Elaboration d'un chatbot pour le service support

Rapport de stage (14/03/2022 - 14/09/2022)

présenté par

Alexane JOUGLAR

Entreprise
Collins Aerospace
6 Avenue Didier Daurat, 31700 Blagnac, France

Master 2
Mention « *Sciences du Langage* »
Parcours « *Langue et Informatique* »

dirigé par

Victoria EYHARABIDE

soutenu le
22 septembre 2022

Remerciements

Avant de commencer ce rapport, je souhaite remercier ceux qui m'ont beaucoup appris au cours de ce stage.

Aussi, je remercie Michael Dierkes et Laurence Lamontagne pour m'avoir fait confiance dans l'élaboration de ce chatbot. Je remercie également l'ensemble des membres du service Connectivity, Test, and Support pour leur accueil et les différents employés ayant assisté à mes démonstrations pour leurs retours précieux.

Par ailleurs, je remercie Madame Eyharabide pour son accompagnement, et l'ensemble de l'équipe enseignante du Master Langue & Informatique pour leur professionnalisme et leur disponibilité.

Résumé

Le développement des intelligences artificielles et du TAL (Traitement Automatique des Langues) ont permis aux agents conversationnels de connaître un fort engouement. Sur le site de Blagnac de Collins Aerospace, une idée a émergé : créer un chatbot capable de soulager les ingénieurs du service support dans leur recherche d'information au sein de milliers de pages de documentation. Nous avons alors construit ce chatbot de toute pièce en python et en utilisant un modèle entraîné à la tâche du *question answering*.

Mots clés : chatbot, extraction d'information, *machine learning*, intelligence artificielle, BERT

Table des matières

Résumé ii

Table des figures 3

Liste des tableaux 4

Liste des abréviations 5

Introduction générale 6

1 Présentation de l'entreprise et du stage 7
- 1.1 Collins Aerospace . . . 7
- 1.2 L'équipe *Connectivity, Test and Support* (CTS) . . . 9
- 1.3 Présentation du stage et des ressources . . . 9
  - 1.3.1 Enjeu du stage . . . 9
  - 1.3.2 Corpus . . . 11

2 Méthodologie 15
- 2.1 Principe du chatbot réalisé lors de ce stage . . . 15
  - 2.1.1 Les outils . . . 15
  - 2.1.2 Principe du chatbot . . . 18
- 2.2 L'organisation du travail . . . 20
  - 2.2.1 Environnement de travail et outils . . . 20
  - 2.2.2 Première étape : avec une base de données rudimentaire . . . 21
    - 2.2.2.1 Frameworks et interfaces . . . 21
    - 2.2.2.2 Mesures de similarité et machine learning . . . 22
    - 2.2.2.3 La base de données du projet . . . 23
    - 2.2.2.4 Les démo . . . 24
  - 2.2.3 Deuxième étape : avec une base de données en Elasticsearch . . . 24
    - 2.2.3.1 Interface . . . 24
    - 2.2.3.2 La base de données . . . 25

2.3 Les verrous . . . 35
2.3.1 Les verrous liés aux données . . . 35
2.3.1.1 La structuration des données . . . 35
2.3.1.2 Le manque de données . . . 36
2.3.2 Les limites de BERT . . . 37
2.3.3 Autres limitations . . . 38

3 Résultats **39**
3.1 Les résultats . . . 39
3.1.1 Limites du système . . . 39
3.1.1.1 Temps d'exécution d'une requête . . . 39
3.1.1.2 Fiabilité de la réponse . . . 39
3.1.2 Evaluation . . . 40
3.1.2.1 Avec des questions simples . . . 40
3.1.2.2 Avec des questions complexes . . . 41
3.1.2.3 Risques et intérêts . . . 41
3.2 Améliorations possibles . . . 42
3.2.1 Une base de donnée agrandie . . . 42
3.2.1.1 Précision de la base de donnée . . . 42
3.2.1.2 Une base de données de questions/réponses . . . 42
3.2.2 Autres améliorations . . . 43
3.2.2.1 Figures et tableaux . . . 43
3.2.2.2 Amélioration du "Know more" . . . 43
3.3 Continuation . . . 44

Conclusion **46**

Bibliographie **47**

Annexes **49**

Table des figures

1.1 Logo de l'entreprise. 7
1.2 Types de questions posées aux clients. 11

2.1 Nuage de mots représentant le même texte avant et après suppression des *stop-words* (figure originale). 18
2.2 Double filtrage menant de la question à la réponse. 19
2.3 Capture d'écran de la première interface du chatbot faite avec Tkinter. 22
2.4 Capture d'écran de l'interface en streamlite. 25
2.5 Capture d'écran de l'interface définitive. 26
2.6 Schéma de création du fichier NDJSON (figure originale). 27
2.7 Extrait du fichier NDJSON. 31
2.8 Capture d'écran du mapping réalisé sur Kibana. 33
2.9 Capture d'écran du résultat de l'indexation. 33
2.10 Capture d'écran du résultat de l'indexation en Kibana. 34
2.11 Capture d'écran d'un texte propre. 36
2.12 Capture d'écran de bruit. 37

3.1 Visualisation du "Know more". 44
2 Exemple d'un document contenant des exigences. 50
3 Exemple de document structuré en parties. 51
4 Poster de présentation. 52

Liste des tableaux

1.1 Comparaison des corpus au format PDF et DOCX 12
1.2 Caractéristiques du corpus au format PDF. 12
1.3 Exemple d'échange avec le service support. 14

2.1 Librairies Python principales et leur version. 20
2.2 Description des codes permettant d'obtenir le JSON fourni à Elasticsearch. . . 32
2.3 Informations inscrites dans le mapping réalisé. 33

3.1 Temps moyen de réponse à une requête. 40
3.2 Résultats bruts du test fait sur le chatbot sans Elasticsearch. 41
3.3 Résultats bruts du test fait sur le chatbot avec Elasticsearch. 41

Liste des abréviations

BERT	Bidirectional Representations from Transformers
CTS	Connectivity, Test and Support
CUI	Conversational User Interface
IA	Intelligence Artificielle
IMO	Information Management On-Board
JSON	JavaScript Object Notation
NDJSON	Newline-Delimited JavaScript Object Notation
OSF	Open World Server Function
SQuAD	Stanford Question Answering Dataset
TAL	Traitement Automatique des Langues
UTC	United Technologies Corporation

Introduction générale

La deuxième année du master Langue & Informatique parcours professionnel à Sorbonne Université est l'occasion pour les étudiants de s'insérer dans le monde de l'entreprise. C'est également l'occasion d'investiguer sur les différents domaines du Traitement Automatique des Langues (TAL) et de déterminer vers lequel de ces domaines ils souhaiteraient préférablement s'orienter. Notre rercherche pour le stage dont il est question dans ce rapport cherchait à combiner deux aspects du TAL : le travail sur la langue écrite et le travail lié à la grande tâche d'extraction d'information. Une offre provenant de Collins Aerospace a alors attiré mon attention. Le sujet du stage portait sur la possibilité de trouver automatiquement des réponses à des questions sur la base de documents. Il trouve son origine dans la volonté pour le service support client d'alléger un peu la quantité de travail liée aux demandes dont les réponses se trouvent dans les documentations fournies aux clients. Ce stage a été pensé par Michael Dierkes, ingénieur logiciel et tuteur de ce stage, et Béatrice Chateau, ingénieur système travaillant au service support. Leur idée était de créer un outil auquel on pourrait poser des questions en langage naturel et qui serait en mesure de répondre. Cet outil fournirait la réponse sur la base des documentations susmentionnées. L'outil en question serait un chatbot et c'est de sa construction dont nous allons parler dans ce rapport. Nous structurerons notre développement de manière à contextualiser le stage en présentant l'entreprise, l'équipe intégrée et le sujet du stage en détail. Nous pourrons alors nous atteler à la construction de l'outil demandé. Enfin, nous parlerons des problèmes rencontrés et présenterons les résultats.

Chapitre 1

Présentation de l'entreprise et du stage

1.1 Collins Aerospace

UTC, devenu Raytheon Technologies par fusion avec Raytheon en 2020, fait l'acquisition de Rockwell Collins en 2018. Raytheon Techonologies est un conglomérat multinational industriel opérant principalement dans les secteurs de l'aéronautique, de la défense, et de la cybersécurité. Suite à cette acquisition, Rockwell Collins devient Collins Aerospace (le logo de l'entreprise est disponible en figure 1.1).

FIGURE 1.1 – Logo de l'entreprise.

Le siège de Collins Aerospace est situé à Waltham, dans le Massachusetts, aux Etats-Unis. Forte d'une expérience de plus d'un siècle, la société Collins Aerospace s'est établie parmi les leaders mondiaux dans les domaines de l'aéronautique et de la défense.

L'entreprise livre d'ailleurs en première page de son site internet [1] le constat suivant à son propos :

> « Collins Aerospace joue un rôle essentiel dans le développement, l'intégration et la maintenance de systèmes de haute technologie pour l'aviation militaire et commerciale en France. En tant qu'un des principaux acteurs aéronautiques du pays, nous sommes fiers de contribuer aux exportations françaises, à l'emploi et à la création de propriété intellectuelle française. »

1. Collinsaerospace.com

Collins Aerospace et ses filiales disposent de dix-huit sites industriels en France dans plusieurs domaines :

— les hélices (à Figeac notamment)

— l'avionique [2], (avec notamment le site de Blagnac)

— « les commandes de vol primaires et treuil pour hélicoptères et avions de transport militaires » [3] (site de Saint-Ouen-l'Aumône)

— les « systèmes de protection anti-feu pour hélicoptères » (site d'Antony)

Collins Aerospace collabore avec des institutions étatiques comme la Direction Générale de l'Aviation Civile (DGAC), ou l'Office National d'Etudes et de Recherches Aérospatiales (ONERA).

Ces sites et notamment le site de Blagnac sont entre autres des fournisseurs privilégiés de l'entreprise Airbus depuis les années 1970. L'entreprise a fabriqué et fabrique encore diverses pièces d'équipement pour de nombreux modèles d'appareils tel que l'A320Neo ou l'A350XWB. Cela comprend les capacités de navigation et de réseau de données, les systèmes de guidage à l'atterrissage, l'équipement de commande de vol, la gestion de l'information et les communications, ainsi que les responsabilités d'intégration du système.

La société fournit également des équipements et des services à plusieurs fabricants d'avions d'affaires et à des compagnies aériennes commerciales. La spécificité du site de Blagnac est d'être multifonction. En effet, il sert d'abord en tant que site industriel destiné aux ateliers de fabrication de pièces. Mais, par ailleurs, il dispose également d'une équipe d'ingénieurs travaillant au support client, et de laboratoires disposant de simulateurs.

En France, Collins Aerospace emploie environ 700 personnes et dispose d'une expérience longue de 60 ans dans le domaine du support client, que ce soit au niveau régional ou mondial. De la cabine au cockpit, du nez à la queue de l'avion et à travers l'expérience de vol des dernières décennies, Collins Aerospace s'est imposée comme l'un des plus grands fournisseurs de systèmes destinés à l'aérospatial, de systèmes d'intérieur tant pour le cockpit que la cabine, de services de système de gestion pour l'aviation commerciale. Ses clients sont diversifiés et incluent des compagnies aériennes, des constructeurs aéronautiques, le Ministère de la Défense, les Forces armées, « les organisations de sécurité civile », « les compagnies d'avions cargo », « les exploitants d'avions d'affaires » [4].

2. L'avionique fait référence aux équipements aidant au pilotage des avions et fusées ; ces équipements sont électroniques, électriques ou encore informatiques.
3. Collinsaerospace.com
4. idem

1.2 L'équipe *Connectivity, Test and Support* (CTS)

J'ai été intégrée à l'équipe CTS, sous la direction de Laurence Lamontagne. Cette équipe comporte treize personnes travaillant sur le développement de solutions de tests pour les communications avioniques et l'organisation du support client. Cette équipe existe au sein d'un département nommé *Connectivity Solutions* travaillant à la liaison entre les différents équipements de l'avion avec le sol.

Au sein de cette équipe d'ingénieurs systèmes et logiciels, le sujet du stage formait une activité nouvelle. En proposant à l'équipe *support engineering* un outil pour les aider à gagner du temps dans leur travail, le sujet du stage représentait pour le service une volonté d'innover. Cette innovation tient tant en la tâche ainsi présentée qu'au domaine dont il est question. Il était envisagé de pouvoir étendre cette innovation au-delà du service.

Au sein de Collins Aerospace, il n'existe pas de service dédié au TAL ou à la science des données bien que certains employés s'y intéressent assidument. Nous avons notamment eu la chance de travailler en partie avec Stéphane Lelièvre, ingénieur logiciel suivant en parallèle de son travail une formation en *data science*, et avec qui j'ai pu préparer une présentation toute particulière pour une journée « portes ouvertes » organisée au sein de l'entreprise. On trouvera en annexe une copie du poster réalisé à l'occasion de cette journée et qui a servi à présenter le projet à l'ensemble de l'entreprise.

