Similarité textuelle pour le ticketing

Ivanhoé Botcazou
Université des sciences d'Angers
i.botcazou@gmx.fr

Année universitaire : 2023-2024

Table des matières

1 Introduction

La gestion quotidienne de plus de 500 tickets issus de divers services représente un défi majeur en termes d'efficience opérationnelle pour les employés du Centre Logistique de Pièces de Rechange international (CLPR). L'accès rapide à des informations pertinentes et déjà traitées est crucial pour les équipes métier afin d'optimiser leur réactivité et leur efficacité. Ce projet de fin d'études vise à explorer et développer des méthodes avancées de traitement du langage naturel (NLP). Nous aimerions étudier la similarité entre les données textuelles : des tickets issus du logiciel Assist. Nous aimerions créer un outil pivot pour faciliter la recherche d'informations pertinentes pour les équipes du support technique. Soit un tableau de bord interactif pour que les utilisateurs puissent consulter efficacement la base de données des tickets Assist et ainsi identifier rapidement la solution à leur problème rencontré.

Pour répondre à cet enjeu, il est essentiel de disposer d'une base de données exhaustive des tickets, comprenant la question et la réponse associées à chaque interaction. Des informations spécifiques telles que les détails techniques des machines concernées, les pièces demandées, ou le service responsable du ticket nous serviront de fondement à l'application de filtres avancés sur le tableau de bord.

Notre travail se concentrera autour de plusieurs questions de recherche clés, notamment :

- Comment traiter et interpréter des données textuelles complexes pour qu'elles soient intelligibles par des modèles informatiques ?
- De quelle manière peut-on identifier des correspondances pertinentes entre les demandes actuelles et les tickets passés ?
- Comment adapter un modèle utilisant l'architecture des Transformeurs pour répondre aux besoins des équipes ?

Cette recherche s'inscrit dans un contexte où l'efficacité de la gestion des tickets est directement liée à la qualité du service offert par le CLPR. En développant un outil capable de réduire significativement le temps nécessaire à la recherche d'informations, ce projet entreprend la transformation des opérations quotidiennes en apportant une réponse concrète aux défis posés par le volume et la complexité des tickets traités. Nous souhaiterions ainsi améliorer les performances opérationnelles du CLPR mais aussi développer les connaissances dans le domaine du traitement automatique du langage naturel appliqué à un environnement professionnel.

2 tokenisation et embeddings

Les données textuelles dans leur forme brute, ne sont pas directement exploitables par les modèles de machine learning. Pour permettre la classification de textes, la génération de résumé, ou encore l'analyse de similarité entre deux documents, il est essentiel de prétraiter ces données afin de les convertir en vecteurs numériques. Nous décrirons dans cette première partie les principales étapes de ce prétraitement. Commençons par la normalisation des données, cette étape est cruciale car elle permet de nettoyer les données. Comment peut-on obtenir un texte sous une forme canonique ? Vient ensuite la segmentation (tokenisation) qui consiste à structurer les données pour créer un dictionnaire de valeurs textuelles. Cette étape est particulièrement complexe et peut varier selon les méthodes employées. Nous examinerons certains aspects sans rentrer explicitement dans les détails de toutes les méthodes existantes. La dernière partie consiste en la vectorisation, l'encodage des données (embedding) pour obtenir une représentation vectorielle de chaque Token.

2.1 Normalisation

La normalisation des données joue un rôle essentiel dans la réduction de la taille du vocabulaire. La conversion de tout le texte en minuscules, par exemple, permet d'unifier les variantes d'un même mot (comme "Hello" et "hello"). Cette opération peut cependant masquer certains sens implicites ou nuances. Prenons l'exemple de : "I am waiting for my parts URGENTLY". Le passage en minuscules dilue le sentiment d'urgence et l'impatience du message envoyé par notre client. Par conséquent, une connaissance métier des données est importante pour la normalisation des données textuelles.

Lors de la normalisation, il peut être judicieux d'éliminer les URL, les adresses e-mail, et les caractères non reconnus. Il convient également de supprimer les sauts de ligne et les espaces en trop. Nous pouvons aussi gérer les répétitions et effets de style ("coooool" ou "Hiiiii" ou encore "$alut"). Il est crucial de trouver un juste équilibre entre une normalisation excessive, qui pourrait occulter certains sens et une normalisation insuffisante qui serait susceptible d'augmenter inutilement la taille du dictionnaire de référence.

2.2 Dictionnaire ou tokenisation

L'étape de tokenisation que nous avons explorée en début de recherche, s'est révélée bien plus variée en termes de méthodologies que nous ne l'avions imaginée. La tokenisation consiste à créer un vocabulaire de référence pour notre corpus en associant à chaque élément textuel (mot, lettre, séquence de caractères, ...) un identifiant numérique. Par exemple via un dictionnaire python :

$$Dico = \{"hello" : 1, "engine" : 2, "mlt" : 3, \cdots, "?" : 715\}$$

2.2.1 Tokenisation basée sur les espaces et la ponctuation

Une méthode élémentaire de tokenisation repose sur l'utilisation des espaces et des caractères de ponctuation pour segmenter le texte en tokens. Cette technique présente des limites en revanche, notamment son incapacité à reconnaître correctement des groupes de mots comme "don't" ou "U.S.A", ces chaînes de caractères spécifiques ne pouvant pas être capturées par cette approche trop simpliste.

2.2.2 Tokenisation en sous-mots

La *tokenisation en sous-mots* est une méthode de tokenisation qui attribue un identifiant numérique à chaque mot fréquent tout en décomposant les mots les plus rares en plusieurs autres mots plus petits. Par exemple, le mot "hugely" serait découpé en deux tokens "huge" et le sufixe "#ly". Cette méthode est inspirée de la théorie de l'information étudiée dans le cadre de nos cours, l'idée étant d'utiliser un découpage à longueur variable pour représenter un mot ou une expression. Un exemple notable de cette approche est le Byte Pair Encoding (BPE) [9]. Voici une explication de l'algorithme :

Algorithm 1 : Byte Pair Encoding (BPE)

Initialisation : Tous les caractères Unicode sont identifiés et listés comme tokens initiaux du vocabulaire.
Itération : Identification de la paire de tokens (bigramme) la plus fréquente dans le corpus. Ajouter la fusion des deux tokens pour créer un nouveau token et l'ajouter au dictionnaire.
Finalisation : Le processus se termine lorsque la taille du vocabulaire atteint sa limite prédéfinie, incluant une diversité de mots entiers et de sous-mots.

Enfin, il faut distinguer l'utilisation des tokens représentant des entités autonomes et les tokens ayant un rôle en tant que composant de mots complexes. Le token "cat" ne sera potentiellement pas le même que le token "#cat" qui compose le mot "mecatronica". L'ajout de suffixes permet d'améliorer la finesse du vocabulaire et facilite également la compréhension des nuances linguistiques présentes dans le corpus. Cette approche raffine la représentation vectorielle des mots, permettant aux modèles de machine learning de mieux saisir les subtilités contextuelles et sémantiques du texte traité.

Pour finir, nous aimerions présenter une des méthodes de tokenisation les plus populaires : "Wordpiece" [8]. Ce tokenizer est devenu célèbre par son utilisation dans le modèle Bert de Google. L'approche de Wordpiece reste relativement semblable à celle de BPE, voici une explication de l'algorithme :

Algorithm 2 : Wordpiece

Initialisation : Tous les caractères Unicode sont identifiés et listés comme tokens initiaux du vocabulaire.
Itération : Identification de la paire de tokens (bigramme) qui a la plus grande information mutuelle. Ajouter la fusion des deux tokens pour créer un nouveau token et l'ajouter au dictionnaire.
Finalisation : Le processus se termine lorsque la taille du vocabulaire atteint sa limite prédéfinie, incluant une diversité de mots entiers et de sous-mots.

Voici quelques éléments possibles pour expliquer la partie itérative de la méthode Wordpiece, le code de cet algorithme n'étant pas opensource, il s'agit ici d'une interprétation proposé sur le site d'Huggingface [19]. Notons $\mathcal{C}$ le corpus de référence connu par le modèle pour tester les scores d'information mutuelle et ainsi constituer le dictionnaire final. Notons $\mathcal{D}_n = \{T_1, T_2, \dots, T_n\}$ le dictionnaire contenant l'ensemble des n caractères unicode constituant $\mathcal{C}$ listés comme tokens initiaux. Pour chaque étape $i \geq 0$ et $k \in [\![1, \dots, n+i]\!]$, il est possible de définir la fréquence P_k du token T_k dans le corpus $\mathcal{C}$. Pour $k, l \in [\![1, \dots, n+i]\!]$, nous noterons P_{kl} la fréquence dans le corpus $\mathcal{C}$ du bigramme T_kT_l. La formule de l'information mutuelle entre deux tokens consécutifs T_k, T_l dans le corpus $\mathcal{C}$ au temps n est donnée par :

$$I_n(T_k, T_l) = \ln(P_{kl}) - \ln(P_k) - \ln(P_l) = \ln\left(\frac{P_{kl}}{P_k \times P_l}\right)$$

Enfin, pour trouver le token à ajouter pour constituer $\mathcal{D}_{n+i+1}$, on maximise sur tous les couples de tokens consécutifs possibles à l'étape n en lien avec le corpus $\mathcal{C}$. Par croissance de la fonction logarithme, il suffit de trouver le couple "T_kT_l" avec le meilleur score comme ci dessous :

$$T_{n+i+1} = \underset{\text{"}T_kT_l\text{"}}{Argmax}\left(\frac{P_{kl}}{P_k \times P_l}\right)$$

Remarque 1. *Comme pour la méthode BPE, la méthode Wordpiece est itérative, elle incrémente d'un élément le vocalulaire à chaque étape. Enfin pour un dictionnaire donné, il est possible de donner la tokenisation d'une phrase comme sur l'exemple suivant :*

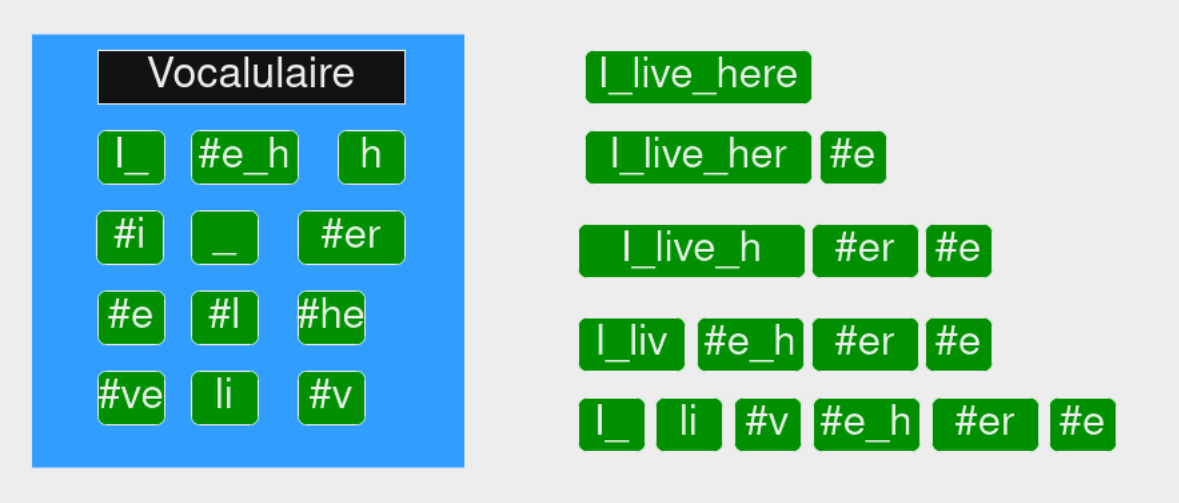


FIGURE 1 – Étapes pour la tokenisation

Remarque 2. *Quand on demande à ChatGPT de nous donner le tokenizer qu'il utilise, il nous répond :*

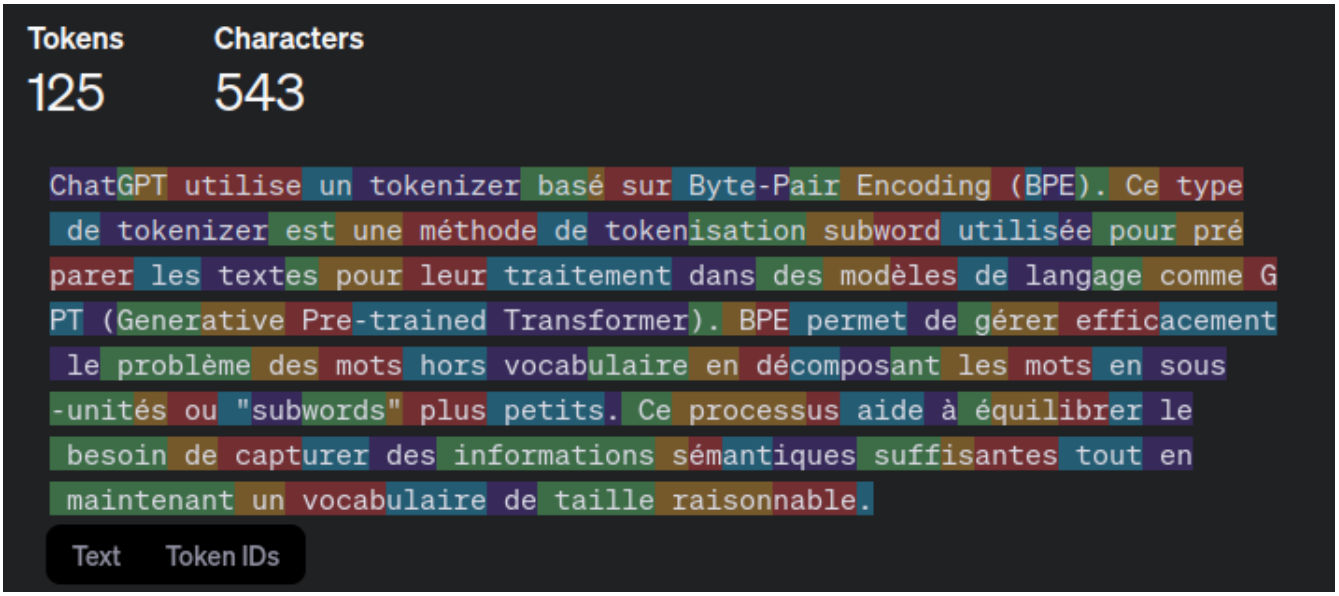


FIGURE 2 – Tokenisation de la réponse de ChatGPT [10]

2.3 Embedding pour les tokens

Maintenant nous savons comment segmenter un corpus en une liste de tokens, en revanche jusqu'à présent, les tokens ne contiennent pas d'information sémantique. L'objet est de convertir chaque token en une donnée vectorielle qui intégrerait le sens sémantique des mots. Il existe de nombreux mécanismes pour obtenir une vectorisation de chaque token, nous présenterons dans cette partie les algorithmes du type "Word2Vec" afin d'avoir une idée plus intuitive du fonctionnement de l'étape d'embedding. Cependant, avant de donner une approche profonde sur la méthodologie, nous proposons une expérience fictive pour donner l'intuition sur la vectorisation des tokens [17]. Cette expérience fictive, inspirée des neurosciences proposerait de placer des électrodes sur le crâne de plusieurs patients afin de mesurer l'activité des différentes zones du cerveau suite à une stimulation auditive. À l'écoute de certains mots, nous pourrions mesurer l'activité cérébrale moyenne en chacune des zones du cerveau. Les réponses électriques issues de ces stimulations auditives pourraient nous renvoyer des vecteurs dans un espace vectoriel multidimensionnel. Une représentation 2d via une projection, pourrait nous donner une visualisation des mots avec leur sens sémantique.

