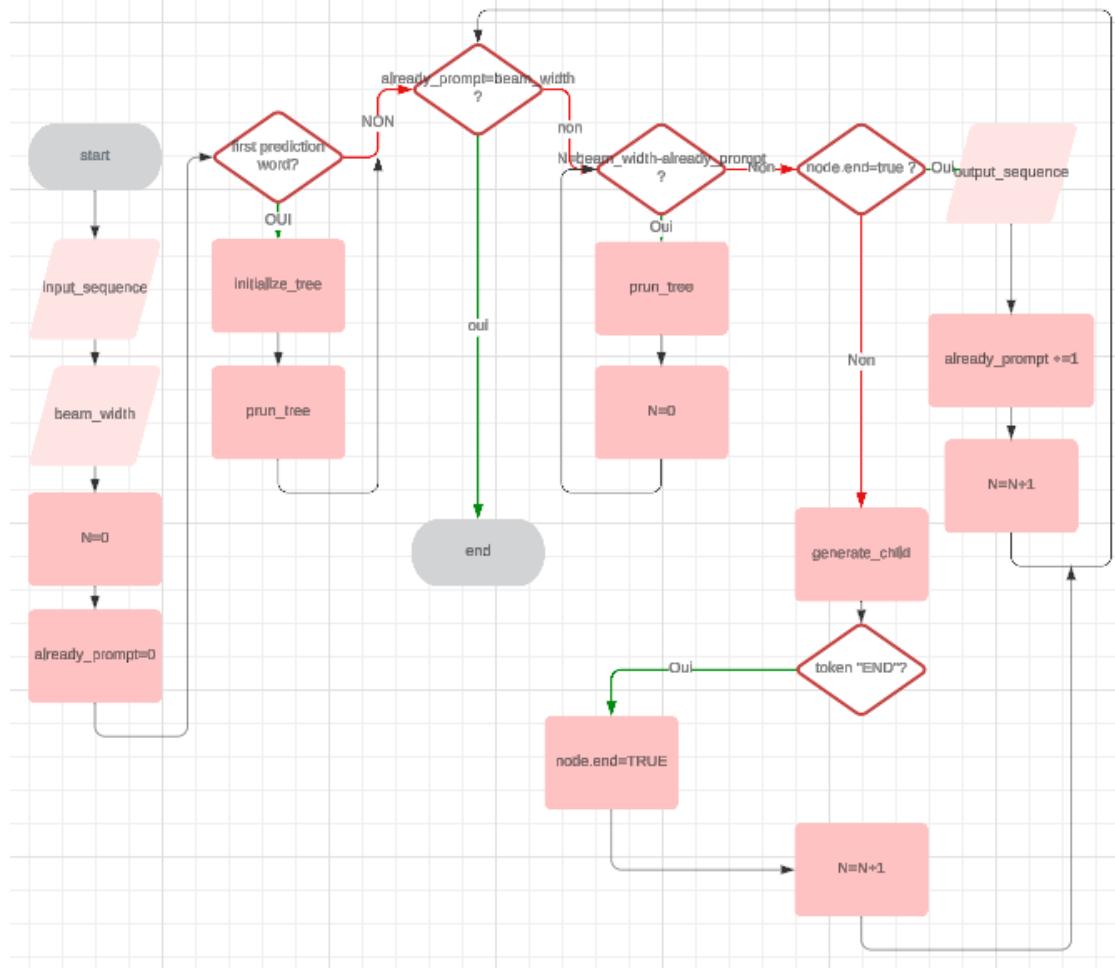


FIGURE 28 – Beam Search Tree

Références

- [1] *Emerging Trends :Word2Vec*, 2016.
- [2] *GloVe : Global Vectors for Word Representation*
- [3] *BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding*
- [4] *Siamese CBOW : Optimizing Word Embeddings for Sentence Representations*
- [5] *Word2Vec Tutorial - The Skip-Gram Model*, 2016

- [6] *Building Chatbots with Python : Using Natural Language Processing and Machine Learning*, Apress, 2019.
- [7] *Building an Enterprise Chatbot : Work with Protected Enterprise Data Using Open Source Frameworks*, 2019.
- [8] *ChatBots : Deep Learning for Natural Language Processing*, Apress, 2018.
- [9] *Machine Learning with TensorFlow*, 2018.
- [10] *LLama 2 : Open Foundation and Fine-Tuned Chat Models*, 2023.
- [11] *MiniLMv2 : Multi-Head Self-Attention Relation Distillation for Compressing Pretrained Transformers*, 2020.
- [12] *Billion-scale similarity search with GPUs*, 2017.
- [13] *Searching in one billion vectors : Re-Rank with Source Coding*, 2011.

Résumé : Le présent rapport explore le développement d'un ChatBot axé sur la scolarité, utilisant les technologies de traitement du langage naturel et d'apprentissage automatique pour répondre aux questions des étudiants et des enseignants. Après avoir examiné l'état de l'art des chatbots et des modèles pré-entraînés comme LLama 2, le rapport détaille notre approche initiale et nos expérimentations avec l'embedding en NLP via des méthodes telles que Word2Vec et BERT. Bien que les premières tentatives de modèles basés sur la récupération simple aient montré des limites, notamment en matière de contextualisation, la transition vers des architectures plus sophistiquées telles que Seq2seq et sa variante avec mécanisme d'attention a marqué une amélioration notable dans la génération de réponses pertinentes.

L'introduction de l'approche RAG (Retrieval-Augmented Generation) a constitué un tournant, alliant récupération d'informations et génération de texte pour fournir des réponses précises et contextuellement appropriées. La segmentation des textes, le processus d'embedding, et la méthode de récupération des réponses ont été méticuleusement élaborés pour optimiser la performance du ChatBot.

Une évaluation pratique du ChatBot a révélé une capacité à fournir des réponses correctes, bien que certains défis, tels que la sensibilité aux formulations de questions et les limitations liées au traitement des données, aient été identifiés. Une interface utilisateur a été développée pour faciliter l'interaction avec le ChatBot, offrant une expérience utilisateur intuitive.

Mots clés : ChatBot, Auto encodeurs, LLM, NLU, NLP, Generative models, Retrieve models, Transformers, Embedding, Llama2

École centrale de Lyon
36, Avenue Guy de Collongue
LIRIS
69134 Écully