1.3 Présentation du stage et des ressources

1.3.1 Enjeu du stage

Le service support mentionné précédemment est constitué d'ingénieurs système assurant la maintenance des produits vendus par Collins Aerospace. Sorte de service après-vente de l'aéronautique, ils sont confrontés à une multitude de questions posées par les clients. Les réponses à ces questions se trouvent parfois telles quelles dans la documentation fournie avec le produit. Mais il s'agit là de documentations longues de centaines, voire de milliers de pages que les clients n'ont pas nécessairement envie de consulter.

Béatrice Chateau, ingénieur système membre du service support, et Michael Dierkes, ingénieur logiciel, ont alors eu l'idée d'un outil qui permettrait de gagner du temps sur la recherche des réponses au sein des documents. Cet outil saurait capter le sens d'une question posée en langage naturel. Puis il saurait chercher la réponse exacte dans la documentation. En conséquence, ils ont pensé au chatbot comme solution au problème posé.

On nomme chatbot un logiciel apte à dialoguer avec un utilisateur. Inventé en 1966 par Joseph Weizenbaum, ELIZA est le premier chatbot connu. Reformulant les affirmations de l'utilisateur en questions, ce chatbot se voulait simulation d'un psychothérapeute rogérien. L'usage des chatbots s'est largement répandu ces dernières années et couvre un large éventail d'utilisation.

Il existe en effet différents types de chatbots :

— les agents conversationnels comme c'est le cas de Xiaolce « pensé comme une intelligence artificielle amie ayant une connexion émotionnelle pour satisfaire les besoins humains de communication, d'affection et d'appartenance sociale. » [5] [ZGLS20]

— les chatbots informatifs (agents rationnels [Cah17]) destinés à fournir des réponses en sa basant sur des documents

— les chatbots « réalisant une tâche spécifique comme réserver un avion ou aider quelqu'un » [6] [AM20] et que l'on pourrait alors qualifier d'assistants

— chatbots éducatifs

Les recherches existant sur les outils permettant à un humain de dialoguer avec une machine sont influencées par le test de Turing (donner l'illusion à l'utilisateur qu'un programme pense). Pour le chatbot dont il est question dans ce rapport, le travail effectué ne cherchait pas spécifiquement à donner l'illusion d'un dialogue sensé. Il s'agissait simplement de pouvoir renvoyer la partie d'un document contenant la réponse voulue.

Pour comprendre en quoi nous pourrions trouver réponse à certaines de ces questions dans les documentations sans passer par une interprétation, il faut analyser les questions précédemment posées au service support. Elles se divisent en trois catégories. La figure 1.2 permet d'en visualiser la répartition. Le diagramme a été obtenu d'après un document contenant des échanges entre le service support et les clients (2 107 échanges au total). Au sein de ce document chaque question a été labellisée ; les trois labels disponibles sont : « Analyse », « Information », et « Assistance ». Nous avons recueilli l'ensemble de ces labels et avons calculé la part représentée par chaque type de label. Le document en question est présenté de manière plus précise dans le chapitre 2 au sein de la première section portant sur le corpus utilisé pour le stage.

Grâce au diagramme, il est nous est aisé de visualiser les proportions de chaque label. Nous remarquons en particulier que 23% des questions posées au service support (question labelisées «assistance») sont suceptibles de trouver leur réponse telle quelle dans les documents fournis

5. "XiaoIce is uniquely designed as an artifical intelligence companion with an emotional connection to satisfy the human need for communication, affection, and social belonging"

6. "Task-based chatbots perform a specific task such as booking a flight or helping somebody."

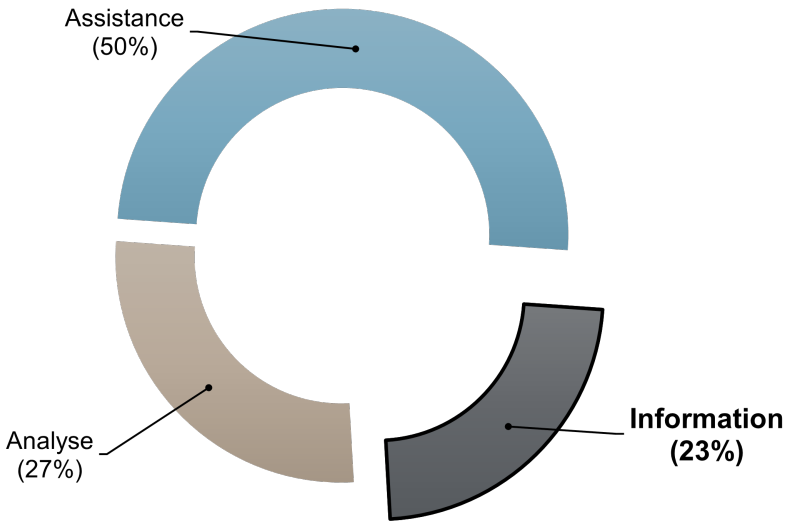


FIGURE 1.2 – Types de questions posées aux clients.

par le service support. Quant aux autres types de question, il est possible que l'on puisse trouver une réponse dans les documents, mais nous ne pouvons pas en quantifier l'importance pour le moment.

1.3.2 Corpus

Dès les premiers jours du stage, nous avons pu obtenir un accès à une base de données constituée de documents relatifs au projet IMO (*Information Management On-Board*), et en particulier à l'une de ses branches nommées OSF (*Open World Server Function*).

IMO est un système permettant de collecter les informations de l'A350 et de les rendre disponible au pilote, au copilote et aux personnes effectuant la maintenance (les *mainteners*). Ce système permet donc avant tout chose d'héberger toutes les applications nécessaires aux activités de ces trois rôles (pilote, copilote, *maintener*).

La base de donnée à laquelle nous avons eu accès est régulièrement mise à jour. Nous y trouvons entre autres les documentations des produits vendus par Collins Aerospace dans le cadre du projet susmentionné. Les documentations telles que disponibles sur cette base ne sont pas exactement celles dont les clients disposent à l'achat d'un produit. En effet, étant en constante évolution, elles sont actuellement un mélange entre la version S6 précédemment sortie et la version S7 qui n'est pas encore achevée.

Les documents sont disponibles en deux formats différents : DOCX ou PDF. Comme nous pouvons l'observer 1.1, à contenu identique, les DOCX sont plus lourds que les PDF donc le

travail a été effectué sur les fichiers au format PDF.

Format	PDF	DOCX
Nom du projet	IMO_OSF	IMO_OSF
Nombre de fichiers	6	6
Taille totale du corpus	67,7 MB	91,8 MB

TABLE 1.1 – Comparaison des corpus au format PDF et DOCX

Les premières observations du corpus me permettent de faire les remarques suivantes :

— les documents sont écrits en langage technique

— tous les documents n'ont pas la même structure

— les documents ne contiennent pas uniquement du texte, mais aussi des figures et des tableaux, ce qui constituera du bruit dans notre corpus

— les documents contiennent beaucoup d'information contextuelle inutile qu'il sera bon de filtrer lors des prétraitements

Le corpus contient des documents correspondant à des spécifications. Il existe au sein de ce corpus deux grands types de documents. Les plus fréquents (quatre documents) sont des documents mélangeant texte brut, figures et tableaux. Il n'existe pas au sein de ces documents d'autre structure que celle offerte par la table des matières. Les autres documents, les moins fréquents (deux documents), sont des exigences. Nous distinguons ces deux types de documents et cela pourrait avoir un impact sur la manière de les traiter. Nous avons décidé de créer un processus de traitement commun à tous mais nous aurions pu décider de faire autrement. Nous affichons en annexe des extraits floutés de ces deux types de documents afin d'en visualiser la disposition (figures 2 et 3).

On trouvera dans le tableau 1.2 des indications supplémentaires sur le corpus PDF seul [7]. On notera à cet égard que le corpus est en anglais.

Format	PDF
Nombre de page	6320
Nombre de tokens	1388832
Langue du corpus	anglais

TABLE 1.2 – Caractéristiques du corpus au format PDF.

7. Les chiffres donnés ont été calculés une fois le corpus nettoyé des pieds de page, entêtes et indications latérales.

Il nous a également été fourni un accès à une base de données sous la forme d'un tableur comprenant entre autres une colonne dédiée aux échanges entre le service support et le client rencontrant un problème ou ayant une demande et le type de problème. Nous appelons « échange » un dialogue entre les deux parties se présentant de la manière suivante :

— le message envoyé au service support par mail et contenant un problème, ou encore une question.

— la réponse donnée par le service support par mail.

Il arrive qu'une case comprenne plusieurs échanges si le client a d'autres questions au même sujet. Nous faisons figurer dans le tableau 1.3, un exemple d'échange entre un client et le service support. Quelques remarques sur ce tableau :

— Les inscriptions en rouge ont été ajoutées pour la bonne compréhension de l'échange. Pour cette raison, elles sont en français. Le reste du tableau est anglais comme l'est le corpus fourni.

— Le tableau a été transposé par rapport au document original. La colonne de gauche est normalement la première ligne du tableau.

— Certaines spécifications n'ont pas été indiquées (par exemple, la personne étant intervenu dans la résolution du problème)

— les informations sensibles ont été remplacées par des « X ».

Nous ne manquerons pas d'indiquer par ailleurs que le document contenant les échanges et celui dont nous nous sommes servis pour définir la quantité de questions à laquelle nous pourrions potentiellement répondre et dont le diagramme représentatif a été donné en 1.2.

La case échange ("Description of the encountered issue") disponible sur le tableur fourni n'est pas toujours complétée, ce qui rend le fichier difficilement exploitable au-delà de l'évaluation de la répartition des questions par catégorie. L'observation des questions lorsqu'elles sont disponibles nous permet de faire le constat suivant : elles ne sont pas posées telles quelles au service support, mais *via* un échange de mail relatant un problème depuis son origine (ceci incluant des détails tels que les heures, les intervenants, les réflexions). Nous pouvons donc imaginer que toute transition vers un système automatique dans le cadre d'un support de niveau 1 demandera la refonte du système de question vers le service support (nous pensons à une question concise et correctement posée).

ID	2066	
Status	7-Closed	
Summary	Support request for X Ground Deployment	
Priority	3-High	
Issue Type	Assistance	
Description of the encountered issue	Demande au service support	X Ground deployment including X is planned on january 22th at 09 :00 AM and AIB lab team need RCI support. Could someone from Collins can be present on Airbus site X please ?
	Réponse du service support	Deployment with X will start at 930 they will perform the deployment in remote Lab team and COllins will follow the deployment from the bench using webex.
Environment description	X	
Original Requester	X	
Project	OSFC	

TABLE 1.3 – Exemple d'échange avec le service support.

Chapitre 2

Méthodologie

2.1 Principe du chatbot réalisé lors de ce stage

2.1.1 Les outils

Plusieurs outils ont été utilisés, à commencer par BERT.

BERT (*Bidirectional Representations from transformeurs*)[DCLT18] est un modèle de langage développé par Google qui, comme l'indique son nom, est issu des transformeurs puisqu'il en reprend la partie *encoding*.

Un transformeur est un réseau de neurones originellement créé pour résoudre le problème de la traduction d'une langue à une autre. Les transformeurs sont expliqués dans l'article [VSP+17]. Sa particularité est qu'il peut apprendre le contexte d'un mot de gauche à droite et de droite à gauche de manière simultanée. L'architecture du transformeur est faite de deux parties :

1. la partie *encoding* qui se charge de générer des *embedding* [1]. L'encodeur apprend le contexte. Si l'on stocke les encodeurs on obtient la structure de BERT.
2. la partie *decoding* qui associe ces *embeddings* à des mots. Le décodeur apprend à cartographier. Si l'on stocke les décodeurs on obtient la structure de GPT.

En se basant sur les *transformeurs* classiques, BERT ne peut prendre en compte que 512 tokens par requête. Afin d'augmenter cette capacité, nous utilisons un dérivé du transformeur appelé *longformer*.

Avant d'introduire les *longformers*, nous devons évoquer rapidement le mécanisme d'attention. De la même manière que les humains ne prêtent pas une attention égale aux différents

1. Vecteurs qui encapsulent le sens d'un mot.

composants d'une image ou d'un texte qu'ils observent, les intelligences artificielles en TAL utilisant ce type de mécanisme vont tenter de sélectionner au sein de larges quantités d'informations, celles qui sont les plus susceptibles d'être intéressantes. Ceci permet entre autres d'obtenir des systèmes plus performants. Il existe plusieurs types de mécanismes d'attention. Nous en notons deux :

— le *sliding window attention*

— le *global attention*

Le *sliding window attention* prend en compte pour chaque token un contexte fixe à sa gauche et à sa droite. Plusieurs couches d'attention vont être superposées, de telle manière que les couches supérieures ont accès à tous les éléments inférieurs « et ont la capacité de construire des représentations intégrant l'ensemble de l'information » [2].

Le *global attention* est « une extension du modèle encodeur-decodeur propre aux réseaux de neurones récurrents » [3].