Word	Numbers	
Apple	5	5
Banana	6	5
Strawberry	5	4
Cherry	6	4
Soccer	0	6
Basketball	1	6
Tennis	1	5
Castle	1	2
House	2	2
Building	2	1
Bicycle	5	1
Truck	6	1

FIGURE 3 – Tableau des embeddings en 2d

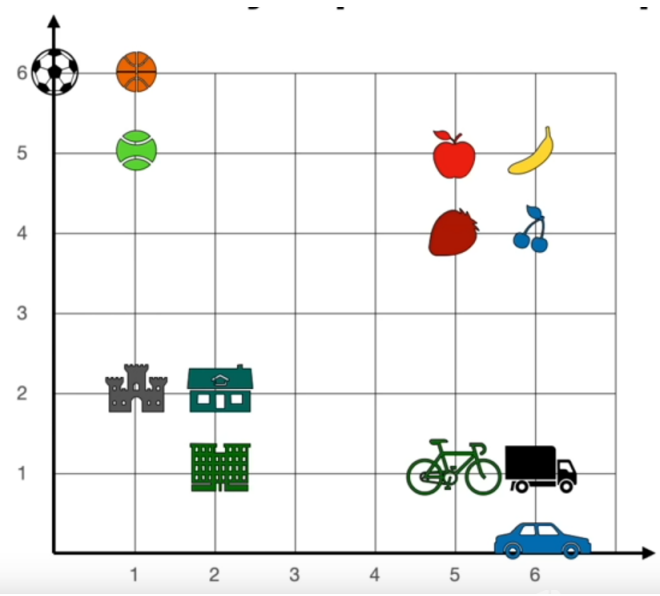


FIGURE 4 – Projection 2d des embeddings

2.3.1 Word2vec

Les méthodes de type Word2Vec [18] reposent sur le principe suivant : les mots apparaissant régulièrement à proximité les uns des autres dans un corpus partagent un sens sémantique similaire.

Ainsi, ils doivent être représentés par des vecteurs proches les uns des autres selon une certaine métrique (ex : la similarité cosinus). Pour obtenir des représentations vectorielles pour chaque token, nous allons utiliser la matrice des poids W_1 de la première couche d'un réseau de neurones de type MLP entraîné sur des tâches de classification. Le réseau de neurones est entraîné à prédire un ou plusieurs mots selon la tâche de classification. La fonction de perte d'entropie croisée sert à évaluer les prédictions à travers un coût et permet d'ajuster les poids du modèle via la rétro-propagation du gradient.

Les embeddings obtenus sont généralement de quelques centaines de dimensions et le réseau de neurones entraîné comporte souvent deux à trois couches. Dans le cadre des modèles Word2Vec, les tâches d'entraînement pour le réseaux de neurones sont de deux types :

— Bag of Words : Pour une fenêtre donnée (exemple : 9 mots), on construit un modèle capable de prédire le token central (ex : 5eme mot).

— Skip-gram : Pour un mot en entrée et une fenêtre de prédiction donnée (exemple : 4 mots), on construit un modèle capable de prédire les mots voisins.

Remarque 3. *Ces techniques d'apprentissage supervisé utilisent le corpus de référence pour ajuster les poids à chaque étape. Le schéma ci-dessous représente un réseau de neurone MLP simple de type Skip-Gram. Via une entrée vectorielle en one-hot encodding, le modèle prédit des probabilités de voisinages.*

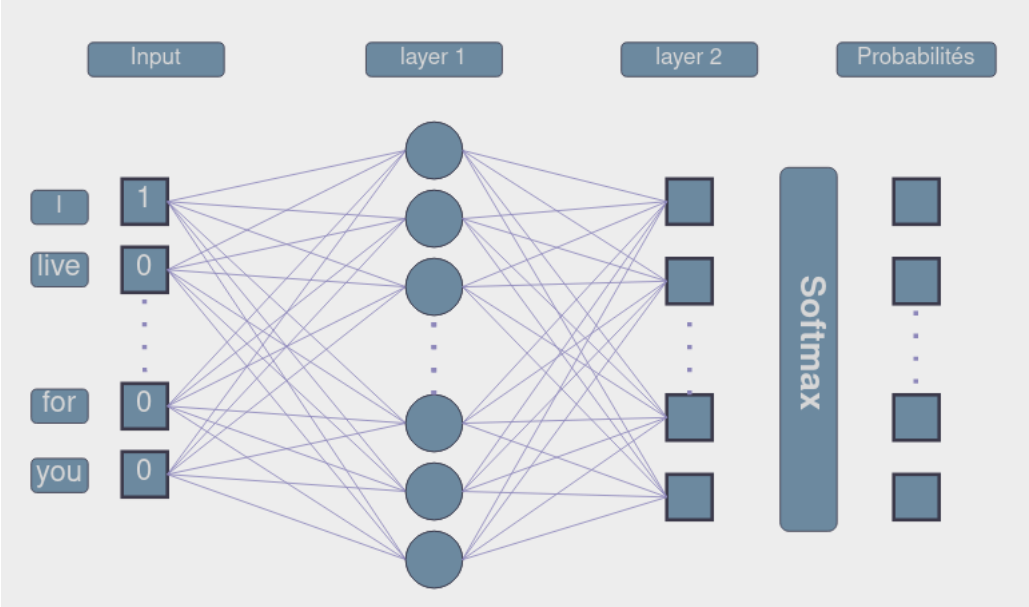


FIGURE 5 – Word2vec du type Skip-Gram

En entrée du modèle, nous utilisons une représentation one-hot encoding des mots. En sortie du modèle, nous obtenons des probabilités via l'utilisation de la fonction Softmax. Pour une dictionnaire de taille N et $p \geq 100$ neurones dans la première couche du réseau, notons $W_1 \in \mathcal{M}_{N,p}(\mathbb{R})$ la

matrice des poids associés à la couche 1. Après entraînement du modèle, l'embedding de chaque mot se retrouve via le produit matriciel : $V_k W_1 \in \mathcal{M}_{1,p}(\mathbb{R})$ ($V_k \in \mathcal{M}_{1,N}(\mathbb{R})$ est la représentation one-hot encoding du mot k).

Pour l'entraînement du modèle de type Skip Gram ci dessus, sur un corpus de référence (ex : $(w_i)_i$ une suite de mots), avec un Batch Size $T \geq 1$ et une fenêtre de référence $c \geq 2$, la fonction de coût à minimiser par ajustement itératif des poids $(W_1, W_2, \ldots, W_n)$ est la suivante :

$$\mathcal{L}oss(W_1, W_2, \ldots, W_n) = \frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-c \leq j \leq c \\ j \neq 0}} \ln(p(w_{t+j}|w_t))$$

Avec les mots w_{t+j}, w_t associés respectivement aux tokens T_k, T_l on a :

$$p(w_{t+j}|w_t) = p(T_k|T_l) = \frac{\exp\left(Layer_n(Layer_{n-1}(\ldots Layer_1(V_l)))_k\right)}{\sum_{i=1}^{N} \exp\left(Layer_n(Layer_{n-1}(\ldots Layer_1(V_l)))_i\right)}$$

Remarque 4. *Les méthodes Word2vec ont perdu en popularité depuis l'arrivée des réseaux de neurones type Transformers et l'emploi de tables d'embeddings qui se forgent au moment de l'entraînement du modèle. Les tables d'embeddings sont ainsi des poids aléatoirement initialisés, ils évoluent ensuite au cours de la phase d'apprentissage. Ces nouveaux modèles récupèrent mieux le sens sémantique des mots aux travers des embeddings construits.*

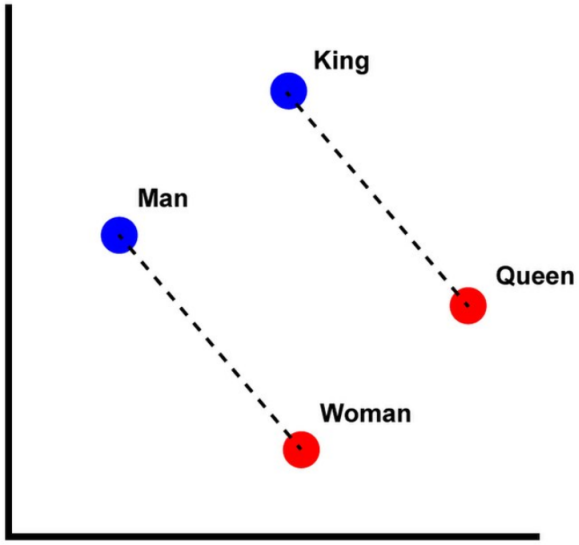


FIGURE 6 – Obtenir un concept de royauté avec les embeddings issus des modèles type Word2vec : King + Woman - Man = Queen [4]

3 Un moteur de recherche sans modèle Transformers

Le premier objectif pour pouvoir construire un moteur de recherche, serait d'être en mesure de comparer chaque nouvelle question client avec les questions clients du passé. Pour cela, il faut trouver une méthodologie permettant de plonger l'information d'une phrase entière dans un $\mathbb{R}$ espace vectoriel. Nous présenterons ensuite la similarité cosinus qui est une métrique permettant d'identifier la proximité entre deux vecteurs.

3.1 La méthode : Term Frequency-Inverse Document Frequency

La méthode TF-IDF, acronyme de "Term Frequency-Inverse Document Frequency", constitue une technique fondamentale en recherche d'information appliquée aux données textuelles. Utilisée principalement par les moteurs de recherche ces dernières années, cette méthode évalue la pertinence d'un document en fonction de l'occurrence d'un terme, d'une expression, ou d'un ensemble de termes. Initialement conçue pour améliorer la pertinence des résultats de recherche internet en lien avec une requête spécifique, nous adopterons dans un premier temps cette méthode afin de classer les questions antérieures au regard des nouvelles problématiques de nos clients.

Le principe de la méthode TF-IDF repose sur une mesure statistique attribuant à chaque terme d'un document (ou "token") un poids donné par le produit de sa fréquence dans le document donné et le logarithme de son importance inverse au sein du corpus d'apprentissage. Pour chaque question antérieure, l'importance d'un terme est calculée selon sa fréquence dans cette question et sa distribution dans l'ensemble du corpus initial. Le corpus de référence, noté $Q \coloneqq \{q_1, q_2, \ldots, q_N\}$ englobe toutes les questions passées. Le choix d'un tokenizer nous donnera pour l'ensemble des questions Q un dictionnaire de référence, celui-ci inclut tous les mots ou chaînes de caractères utilisés dans le corpus, il correspond à l'ensemble des Tokens que nous noterons $T \coloneqq \{T_1, T_2, \ldots, T_K\}$.

Dans ce cadre, l'information contenue dans chaque interrogation client (un document) est convertie en un vecteur de $\mathbb{R}^K$. Cette méthode met l'accent sur les tokens indépendamment de leur position dans le texte. Par la suite, notre étude s'attachera également à explorer d'autres méthodes de vectorisation de documents plus sophistiquées.

Remarque 5. *Avant de détailler le calcul des coefficients de la matrice des poids de référence, il convient de mentionner une étude de 2015 [11] qui estimait qu'un grand nombre des systèmes de recommandation basés sur le texte s'appuyaient sur la méthode TF-IDF. Cette méthode offre également des perspectives intéressantes pour la synthèse de textes. Cela se fait par sélection des phrases les plus significatives d'un corpus, cette application spécifique ne sera pas abordée dans le cadre de notre projet mais cet article [7] peut vous donner plus de renseignements.*

Abordons maintenant l'explication détaillée de la méthode TF-IDF, en mettant l'accent sur le calcul des poids attribués à chaque terme (ou "token") dans un document. La méthode se divise en deux composantes principales : la fréquence du terme (TF) et la log-fréquence inverse du document (IDF). Nous noterons dans la suite $P \in \mathcal{M}_{N,K}(\mathbb{R})$ la matrice des poids.

Soient $q_i \in Q$ et $T_j \in T$, on a $P_{i,j} = TF_{i,j} \times IDF_j$

$$P = \begin{pmatrix} P_{1,1} & P_{1,2} & \cdots & P_{1,K} \\ P_{2,1} & P_{2,2} & \cdots & P_{2,K} \\ \vdots & \vdots & \ddots & \vdots \\ P_{N,1} & P_{N,2} & \cdots & P_{N,K} \end{pmatrix}$$

1. **Fréquence du Terme (TF) :**

   Cette composante mesure la fréquence d'un terme T_j dans un document i donné. La fréquence est calculée selon la formule suivante :

   $$TF_{i,j} = \frac{\#\{t \in q_i \mid t = T_j\}}{\sum_{k=1}^{K} \#\{t \in q_i \mid t = T_k\}}$$

   La fréquence d'un token dans la question permet d'attribuer un poids plus élevé aux tokens qui apparaissent fréquemment dans celle-ci, reflétant donc leur importance relative dans ce contexte spécifique.

   Remarque 6. *L'appartenance d'un token t à une question q sous entend une tokenisation de la question en amont. Pour simplifier la lecture nous avons fait le choix de noter : "$t \in q$" ci-dessus au lieux de : "$t \in Tokenisation(q)$".*

2. **Fréquence Inverse du Document (IDF) :**

   Cette composante vise à diminuer le poids des termes qui apparaissent trop fréquemment dans l'ensemble du corpus, elle est calculée par la formule suivante :

   $$IDF_j = \ln\left(\frac{N}{\sum_{i=1}^{N} \mathbb{1}_{\{T_j \in q_i\}}}\right)$$

   Ainsi, un terme rare dans l'ensemble du corpus aura un poids IDF plus élevé, tandis que les termes communs à de nombreux documents auront un poids IDF réduit.

Remarque 7. *Le terme* $\frac{N}{\sum_{i=1}^{N} \mathbb{1}_{\{T_j \in q_i\}}}$ *est compris entre* 1 *et* N*. Donc* $0 \leq IDF_j \leq \ln(N)$*.*

Si le token est présent dans chaque document on a : $IDF_j = 0$*.*
Si le token est présent dans un unique document on a : $IDF_j = \ln(N)$

L'efficacité de la méthode TF-IDF repose sur son aptitude à équilibrer l'importance des termes en fonction de leur fréquence dans un document particulier et leur unicité dans l'ensemble du corpus. Les termes rares au niveau du corpus mais fréquents dans un document spécifique reçoivent un poids élevé, ce qui permet de valoriser leur pertinence spécifique. À l'inverse, les mots très communs, tels que les prépositions ("a", "the", "I", ...) ou les formules de politesse ("hello", "hi",...) n'apportent pas d'information significative sur le contenu du document, ils sont pénalisés par un poids réduit. Pour chaque terme dans un document, le poids TF-IDF est calculé comme le produit de TF et IDF. Cette opération donne un poids qui caractérise l'importance relative du terme dans le contexte du document par rapport à l'ensemble du corpus. Nous obtenons enfin une matrice de poids où les lignes représentent les documents du corpus initial et les colonnes les tokens.

3.1.1 La méthode TF-IDF et le lien avec la théorie de l'information

La méthode TF-IDF est souvent présentée comme une méthode empirique avec de nombreuses variantes dans sa mise en pratique. En s'inspirant de l'article *"An information-theoretic perspective of Tf-Idf"* [6] et par nos connaissances de la théorie de l'information, nous allons proposer une justification mathématiques pour le calcul des poids vu précédemment. Commençons par définir certains objets mathématiques iconiques de la théorie de l'information :

Définition 1.

1. Soit $\mathcal{X}, \mathcal{Y}$ deux ensembles discrets, tels que $\mathcal{X} \times \mathcal{Y}$ soit muni d'une mesure de probabilité marginale $P(x_i, y_j)$ donnant respectivement des mesures de probabilités sur $\mathcal{X}$ et $\mathcal{Y}$ telles que , :
$$\forall x_i \in \mathcal{X}, \forall y_j \in \mathcal{Y}, \quad P(x_i) = \sum_{y_j \in \mathcal{Y}} P(x_i, y_j) \ , \ P(y_j) = \sum_{x_i \in \mathcal{X}} P(x_i, y_j)$$
2. Pour tout élément $x_i \in \mathcal{X}$, la *quantité d'information* est définie par : $\ln\left(\frac{1}{P(x_i)}\right) = -\ln\left(P(x_i)\right)$.
3. Soit X, Y deux variables aléatoires définies sur un espace probabilisé $(\Omega, \mathscr{A}, \mathbb{P})$ à valeur respectivement dans $\mathcal{X}$ et dans $\mathcal{Y}$, dont la loi du couple (X, Y) s'identifie à la mesure de probabilité jointe sur $\mathcal{X} \times \mathcal{Y}$ vue précédemment :
$$\mathbb{P}(X = x_i, Y = y_j) = P(x_i, y_j) \ , \ \mathbb{P}(X = x_i) = P(x_i) \ , \ \mathbb{P}(Y = y_j) = P(y_j)$$

— On définit l'*entropie* (self-entropy) de la variable aléatoire X par la formule suivante :

$$H(X) = -\sum_{x_i \in \mathcal{X}} \mathbb{P}(X = x_i) \ln(\mathbb{P}(X = x_i)) = -\sum_{x_i \in \mathcal{X}} P(x_i) \ln(P(x_i))$$

— On définit l'*entropie conjointe* (joint entropy) du couple de variables aléatoires (X, Y) par la formule suivante :

$$H(X,Y) = -\sum_{x_i \in \mathcal{X}} \sum_{y_j \in \mathcal{Y}} P(x_i, y_j) \ln(P(x_i, y_j))$$

— On définit l'*entropie conditionnelle* de la variable X sachant la variable Y par la formule suivante :

$$H(X|Y) = -\sum_{y_j \in \mathcal{Y}} P(y_j) \sum_{x_i \in \mathcal{X}} P(x_i|y_j) \ln(P(x_i|y_j))$$

Remarque 8. *L'entropie reflète le degré d'incertitude (le désordre) d'une variable aléatoire face à ses réalisations possibles. Pour une variable aléatoire constante, l'entropie est nulle. Pour une variable aléatoire Z discrète avec N états équidistribués (loi uniforme) on a :*

$$H(Z) = -\sum_{i=1}^{N} \mathbb{P}(Z = z_i) \ln(\mathbb{P}(Z = z_i)) = -\sum_{i=1}^{N} \frac{1}{N} \ln\left(\frac{1}{N}\right) = \ln(N)$$

4. L'*information mutuelle mutuelle entre deux états* x_i, y_j est donnée par la formule :

$$\ln\left(\frac{P(x_i, y_j)}{P(x_i)P(y_j)}\right)$$

Remarque 9. *En cas d'indépendance des évènements $\{X = x_i\}$ et $\{Y = y_j\}$, l'information mutuelle entre les deux états est nulle.*

5. L'*information mutuelle mutuelle entre deux variables aléatoires* X et Y est donnée par la formule :

$$\begin{aligned}
I(X,Y) &= \sum_{x_i \in \mathcal{X}} \sum_{y_j \in \mathcal{Y}} P(x_i, y_j) \ln\left(\frac{P(x_i, y_j)}{P(x_i)P(y_j)}\right) \\
&= \sum_{x_i \in \mathcal{X}} \sum_{y_j \in \mathcal{Y}} P(x_i, y_j) \ln\left(P(x_i, y_j)\right) - \sum_{x_i \in \mathcal{X}} \ln\left(P(x_i)\right) \sum_{y_j \in \mathcal{Y}} P(x_i, y_j) - \sum_{y_j \in \mathcal{Y}} \ln\left(P(y_j)\right) \sum_{x_i \in \mathcal{X}} P(x_i, y_j) \\
&= -\sum_{x_i \in \mathcal{X}} \ln\left(P(x_i)\right) P(x_i) - \sum_{y_j \in \mathcal{Y}} \ln\left(P(y_j)\right) P(y_j) + \sum_{x_i \in \mathcal{X}} \sum_{y_j \in \mathcal{Y}} P(x_i, y_j) \ln\left(P(x_i, y_j)\right) \\
&= H(X) + H(Y) - H(X,Y)
\end{aligned}$$

or

$$\begin{aligned}
H(X) - H(X,Y) &= -\sum_{x_i \in \mathcal{X}} P(x_i) \ln\left(P(x_i)\right) + \sum_{x_i \in \mathcal{X}} \sum_{y_j \in \mathcal{Y}} P(x_i, y_j) \ln\left(P(x_i, y_j)\right) \\
&= -\sum_{x_i \in \mathcal{X}} \sum_{y_j \in \mathcal{Y}} P(x_i, y_j) \ln\left(P(x_i)\right) + \sum_{x_i \in \mathcal{X}} \sum_{y_j \in \mathcal{Y}} P(x_i, y_j) \ln\left(P(x_i, y_j)\right) \\
&= \sum_{x_i \in \mathcal{X}} \sum_{y_j \in \mathcal{Y}} P(x_i, y_j) \ln\left(\frac{P(x_i, y_j)}{P(x_i)}\right) = \sum_{x_i \in \mathcal{X}} \sum_{y_j \in \mathcal{Y}} P(y_j|x_i) P(x_i) \ln\left(\frac{P(y_j|x_i) P(x_i)}{P(x_i)}\right) \\
&= \sum_{x_i \in \mathcal{X}} P(x_i) \sum_{y_j \in \mathcal{Y}} P(y_j|x_i) \ln\left(P(y_j|x_i)\right) \\
&= -\sum_{x_i \in \mathcal{X}} P(x_i) H(Y|X = x_i) \\
&= -H(Y|X)
\end{aligned}$$

de même on trouver que

$$H(Y) - H(X,Y) = -H(X|Y)$$

Enfin

$$I(X,Y) = H(X) + H(Y) - H(X,Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

Remarque 10. *L'information mutuelle entre deux variables aléatoires est une application symétrique :* $I(X,Y) = I(Y,X)$

Cherchons maintenant à trouver un lien entre le calcul des poids dans la méthode TF-IDF et les concepts de la théorie de l'information vus précédemment. Nous reprenons dans cette partie la démonstration proposée dans l'article *"An information-theoretic perspective of Tf-Idf"* [6]. Soit $\mathcal{Q} = \{q_1, \ldots, q_N\}$, un ensemble de questions client, et $\mathcal{T} = \{t_1, \ldots, t_K\}$, le dictionnaire de référence reprenant tous les mots présents dans l'ensemble des questions $\mathcal{Q}$. Soit D et T deux variables aléatoires à valeurs dans $\mathcal{Q}$ et dans $\mathcal{T}$, modélisant le tirage respectivement d'une question et d'un mot selon les lois empiriques suivantes :

$$P(D = q_i) = \frac{1}{N} \ , \ P(T = t_j) = \frac{1}{N} \sum_{i=1}^{N} f_{i,j}$$

On rappelle que $f_{i,j}$ est la fréquence du mot j dans la question i donnée par la formule de coefficient $TF_{i,j}$.

Considérons l'information mutuelle entre D et T, on trouve ainsi :

$$I(D,T) = H(D) + H(T) - H(D,T) = H(D) - H(D|T)$$

Or :

$$H(D) = -\sum_{i=1}^{N} P(D = q_i) \ln(P(D = q_i)) = -\sum_{i=1}^{N} \frac{1}{N} \ln\left(\frac{1}{N}\right) = \ln(N)$$

Intéressons nous à la quantité :

$$H(D|T) = -\sum_{j=1}^{K} P(T = t_j) H(D|T = t_j)$$

On a :

$$H(D|T = t_j) = \sum_{i=1}^{N} P(D = q_i|T = t_j) \ln(P(D = q_i|T = t_j))$$

Soit $t_j \in \mathcal{T}$, notons N_j le cardinal du sous-ensemble $\mathcal{Q}_j$ des documents contenant le mot t_j.

Or :

$$\begin{cases} P(D = q_i|T = t_j) = \dfrac{1}{N_j} & \text{si} \quad q_i \in \mathcal{Q}_j \\ P(D = q_i|T = t_j) = 0 & \text{si} \quad q_i \notin \mathcal{Q}_j \end{cases}$$

Donc :

$$\begin{aligned} H(D|T = t_j) &= \sum_{q_i \in \mathcal{Q}_j} P(D = q_i|T = t_j) \ln(P(D = q_i|T = t_j)) + \sum_{q_i \notin \mathcal{Q}_j} P(D = q_i|T = t_j) \ln(P(D = q_i|T = t_j)) \\ &= \sum_{q_i \in \mathcal{Q}_j} \frac{1}{N_j} \ln\left(\frac{1}{N_j}\right) + 0 = N_j \times \frac{1}{N_j} \ln\left(\frac{1}{N_j}\right) = \ln\left(\frac{1}{N_j}\right) \end{aligned}$$

On trouve donc :

$$\begin{aligned} I(D,T) = H(D) - H(D|T) &= \ln(N) + \sum_{j=1}^{K} P(T = t_j) \ln\left(\frac{1}{N_j}\right) \\ &= \sum_{j=1}^{K} P(T = t_j) \ln(N) + \sum_{j=1}^{K} P(T = t_j) \ln\left(\frac{1}{N_j}\right) \\ &= \sum_{j=1}^{K} P(T = t_j) \ln\left(\frac{N}{N_j}\right) = \sum_{j=1}^{K} P(T = t_j) \times IDF_j \\ &= \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{K} f_{i,j} \times IDF_j = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{K} TF_{i,j} \times IDF_j \end{aligned}$$

Cette dernière égalité nous montre comment retrouver les poids calculés via la méthode TF-IDF grâce à l'expression de l'information mutuelle entre les variables aléatoires D et T.

3.2 Indice de similarité entre deux documents

Nous avons exploré plusieurs méthodes pour convertir des données textuelles en vecteurs. L'objectif de cette partie est de proposer un moyen pour comparer la proximité sémantique entre les mots ou les questions. Quelle métrique pourrions-nous employer pour obtenir un indice de similarité entre des mots ou des documents ? Nous exploiterons les propriétés géométriques des espaces euclidiens pour introduire la similarité cosinus.

Dans les parties précédentes de ce rapport, nous avons proposé plusieurs méthodes pour plonger des données textuelles dans un espace vectoriel de grande dimension. Nous avons présenté l'exemple de la méthode TF-IDF, capable de fournir pour chaque question un embedding de la taille du dictionnaire. Chaque question est assimilée à un vecteur de $\mathbb{R}^K$, où K est la taille du dictionnaire. Rappelons que $\mathbb{R}^K$ est un espace vectoriel réel de dimension finie et que nous pouvons le munir du produit scalaire usuel $\langle \cdot, \cdot \rangle$ (une forme bilinéaire, symétrique et définie positive sur $\mathbb{R}^K \times \mathbb{R}^K$ à valeurs dans $\mathbb{R}$).

Propriété 1 (Inégalité de Cauchy-Schwarz)**.**
Pour tous vecteurs $x, y \in \mathbb{R}^K$, l'inégalité suivante est satisfaite :

$$\langle x, y \rangle \leq |\langle x, y \rangle| \leq \|x\| \cdot \|y\|,$$

avec égalité si et seulement si x et y sont colinéaires.

Démonstration. Soit $x, y \in \mathbb{R}^K$,

on a :

$$\begin{aligned} \|x + \lambda y\|^2 = \langle x + \lambda y, x + \lambda y \rangle &= \langle x, x \rangle + 2\lambda \langle x, y \rangle + \lambda^2 \langle y, y \rangle \\ &= \|x\|^2 + 2\lambda \langle x, y \rangle + \lambda^2 \|y\|^2 \end{aligned}$$

La fonction $\lambda \mapsto \|x\|^2 + 2\lambda \langle x, y \rangle + \lambda^2 \|y\|^2$ est polynomiale de degré deux, elle n'est jamais négative car : $\forall \lambda \in \mathbb{R}, \|x + \lambda y\| \geq 0$. On en déduit que sont discriminant $\Delta \leq 0$.

Or : $\Delta = 4\langle x, y\rangle^2 - 4\|x\|^2 \cdot \|y\|^2$ Donc :

$$\begin{aligned} & \Delta \leq 0 \\ \Leftrightarrow \quad & 4\langle x, y\rangle^2 - 4\|x\|^2 \cdot \|y\|^2 \leq 0 \\ \Leftrightarrow \quad & 4\langle x, y\rangle^2 \leq 4\|x\|^2 \cdot \|y\|^2 \\ \overset{(*)}{\Leftrightarrow} \quad & |\langle x, y\rangle| \leq \|x\| \cdot \|y\| \end{aligned}$$

(*) Par croissance de la fonction carrée sur $\mathbb{R}^+$.

$\Rightarrow$ Dans le cas où x et y sont colinéaires, par bilinéarité du produit scalaire nous donne l'égalité.
$\Leftarrow$ Dans le cas de l'égalité $|\langle x, y\rangle| = \|x\| \cdot \|y\|$, on en déduit que $\Delta = 0$. Ainsi il existe un unique λ_0 tel que $\|x + \lambda_0 y\| = 0$. Par le caractère définie de la norme on a donc : $x + \lambda_0 y = 0$. Donc x et y sont colinéaires. □

Propriété-définition 1 (Cosine Similarity)**.**
Soit $x, y \in \mathbb{R}^K \backslash \{0_K\}$, on trouve que :

$$-1 \leq \frac{\langle x, y\rangle}{\|x\| \cdot \|y\|} \leq 1$$

La Cosine Similarity entre les deux vecteurs x et y est donnée par la valeur : $\dfrac{\langle x, y\rangle}{\|x\| \cdot \|y\|}$.

Par bijectivité de la fonction $\cos : [0, \pi] \to [-1, 1]$, il existe un unique $\theta \in [0, \pi]$ tel que :

$$\cos(\theta) = \frac{\langle x, y\rangle}{\|x\| \cdot \|y\|}$$

Cette propriété nous permet de retrouver une notion de géométrie, θ est l'angle entre les vecteurs x et y semblable à l'intuition que nous pouvons avoir si nous étions dans le plan.

Remarque 11. *Nous utiliserons ce score de ressemblance dans le moteur de recherche que nous allons créer afin d'indexer les questions par ordre de similarité croissante selon la similarité cosinus. Le choix de cette métrique est motivé par son efficacité avec des données creuses.*

Exemple 1 (Cosine Similarity with Python [3])**.**

```python
import numpy as np

u = np.array([1, 1, 0, 0])
v = np.array([1, 0, 0, 0])
w = np.array([2, 1, 0, 0])

def cosine_similarity(x, y):
    # Compute the dot product between x and y
    dot_product = np.dot(x, y)
    # Compute the L2 norms (magnitudes) of x and y
    magnitude_x = np.sqrt(np.sum(x**2))
    magnitude_y = np.sqrt(np.sum(y**2))
    # Compute the cosine similarity
    cosine_similarity = dot_product / (magnitude_x * magnitude_y)
    return cosine_similarity

print(cosine_similarity(u, v))
    #-> 0.707
print(np.arccos(cosine_similarity(u, v)))
    #-> 0.013
print(np.linalg.norm(u-v))
    #-> 1.0

print(cosine_similarity(u, v))
    #-> 0.948
print(np.arccos(cosine_similarity(u, v)))
    #-> 0.005
print(np.linalg.norm(u-v))
    #-> 1.0

print(0.013/0.005)
#-> 2.44
```

En effet, les embeddings de documents contiennent souvent de nombreux zéros, la similarité cosinus mesure la proximité angulaire plutôt que l'amplitude des vecteurs.Considérez l'exemple suivant où les vecteurs v et w sont tous deux à une distance euclidienne de 1 du vecteur u. Cependant, il existe un facteur de 2 de différence dans leur proximité angulaire. Dans ce cas, nous dirions que le vecteur w est deux fois plus semblable à u selon le score de similarité cosinus que le vecteur v.