Les *longformers* utilisent ces deux types de mécanisme d'attention. Les *transformeurs* ont permis de résoudre plusieurs tâches relatives au TAL comme la génération de texte ou bien la traduction, notamment grâce à leur mécanisme d'attention. Mais ils sont très chronophages lorsque appliqués à de longues séquences :

> « Bien que puissants, la mémoire et la puissance de calcul nécessaires pour le mécanisme d'attention croît quadratiquement avec la longueur de la séquence, ce qui rend impossible (ou très couteux) de prendre en compte de longues séquences. » [4] [BPC20]

Les *longformers* au contraire présentent un mécanisme d'attention dont la mémoire et la puissance de calcul nécessaires croît linéairement.

Nous pouvons alors introduire BERT. BERT est entrainé en deux étapes. La première étape consiste à apprendre la langue (*pre-training*) ; cela se fait grâce à deux tâches : le *Masked Language Modelling* où le programme cherche à trouver des mots qui ont été cachés, et le *Next Sentence Prediction* où le programme cherche à prédire la phrase suivante. Cet entrainement s'est fait sur tout Wikipedia en anglais et Google Books pour un total de 3,3 milliards de mots. La deuxième étape, (*fine-tuning*), consiste à apprendre la tâche pour laquelle on veut utiliser le

2. paperswithcode.com/method/sliding-window-attention
3. machinelearningmastery.com//global-attention-for-encodeur-decoder-recurrent-neural-networks/
4. "While powerful, the memory and computational requirements of self-attention grow quadratically with sequence length, making it infeasible (or very expensive) to process long sequences."

modèle. L'entrainement va donc dépendre de la tâche : question/réponse, résumés, prédiction de phrases, résolution de polysémie et de coréférence, désambiguïsation de mots, analyse de sentiments, etc. Dans le cas qui nous concerne, il s'agit de répondre à des questions. Pour entrainer le modèle à une tâche, il lui faut une grande quantité de données (quelques centaines de milliers de questions/réponses) et une bonne puissance de calcul. En conséquence, nous avons choisi un modèle déjà apte à la tâche ; ce modèle est disponible sur Hugging Face. Hugging Face[WDS+19] est une plateforme open source proposant des modèles de *machine learning*.. L'avantage du modèle SQuAD est qu'il est capable de prendre en compte de longues séquences de tokens (jusqu'à 4096), ce qui est considérable par rapport au maximum habituel (512 tokens). Ceci est possible grâce aux *longformers*, une variation des transformeurs.

Pour le *fine-tuning* nous avons utilisé un jeu de données nommé SQuAD (Stanford Question Answering Dataset) [RZLL16]. C'est un jeu de données créé par l'université de Stanford à partir de Wikipédia et destiné à entrainer des algorithmes répondant à la tâche du *question answering*. Il est composé de « questions posées par des micro-travailleurs sur un ensemble d'articles Wikipedia où la réponse à chaque question est un segment de texte ou *span*, issu du passage correspondant, ou bien la question ne trouve pas de réponse. » [5]. Il existe deux versions du SQuAD. La première version contient 100 000 paires de question-réponse sur plus de 500 articles. La deuxième version reprend ces paires-là et y ajoute 50 000 questions qui ne trouvent pas de réponse dans les articles [RJL18]. Nous avons testé ces deux jeux de donnée. Les résultats disponibles dans le chapitre suivant ne donnent les chiffres que pour la version 1 du SQuAD en raison d'un nombre de réponses proche de 0 lorsqu'on utilise la version 2.

DistilBERT est le résultat d'une modification du modèle BERT original permettant de le rendre plus léger [SDCW19].

> « DistilBERT est un petit transformeur, rapide, léger et bon marché créé par distillation de la base BERT. Il a 40% de moins de paramètres que BERT-base-uncased, fonctionne 60% plus vite tout en préservant plus de 95% des performances de BERT mesurées sur le benchmark de compréhension du langage GLUE. » [6]

Pour les mêmes raisons qu'avec la version 2 du SQuAD, nous ne prendrons pas en compte les résultats donnés par le DistilBERT entrainé sur la tâche de question/réponse.

Nous avons fait le choix de BERT car c'est un modèle dont nous avions déjà entendu parler et c'est également un modèle utilisé par une large communauté. Compte-tenu du fait que ce

5. https://rajpurkar.github.io/SQuAD-explorer/
6. "DistilBERT is a small, fast, cheap and light transformeur model trained by distilling BERT base. It has 40% less parameters than BERT-base-uncased, runs 60% faster while preserving over 95% of BERT's performances as measured on the GLUE language understanding benchmark." in huggingface.co

travail a été réalisé sans aide, il était important de pouvoir aisément trouver des réponses aux éventuels problèmes posés en quelques recherches sur internet.

Enfin, nous ferons une parenthèse sur les tokens et la toknisation car, nous le verrons, c'est un sujet de grande importance pour ce projet. La toknisation est le processus de séparer un texte en morceaux. Ces morceaux peuvent correspondre à des mots ou des phrases. On appelle ces morceaux des tokens. Suite à la toknisation, nous effectuons un nettoyage des *stopwords* (« mots vides ») qui permet de ne garder que les mots importants. La figure 2.1 montre l'importance du nettoyage des stopwords.

FIGURE 2.1 – Nuage de mots représentant le même texte avant et après suppression des *stopwords* (figure originale).

2.1.2 Principe du chatbot

A présent, nous pouvons évoquer le principe du chatbot. La question entrée par l'utilisateur dans la section prévue à cet effet est mise en relation avec la base de donnée. De là, le chatbot fonctionne sur la base d'un double filtrage :

1. Filtrage par mots-clés. Le document ayant le plus de mots-clés en commun avec la question est extrait de la base de données. Le document candidat est alors stocké.
2. Filtrage de la réponse grâce à BERT. On applique un modèle BERT entrainé au document candidat et on en ressort la réponse exacte. Le code utilisé est en partie fourni par Hugging Face [7] pour l'implémentation du modèle entraîné à la tâche.

7. https://huggingface.co/valhalla/longformer-base-4096-finetuned-squadv1

La réponse est ensuite renvoyée à l'utilisateur dans la fenêtre principale. Nous pouvons voir à quoi cela ressemble dans la figure 2.3. La réponse est accompagnée d'un lien ("Show more", en bleu sur la capture d'écran) qui, lorsque l'on clique dessus, mène vers le document PDF contenant la réponse.

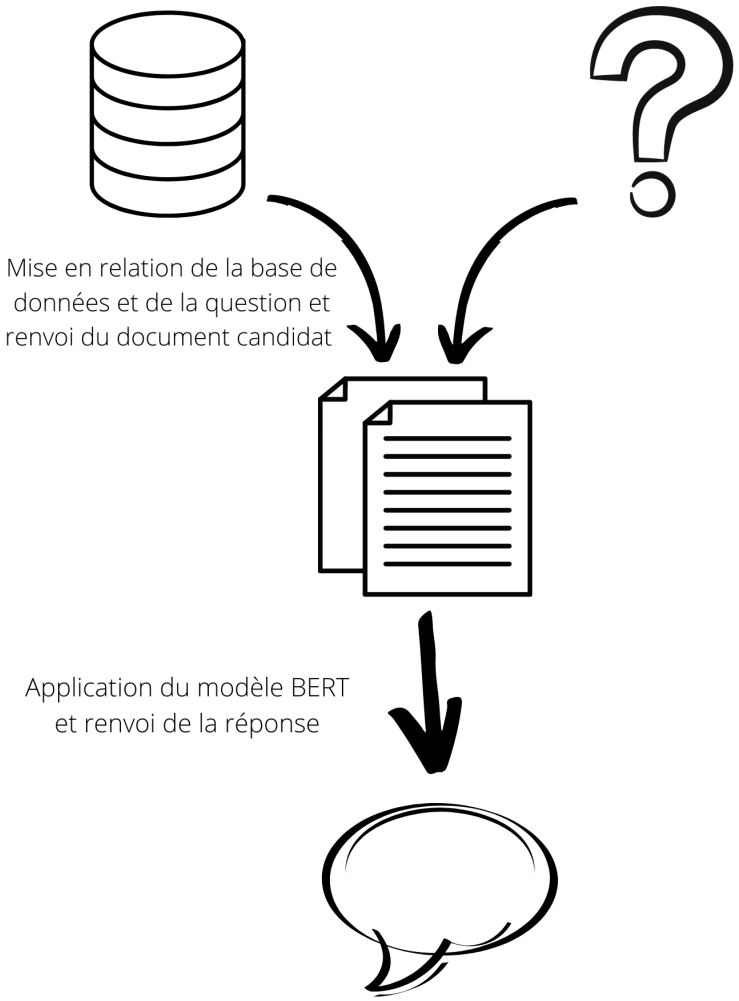


FIGURE 2.2 – Double filtrage menant de la question à la réponse.

Le principe expliqué ci-dessus est le même dans la première et la deuxième version du chatbot (dont il est question plus tard). Ce qui change fondamentalement entre les deux versions est la manière dont l'on stocke la base de données.

2.2 L'organisation du travail

2.2.1 Environnement de travail et outils

Le travail fut effectué sous deux environnements différents. Les tout premiers pas furent faits sur Windows, mais je suis très vite passée sous Linux lorsque le service Intelligence Artificielle (IA) de Collins Aerospace m'a prêté un ordinateur avec davantage de puissance de calcul pour faire des expériences. Puis je suis à nouveau passée sous Windows lorsqu'il m'a été indiqué que mon chatbot devait avant tout fonctionner sous Windows, car il s'agit du système d'exploitation utilisé par le service support. Les expériences ont donc été menées sur deux machines :

— un notebook Alienware 17 R3 (version 1.2.3)

— un ordinateur fixe Dell sous Windows 10

Les temps d'exécution étaient plus rapides sur l'Alienware, mais a terme il a fallu que les codes soient adaptés à Windows puisque c'est le système d'exploitation utilisé par les ingénieurs système du support client.

Les librairies Python principales utilisées dans les différents codes permettant de mettre en place la base de données ou de lancer le chatbot sont disponibles dans le tableau 2.1.

Etape d'utilisation	Librairie	Version
prétraitements	PyPDF2	2.8.1
	Fitz	x
Chatbot	Streamlit	1.10.0
	Streamlit-chat	0.0.2.1
	Tkinter	x
	Nltk	3.7
	Sklearn	1.0.2
	Torch	1.10.2+cpu
	transformeurs	4.13.0
	BERT-tensorflow	1.0.4
	Elasticsearch	7.10.0

TABLE 2.1 – Librairies Python principales et leur version.

Les librairies Python plus générales ayant été utilisées sont les suivantes :

— JSON

— codecs

— glob

— re

— os
— shutil
— math
— string
— itertools

On travaille dans un environnement virtuel. Toutes les expériences sont menées en langage Python.

2.2.2 Première étape : avec une base de données rudimentaire

2.2.2.1 Frameworks et interfaces

Lorsqu'il s'agit de chatbot, plusieurs *frameworks* sont disponibles sur internet pour développer son propre chatbot. Ces outils se composent premièrement d'une interface utilisateur ou *Conversational User Interface* (CUI). Il s'agit d'une fenêtre permettant à l'utilisateur d'insérer sa question et de visualiser la réponse. Ils disposent également d'une API Web qui contient la « logique métier » permettant « l'interaction avec l'utilisateur »[8]. Par ailleurs, ils disposent éventuellement d'un moteur permettant de « transformeur les phrases de votre utilisateur en intentions et en entités permettant à votre ChatBot de comprendre un utilisateur qui parle comme s'il parlait à un autre humain »[9]. Enfin, ils disposent de données sur lesquelles le chatbot va venir piocher pour trouver les réponses.

Les *frameworks* de chatbot sont généralement payant lorsque l'utilisation est destinée aux entreprises et/ou insuffisants pour la tâche souhaitée. Or, il m'a été demandé d'utiliser des outils open-source, et gratuits dans le cadre d'une utilisation en entreprise, éventuellement à large échelle. Aucun outil ne m'avait alors été indiqué ou conseillé ; ceci est majoritairement dû au fait que je n'intégrais pas une équipe de TAL où mes collègues auraient eu une vision précise du produit attendu. J'avais la possibilité de faire mes propres choix. En conséquence, puisqu'il s'agissait de construire un outil capable de répondre sur la base de documents existants, il ne m'a pas semblé indispensable de trouver un tel outil et j'ai décidé de mener la tâche pas à pas et de zéro. La seule question qui restait était de savoir quelle librairie ou quel outil utiliser pour l'interface. Il a été fait le choix de la librairie Tkinter qui n'offre *a priori* pas un avantage esthétique certain mais qui permet de créer rapidement et aisément une interface capable d'être présentée lors d'une démonstration. La figure 2.3 montre à quoi ressemblait la première interface. Celle-ci avait été créée rapidement et sans souci de joliesse. En conséquence, l'idée d'utiliser un *framework* pour réaliser mon chatbot a très vite été écartée.