3.3 Limites de la méthode TF-IDF

Pour créer le moteur de recherche, nous avons développé une première version de l'application en utilisant les embeddings de documents produits par la méthode TF-IDF. Les résultats étaient satisfaisants et meilleurs qu'avec un simple filtre par mots-clés. Cependant, les faiblesses de cette méthode résident principalement dans la non-prise en compte de l'emplacement des mots dans la phrase et la production d'embeddings dont la dimension est égale à la taille du dictionnaire total.

La méthode TF-IDF traite chaque terme indépendamment, ce qui signifie qu'elle ne capture pas les informations syntaxiques ni sémantiques qui peuvent être dérivées de l'ordre des mots ou de leur contexte dans la phrase. Par exemple, "Manitou's team awaits a response from the dealer" et "The dealer awaits a response from Manitou's team " produiront le même vecteur avec TF-IDF malgré leurs significations différentes.

De plus, la dimensionnalité des embeddings créés par TF-IDF est directement liée à la taille du dictionnaire, ce qui peut entraîner des vecteurs très grands, particulièrement dans le cas de grands corpus de texte. Cela peut non seulement augmenter les besoins en stockage et en calcul mais aussi réduire l'efficacité du moteur de recherche en augmentant le temps nécessaire pour comparer les vecteurs.

Pour surmonter ces limites, des techniques plus avancées comme les modèles basés sur les Transformers peuvent être envisagées. Ces méthodes prennent en compte le contexte des mots et génèrent des embeddings de dimension réduite, ce qui améliore la gestion de la sémantique.

4 Mécanisme d'attention dans le deep learning

Dans cette partie, nous aborderons les réseaux de neurones du type *Transformer* pour le traitement des données séquentielles. Nous traiterons des données textuelles grâce à des méthodes de deep learning basées sur le mécanisme d'attention présenté dans l'article "Attention is All You Need" [1]. Contrairement aux réseaux de neurones récurrents (RNN) qui traitent aussi des données séquentielles en tenant compte de l'ordre d'arrivée des tokens et en utilisant un processus de mémoire, les modèles Transformer traitent les séquences dans leur ensemble sans prendre en compte l'ordre initial des tokens dans la phrase. Une innovation clé de ce type de modèle est le mécanisme d'attention que nous décrirons en détail plus loin dans ce rapport. De plus, l'architecture des Transformers a été conçue pour permettre la parallélisation des calculs, un avantage non disponible avec des RNN tels que les LSTM et les GRU. Nous pouvons classer les modèles de type Transformer en trois catégories : les modèles d'encodage uniquement tels que "BERT", les modèles de décodage uniquement tels que "GPT", et les modèles combinant encodeur et décodeur comme celui présenté dans l'article "Attention is All You Need". Nous présenterons les modèles Transformers du type encodeur, cette approche n'est pas celle chronologique de leur arrivée.

4.1 Les modèles Transformers de type encodeur

Le premier modèle Transformer a été proposé en 2017 et présenté dans l'article "Attention is All You Need" [1]. Son architecture repose sur une combinaison d'un encodeur et d'un décodeur en utilisant un empilement de blocs d'attention. Nous choisirons dans ce rapport de présenter les modèles de type encodeur simple, ils offrent une représentation bidirectionnelle des données textuelles qui sera utile pour construire un moteur de recherche sur les tickets Assist. Nos explications suivantes reposent sur la lecture des articles : "Attention is All You Need" [1] et de l'article "BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding" [2]. La vidéo "Seq. 08 / Transformers" proposée par les équipes de l'IDRIS [14] nous a aussi beaucoup inspirée et aidée dans la rédaction des parties suivantes.

4.1.1 Données d'entrée : embeddings et positional encoding

Avant de rentrer dans l'architecture du modèle en lui même, considérons-le d'abord comme une boîte noire et concentrons-nous sur les données d'entrée et de sortie de celui-ci. Pour une séquence de tokens (des "mots" dans notre cas), nous allons utiliser une table d'embeddings fournie par le modèle pour convertir chaque token en un vecteur de taille fixe ($d_{model} = 768$, dans le cas d'un BERT de base). À l'inverse des modèles récurrents qui traitent les tokens séquentiellement, les Transformers manipulent directement une séquence de taille fixe ($n = 512$ pour un BERT de base). Pour conserver une notion de position des tokens dans la séquence, une solution a été d'introduire des embeddings positionnels de même taille d_{model}. Ces embeddings positionnels (PE : positional encoding) sont combinés aux embeddings de tokens pour former l'entrée du modèle grâce à une addition deux à deux. Une technique d'encodage positionnel que nous proposons est celle présentée dans l'article "Attention is All You Need". Elle utilise des fonctions cosinus et sinus comme présenté ci dessous :

$$PE_{(pos,j)} = \begin{cases} \sin\left(pos/10^{4\times\frac{2\times\lfloor\frac{j}{2}\rfloor}{d_{model}}}\right) & \text{si : } j \equiv 0 \ [2] \\ \cos\left(pos/10^{4\times\frac{2\times\lfloor\frac{j}{2}\rfloor}{d_{model}}}\right) & \text{si : } j \equiv 1 \ [2] \end{cases}$$

Remarque 12. *Ce type d'encodage pourrait aider le modèle à mieux interpréter les séquences translaté d'un facteur k [1].*

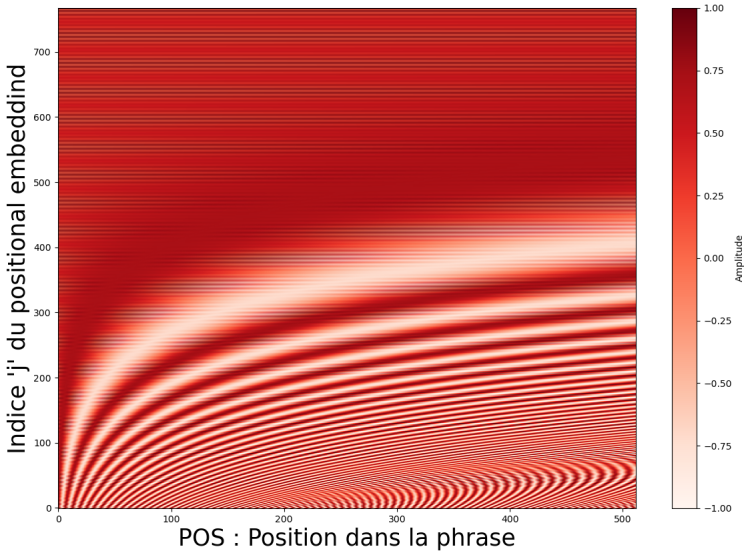


FIGURE 7 – Représentation graphique des vecteurs de positions avec $d_{model} = 768$ et $n = 512$

Une couche de réseau de neurones dense est souvent ajoutée à la suite du modèle Transformer. Cette couche permet, par exemple d'effectuer des tâches de classification, ce qui facilite l'entraînement des poids du modèle lors de la phase initiale de pré-entraînement ainsi que durant les phases de finetuning destinées à spécialiser le modèle. Nous examinerons plus en détail cette composante avec le modèle BERT ultérieurement.

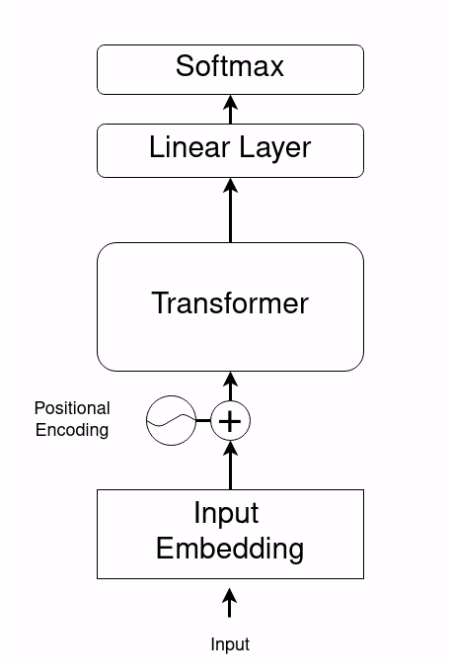


FIGURE 8 – Schéma représentatif d'un modèle Transformer avec une couche dense et un softmax pour faire de la classification multi-classes

Une fois les embeddings passés au travers du modèle, celui-ci renvoie une séquence de nouveaux vecteurs qui peuvent être considérés comme de nouvelles variables ("hidden states" ou "features") enrichies des informations de la séquence initiale. Ces vecteurs de sortie contiennent des représentations intégrées des interactions entre tous les éléments de la séquence d'entrée grâce au mécanisme d'attention.

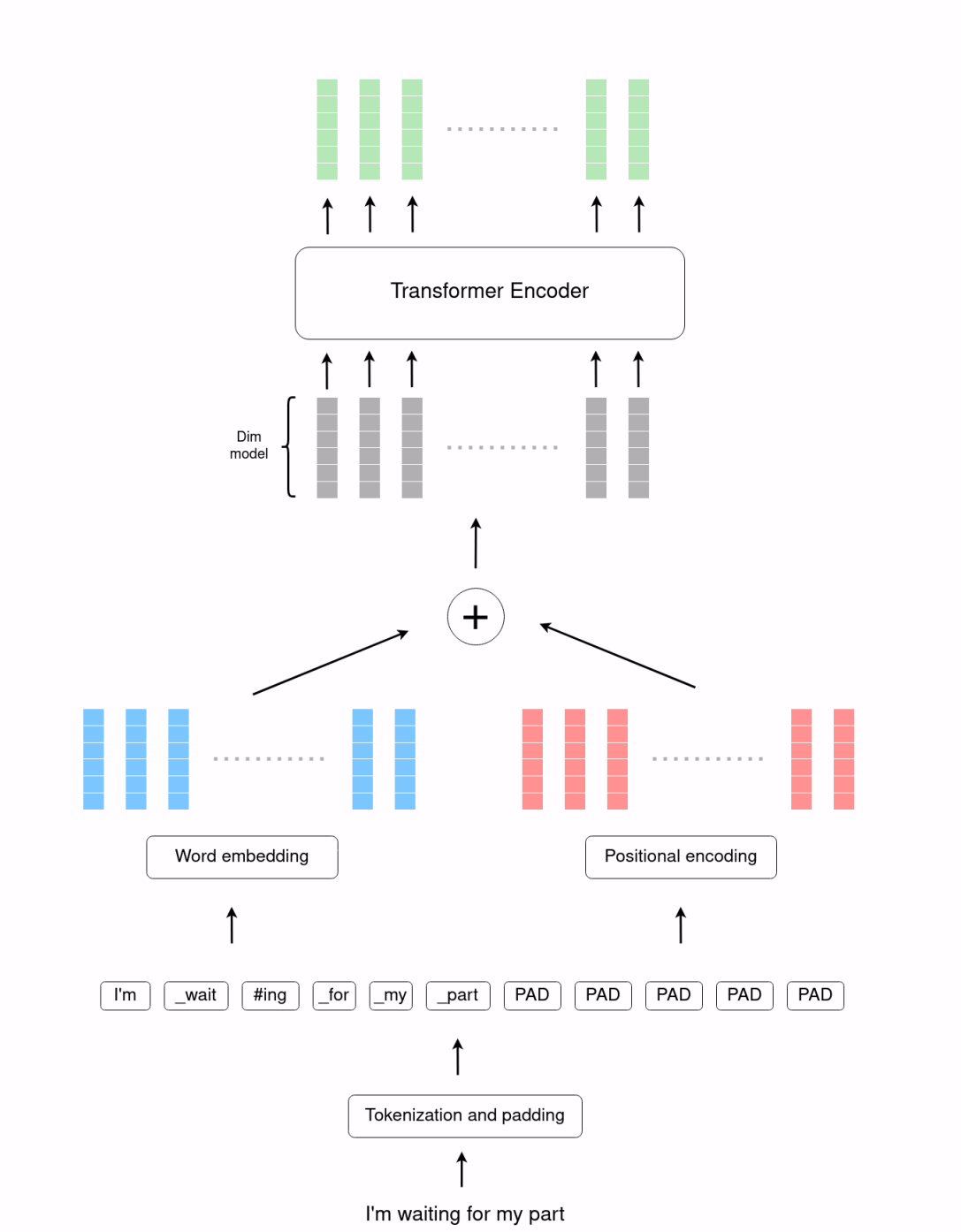


FIGURE 9 – Schéma représentatif des entrées et sorties d'un modèle Transformer type encoder

Remarque 13. *Certaines phrases courtes, une fois tokenisées, ne remplissent pas nécessairement le nombre maximal de tokens que le modèle peut recevoir en entrée. Pour pallier ce manque de tokens, il est possible de rajouter du padding, ce qui permet de combler ce vide. Le token de padding (« PAD ») possède son propre embedding qui sera également appris durant l'entraînement du modèle. L'introduction de nouveaux tokens est une pratique que nous aurons l'occasion de revoir dans la présentation du modèle BERT.*

4.1.2 Architecture d'un modèle de type encodeur

Rentrons maintenant dans les détails d'un réseau de neurones Transformer de type encodeur avec des embeddings de dimension d_{model} (*hidden size*). Comme nous l'avons vu précédemment, le modèle prend en entrée une séquence de vecteurs $(x_1, x_2, \ldots, x_n)$ et retourne en sortie une séquence de vecteurs $(y_1, y_2, \ldots, y_n)$. Dans notre présentation, nous aurons $x_i, y_j \in \mathbb{R}^{d_{\text{model}}}$, bien que la dimension des vecteurs de sortie puisse être différente. En comparaison avec les réseaux de neurones convolutifs (CNN) qui empilent des blocs de convolutions successifs, les modèles de type Transformer empilent des blocs de Transformers ($\times N$).

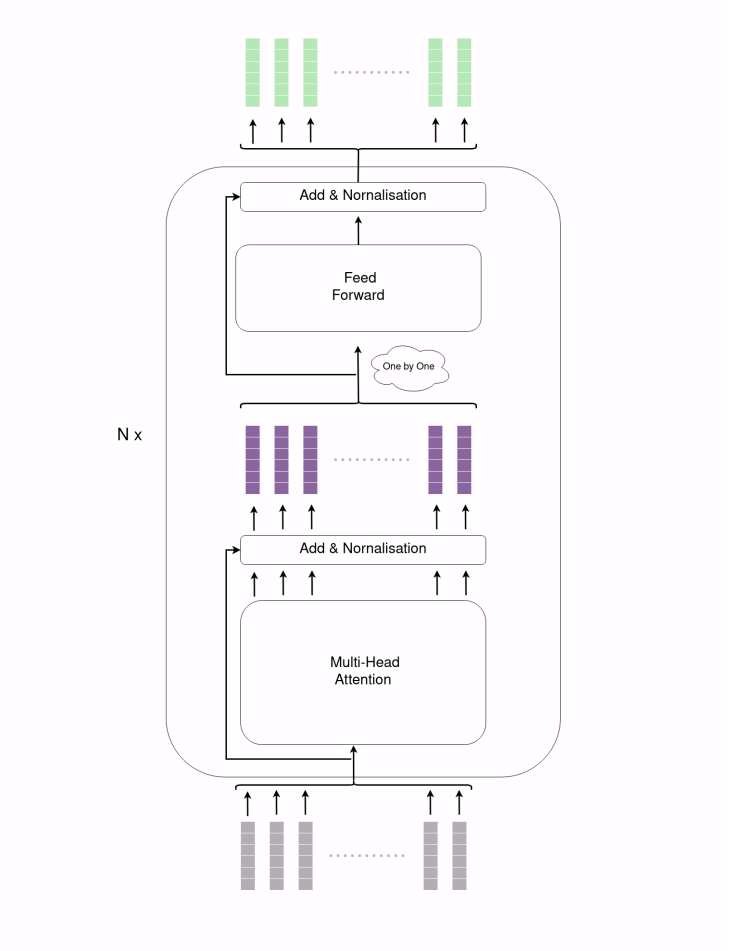


FIGURE 10 – Schéma représentatif des N blocs de Transformer mis bout à bout

Chaque bloc de Transformer possède la même architecture et se compose de deux parties distinctes : la première correspond à la multi-tête d'attention (*Multi-Head Attention*) et la seconde à une couche dense linéaire de neurones (*Feed Forward*) modifiant la représentation des vecteurs qui la traversent. Après chacune de ces parties, des connexions résiduelles entre les entrées et les sorties sont ajoutées, ainsi qu'une phase de normalisation des tableaux de valeurs. Ces pratiques permettent d'éviter l'explosion ou la disparition du gradient dans les phases d'apprentissage du modèle. La famille de vecteurs en entrée sera traitée dans son ensemble dans le bloc de *Multi-Head Attention*, d'où ressortira une famille de vecteurs de même dimension et de même cardinal. Ensuite, chaque vecteur passera chacun son tour dans un bloc dense pour obtenir une nouvelle représentation de ceux-ci dans l'espace $\mathbb{R}^{d_{\text{model}}}$ (*Feed Forward* est un réseaux de neurones linéaires avec une ou deux couches) .

4.1.3 Mécanisme d'attention et multi-têtes

Nous entrons maintenant dans la phase principale qui caractérise les modèles Transformers : le mécanisme d'attention. Nous allons décrire dans un premier temps la structure d'un bloc d'attention, puis expliquer le principe qui se cache derrière les têtes d'attention multiples.

Le mécanisme d'attention :

Le mécanisme d'attention prend en compte la séquence entière de vecteurs et la traite de manière globale pour en extraire les relations profondes entre les vecteurs d'entrée. Ce mécanisme introduit trois matrices essentielles : la matrice des *keys* (notée K), la matrice des *queries* (notée Q) et la matrice des *values* (notée V).

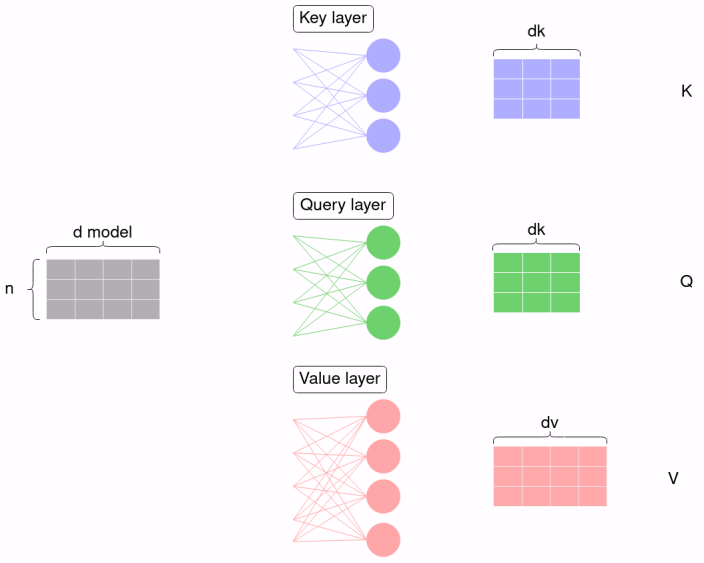


FIGURE 11 – Trois réseaux de neurones denses distincts pour obtenir les matrices K, Q et V

Ces matrices sont obtenues à partir de trois réseaux de neurones denses distincts (chacun avec une couche). Chaque vecteur d'entrée est transformé par un produit matriciel avec les matrices de poids $W_K, W_Q \in \mathcal{M}_{n,d_k}(\mathbb{R})$ et $W_V \in \mathcal{M}_{n,d_v}(\mathbb{R})$, où d_k et d_v sont des dimensions correspondant à des hyperparamètres du réseau et n est la longueur de la séquence. Une fois les matrices K, Q et V obtenues, il y a un produit matriciel entre Q et K^T normalisé par $\sqrt{d_k}$. Ceci permettra d'obtenir une matrice que nous noterons M_{sca}, elle correspond à la matrice des produits scalaires entre les vecteurs $Q_{i,:}$ et $K_{j,:}$ qui sont des représentations différentes des vecteurs initiaux dans l'espace $\mathbb{R}^{d_k}$.

Remarque 14. *La normalisation par un facteur $\sqrt{d_k}$ est présentée dans l'article "Attention Is All You Need" [1], elle a été introduite dans le but d'améliorer les phases d'entraînement du modèle en diminuant l'amplitude des valeurs prises dans la matrice QK^T. Cette pratique permet de ne pas écraser certaines valeurs de la matrice M_{sca} par le passage au Softmax à l'étape suivante. En effet, lorsque la dimension d_k augmente, l'apprentissage du modèles sans normalisations perd en stabilité [1].*

Nous proposons ici de développer la remarque en bas de page 4 de l'article "Attention Is All You Need" [1] :

Pour $0 \leq i, j \leq n$, supposons que $K[i,:]$ et $Q[j,:]$ représentent deux vecteurs aléatoires indépendants, constitués respectivement des variables iid $(K_{il})_l$, $(Q_{jl})_l$, centrées et réduites. Alors :

$$\langle K[i,:], Q[j,:] \rangle = \sum_{l=1}^{d_k} K_{il} \times Q_{jl}$$

$$\mathbb{E}\left[\langle K[i,:], Q[j,:] \rangle\right] = \sum_{l=1}^{d_k} \mathbb{E}\left[K_{il} \times Q_{jl}\right] = \sum_{l=1}^{d_k} \mathbb{E}\left[K_{il}\right] \times \mathbb{E}\left[Q_{jl}\right] = 0$$

$$\mathbb{V}\left[\langle K[i,:], Q[j,:] \rangle\right] = \mathbb{E}\left[\langle K[i,:], Q[j,:] \rangle^2\right] = \sum_{l=1}^{d_k} \sum_{s=1}^{d_k} \mathbb{E}\left[K_{il} \times K_{is} \times Q_{jl} \times Q_{js}\right]$$
$$= \sum_{l=s}^{d_k} \mathbb{E}\left[K_{il}^2\right] \times \mathbb{E}\left[Q_{jl}^2\right] + \sum_{l \neq s}^{d_k} \mathbb{E}\left[K_{il}\right] \times \mathbb{E}\left[K_{is}\right] \times \mathbb{E}\left[Q_{il}\right] \times \mathbb{E}\left[Q_{js}\right] = \sum_{l=s}^{d_k} 1 + \sum_{l \neq s}^{d_k} 0 = d_k$$

$$\mathbb{V}\left[\frac{\langle K[i,:], Q[j,:] \rangle}{\sqrt{d_k}}\right] = \frac{1}{d_k} \times \mathbb{V}\left[\langle K[i,:], Q[j,:] \rangle\right] = 1$$

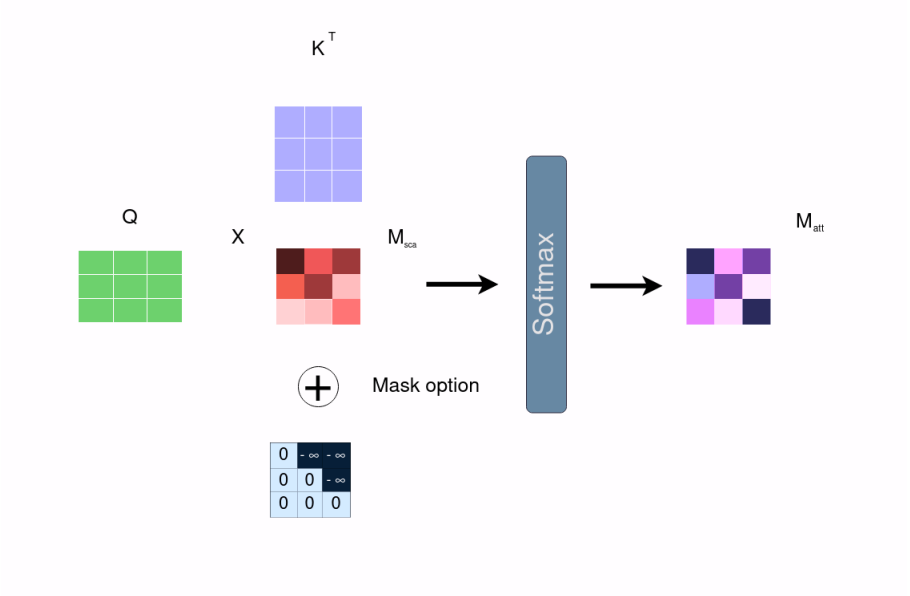
On trouve ainsi que la normalisation par $\sqrt{d_k}$ permet de réduire sous certaines hypothèses l'amplitude de la variable aléatoire $\langle K[i,:], Q[j,:] \rangle$.

Chaque ligne de la matrice M_{sca} passe ensuite par la fonction Softmax afin d'obtenir des poids d'attentions, la somme sur chaque ligne faisant alors 1. La matrice obtenue, notée M_{att}, s'appelle la *matrice d'attention.*

$$M_{\text{sca}} = \frac{QK^T}{\sqrt{d_k}}$$

$$M_{\text{att}} = Softmax(M_{\text{sca}}, axis = 1)$$

$$M_{\text{att}}(i,j) = \frac{\exp(M_{\text{sca}}(i,j))}{\sum_{k=1}^{n} \exp(M_{\text{sca}}(i,k))}$$



Remarque 15. *Il est possible de rajouter un masque à droite (*"Mask otion"*) avant le passage à la fonction softmax. Cette pratique constitue la principale distinction entre un modèle Transformers de type encodeur et un modèle Transformers de type décodeur. Dans le cas d'un modèle Transformers de type encodeur (sans Mask attention), nous parlons d'un bloc d'attention bidirectionnel, tandis que dans un modèle Transformers de type décodeur (avec Mask attention), nous parlons d'un bloc d'attention masqué unidirectionnel. Pour ce faire, il suffit d'ajouter à la matrice M_{sca}, une matrice triangulaire strictement supérieure avec des valeurs de $-\infty$ et des zéros sur la partie inférieure. Après application de la fonction softmax sur les lignes, les poids strictement au-dessus de la diagonale seront tous égaux à zéro. Cette opération force le mécanisme d'attention à ne pas prendre les tokens de droite pour déterminer les poids contextuels. Ce procédé de masquage dans les matrices d'attention est la principale différence d'architecture entre un modèle Transformers de type encodeur (BERT) et un modèle Transformers de type décodeur (GPT).*

Remarque 16. *Une troisième architecture de modèles Transformer du type encodeur-décodeur n'est pas la concaténation de ces deux types de modèles précédents. Une attention croisée avec les features issues de la partie encodeur est incluse dans la partie décodeur du modèle encodeur-décodeur.*

Remarque 17. *Il ne faut pas confondre le procédé de masquage dans la construction de la matrice d'attention (*Masked Multi-Head Attention*) avec l'introduction d'un token "MASK" que nous présenterons lors du pré-entraînement du modèle BERT.*

Une fois la matrice d'attention obtenue, nous obtenons les sorties du bloc d'attention grâce au produit matriciel entre la matrice M_{att} et la matrice de values V. Nous obtenons une matrice de $\mathcal{M}_{n,d_v}(\mathbb{R})$ où chaque ligne correspond à une combinaison linéaire des lignes de la matrice V. Nous pouvons décrire ce procédé comme une pondération de chaque token en fonctions des poids de la matrice d'attention et des autres tokens dans la séquence.

$$Attention(K,Q,V) = Softmax\left(\frac{QK^T}{\sqrt{d_k}}, axis = 1\right)V$$

On a donc pour chaque ligne $0 \leq i \leq n$:

$$Attention(K,Q,V)[i,:] = M_{\text{att}}[i,:] \cdot V = \sum_{i=1}^{n}\sum_{j=1}^{n} M_{\text{att}}[i,j] \times (E_{ij} \cdot V)\,[i,:] = \sum_{i=1}^{n}\sum_{j=1}^{n} M_{\text{att}}[i,j] \times V[j,:]$$

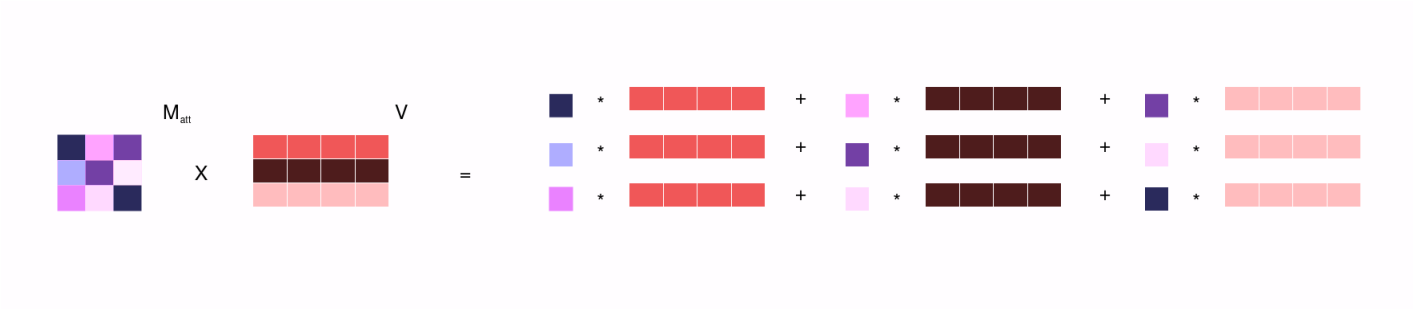


FIGURE 12 – Pondération contextuelle des vecteurs values avec les poids de la matrice d'attention

Multi-tête d'attention :

Soit h un entier, l'architecture des modèles Transformers propose d'effectuer h fois le mécanisme d'attention dans chaque bloc de Transformer, celui-ci s'effectue sur des sous représentations des matrices K, Q et V. Une tête d'attention $((head(K,Q,V)_i)_{1\leq i\leq h})$ effectue le même procédé d'attention que nous avons vu précédemment avec les matrices K_i, Q_i et V_i constituées respectivement des projections des lignes des matrices K,Q et V dans des espaces de dimensions $\hat{d_k}, \hat{d_k}$ et $\hat{d_v}$. Ces projections sont effectuées par des réseaux de neurones denses associées aux matrices de poids suivantes : $W_i^Q, W_i^K \in \mathcal{M}_{d_k,\hat{d_k}}(\mathbb{R})$ et $W_i^V \in \mathcal{M}_{d_v,\hat{d_v}}(\mathbb{R})$.