8. dotnet.devloppez.com
9. idem

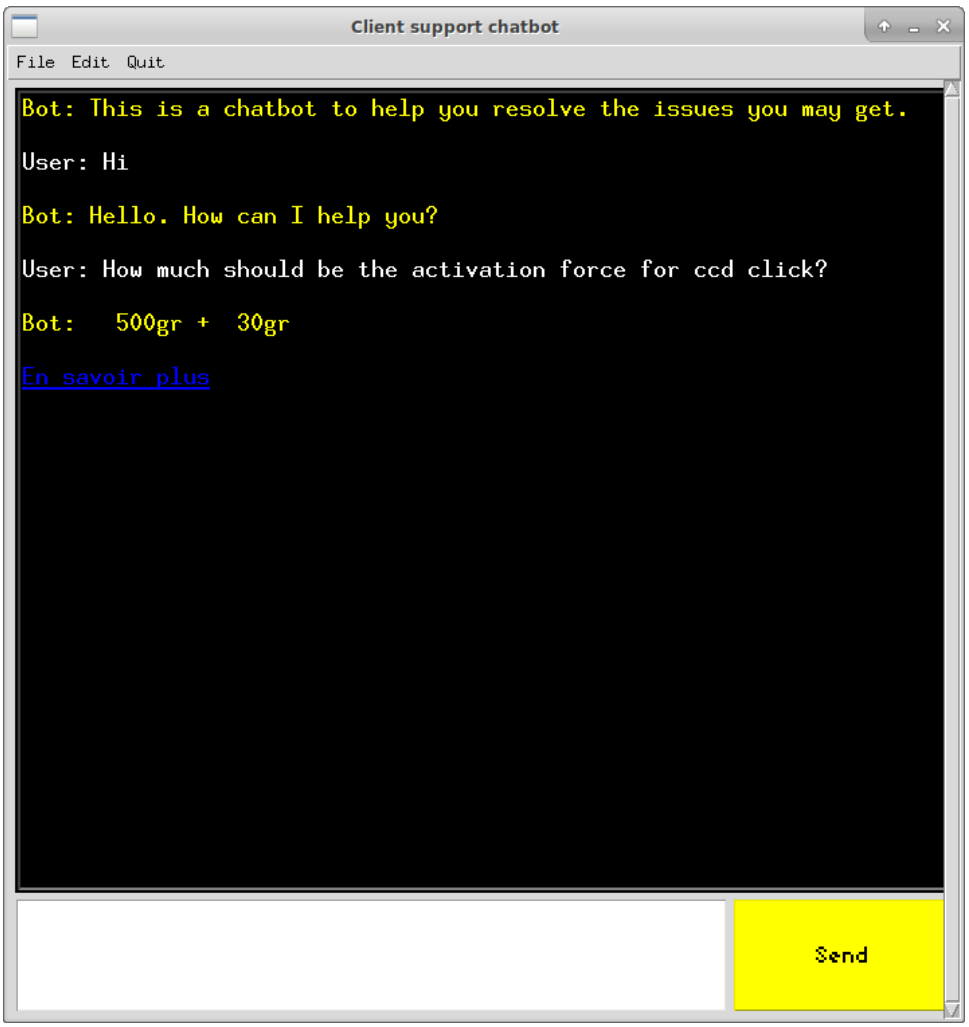


FIGURE 2.3 – Capture d'écran de la première interface du chatbot faite avec Tkinter.

2.2.2.2 Mesures de similarité et machine learning

Ma première piste pour le chatbot était de travailler avec des mesures de similarité :

> « Si l'on se restreint aux données textuelles, le besoin est avant tout venu de besoin pour la recherche d'information. Il s'agit en effet de pouvoir d'une part mesurer le degré de proximité entre des documents et d'autre part de pouvoir identifier et ordonner la liste des documents les plus pertinents à offrir en réponse à une requête dans un moteur de recherches. » [BFG+20]

Idéalement, nous aurions fait une liste de questions tests (par exemple, cent questions). Ces questions auraient dû ressembler à celles posées par les clients (quant à la possibilité d'avoir de vraies questions, il en est question antérieurement dans ce rapport, dans la partie propre au corpus). On aurait alors utilisé différentes mesures de similarité (Jacquard par exemple) pour déterminer la similarité entre la question et une partie de la documentation. Dans les premiers jours, nous avons mené une telle expérience (sur quelques questions seulement). Nous avons

donc utilisé des mesures de similarité pour associer un *input* (une question) à un extrait de la documentation. Cet extrait est composé d'un élément d'une liste faite des différentes parties des documentations. Nous avons pris pour chaque *input* la réponse associée la plus susceptible d'être la bonne sur la base de la mesure de similarité utilisée (c'est-à-dire la plus haute). Nous avons mené une telle expérience en utilisant l'ensemble des tokens, ou bien en supprimant les mots vides ou la ponctuation, ou bien en utilisant les n-grammes, ou bien en combinant plusieurs de ces caractéristiques. Ensuite, nous avons regardé une à une si les réponses trouvées étaient effectivement correctes. Nous n'avons pas décidé de poursuivre avec cette approche, car ses limites se sont très vites imposées à nous :

— les résultats étaient peu concluants. Il faut bien noter qu'aucune notion de synonymie n'était prise en compte, ce qui limite les associations.

— la liste composée des éléments de la documentation et produisant les réponses possibles était trop hétérogène en raison de la nature disparate des documents.

Dès les premiers jours, nous avons pensé au modèle BERT comme possible solution et c'est sur cette piste que nous avons travaillé tout le temps restant. Nous avons premièrement pensé à *fine-tuner* un modèle mais il nous fallait davantage de puissance de calcul et davantage de données (des milliers, voire des centaines de milliers de paires questions/réponses avec le texte dans lequel s'inscrit la réponse).

2.2.2.3 La base de données du projet

Dans un premier temps, correspondant aux trois premiers mois du stage, le chatbot fut développé à partir d'une base de données rudimentaire. Il s'agissait en effet d'un dossier dans lesquels se trouvaient les documents. Nous allions interroger cet ensemble de documents sans intermédiaire.

Les étapes de création de la base de données furent les suivantes :

1. Conversion des PDF en TXT
2. Division par « thème » (c'est-à-dire en fonction des parties du document, grâce à la table des matières)
3. Division des fichiers de plus de 13 kB (qui représentent environ 4000 tokens, maximum pris en charge par le modèle)

Nous effectuons en effet une double division des documents. La première division trouve son origine dans la volonté de garder *a minima* la structure offerte par les PDF *via* la table des matières. Malgré tout, cette division n'était pas suffisante. La plupart des blocs de texte obtenus

font plus de 4096 tokens (quantité maximale de tokens prise en compte par le modèle BERT). Il nous a donc fallu effectuer une deuxième division dont le critère seul est la taille des blocs de texte. Chaque bloc obtenu précédemment a fait l'objet d'une seconde division si sa taille était supérieure à 13 kB, ce qui correspondait plus ou moins à 4 000 tokens.

Le résultat fut l'obtention de 3 351 fichiers d'extension TXT pour une taille totale de 13,4 MB. Ces fichiers sont contenus dans dossier en local. C'est ce dossier qui constitue la base de données.

2.2.2.4 Les démo

Nous avons présenté une première version du chatbot deux mois après avoir commencé le stage. Nous avons reçu plusieurs retours permettant l'amélioration du chatbot. Les voici :

— Réaliser une base de données en Elasticsearch
— Améliorer le « en savoir plus » pour qu'il mène vers l'extrait précis au sein du PDF
— Ajouter une barre de progression lorsque la réponse est en cours de recherche par le programme

Plusieurs autres présentations ont suivi à la même période, notamment lors de la journée portes ouvertes de l'entreprise. Lors de cette journée, les exposants présentent pendant deux heures leur projet à différents groupes de personnes ; lors de cet événement, beaucoup d'idées ont été reçues de la part des participants qui voyaient plusieurs domaines d'application au chatbot : une aide pour les employés qui doivent naviguer au sein de grandes quantités de documents, support de niveau 1 généralisé, faciliter le *on-boarding* projet. Cette présentation s'est faite à l'aide d'un poster dont on trouvera une copie en annexe.

2.2.3 Deuxième étape : avec une base de données en Elasticsearch

2.2.3.1 Interface

Dans la deuxième partie du stage, et suite aux recommandations qui m'avaient été faites, j'ai cherché à améliorer l'aspect esthétique de mon interface. J'ai utilisé la librairie Streamlit [10]. On pourra observer en 2.4 le rendu de cette interface pour le chatbot.

Bien que satisfaisante d'un point de vue esthétique, cette interface ne semble pas adaptée à la lourdeur du processus se déroulant en arrière-plan. A terme, cette interface, très lente et sujette aux erreurs d'exécution, a donc été remplacée par la précédente, celle réalisée avec Tkinter, moins esthétique, mais plus efficace. Malgré tout, nous avons tenté d'améliorer l'aspect esthétique de l'interface en Tkinter. On en trouvera le résultat en 2.5.

10. streamlit.io

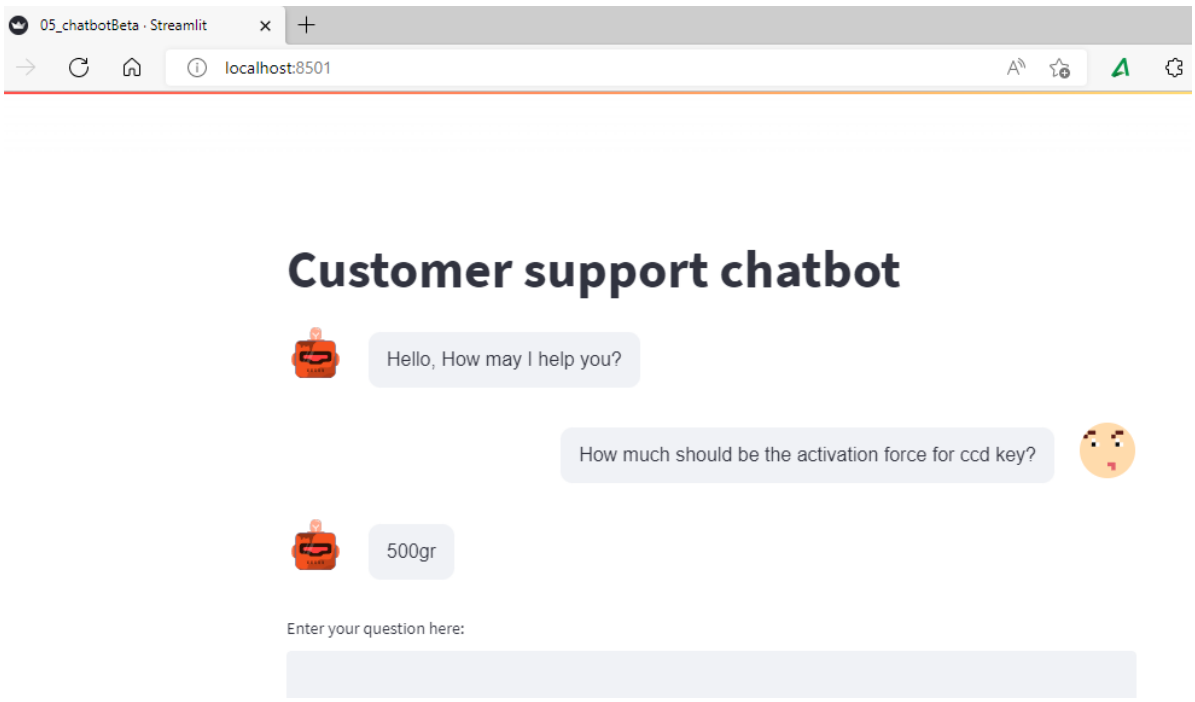


FIGURE 2.4 – Capture d'écran de l'interface en streamlite.

2.2.3.2 La base de données

La base de données utilisée pour la deuxième version de notre chatbot est foncièrement différente de la première, mais nous nous sommes basés sur les codes précédemment créés pour le prétraitement des documents. Nous faisons apparaitre pour la première fois de ce rapport des codes. Ils ne sont que partiellement retranscrits et ne servent qu'à mettre en valeur les parties les plus importantes.

Le schéma 2.6 reprend les trois grandes étapes nécessaires que nous avons décidé de suivre pour créer la base de données. Le résultat de ce processus est un fichier NDJSON indexable en Elasticsearch.