Ainsi :

$$K_i = K \cdot W_i^K, \quad Q_i = Q \cdot W_i^Q, \quad V_i = V \cdot W_i^V$$

$$head_i(K,Q,V) = Attention(K_i, Q_i, V_i)$$

Remarque 18. *Nous avons introduit les notations $\hat{d}_k$ et $\hat{d}_v$ qui ne sont pas présentes dans l'article "Attention is all you need". Avec nos notations et en lien avec ce qui est présenté dans l'article [1], un exemple d'hyper-paramètres proposés est le suivant :*

$$d_{model} = 512, \quad d_k = d_v = d_{model}, \quad h = 8, \quad \hat{d}_k = \hat{d}_v = d_{model}/h = 64$$

Enfin, les n sorties de dimension $\hat{d}_v$ issues de chacune des têtes sont concaténées pour obtenir n vecteurs de dimension $h \times \hat{d}_v$. Une représentation de chacune des sortie dans $\mathbb{R}^{d_{model}}$ est alors faite à l'aide d'un réseau de neurones dense, notons $W^O \in \mathcal{M}_{h\times\hat{d}_v, d_{model}}(\mathbb{R})$ les poids associés, ceci afin de retrouver n vecteurs à la sortie du blocs "MultiHead" de dimension d_{model}.

$$MultiHead(Q,K,V) = Concat(head_1(K,Q,V), \cdots, head_h(K,Q,V)) \cdot W^O$$

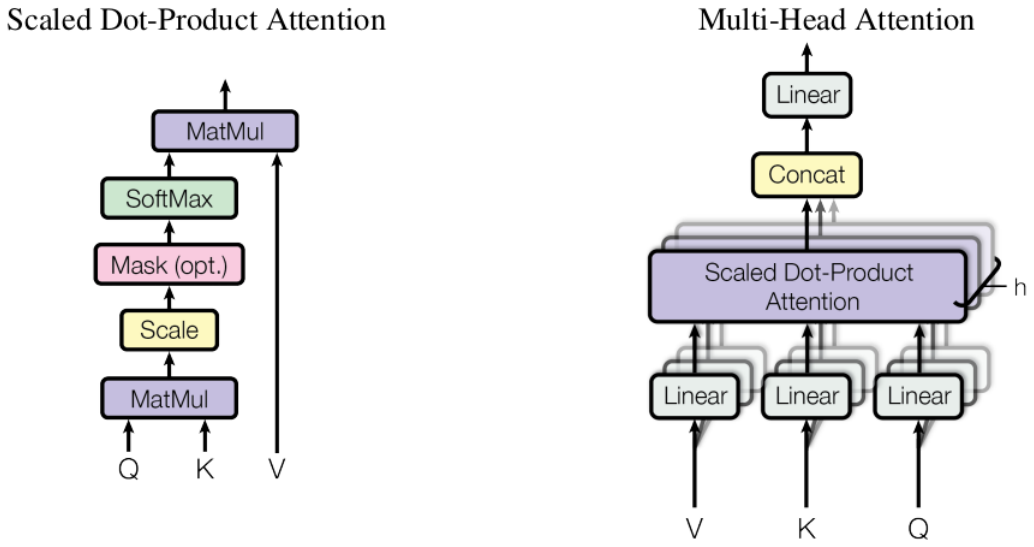


FIGURE 13 – Schémas issus de l'article "Attetion is all you need" [1] reprenant le principe d'attention et de "Multi Head"

Ces schémas concluent la partie présentant l'architecture des modèles Transformers de type encodeur. Nous avons seulement présenté à travers une remarque les modèles Transformers de type décodeur, qui sont plus adaptés pour la génération de contenu et l'interaction avec un utilisateur. Nous vous invitons à lire l'article "Improving Language Understanding by Generative Pre-Training" [5] pour plus de renseignements. Initialement, le premier modèle Transformer avait une architecture de type encodeur-décodeur. Nous ne rentrerons pas dans les détails de son architecture, mais nous vous proposons les lectures "Attention Is All You Need" [1] et "Understanding and Coding Self-Attention, Multi-Head Attention, Cross-Attention, and Causal-Attention in LLMs" [16] pour appréhender le mécanisme d'attention croisée présent dans la partie décodeur du modèle.

4.2 Le modèle BERT

Nous avons vu précédemment l'architecture des modèles Transformers, leur arrivée a bouleversé les manières de traiter les séquences de données comme le texte et les images. Les résultats qu'ils offrent sont impressionnants et leur marge de progression reste à venir. Dans cette partie, nous allons présenter le premier modèle de Transformers de type encodeur ayant fait sensations dans le milieu du NLP. Il s'agit du modèle BERT (Bidirectional Encoder Representations from Transformers) proposé par les équipes de Google en octobre 2018. Nous présenterons rapidement l'architecture de $BERT_{base}$ pour ensuite accentuer notre présentation sur le pré-entraînement du modèle de manière auto-supervisée. Enfin, nous proposerons quelques tâches pour lesquelles ce modèle pré-entraîné a pu être finetuné et dont les évaluations des résultats ont été probantes. Nous tenons aussi à souligner qu'au travers de nos lectures, nous avons intégré l'importance du pré-entraînement d'un modèle. L'architecture des Transformers, bien qu'innovante, n'est pas la seule clé de réussite. La qualité des données et leur quantité sont également cruciales. De plus, trouver une bonne manière de pré-entraîner un modèle reste un facteur majeur dans l'essor de celui-ci. Nous proposons de détailler dans les deux prochaines parties, les points majeurs que nous avons compris de l'article de référence sur le sujet : "Pre-training of Deep Bidirectional Transformers for Language Understanding" [2]

4.2.1 Architecture du modèle BERT

Comme nous les avons présentés dans la section précédente,le modèle BERT s'inscrit dans la catégorie des modèles Transformeurs du type encodeurs. Le modèle $BERT_{base}$ prend $n = 512$ vecteurs en entrée et retourne n vecteurs en sortie. Il se compose de $L = 12$ blocs de Transformers empilés (Transformers layers). Chaque bloc possède une multi-têtes d'attention comportant $h = 12$ têtes. Les modèles basés sur BERT utilisent le tokenizer WordPiece qui a été développé par les équipes de Google. Les tables d'embeddings sont initialement aléatoires, les poids pour chaque embedding de token seront appris durant la phase d'entraînement du modèle. À l'inverse de tables préalablement fixées, cette pratique laisse la liberté au modèle de changer la représentation des "mots" dans l'espace $\mathbb{R}^{d_{model}}$ ($d_{model} = 768 = 2^8 \times 3$). L'affinement des représentations vectorielles des tokens à travers la table d'embeddings se fait au fur et à mesure de l'entraînement du modèle. Dans le cas du modèle BERT, le dictionnaire contient $n_{dico} = 30K$ tokens de référence. Chaque token admet une représentation $t_i \in \{0,1\}^{n_{dico}}$ sous une forme one-hot encoding. Celle-ci peut être rattachée à son embedding dans la table du modèle via un produit matriciel $t_i \times W_{emb}$, où W_{emb} est la matrice d'embeddings de dimension n_{dico} lignes et d_{model} colonnes.

4.2.2 Introduction de nouveaux tokens

Une des innovations des équipes de Google fut d'introduire des nouveaux tokens non associés à des chaînes de caractères textuelles et ayant leur propre représentation dans l'espace $\mathbb{R}^{d_{model}}$.

$[MASK]$

Le *token de masque* : il remplace de manière aléatoire certains tokens textuels, donnant ainsi au modèle l'information de masquage des tokens textuels initialement à cette place.

$[SEP]$

Le *token de séparation* : le modèle BERT a été conçu pour avoir une grande flexibilité face aux différents contenus textuels d'entrée. Le modèle peut recevoir un paragraphe ou encore un couple paragraphe A/paragraphe B (exemple : question/réponse). Pour séparer, si besoin, ces deux entités, le modèle possède un token de séparation.

$[PAD]$

Le *token de padding* : comme nous l'avons déjà présenté, le paragraphe donné au modèle une fois tokenisé donne rarement le nombre $n = 512$ de tokens attendu par le modèle. L'input du modèle est alors compléter avec des tokens de padding, ayant leur propre représentation.

$[CLS]$

Le *token de classification* : l'innovation la plus remarquable a été d'introduire un token spécial de classification. Ce token, noté entre crochets [CLS], sera rajouté au début de chaque séquence passant par le modèle BERT. À la sortie du modèle, le premier token de sortie, que nous noterons C, pourra être utilisé par la suite comme un vecteur informatif pour des tâches de classification. L'idée à travers ce nouveau token était de trouver dans l'espace $\mathbb{R}^{d_{model}}$ une représentation « résumé » de la séquence d'entrée.

Remarque 19. *Le modèle BERT utilise comme nous l'avons vu dans la section précédente des embeddings de position. Pour différencier cependant l'appartenance au paragraphe A et au paragraphe B, des embeddings spécifiques aux deux paragraphes sont ajoutés au sein des embeddings de tokens et des embeddings de position. Nous les noterons respectivement E_A et E_B.*

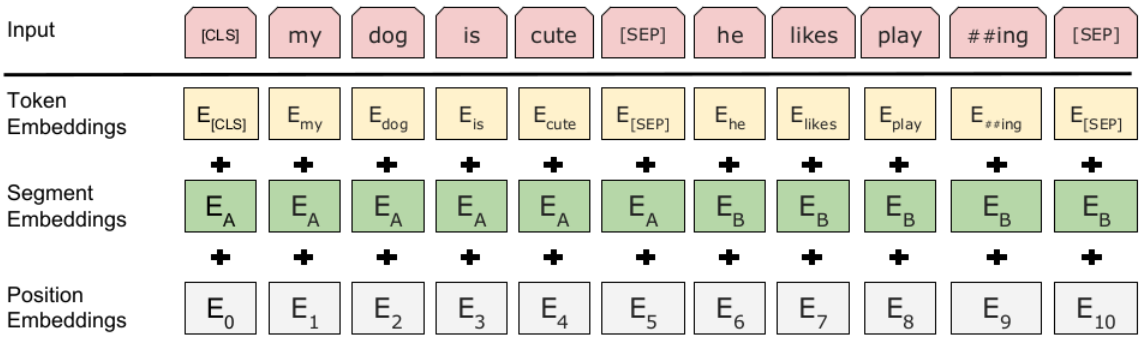


FIGURE 14 – Représentation de l'Input du modèle BERT issue de l'article [2]

4.2.3 Pré-entraînement du modèle BERT pour la classification

Dans un premier temps, le modèle BERT a été entraîné à comprendre le langage naturel, cette partie est essentielle et nécessite une grande quantité de données. Pour ce faire, les données issues d'un corpus de livres "BookCorpus"(800M de mots) ainsi que l'ensemble des données textuelles issues de Wikipédia en anglais (2500M de mots) ont été utilisées. Pour les données présentées, la phase de pré-entraînement suit une méthode d'autosupervision probabiliste.

Définition 2. L'**autosupervision** est une technique d'apprentissage automatique où un modèle utilise des données non annotées pour créer ses propres étiquettes ou signaux d'entraînement. Cela permet au modèle de s'entraîner sans avoir besoin de données préalablement étiquetées par des humains.

L'approche du pré-entraînement du modèle BERT se fait via deux tâches de classification. La première est une classification multi-classe sur l'ensemble des mots du vocabulaire, la seconde est une classification binaire en lien avec la cohérence sémantique entre la phrase A et la phrase B. Des réseaux de neurones de type MLP (Multilayer Perceptron) sont rajoutés à la sortie du modèle Transformers. Une fonction softmax (ou sigmoïde) permet d'obtenir des probabilités. Pour $n = 512$ tokens en entrée, le token $[CLS]$ sera toujours le premier token du modèle. Un token $[SEP]$ sera rajouté après chaque fin de chaque paragraphe issue du corpus de référence. En sortie du modèle, le premier token, noté $C \in \mathbb{R}^{d_{model}}$, sera un vecteur pivot pour effectuer la deuxième tâche de classification binaire durant le pré-entraînement. Pour tout indice $2 \leq i \leq n$, le token T_i de sortie sera associé au token t_i que nous avons mis en entrée à l'indice i. La fonction de coût (loss function) utilisée pour l'entraînement des poids est l'entropie croisée.

Définition 3. Pour p et q deux mesures de probabilités sur un même espace discret Ω, l'entropie croisée (cross hentrpy) entre p et q notée $H(p,q)$ est donnée par :

$$H(p,q) = -\sum_{i=1} p_i \ln(q_i) = -\sum_{i=1} p_i \ln\left(\frac{q_i \times p_i}{p_i}\right) = -\sum_{i=1} p_i \ln(p_i) - \sum_{i=1} p_i \ln\left(\frac{q_i}{p_i}\right) = H(p) + D_{KL}(p||q)$$

où $D_{KL}(p||q)$ est la divergence de Kullback-Leibler entre p et p. La divergence de Kullback-Leibler est une application à valeur positive, non symétrique, qui induit une notion de proximité entre deux mesures de probabilité sur un même espace.

Remarque 20. *La rétropropagation (Back propagation) implémentée via la méthode de descente de gradient, permet de faire évoluer les poids du modèle à chaque itération lors de la période d'entraînement. Une partie entière serait nécessaire pour expliquer le fonctionnement de l'optimiseur Adam qui est utilisé pour le pré-entraînement du modèle BERT. Nous ne développerons pas cet axe dans ce rapport.*

Tâche 1 : Modèle de langage masqué (MLM)

Pour deux phrases A et B données en entrée du modèle, une partie des tokens textuels (15 %) sont masqués de manière aléatoire. L'objet de cette tâche est de récupérer les tokens de sortie associés aux tokens masqués, afin de déduire le mot qui a été masqué. Dans 80 % des cas de masquage pour un token, celui-ci est remplacé par le token [MASK]. Dans 10 % des cas, le token n'est pas échangé et dans les 10 % restants, le token est remplacé par un autre token tiré aléatoirement dans le dictionnaire de référence. Ce procédé est utilisé pour forcer le modèle à comprendre le contexte des prédictions des tokens sans forcément lui indiquer la présence du token [MASK]. Le réseau de neurones à la sortie du modèle BERT est une projection du vecteur de sortie afin de lui associer un mot dans le vocabulaire de référence.

Remarque 21. *Soit D_{dico} la taille du dictionnaire et $k \in \Omega = [\![1, D_{dico}]\!]$. Pour t^k un token masqué à l'indice i_0 dans la séquence de tokens $S = (t_1, t_2, \cdots, t_n)$, nous pouvons lui associer un coût lié à l'entropie croisée entre la probabilité cible δ_k (un Dirac en k sur Ω) et la mesure de probabilité q retournée par le modèle BERT + MLP + Softmax. Pour S fixée, la mesure de probabilité q est une fonction des poids W du modèle. Ainsi, l'entropie croisée entre la mesure de probabilité δ_k et la mesure de probabilité q est une fonction régulière en les poids W du modèle, elle est donnée par l'expression suivante :*

$$W \longmapsto H(\delta_k, q(W))$$

où :

$$\begin{aligned} H(\delta_k, q(W)) &= -\sum_{i=1}^{D_{dico}} \delta_k(i) \times \ln(q(W)(i)) \\ &= -\ln(q(W)(k)) = 0 - \ln(q(W)(k)) \\ &= H(\delta_k) + D_{KL}\left(\delta_k \| q(W)\right) \end{aligned}$$

La régularité (différentiabilité et convexité) de la fonction de perte permet d'utiliser des méthodes d'optimisation non linéaires, telles que la méthode de descente de gradient, afin de trouver les poids W^* minimisant l'entropie croisée. Minimiser la fonction $W \mapsto H(\delta_k, q(W))$ revient donc à trouver les poids minimisant la divergence de Kullback-Leibler entre les deux mesures de probabilité δ_k et $q(W)$.

En pratique, l'optimisation des poids se fait de manière itérative, sur plusieurs tokens masqués selon un certain "batch size" de séquences.