Elasticsearch est un moteur de recherche et d'analyse de données utilisant Lucene, une bibliothèque Java open source permettant d'indexer et de chercher du texte. Elasticsearch est distribué sous licence Elastic. « Cette licence confère gratuitement le droit d'utilisation, de modification, de création d'œuvres dérivées et de redistribution » ; cette utilisation présente trois interdictions. Il est est interdit de « fournir les produits en tant que service géré », « contourner la principale fonctionnalité de cette licence [ou d'] obscurcir ou supprimer les fonctionnalités protégées par les clés de licence », « supprimer [ou d'] obscurcir toute notification de licence, de droit d'auteur ou autre » [11]. Le logiciel Elasticsearch est, entre autres, associé à Kibana qui permet notamment de visualiser des données. Les données indexées en Elasticsearch sont au

11. elastic.co/fr/pricing/faq/licensing

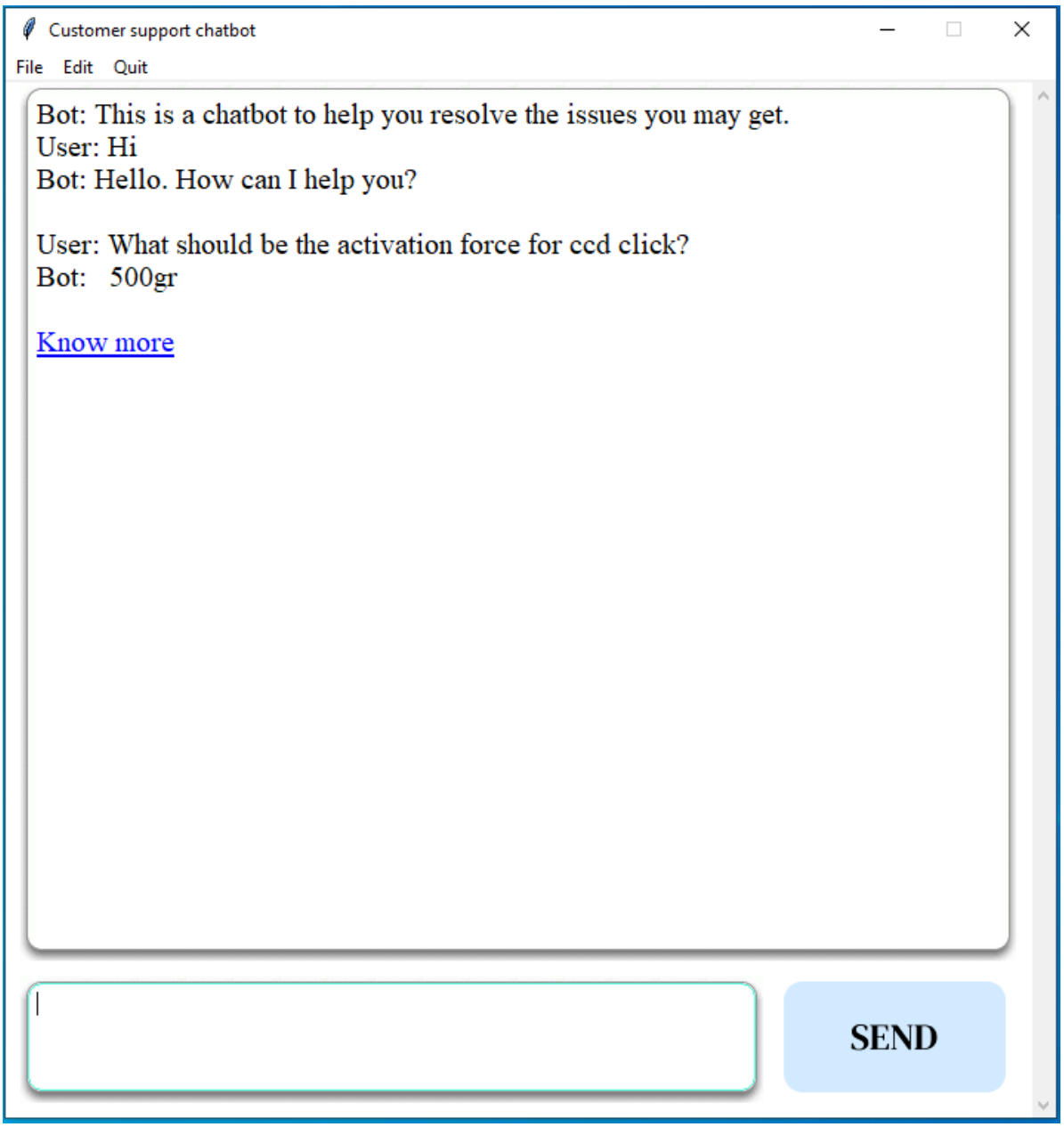


FIGURE 2.5 – Capture d'écran de l'interface définitive.

format NDJSON.

Nous avons choisi d'utiliser ce logiciel suite aux recommandations du service IA de Collins Aerospace qui l'utilise et le connait déjà.

Pour la conversion des PDF en TXT, nous avons notamment utilisé deux librairies : PyPDF2 et Fitz. La première permet de récupérer le contenu du PDF tandis que la deuxième nous permet de récupérer le sommaire. Pour chaque fichier PDF initial, nous obtenons un fichier TXT avec le texte et un fichier JSON avec le sommaire. Nous créons alors un JSON qui permet de lier ces

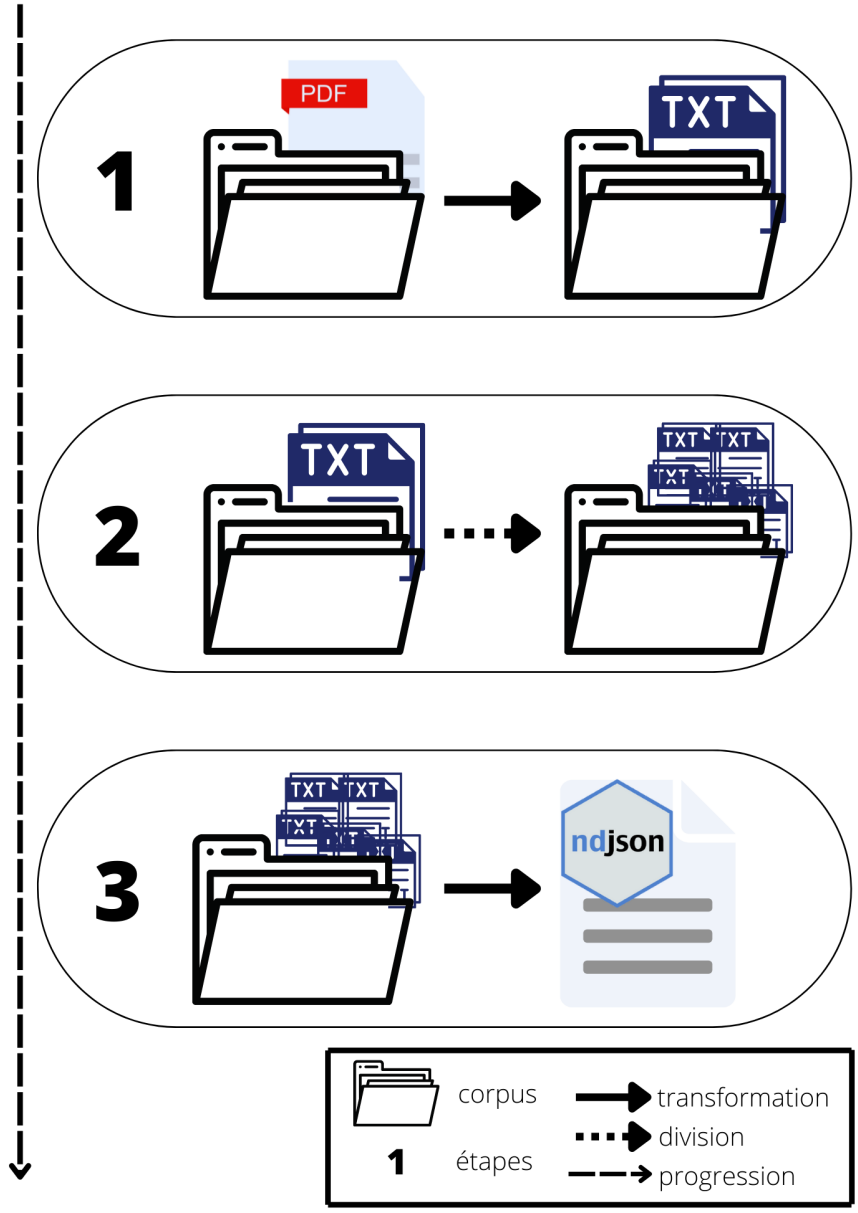


FIGURE 2.6 – Schéma de création du fichier NDJSON (figure originale).

deux ensemble. Le code est visible ci-après :

```
def get_tableOfContent(self):
    #obtenir la table des matieres
        tata = self.tata
        toc = tata.get_toc()
        dico_toc = {}
        for i in toc:
            tutu = i[0]
```

```
            tuti = i[1]
            dico_toc[tuti] = tutu
        #on met la table des matieres dans un fichier :
        self.write_JSON(self.toc_link, dico_toc)


    def PDF_to_str(self):
    #recuperer le contenu d'un PDF
        document = PDFFileReader(open(self.filePath, 'rb'))
        PDFtext = ""
        for page in range(document.numPages):
            pageObj = document.getPage(page)
            PDFtext += pageObj.extractText().replace('\n', '')
        return PDFtext


    def write_TXT_file(self, TXT_path, text):
        text_file = open(TXT_path, "w", encoding='utf-8')
        text_file.write(text)
        text_file.close()


    def infos_to_JSON(self):
        TXT_content = self.PDF_to_str()
        TXT = self.write_TXT_file(self.TXT_link, TXT_content)
        self.get_tableOfContent()
        infos_dico = {}
        infos_dico[self.fileName] = [self.toc_link, self.TXT_link]
        self.write_JSON(self.pathName+self.fileName+"_info.JSON",
                        infos_dico)
```

Nous faisons ci-après figurer un code permettant de diviser les fichiers en blocs de texte cohérents. Une fonction nous permet d'obtenir une liste des titres nous intéressant. Cette liste est constituée des titres les plus petits dans la hiérarchie de chaque partie. Par exemple, si une partie ne contient pas de sous-titre de niveau 3, nous gardons celui de niveau 2. La liste ainsi obtenue nous permet de découper le fichier texte précédemment obtenu en blocs cohérents, c'est-à-dire en blocs de texte qui appartiennent à un même thème.

```
    def retrieve_headings_with_text(self):
        headingsDico, TXTContent = self.retrieve_toc_and_text()
        #on recupere le dico et le texte
        #maintenant, on va prendre chaque titre

        TXTContent = TXTContent.lower()
        heading_list = self.retrieve_headings(headingsDico)
        blocks_dico = {}

        for heading in heading_list:
            heading = heading.lower()
            try:
                index_heading = heading_list.index(heading)
                index_next = index_heading + 1
                next_heading = heading_list[index_next]
                next_heading = next_heading.lower()
                #on a nos titres
                block = ''
                string_to_match = re.search("%s(.*)%s"%(heading,
                                        next_heading), TXTContent)
                block+=string_to_match.group(1)
                blocks_dico[heading]=block
            except:
                try :
                    block = ''
                    string_to_match = re.search("%s(.*)$"%heading,
                                                    TXTContent)
                    block+=string_to_match.group(1)
                    block = re.sub("\[|\]|\"|\"|\\\\", "", block)
                    blocks_dico[heading]=block
                except :
                    pass
        return blocks_dico
```

Les blocs de texte cohérents peuvent toutefois s'avérer trop longs (c'est-à-dire faire plus de 4096 tokens). Pour ces blocs trop longs, un troisième code est utile et visible ci-dessous. Ce code permet de diviser les blocs de texte faisant plus de 4 000 tokens. Pour des questions de sécurité, la tokenisation n'étant pas la même en Elasticsearch que celle que nous pouvons

réaliser nous-mêmes, la limite de 4 000 est en réalité abaissée à 2 500. Les blocs de texte qui sont suffisamment courts sont gardés tels quels.

```
for i in glob.glob(text_corpus):
toto = re.match(r"secondProcess/TXTBlocks\\(.*).TXT", i)
file_name = toto[1]
TXT_block = open_file(i)
toks = TXT_block.split()
if len(toks) < 2500:
    shutil.move(i, "thirdProcess/corpus/")
if len(toks) > 2500:
    toks_count = len(toks)/2500
    titi = math.floor(toks_count)
    count = 1
    while count <= titi:
        start = (count-1)*2500
        end = count*2500
        span = toks[start:end]
        TXT = ' '.join(span)
        name = "thirdProcess/corpus/" + file_name + "__" +
                                        str(count) + ".TXT"
        write_file(TXT, name)
        count+=1
    else:
        span = toks[titi*2500:]
        TXT = ' '.join(span)
        name = "thirdProcess/corpus/" + file_name + "__" +
                                        str(count+1) + ".TXT"
        write_file(TXT, name)
```

Exécutés l'un après l'autre, ces codes permettent de générer un fichier d'extension JSON au format NDJSON (voir 2.7 pour un extrait) tel que demandé par Elasticsearch 2.2. La partie texte a été tronquée pour prendre un minimum de place. Il s'agit de montrer la structure du NDJSON.

Voici le code permettant de générer le fichier JSON :

```
file_dico = ''
_id = 0
```

```
{"index":{"_index":"chatbotdatabase","_id":0}}
{"name":"945-8944_IMO-CCP_0__1","pdf_link":
"nouveauCorpus/945-8944_IMO-CCP_0_.pdf","text":"no nevolutions are typically
[...]"}
{"index":{"_index":"chatbotdatabase","_id":1}}
{"name":"945-8944_IMO-CCP_0__2","pdf_link":
"nouveauCorpus/945-8944_IMO-CCP_0_.pdf","text":"criteria outlined above in
section 4.1.3.2. nthe frequency of ccb meetings is tailored by the project team
to match the level of development activity. [...]"}
{"index":{"_index":"chatbotdatabase","_id":2}}
```


FIGURE 2.7 – Extrait du fichier NDJSON.

```
JSON_path = JSON_corpus + "JSON_corpus" + ".JSON"
for i in glob.glob(corpus):
    match = re.match(r"thirdProcess/corpus\\(.*).TXT", i)
    name = match[1]
    match2 = re.match(r"thirdProcess/corpus\\(.*)_\d*__\d*.*TXT", i)
    PDF_link = match2[1]
    text_block = open_file(i)
    indexLine = '{"index":{"_index":"chatbotdatabase",  "_id":%s}}\n' %_id
    docLine = '{"name":"%s","PDF_link":"%s","text":"%s"}\n' %(name,
                                                PDF_link, text_block)
    file_dico += indexLine
    file_dico += docLine
    _id += 1
```

La base de donnée fut alors passée en Elasticsearch [Ela18].

Elasticsearch est un moteur de recherche open-source faisant partie de la suite Elastic. Il est basé sur la librairie Apache Lucene et offre la possibilité d'une recherche rapide au sein d'une base de donnée construite sur la base de documents JSON. Nous l'avons utilisée à travers sa librairie Python, Kibana et PowerShell. Kibana est une interface utilisateur open-source qui fait également partie de la suite Elastic. Elle permet de visualiser une base de données et de faire des requêtes grâce au *Dev Tools*.

Quatre codes ont donc été écrits : le tableau 2.2 reprend l'ensemble de ces codes.

Le code permettant de mettre en relation la base de donnée en Elasticsearch et la question est disponible ci-après :

```
def get_intersection(question):
    clean_quest = question.lower()
    clean_quest = tokenize_and_clean(clean_quest)
```

Nom du fichier	Explication
01_from_PDF_to_TXT.py	Prend en compte un corpus dont les fichiers sont au format PDF, et créé un dossier contenant trois sous-dossiers qui stockent respectivement : le contenu au format TXT, la table des matières au format JSON, un fichier JSON faisant le lien entre le contenu et la table des matières.
02_from_TXT_to_TXT.py	Prend en compte le résultat du premier code pour produire un dossier contenant deux sous-dossiers avec respectivement : des blocs de texte cohérents, un fichier JSON liant le nom du bloc créé à sa partie.
03_bigFilesSplitting.py	Prend en compte le résultat du deuxième code pour produire un dossier contenant un seul sous-dossier avec le corpus complet sous forme de blocs de texte (chacun au format texte) et ne dépassant pas 4000 tokens.
04_generatingJSONCorpus.py	Prend en compte le résultat du troisième code pour produire un dossier contenant un seul fichier au format JSON et respectant les règles d'écriture du NDJSON demandé par Elasticsearch.

TABLE 2.2 – Description des codes permettant d'obtenir le JSON fourni à Elasticsearch.

```
clean_quest = list(clean_quest.split(" "))
query_body = {
    "query": {
        "terms": {
            "text": clean_quest
            }
        }
}
res = es.search(index="chatbotdatabase", body=query_body)
PDF_link = res["hits"]["hits"][0]["_source"]["PDF_link"]
text = res["hits"]["hits"][0]["_source"]["text"]

return PDF_link, text
```

Pour indexer le NDJSON en Elsaticsearch, il faut créer un mapping. La figure 2.8 montre le mapping tel qu'il a été réalisé pour ce projet. Le mapping a été fait sur Kibana grâce à la section "Dev Tools".

Les éléments du mapping sont décrits dans le tableau 2.3.

On vérifie sur Kibana que le mapping a fonctionné grâce à la commande suivante :

FIGURE 2.8 – Capture d'écran du mapping réalisé sur Kibana.

name	nom du bloc de texte
PDF_link	nom du PDF dans lequel se trouve le bloc de texte
text_block	le bloc de texte tel que découpé grâce aux codes python décrits

TABLE 2.3 – Informations inscrites dans le mapping réalisé.

HEAD /chatbotdatabase

L'indexation a été réalisée en powershell grâce à la commande suivante :

Invoke-RestMethod "http ://localhost :9200/chatbotdatabase/_bulk ?pretty" [12] -Method Post -ContentType 'application/x-NDJSON' -InFile "JSON_corpus.JSON" [13]

La figure 2.9 montre le résultat de l'indexation. On surveille en particulier la colonne *errors* qui nous indique si l'indexation a été un succès. Les erreurs d'indexation sont liées à un fichier NDJSON présentant des défauts, souvent liés à des caractères ambigus dans le texte. Dans le cas de ce projet, tout caractère engendrant des erreurs d'indexation a été supprimé lors du passage des fichiers PDF aux fichiers TXT.

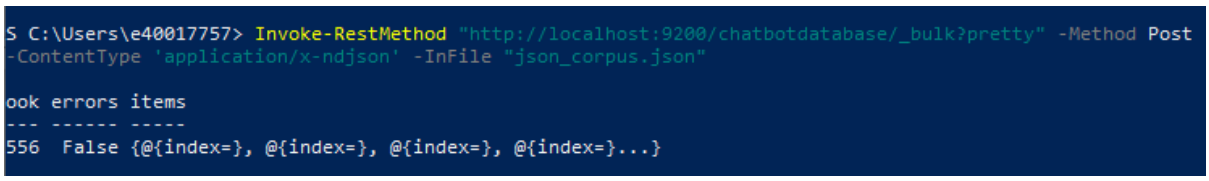


FIGURE 2.9 – Capture d'écran du résultat de l'indexation.

On peut également vérifier sur Kibana que l'indexation est complète grâce à la commande suivante :

12. lien local de la base de données dont le mapping a été créé
13. chemin vers le fichier NDJSON créé

POST chatbotdatabase/_count

Le résultat renvoyé comme visible en 2.10 nous indique que c'est bien le cas.

FIGURE 2.10 – Capture d'écran du résultat de l'indexation en Kibana.

En dernier lieu, il faut pour l'utilisateur pouvoir visualiser la réponse obtenue grâce aux processus précédents. La réponse est en effet dans un dernier temps affichée sur l'interface de l'utilisateur (grâce au code retranscrit ci-dessous). Cette réponse est accompagnée d'un lien "Know more" sur lequel l'utilisateur peut cliquer pour avoir accès au document complet dont a été extraite la réponse.

```
def send_to_chat(event=None):
    usr_input = mes_win.get("1.0", END)
    usr_input = usr_input.lower()
    textcon.insert(END, f'User: {usr_input.capitalize()}','usr')
    if usr_input.strip() in exit_list:
        textcon.config(fg='black')
        textcon.insert(END,"Bot: I hope it was useful, bye!\n")
        delete_input()
        #return root.destroy()
    else:
        textcon.config(fg='black')
        if basic_politeness(usr_input) != None:
            lab=f"Bot: {basic_politeness(usr_input)}"+'\n'
            textcon.insert(END,lab)
            delete_input()
        else:
            bot_answer, file_path = generate_answer(usr_input)
```

```
            lab = f"Bot: {bot_answer}"+'\n'
            hyperlink= HyperlinkManager(textcon)
            textcon.insert(END, lab)
            if file_path != "no_path":
                textcon.insert(END, "Know more",
                        hyperlink.add(partial(webbrowser.open,file_path)))
            textcon.insert(END, "\n")
            textcon.see(tk.END)
            delete_input()
```

Enfin, nous faisons figurer le code permettant de générer la réponse à partir du document candidat :

```
def get_answer_from_file(question):
    file_link, text = get_intersection(question)
    encoding = tokenizer.encode_plus(question, text,
                                return_tensors="pt")
    input_ids = encoding["input_ids"]
    attention_mask = encoding["attention_mask"]
    start_scores, end_scores = model(input_ids,
                attention_mask=attention_mask)
    all_tokens = tokenizer.convert_ids_to_tokens(input_ids[0].tolist())
    answer_tokens = all_tokens[torch.argmax(start_scores) :torch.argmax(end_scores)+1]
    answer = tokenizer.decode(tokenizer.convert_tokens_to_ids(answer_tokens))
    return answer, file_link
```

2.3 Les verrous

Lors de la conception du chatbot, plusieurs difficultés se sont posées. Nous avons notamment rencontré des difficultés liées à la manière dont le corpus était structuré, des difficultés liées au manque de données, et puis des difficultés moindres telles que celles inhérentes à l'utilisation de BERT ou à la contrainte posée par le proxy de l'entreprise.

2.3.1 Les verrous liés aux données

2.3.1.1 La structuration des données

Il est très rare de disposer d'un corpus parfaitement bien construit, facile à exploiter et dénué de bruit. Ceci dit, nous avions pensé lors de la présentation du sujet de stage pouvoir nous servir

de la structure des documents pour les parser et aisément obtenir des résultats dans les premiers jours. Il aurait existé une intention logique structurant l'ensemble des documents et permettant leur exploitation organisée. L'observation du corpus et les premiers travaux de prétraitements nous ont convaincu que la tâche n'était pas aussi simple qu'elle n'y paraissait.

Il s'est trouvé en effet que la structuration des données posait problème. Bien qu'il ait été question en amont du stage d'une documentation extrêmement codifiée et dont l'intention était aisément perceptible, il est apparu dans la pratique que chaque document disposait de sa propre structure comme cela a été évoqué dans le chapitre 1 et que les différents documents n'avaient rien en commun sauf pour le cas des documents présentant des exigences (comme visible en figure 4) dont la structure reste toujours la même. La plupart des documents ne présente en guise de structure que du texte, des tableaux et des figures, le tout divisé en différentes parties.

Nous avons tenté de garder la cohérence de ces parties autant que faire se peut en utilisant le sommaire. Le résultat laisse malgré tout beaucoup à désirer. Nous obtenons des résultats très propres souvent comme en figure 2.11. Mais il arrive aussi parfois d'obtenir des choses illisibles comme en 2.12.

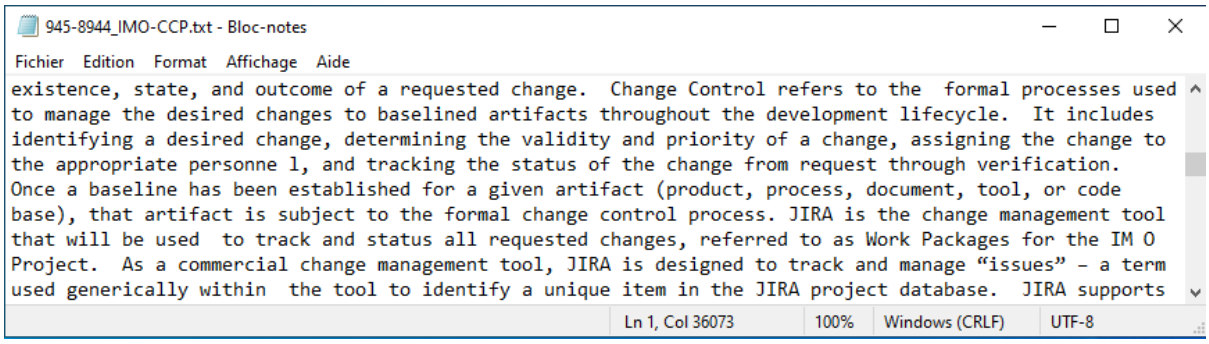


FIGURE 2.11 – Capture d'écran d'un texte propre.

2.3.1.2 Le manque de données

Nous avons utilisé le modèle BERT au sein de notre chatbot. Nous avons expliqué dans le deuxième chapitre de ce rapport la manière dont BERT fonctionne. La deuxième phase de son entrainement correspond à l'apprentissage d'une tâche, (*fine-tuning*). Cet apprentissage nécessite beaucoup de données. Il nous aurait fallu quelques centaines de milliers de paires de question-réponse. Parmi ce jeu de données, on incorporerait à la fois des données faisant référence à des connaissances générales, et des données faisant référence au domaine spécifique dont il est question. En effet, d'après un article publié dans le Computing Research Repository (CoRR) et porté sur l'amélioration de la tâche du *question answering* les performances liées à cette tâche semblent être meilleures avec des données hétérogènes [MBP+19].

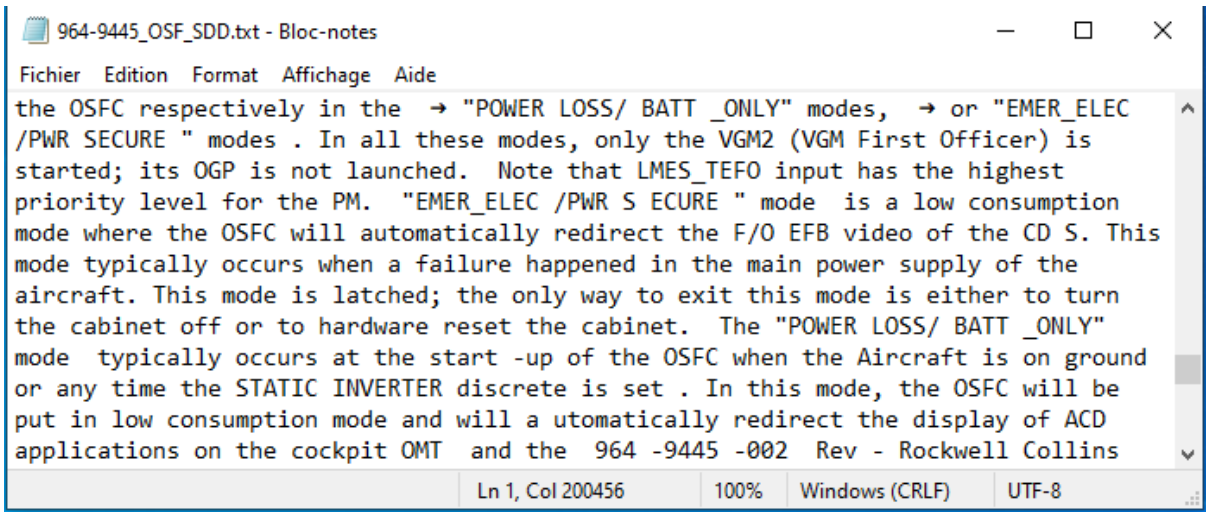


FIGURE 2.12 – Capture d'écran de bruit.