Tâche 2 : Cohérence sémantique entre deux phrases (NSP : next sentence prediction)

La partie NSP consiste à fournir deux phrases A et B au modèle BERT, telles que la phrase B ne soit pas toujours la suite logique de la phrase A. 50 % du temps, la phrase B est la suite logique de la phrase A et dans les 50 % restants, la phrase B est une phrase choisie aléatoirement dans le corpus de référence. Le vecteur de sortie C issu du token [CLS] est ensuite passé dans un réseau dense pour effectuer une tâche de classification binaire (1 = IsNext ou 0 = NotNext).

Remarque 22. *Cette étape simple, est décrite comme bénéfique dans l'article initial de BERT. Elle sera cependant remise en question avec l'arrivée du modèle RoBERTa dans l'article suivant p.5 : "RoBERTa : A Robustly Optimized BERT Pretraining Approach" [12]*

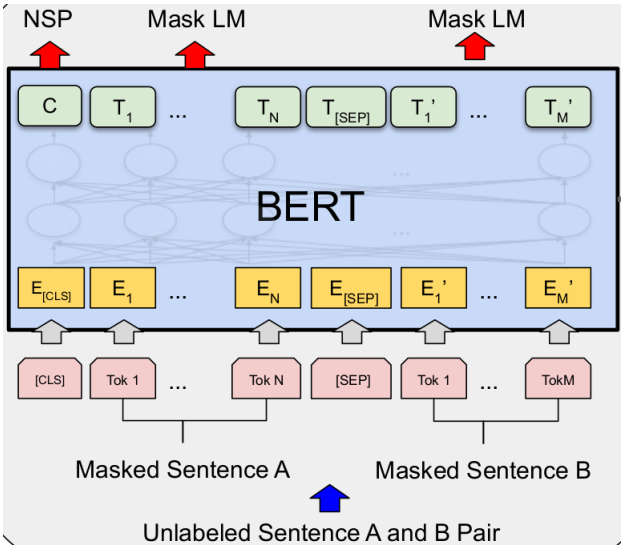


FIGURE 15 – Représentation graphique du pré-entraînement du modèle BERT issue de l'article [2]

4.2.4 Fine-tuning du modèle BERT pour des tâches de classification

Dans l'article de référence du modèle BERT, les phases de fine-tuning sont décrites comme rapides et relativement simples. Cette simplicité est largement due à l'architecture de BERT, qui permet d'adapter facilement le modèle à diverses tâches de traitement du langage naturel (NLP) en ajoutant seulement quelques couches supplémentaires.

Un exemple de fine-tuning présenté dans l'article est le fine-tuning du modèle sur les tâches associées au benchmark GLUE. Le benchmark GLUE (General Language Understanding Evaluation) est un recueil de neuf tâches de classification et de régression qui évaluent la compréhension du langage naturel par les modèles.

Quelques exemples de tâches :

— CoLA (Corpus of Linguistic Acceptability) : Prédire si une phrase est grammaticalement correcte.
— SST-2 (Stanford Sentiment Treebank) : Analyser la polarité du sentiment (positif ou négatif) d'une phrase.
— MRPC (Microsoft Research Paraphrase Corpus) : Déterminer si deux phrases sont des paraphrases.
— STS-B (Semantic Textual Similarity Benchmark) : Évaluer la similarité sémantique entre deux phrases sur une échelle continue.
— QQP (Quora Question Pairs) : Déterminer si deux questions posées sur Quora sont des doublons.

Pour le fine-tuning du modèle BERT sur chacune de ces tâches, nous utilisons uniquement le résultat du token [CLS] à la sortie du modèle BERT, soit le premier token C à la sortie du modèle. Ce token est conçu pour capturer l'information globale de la séquence d'entrée.

Le token C est ensuite passé dans un réseau de neurones dense (MLP) afin de prédire des probabilités via une fonction softmax ou sigmoïde, en fonction du caractère binaire ou multiclasse de la tâche. Cette pratique permet de réentraîner tous les poids du modèle, tout en bénéficiant de l'initialisation des poids issue du modèle pré-entraîné sur un large corpus de texte. Cela permet d'adapter rapidement le modèle à une nouvelle tâche spécifique avec un petit nombre d'époques d'entraînement (2 à 3 en général).

Remarque 23. *Les performances des modèles BERT ajustés par fine-tuning se sont avérées remarquables, notamment pour les tâches de classification. BERT atteint des scores élevés sur presque toutes les tâches du benchmark GLUE. En revanche, bien que performants pour la classification, les modèles fine-tunés de BERT ne sont pas optimaux pour la génération de contenu, contrairement à des modèles avec une architecture de transformeurs de type décodeur, tels que GPT ou Llama, mieux adaptés pour cette fonction.*

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

FIGURE 16 – Résultats du modèle Bert sur les tâche GLUE [2]

5 Un moteur de recherche avec un modèle Transformers

Nous avons montré dans une première partie comment nous pouvions comparer la similarité entre deux documents grâce à une approche fréquentielle (TF-IDF). Dans cette partie, nous allons utiliser un modèle spécialement conçu pour créer des embeddings de phrases grâce à la technologie des réseaux de neurones Transformers.

5.1 SBERT : un fine-tuning du modèle BERT pour obtenir des embeddings de phrases

Le modèle SBERT est une version fine-tunée du modèle BERT, employant la méthode des réseaux de neurones siamois. Dans cette section, nous reprenons les résultats exposés dans l'article de référence du modèle SBERT : "Sentence Embeddings using Siamese BERT-Networks" [13]. Les scores obtenus par ce modèle sur certains datasets benchmarks comme STS [15], dépassent les résultats des modèles précédents comme le modèle Universal Sentence Encoder. Depuis l'arrivée des réseaux de neurones Transformers, les tâches de NLP qui consistent à comparer des phrases n'utilisent plus des approches fréquentielles comme la méthode TF-IDF.

5.1.1 BERT et les embeddings de phrase

Nous pouvons commencer cette section en nous posant la question suivante : *comment obtenir une vectorisation d'une séquence de Tokens via les sorties vectorielles du modèle BERT ?*

Rappelons que BERT prend $n = 512$ vecteurs en entrée et renvoie $n = 512$ vecteurs en sortie. Une première idée, qui fonctionne plutôt mal en pratique, serait de prendre le vecteur barycentrique (MEAN) issu des n sorties du modèle BERT. Une autre approche, encore moins performante dans les faits, serait d'utiliser le token C à la sortie du modèle BERT pour en déduire un vecteur de référence pour chaque phrase. Malheureusement, ces approches donnent des scores d'évaluation sur les datasets issus de STS [15] inférieurs en comparaison avec des modèles spécialisés (ex : Universal Sentence Encoder). Le tableau suivant montre que les sorties d'un modèle BERT ne permettent pas directement d'obtenir une représentation pertinente d'une phrase par un simple "pooling" des sorties, le fine tuning est donc nécessaire.

Model	STS12	STS13	STS14	STS15	STS16	STSb	SICK-R	Avg.
Avg. GloVe embeddings	55.14	70.66	59.73	68.25	63.66	58.02	53.76	61.32
Avg. BERT embeddings	38.78	57.98	57.98	63.15	61.06	46.35	58.40	54.81
BERT CLS-vector	20.16	30.01	20.09	36.88	38.08	16.50	42.63	29.19
InferSent - Glove	52.86	66.75	62.15	72.77	66.87	68.03	65.65	65.01
Universal Sentence Encoder	64.49	67.80	64.61	76.83	73.18	74.92	**76.69**	71.22
SBERT-NLI-base	70.97	76.53	73.19	79.09	74.30	77.03	72.91	74.89
SBERT-NLI-large	72.27	**78.46**	**74.90**	80.99	76.25	**79.23**	73.75	76.55
SRoBERTa-NLI-base	71.54	72.49	70.80	78.74	73.69	77.77	74.46	74.21
SRoBERTa-NLI-large	**74.53**	77.00	73.18	**81.85**	**76.82**	79.10	74.29	**76.68**

FIGURE 17 – Tableau des évaluations sur les datasets de STS [15]

5.1.2 SBERT un modèle finetuné de BERT via la méthode des réseaux siamois

L'idée du modèle SBERT est de fine-tuner simultanément deux modèles BERT pré-entraînés, à l'aide de la méthode des réseaux siamois. Cela revient à constituer une architecture en parallèle avec deux modèles BERT et d'entraîner symétriquement les poids des deux modèles sur une tâche de classification consistant à prendre des couples de phrases et à renvoyer des labels de similarité. Au lieu de traiter deux phrases dans un seul modèle, comme peut le faire le modèle BERT avec le token `[SEP]`, le premier modèle BERT (A) traitera uniquement la première phrase (A), et le deuxième modèle BERT (B) traitera uniquement la deuxième phrase (B). À la sortie des deux modèles, une phase de "pooling" sera ajoutée pour transformer les sorties séquentielles en un unique vecteur. Cette phase de "pooling" sera potentiellement la moyenne des vecteurs, le token C (premier vecteur de sortie) ou le max des coefficients sur chacun des vecteurs de sortie. Notons u et v les deux vecteurs de sortie des deux blocs siamois, une étape de concaténation [1] sera rajouté avant de passer ce nouveaux vecteur dans un réseaux dense suivi d'une couche Softmax. Le modèle siamois sera entraîné pour une tâche de classification sur le dataset SNLI [15] avec la fonction de perte d'entropie croisée. La rétropropagation du gradient fera évoluer les poids de manière symétrique au fil de l'entraînement du modèle siamois. Une fois l'entraînement terminé, il suffit de garder l'un des deux modèles (BERT (A) par exemple) et le bloc de "pooling" pour obtenir une représentation vectorielle d'une phrase.

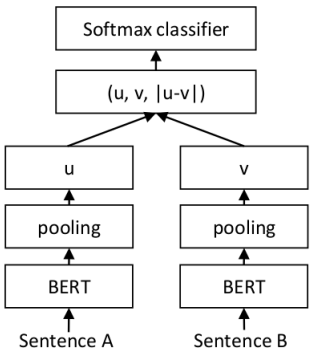


FIGURE 18 – Architecture du modèle siamois durant l'entraînement du modèle SBERT pour des tâches de classifications [13]

1. La concaténation peut être variable, la plus performante utilisée sera celle des vecteurs u , v et $|u - v|$

Remarque 24. *Plusieurs phases d'entraînement du modèle SBERT sont présentées dans l'article de référence [13]. L'entraînement des poids du modèle se fait aussi sur des tâches de régression en utilisant la similarité cosinus entre les vecteurs u et v en sortie. Nous supposons qu'une transformation affine du segment* $[-1, 1]$ *pourrait être effectuée pour fournir des valeurs plus étendues (exemple : un segment inclus dans* $\mathbb{R}$*). Pour une phase d'entraînement à l'aide du dataset STSb, chaque paire de phrases se voit attribuer un score de similarité compris entre 0 et 5.*

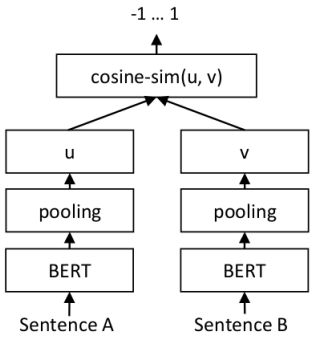


FIGURE 19 – Architecture du modèle siamois durant l'entraînement du modèle SBERT pour des tâches de régressions [13]

Model	MR	CR	SUBJ	MPQA	SST	TREC	MRPC	Avg.
Avg. GloVe embeddings	77.25	78.30	91.17	87.85	80.18	83.0	72.87	81.52
Avg. fast-text embeddings	77.96	79.23	91.68	87.81	82.15	83.6	74.49	82.42
Avg. BERT embeddings	78.66	86.25	94.37	88.66	84.40	92.8	69.45	84.94
BERT CLS-vector	78.68	84.85	94.21	88.23	84.13	91.4	71.13	84.66
InferSent - GloVe	81.57	86.54	92.50	**90.38**	84.18	88.2	75.77	85.59
Universal Sentence Encoder	80.09	85.19	93.98	86.70	86.38	**93.2**	70.14	85.10
SBERT-NLI-base	83.64	89.43	94.39	89.86	88.96	89.6	**76.00**	87.41
SBERT-NLI-large	**84.88**	**90.07**	**94.52**	90.33	**90.66**	87.4	75.94	**87.69**

FIGURE 20 – Tableau d'évaluations sur différents datasets

5.1.3 Une évaluation difficile des performance

Il est important de noter que l'évaluation générale de la qualité des embeddings de phrases issus d'un modèle reste une tâche complexe. Une évaluation possible se réalise grâce à des datasets benchmarks, de manière supervisée sur des tâches de classification ou de régression pour des paire de phrases. Le modèle peut ainsi comparer deux phrases via une tâche de classification (par exemple, évaluer si les phrases sont en contradiction, en implication ou neutres), ou encore produire un score de similarité pour une tâche de régression supervisée. Dans le cadre de notre projet, pour créer un moteur de recherche sur les tickets Assist, nous utiliserons la similarité cosinus afin d'évaluer la proximité de deux phrases vectorisées. Le moteur de recherche que nous développerons pour les questions des clients sera basé sur les embeddings fournis par le modèle SBERT. Nous aurons besoin de l'expertise d'une personne spécialisée sur le support technique pour pouvoir évaluer la qualité des résultats renvoyés par le moteur de recherche.

Remarque 25. *Nous pourrions, avec les collaborateurs métier, constituer un dataset contenant des paires de questions clients, en attribuant une note de similarité allant de 0 à 5. Ces données annotées nous permettraient de mettre en place une méthode d'évaluation basée sur une tâche de classification utilisant des métriques telles que l'accuracy, le F1-score ou encore l'aire sous la courbe ROC. Il serait alors possible d'évaluer de manière rigoureuse la qualité des embeddings de questions générées par le modèle SBERT pour notre secteur d'activité. Ce dataset nous permettrait non seulement de quantifier la performance du modèle en termes de précision des similarités calculées, mais aussi d'affiner les paramètres du modèle pour améliorer ses prédictions en conditions réelles via un possible fine-tuning.*

5.2 Une application suivant les besoins métiers

Le manager du service technique du CLPR nous a contacté pour initier la construction d'un moteur de recherche spécifique pour ses activités. Actuellement, toutes les équipes du plateau partagent le même outil de ticketing, ce qui rend les recherches spécialisées pour le service technique longues en raison de la volumétrie de tous les tickets. Il a remarqué que les besoins des clients sont souvent les mêmes, bien que la formulation de ceux-ci change pour chaque ticket et que les mots employés dépendent de l'interlocuteur. L'idée d'un moteur de recherche capable de comparer la similarité textuelle d'un nouveau ticket avec la base de tous les anciens tickets lui semblait une bonne piste de travail.

Nous avons ainsi développé une application web avec une interface utilisateur simple et utilisant un modèle Transformers de NLP en backend. Nous avons développé la totalité de notre application en Python, ce langage permet le traitement des données, il possède aussi de nombreuses bibliothèques pour le machine learning (PyTorch, Huggingface, etc.). Nous avons également utilisé Dash, un framework open-source en Python, pour construire des applications web interactives et analytiques. Dash est particulièrement populaire pour créer des tableaux de bord interactifs et des visualisations de données.

5.2.1 Fonctionnement du moteur de recherche

L'interface du moteur de recherche que nous avons développée est relativement simple. Une table comportant tous les tickets est présentée au centre de l'application. La table centrale contient par colonne des zones pour filtrer sur chacune des lignes. Une zone de texte au-dessus de la table principale permet d'écrire une nouvelle question client et de l'envoyer via le bouton "Load" au modèle SBERT pour être vectorisée. Une recherche sur les 30 000 plus proches voisins est alors réalisée en quelques secondes grâce l'index Faiss, les résultats de recherche sont ensuite ordonnés avec un score de similarité cosinus. Un bouton "Reset Filter" permet de réinitialiser les filtres et un bouton "Reset" permet de retrouver l'ensemble des tickets.