Il nous manque malheureusement ces dernières. En effet, nous avons reçu de la part du service support un document contenant 2 107 items correspondant chacun un échange qu'il y a eu avec un client ayant rencontré un problème. Parmi ces 2 107 items, 23% seulement correspondent à des demandes d'information. Parmi ces 23%, beaucoup d'items étaient inutilisables, car la case correspondant à l'échange n'avait pas été renseignée. Et parmi les échanges restant (une centaine) il aurait fallu reformuler l'ensemble de ces paires disponibles pour les mettre sous une forme plus utile que l'échange de mails. C'est pourquoi nous avons utilisé un modèle d'ores-et-déjà apte à la tâche.

Concernant le choix du modèle, nous n'avons eu que peu de difficultés. D'une part, peu de choix était disponible. D'autre part, parmi les possibilités qui s'offraient à nous via HuggingFace, quelques essais sur l'interface nous ont indiqué que la version 2 du SQuAD ne convenait pas (comme indiqué dans le chapitre 2), de même que DistilBert. En conséquence, nous sommes restés sur le même modèle entraîné à la tâche : la version 1 du SQuAD.

2.3.2 Les limites de BERT

Premièrement, lorsque je me suis rendu compte que BERT ne pouvait prendre en compte que des morceaux de 500 tokens tout au plus, je me suis renseignée sur la possibilité de pouvoir prendre en compte des morceaux plus grands. Je suis tombée sur un papier nommé CogLTX [DZYT20] présentant un outil réalisant cette tâche. Mais la licence n'était alors pas disponible aussi je n'ai pas pu utiliser cet outil.

La solution a alors été d'utiliser des *longformers* (dont nous avons parlé dans le chapitre 2) pour pouvoir utiliser BERT avec des morceaux de texte de maximum 4 096 tokens et de diviser

les documents en tant de morceaux permettant de satisfaire l'exigence de la taille.

2.3.3 Autres limitations

Par ailleurs, s'étant posé à plusieurs reprises, le problème du proxy est le plus facile à résoudre. Cet obstacle n'a pas été de grande importance. Il ne s'est présenté qu'au début du stage et le temps de comprendre comment fonctionnait son utilisation. En effet, du début à la fin du projet, il a été nécessaire de travailler derrière un proxy. L'autre problème posé par le proxy concerne le temps puisqu'il ralentit tous les processus opérés.

Solutions pour contourner le problème du proxy. Nous pouvons bien entendu supprimer le proxy. Les résultats parviennent plus rapidement. Cela pose toutefois le problème de l'accès au dépôt où se trouvent les PDF originaux et auquel nous avons accès en cliquant sur le lien "Know more". La deuxième solution serait bien entendu une machine plus puissante qui puisse compenser les problèmes de lenteur induits par l'utilisation d'un proxy.

Chapitre 3

Résultats

3.1 Les résultats

3.1.1 Limites du système

3.1.1.1 Temps d'exécution d'une requête

Un problème majeur réside dans le temps d'exécution d'une requête au chatbot.

Lors de ma première présentation du chatbot (environ deux mois et demi après avoir commencé le stage), l'un des retours (venant de la part du service IA) était le suivant : il est opportun d'avoir la compétence chatbot dans la société (sans avoir recours à un organisme externe) mais le problème de la performance temporelle est majeur. Les expériences étaient alors réalisées sur un Alienware sous Xubuntu et le temps de réponse variait entre 12 secondes et 20 secondes sur un petit échantillon de questions.

Il avait alors été indiqué que ce serait idéal si l'on pouvait passer sous la barre des 10 secondes. Il était question de gagner du temps dans le pipeline aux endroit-clés. L'une des suggestions était alors d'utiliser Elasticsearch qui opère déjà une tokenisation sur les documents qui lui sont passés.

Considérons le tableau 3.1. Sans Elasticsearch, les temps de réponse sont nettement plus élevés. Le temps moyen d'exécution d'une requête baisse drastiquement lorsque nous utilisons Elasticsearch, et encore plus sur l'Alienware. En revanche, nous verrons dans la partie suivante, que la fiabilité de la réponse (dont il est question plus tard) est moindre lorsque nous utilisons Elasticsearch.

3.1.1.2 Fiabilité de la réponse

Les réponses ne sont pas toujours fiables. Les deux raisons sont :

Environnement	Windows sur PC classique	Linux (Xubuntu) sur Alienware
Sans Elasticsearch	21 secondes	14,5 secondes
Avec Elasticsearch	12,5 secondes	8 secondes

TABLE 3.1 – Temps moyen de réponse à une requête.

— mauvais *parsing* des documents

— *fine-tuning* non adapté à la tâche

En premier lieu, comme décrit dans la partie corpus, les PDF sont des documents comportant trois types d'information : du texte, des figures et des tableaux. S'il est possible d'entrainer un modèle à une tâche avec toutes sortes d'informations, nous n'avons pas trouvé ce genre de modèle sur HuggingFace. En conséquence, le modèle utilisé ne prenait en compte que le texte. Les figures n'ont pas été conservées lors du passage de PDF à TXT, mais les tableaux sont devenus des lignes de texte sans intérêt qui ont constitué du bruit par la suite. Il arrive que certaines réponses soient contenues dans un grand bloc de texte mal formé en raison de ce bruit-là.

3.1.2 Evaluation

3.1.2.1 Avec des questions simples

Nous faisons apparaître la classification des résultats. Les tableaux 3.2 et 3.3 donnent les résultats bruts lors du test effectué sur le chatbot. Ce test est effectué de la manière suivante :

1. Nous rédigeons des questions simples sur la base des documents. Pour chaque document, nous rédigeons 5 questions pour un total de 30 questions.
2. Les questions sont passées dans le chatbot les unes après les autres.
3. Une réponse est considérée correcte lorsqu'elle ressemble exactement à ce que l'on attendait et on lui assigne la note de 1.
4. Sinon une réponse est considérée incorrecte et on lui assigne la note de 0.
5. Nous en déduisons le taux de bonne réponse pour chaque document.
6. Nous indiquons la moyenne de ces taux en dernière ligne des tableaux.

Nous rappelons que, considérant une liste de réels (x1, ..., xn), nous définissons sa moyenne arithmétique de la manière suivante :

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Document	Taux de bonne réponse
1	60 %
2	60 %
3	60 %
4	40 %
5	60 %
6	40 %
Ensemble des documents	**53%**

TABLE 3.2 – Résultats bruts du test fait sur le chatbot sans Elasticsearch.

Document	Taux de bonne réponse
1	40 %
2	30 %
3	30 %
4	20 %
5	40 %
6	30 %
Ensemble des documents	**32%**

TABLE 3.3 – Résultats bruts du test fait sur le chatbot avec Elasticsearch.

Nous remarquons que le passage en Elasticsearch cause la diminution de la qualité de réponse. Nous supposons que la raison pour laquelle cette qualité est diminuée est la même raison qui contribue à gagner un peu de temps : la tokenisation. En effet, dans le premier système la tokenisation de la question et la tokenisation des documents étaient identiques. Nous n'utilisions pas une méthode de tokenisation différente dans l'un et l'autre cas. En revanche, Elasticsearch opère sa propre tokenisation. Cette différence de tokenisation contribue probablement à l'échec de la première étape du chatbot dans laquelle nous choisissons le document candidat. En effet, lors de cette étape, la tokenisation est cruciale puisqu'il s'agit d'une discrimination de documents sur la base de mots-clés.

3.1.2.2 Avec des questions complexes

Nous n'avons pas eu le temps d'obtenir le retour des membres du service support en raison de la période estivale. Le chatbot est en cours d'essai.

3.1.2.3 Risques et intérêts

Comme nous l'avons vu, les résultats restent faibles. Nous avons dans un cas 1/3 des réponses correctes, et dans l'autre cas la moitié des réponses correctes. Nous rappelons que cet outil est destiné à être utilisé par des ingénieurs dans le cadre de projets complexes et impliquant la responsabilité de beaucoup de personnes, ainsi qu'éventuellement impliquant des problèmes

de sécurité. Le manque de fiabilité actuel de l'outil signifie pour l'ingénieur la nécessité de vérifier une réponse s'il n'est pas sûr de celle-ci. Actuellement, le lien "Know more" ne mène pas directement vers l'extrait de la documentation comportant la réponse, mais vers le document dans son entièreté. Vérifier chaque réponse représente alors une perte de temps (alors que nous souhaitons économiser du temps). En conséquence, l'outil n'est pas mature et nécessite des améliorations et des phases de tests entrainant un taux de fiabilité accru.

Malgré tout, l'intérêt d'un tel outil n'est pas négligeable. Considérons une question rentrée dans l'agent conversationnel et obtenant une bonne réponse : « What do Item1 blocking relationships do ? » (« Item1 » est mis pour ne pas dévoiler le nom d'un composant). En tapant cette question dans l'outil, nous obtenons une réponse exacte (« Blocking relationships prevent one Item1 issue from [...]. »). Cette réponse est renvoyée en moins de 10 secondes. En revanche, avec une recherche manuelle, il faut effectuer plusieurs actions pouvant mener à un travail de plusieurs dizaines de minutes, voire de plusieurs heures d'après le service support :

— Trouver le document où l'on peut dénicher la réponse.

— Faire une recherche textuelle mot par mot, éventuellement avec une expression régulière.

3.2 Améliorations possibles

3.2.1 Une base de donnée agrandie

3.2.1.1 Précision de la base de donnée

Dans le deuxième chapitre de ce rapport, nous avons présenté le corpus qui a servi à réaliser ce travail. Nous avons également indiqué que les documents formant le corpus comprenaient des éléments de la version S6 de ces documents d'ores-et-déjà sortie, ainsi que de la version S7 à venir. Comme ces documents sont en constante évolution et que les questions peuvent intervenir sur des versions diverses de ces documents, il serait intéressant pour la base de données en Elasticsearch d'ajouter pour chaque document du JSON un élément servant à l'indexation. Il s'agirait d'une paire clé-valeur correspondant à la version du document indexé. En quoi cela changerait-il le fonctionnement du chatbot ? Nous pouvons imaginer lors du lancement du chatbot une question préliminaire demandant au client d'indiquer la version des documents sur laquelle porte sa question. Le reste du processus s'effectuerait alors avec la version adéquate.

3.2.1.2 Une base de données de questions/réponses

Deux possibilités s'offrent à nous ici :

1. une base de données de questions/réponses existantes
2. une base de données de questions/réponses autogénérée

Dans un premier temps, il serait éventuellement intéressant d'ajouter au chatbot l'accès à une base de donnée contenant des questions-réponses issues d'échanges précédents avec le robot et dont la réponse était correcte. Bien entendu, ces échanges seraient eux-mêmes dépendants d'une version donnée des documents. Il faudrait donc aussi pour ces documents-là stocker la version dont il était question.

Mais par ailleurs, une base de données de questions/réponses autogénérée mérite d'être testée. En avril 2022, Collins Aerospace a organisé un rendez-vous avec une entreprise toulousaine appelée Synapse. Cette entreprise a fait du chatbot son corps de métier. Sur la base de documents, les ingénieurs y travaillant, sont capables de produire une base de données faites de questions/réponses autogénérées. Par la suite, dès qu'un utilisateur entre une question, celle-ci est associée par mesure de similarité à une question déjà existante afin de produire une réponse adéquate.

3.2.2 Autres améliorations

3.2.2.1 Figures et tableaux

Nous avons parlé précédemment de la nature hétérogène des éléments constituant les documents. Il serait opportun de pouvoir prendre en compte les figures et tableaux dans le chatbot. Il existe par exemple un corpus formé de contenus hétérogènes, HybridQA [CZC+20] :

> « Chaque question est alignée avec un tableau Wikipedia et plusieurs corpus libres liés aux entités dans le tableau. » [1]

On pourrait imaginer fine-tuner un modèle Bert pour la prise en compte de ces différents éléments et ainsi enrichir la réponse apportée à l'utilisateur.

Une autre possibilité serait de ne pas inclure figures et tableaux dans la base de données avec bien entendu l'inconvénient de perdre une partie des informations.

3.2.2.2 Amélioration du "Know more"

Dans l'état actuel du chatbot réalisé, un lien "Know more" permet à l'utilisateur d'accéder au document duquel la réponse renvoyée par le chatbot a été extraite. Ce lien mène vers le document en question, mais ne mène pas vers l'extrait précis du document.

1. "Each question is aligned with a Wikipedia table and multiple free-form corpora linked with the entities in the table. The questions are designed to aggregate both tabular information and text information, i.e., lack of either form would render the question unanswerable."

Amélioration proposée lors de la présentation de la première version du chatbot, l'amélioration du "Know more" permettrait d'arriver à la partie précise du document d'où a été extraite la réponse, comme le schématise la figure 3.1. C'est une amélioration que je n'ai pas eu le temps d'implémenter.

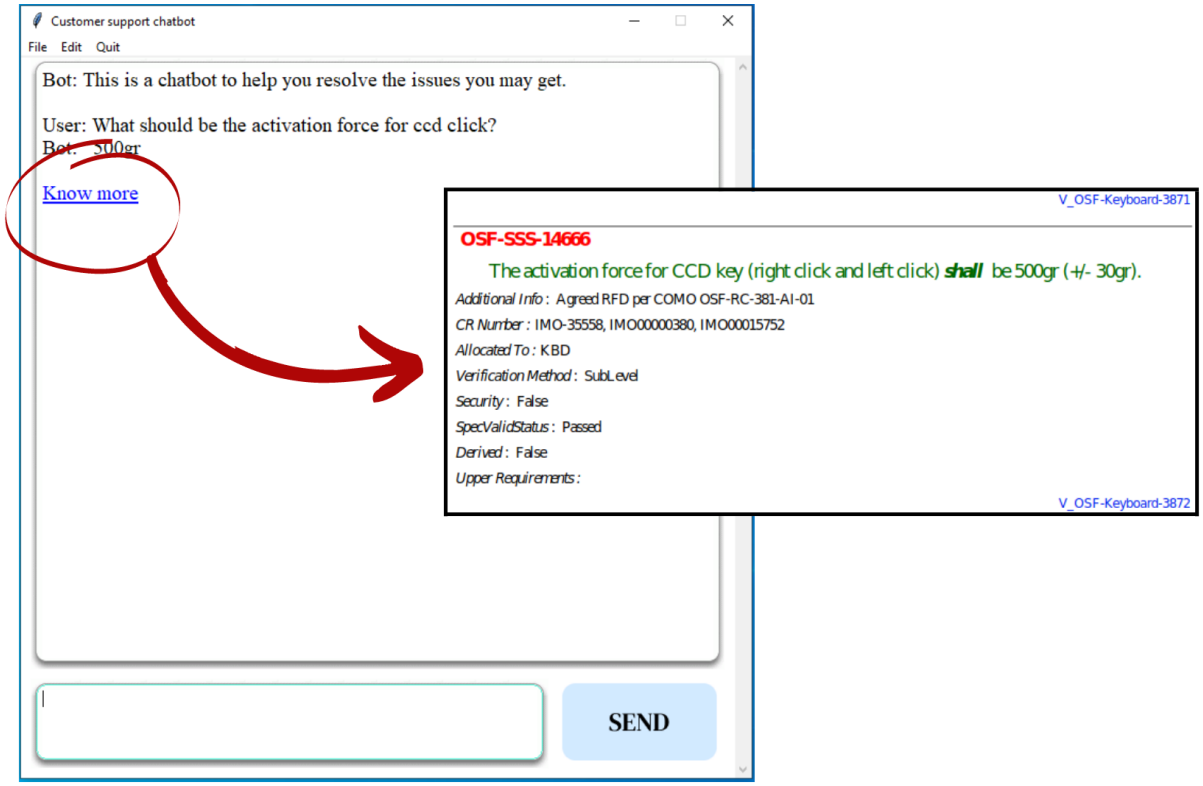


FIGURE 3.1 – Visualisation du "Know more".

3.3 Continuation

Le projet va donner lieu à des poursuites. En particulier, un stage cette année va consister à incruster le chatbot dans une interface web. L'utilisation de serveur plus puissants résoudrait peut-être le problème de la performance. Eventuellement, plus tard, de stages pour implémenter des améliorations à l'outil.

En ce qui concerne les améliorations de l'outil, je conseille donc en premier lieu de modifier la base de donnée en Elasticsearch en changeant le type de tokenisation ; il faut reproduire en Elasticsearch la tokenisation réalisée dans le code pour la question. Comme vu précédemment la tokenisation standard utilisée n'est pas apparemment pas adaptée à la tâche et entraîne des résultats médiocres. Il faudrait que la tokenisation en Elasticsearch soit la même que celle opérée lors du traitement de la question.

Par ailleurs, ayant comparé les performances entre l'ordinateur classique sous Windows et l'Alienware sur Linux, il semble évident que les performances intrinsèques de l'ordinateur influent considérablement sur le temps de réponse de l'outil. Ainsi pour améliorer le temps de réponse, il vaut mieux se tourner vers l'augmentation des performances du *hardware* utilisé.

Enfin, le chatbot est un outil de plus en plus utilisé. Ceci dit, de manière générale, ses performances laissent à désirer. Au sein du domaine du traitement automatique des langues, la tâche de répondre à des questions posées en langage naturel nécessite de faire encore de larges progrès.

Conclusion

Du 14 mars au 14 septembre 2022, j'ai eu la chance d'effectuer un stage au sein d'un des leaders dans le domaine de l'aéronautique, Collins Aerospace. Durant six mois, j'ai eu la chance de travailler dans un cadre agréable sur le site de Blagnac, dans le sud de la France, à côté de Toulouse, et au sein d'une équipe accueillante. Ce stage portait sur la création d'un chatbot capable d'aider les ingénieurs système travaillant au service support en leur fournissant un accès rapide à des réponses contenues dans les milliers de pages de documentation dont ils disposent. Mes attentes en tant que stagiaire ont été parfaitement comblées. En faisant partie de l'équipe CTS dirigée par Laurence Lamontagne, j'ai pu travailler avec des personnes qui avaient vaguement entendu parler de *data science* mais n'avaient jamais entendu parler de TAL. Pour réaliser mon chatbot, je me suis mise en relation avec divers services et différents corps de métier. J'ai par ailleurs pu approfondir mes connaissances en programmation et apprendre l'utilisation de différents softwares comme Elasticsearch. Le résultat fourni à l'entreprise et susceptible d'être amélioré est une interface répondant aux questions d'utilisateurs sur la base de documentations. En théorie, elle peut par ailleurs être adaptée à différents corpus. Il suffit de changer la base de donnée et de lancer les différents codes tel qu'indiqué dans le guide d'installation fourni à l'entreprise. Bien que les réponses ne soient pas toujours exactes, il s'agit là d'un premier pas vers l'automatisation de certaines tâches permettant au service support, voire à d'autres services de gagner du temps.

Bibliographie

[AM20] Eleni Adamopoulou and Lefteris Moussiades. An overview of chatbot technology. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pages 373–383. Springer, 2020. (Cité en page 10)

[BFG+20] Davide Buscaldi, Ghazi Felhi, Dhaou Ghoul, Josepth Le Roux, Gaël Lejeune, and Xudong Zhang. Calcul de similarité entre phrases : quelles mesures et quels descripteurs ? In *6e conférence conjointe Journées d'Études sur la Parole (JEP, 33e édition), Traitement Automatique des Langues Naturelles (TALN, 27e édition), Rencontre des Étudiants Chercheurs en Informatique pour le Traitement Automatique des Langues (RÉCITAL, 22e édition). Atelier DÉfi Fouille de Textes*, pages 14–25. ATALA ; AFCP, 2020. (Cité en page 22)

[BPC20] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer : The long-document transformer. *arXiv preprint arXiv :2004.05150*, 2020. (Cité en page 16)

[Cah17] Jack Cahn. Chatbot : Architecture, design, & development. *University of Pennsylvania School of Engineering and Applied Science Department of Computer and Information Science*, 2017. (Cité en page 10)

[CZC+20] Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Yang Wang. Hybridqa : A dataset of multi-hop question answering over tabular and textual data. *CoRR*, abs/2004.07347, 2020. (Cité en page 43)

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert : Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv :1810.04805*, 2018. (Cité en page 15)

[DZYT20] Ming Ding, Chang Zhou, Hongxia Yang, and Jie Tang. Cogltx : Applying bert to long texts. In *NeurIPS*, 2020. (Cité en page 37)

[Ela18] BV Elasticsearch. Elasticsearch. *Internet : https ://www. elastic. co/pt/,[Sep. 12, 2019]*, 2018. (Cité en page 31)

[MBP+19] Arindam Mitra, Pratyay Banerjee, Kuntal Kumar Pal, Swaroop Mishra, and Chitta Baral. Exploring ways to incorporate additional knowledge to improve natural

language commonsense question answering. *CoRR*, abs/1909.08855, 2019. (Cité en page 36)

[RJL18] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know : Unanswerable questions for squad. *CoRR*, abs/1806.03822, 2018. (Cité en page 17)

[RZLL16] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad : 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016. (Cité en page 17)

[SDCW19] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of BERT : smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019. (Cité en page 17)

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. (Cité en page 15)

[WDS+19] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers : State-of-the-art natural language processing. *arXiv preprint arXiv :1910.03771*, 2019. (Cité en page 17)

[ZGLS20] Li Zhou, Jianfeng Gao, Di Li, and Heung-Yeung Shum. The design and implementation of xiaoice, an empathetic social chatbot. *Computational Linguistics*, 46(1) :53–93, 2020. (Cité en page 10)

Annexes

Produit

Guide d'installation

Le guide d'installation est partiellement retranscrit ci-dessous.

. . . . . .

Pré-requis Installer Elasticsearch et Kibana.

Pré-requis Installer toutes les librairies nécessaires (requirements.txt)

Etape 1 Mettre les documents dans le dossier indiqué.

Etape 2 Exécuter les 3 codes de pré-traitements.

Etape 3 Exécuter le code pour obtenir le Ndjson.

Etape 4 Comment créer le mapping en Kibana.

Etape 5 Commande pour indexer en Powershell.

Etape 6 Exécuter le code pour lancer le chatbot.

Arborescence

Le projet tient dans un dossier contenant :

— les cinq scripts python permettant d'obtenir le fichier NDJSON

— un dossier contenant le corpus (fichiers pdf)

— un script python permettant d'afficher le chatbot

— un fichier *readme.md*

— un fichier *requirements.txt*

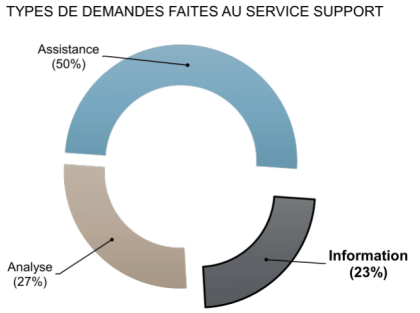
— les produits issus de l'exécution des quatre premiers scripts

FIGURE 2 – Exemple d'un document contenant des exigences.

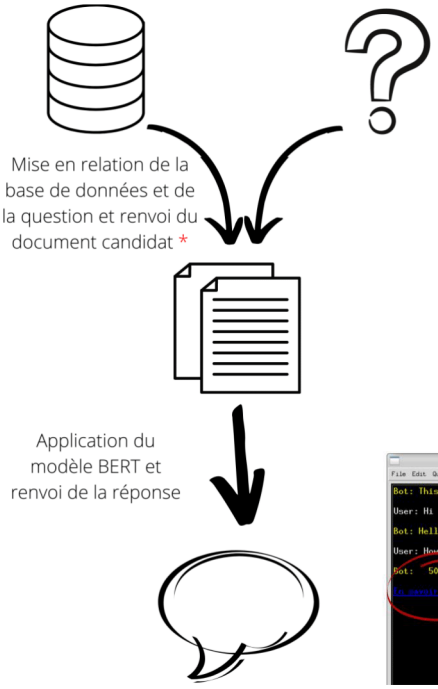
FIGURE 3 – Exemple de document structuré en parties.

Un chatbot pour le service support

Problème : répondre automatiquement à une question en langage naturel

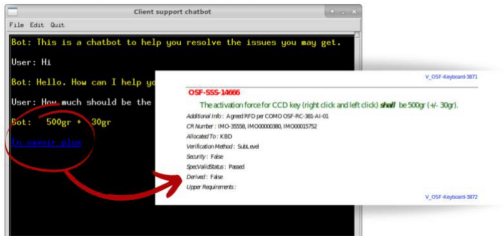


Solution : un chatbot à base d'intelligence artificielle



* Implique des pré-traitements :
- Tokenization
- Suppression des mots vides

Importance du traitement des mots vides



Domaines d'application
- Support de niveau 1
- Recherche avancée dans une documentation volumineuse
- Faciliter le *on-boarding* projet

Améliorations
- Base de données en Elasticsearch
- Amélioration du "en savoir plus"
- Barre de progression

Collins Aerospace

FIGURE 4 – Poster de présentation.