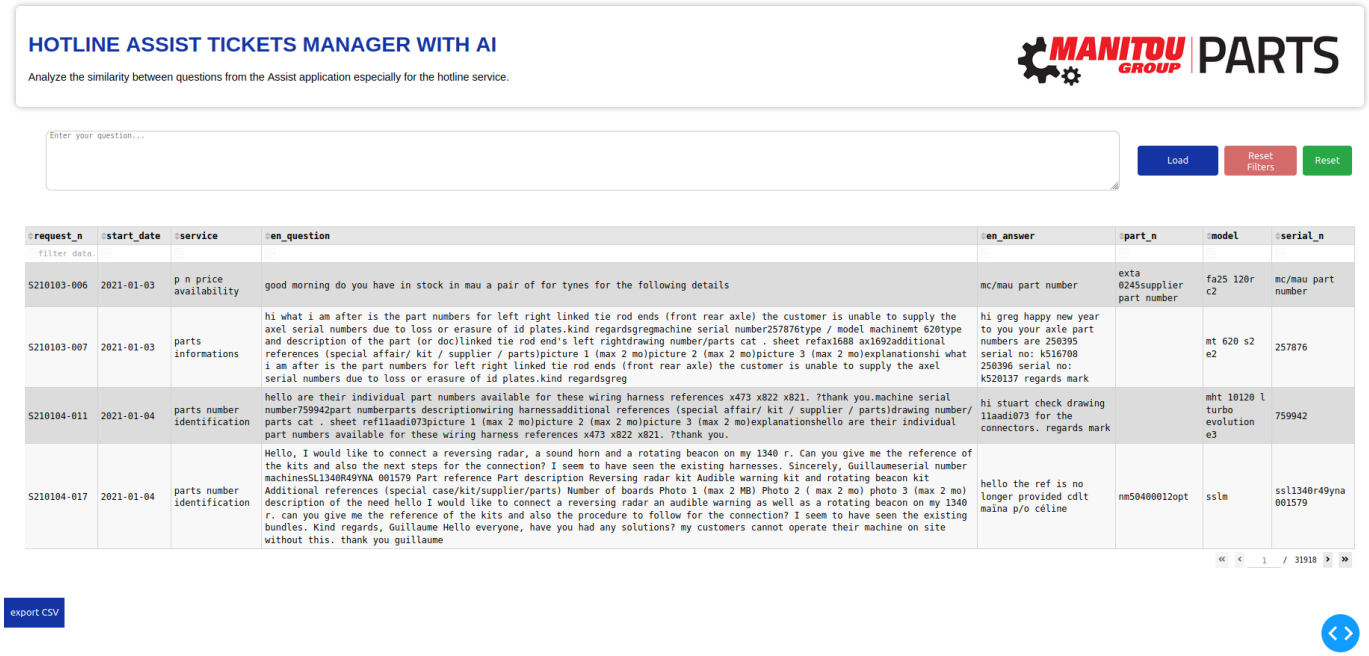


FIGURE 21 – Vue principale du Dashboard sans utiliser le moteur de recherche

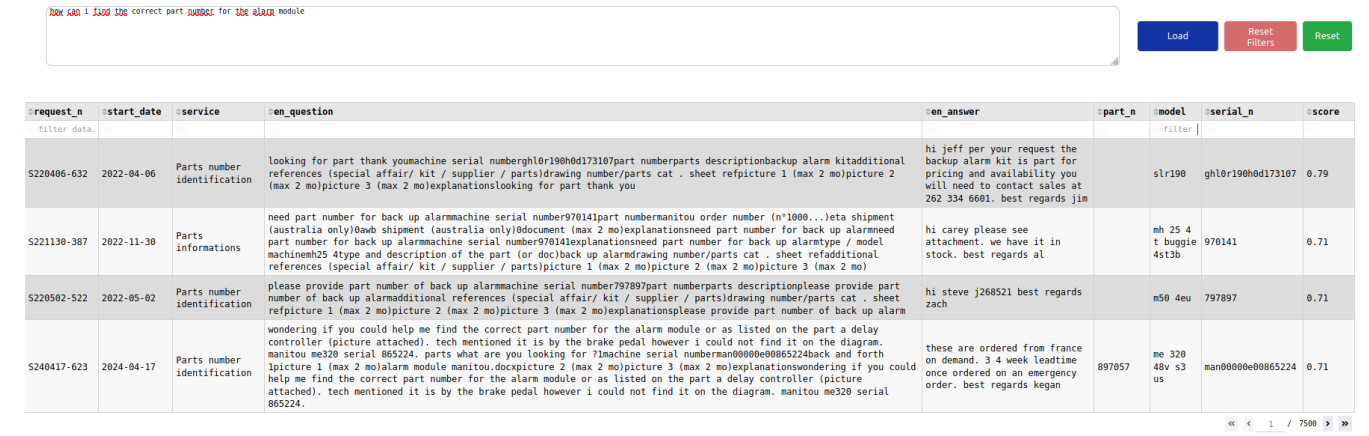


FIGURE 22 – Résultat du moteur de recherche pour une demande client sur un kit alarme

5.2.2 Construction de l'application

Nous présentons dans cette partie les étapes principales du développement de l'application précédemment mentionnée. Initialement, nous avons reçu des données brutes (questions/réponses) des clients issues de différentes langues au format Excel. Ces tableaux de données, allant de janvier 2021 à mars 2024, proviennent d'extractions du logiciel de ticketing Assist. La première partie de notre travail consista à homogénéiser les données textuelles par des méthodes REGEX. Une traduction en anglais de tous les tickets a également été proposée pour faciliter l'emploi international de l'outil. Pour ce faire, nous avons utilisé l'API Google "Deep Translator".

Remarque 26. *Les versions originales sont toujours disponibles dans la table finale même si elles ne sont pas affichées à l'écran via l'application. Nous avons fait des tests unitaires sur les traductions et la qualité nous semble satisfaisante. Cependant, des vérifications par des équipes spécialisées seraient nécessaires.*

Google Colab est l'outil que nous avons utilisé pour charger le modèle SBERT via Huggingface. Après avoir mis nos données sur le drive, nous avons pu vectoriser toutes les questions afin d'obtenir des embeddings. Nous avons chargé l'ensemble de ces vecteurs dans une instance FAISS. La bibliothèque FAISS est idéale pour manipuler des données vectorielles conséquentes (plus de 120 000 questions dans notre cas). Cette bibliothèque propose une instance "index" qui permet de stocker et d'effectuer des recherches des plus proches voisins rapidement pour des métriques telles que la distance euclidienne, la distance de Manhattan ou encore celle que nous avons utilisée, la similarité cosinus. La vectorisation de toutes les questions clients a pris environ 10 minutes sur un processeur GPU NVIDIA A100. Nous avons ensuite téléchargé l'index pour le connecter à l'application développée sous Dash. Les volumes de données n'étant pas trop conséquents, nous avons pu utiliser la bibliothèque Pandas pour manipuler nos tableaux de valeurs. Pour chaque nouvelle question client, nous utilisons le modèle SBERT chargé au lancement de l'application pour la vectoriser. Une opération de recherche des 30 000 plus proches voisins est alors exécutée en quelques secondes, comme nous avons pu le voir précédemment grâce à l'index de la bibliothèque FAISS.

5.2.3 Les pratiques MLOps

Les pratiques MLOps (Machine Learning Operations) visent à standardiser et automatiser le cycle de vie du développement des modèles de machine learning, de leur déploiement à leur maintenance. Pour notre application de moteur de recherche basé sur SBERT, nous pourrions intégrer ces pratiques en mettant en place une pipeline MLOps complète. Cela inclurait l'automatisation du prétraitement des données et de la gestion des versions des modèles. En intégrant les pratiques MLOps, nous améliorerions l'efficacité du développement, nous assurerions aussi la reproductibilité des résultats et nous faciliterions la collaboration entre les équipes de data science.

Améliorations : Afin d'améliorer le moteur de recherche, nous envisageons plusieurs pistes :

- **Affinage du modèle** : En fonction des retours utilisateurs, nous pourrions affiner le modèle SBERT via des séances de fine-tuning supplémentaires sur des datasets plus spécifiques à notre domaine d'activité. Cela permettrait d'améliorer la précision des embeddings et d'affiner nos recherches sur les questions des clients.
- **Optimisation de l'interface utilisateur** : En fonction des retours utilisateurs, une interface plus moderne et réactive améliorerait leur expérience de l'application.

Déploiement : Le déploiement de notre application nécessite plusieurs étapes clés pour garantir son bon fonctionnement et son accessibilité :

- **Plusieurs environnements** : Préparer des environnements de développement, de test et de production en utilisant des outils tels que Docker pour conteneuriser notre application et faciliter son déploiement sur différents serveurs.
- **CI/CD** : Mettre en place une pipeline d'intégration continue et de déploiement continu (CI/CD) pour automatiser les ajouts des nouvelles questions questions client, l'automatisation des tests et le déploiement. Cela permettrait de déployer rapidement et en toute sécurité les nouvelles versions de l'application.
- **Hébergement** : Choisir une solution d'hébergement fiable et évolutive (exemple : AWS, Google Cloud ou Azure) pour héberger l'application et garantir sa disponibilité et sa performance.
- **Sécurité** : Assurer la sécurité de notre application en implémentant des protocoles de sécurité tels que HTTPS, en gérant les accès et en surveillant les vulnérabilités.

Maintenance : Pour garantir la pérennité et la performance de l'application, un plan de maintenance régulier est essentiel :

- **Mises à jour** : Effectuer régulièrement des mises à jour du modèle et des bibliothèques utilisées pour bénéficier des dernières améliorations. S'assurer que les nouvelles mis à jour des bibliothèques n'altèrent pas le fonctionnement de l'application.
- **Augmentation des données** : Enrichir le dataset en y intégrant de nouvelles questions et réponses régulièrement. Plus le dataset sera riche et varié, plus le moteur de recherche pourra proposer des exemples de questions passées.

En suivant ces étapes d'améliorations, de déploiement et de maintenance, nous nous assurons que le moteur de recherche basé sur SBERT reste performant, fiable et adapté aux besoins de notre secteur d'activité.

6 Conclusion

Au vu des premiers retours métier, ce projet démontre l'efficacité d'un moteur de recherche basé sur les modèles Transformers, en particulier SBERT. Cette technologie répond aux besoins spécifiques des équipes CLPR et s'inscrit dans une démarche innovante d'intégration de solutions d'intelligence artificielle dans l'industrie. En utilisant la vectorisation des phrases et des mesures de similarité, nous avons pu améliorer la recherche de tickets, rendant l'accès aux informations plus rapide et pertinent. Pour aller plus loin, nous envisageons plusieurs améliorations significatives :

Pré-réponses automatiques avec un modèle génératif : L'intégration d'un modèle génératif tel que GPT, LLaMA ou encore Mistral, pourrait permettre de fournir des pré-réponses automatiques aux questions des clients. En fine-tunant ce modèle sur le dataset spécifique des questions et réponses, nous pourrions améliorer la pertinence des réponses proposées. Ce modèle génératif pourrait être utilisé en complément du moteur de recherche, offrant ainsi une assistance proactive et réduisant le temps de réponse des équipes.

Utilisation des données sur les pièces machines : Une future amélioration consisterait à intégrer une base de données complète sur les pièces composant chaque machine. Cette base de données permettrait d'enrichir le moteur de recherche en plus des réponses aux questions des clients. L'accès rapide à ces données techniques pourra grandement faciliter le travail des équipes techniques. Un lien avec les niveaux de stock et les statuts des pièces est un idéal vers lequel nous aimerions tendre.

Amélioration de la normalisation des questions : Nous prévoyons de reprendre la normalisation des questions pour éliminer les répétitions et gérer les informations sur les pièces jointes. Une uniformisation des noms de machines est également nécessaire pour garantir une cohérence et une précision accrues dans les recherches via les filtres. La collaboration avec un expert métier est essentielle pour uniformiser les questions et les réponses, assurant ainsi une qualité optimale des données traitées.

Perspectives futures : En intégrant ces axes d'amélioration, le moteur de recherche évoluera vers une solution encore plus performante et adaptée aux besoins des utilisateurs. L'automatisation des réponses, l'enrichissement des données techniques et l'amélioration continue de la qualité des données textuelles contribueront à faire de cet outil un atout pour le CLPR. La collaboration avec des experts métier et l'utilisation de technologies de pointe en NLP permettront de maintenir cette application à la pointe de l'innovation, assurant ainsi une satisfaction maximale des utilisateurs. En conclusion, ce projet constitue une étape importante vers l'optimisation des opérations du CLPR. Les améliorations envisagées ouvriront de nouvelles perspectives pour une gestion toujours plus efficace des tickets et des interactions avec les clients.

Références

[1] Attention is all you need. https://arxiv.org/abs/1706.03762. Accédé le [10-05-2024].

[2] Bert : Pre-training of deep bidirectional transformers for language understanding. https://arxiv.org/pdf/1810.04805. Accédé le [12-05-2024].

[3] Cosine similarity. https://www.learndatasci.com/glossary/cosine-similarity/. Accédé le [08-05-2024].

[4] Distributed representations of words and phrases and their compositionality. https://arxiv.org/pdf/1310.4546. Accédé le [7-05-2024].

[5] Improving language understanding by generative pre-training. https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf. Accédé le [14-05-2024].

[6] An information-theoretic perspective of tf–idf measures. https://ccc.inaoep.mx/~villasen/index_archivos/cursoTL/articulos/Aizawa-tf-idfMeasures.pdf. Accédé le [08-05-2024].

[7] Intelligence artificielle : Résumer un texte grâce au tf idf. https://datascientest.com/tf-idf-intelligence-artificielle. Accédé le [08-02-2024].

[8] Japanese and korean voice search. https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/37842.pdf. Accédé le [14-03-2024].

[9] Neural machine translation of rare words with subword units. https://aclanthology.org/P16-1162.pdf. Accédé le [14-03-2024].

[10] Openai tokenizer. https://platform.openai.com/tokenizer. Accédé le [7-05-2024].

[11] Research-paper recommender systems : a literature survey. https://kops.uni-konstanz.de/entities/publication/861ddd16-fbc6-4ee4-a77f-6bba081041f3. Accédé le [08-02-2024].

[12] Roberta : A robustly optimized bert pretraining approach. https://arxiv.org/pdf/1907.11692. Accédé le [15-05-2024].

[13] Sentence-bert : Sentence embeddings using siamese bert-networks. https://arxiv.org/abs/1908.10084. Accédé le [17-05-2024].

[14] Seq. 08 / transformers. https://www.youtube.com/watch?v=L3DGgzIbKz4&t=3666s. Accédé le [12-05-2024].

[15] Sts benchmark (semantic textual similarity). https://paperswithcode.com/dataset/sts-benchmark. Accédé le [17-05-2024].

[16] Understanding and coding self-attention, multi-head attention, cross-attention, and causal-attention in llms. https://magazine.sebastianraschka.com/p/understanding-and-coding-self-attention. Accédé le [14-05-2024].

[17] Video (min 18) : What are transformer models and how do they work ? https://www.youtube.com/watch?v=qaWMOYf4ri8. Accédé le [14-03-2024].

[18] Word2vec. https://en.wikipedia.org/wiki/Word2vec. Accédé le [15-03-2024].

[19] Wordpiece tokenization : https://huggingface.co/learn/nlp-course/en/chapter6/6. Accédé le [15-03-2024].