

SEARCH ENGINE REPORT

Semantic search engine Based on Transformers architecture.

Mamouni Fatima Zahrae
Bouajaj Zayd
Izem Rachid
Chakir Mohamed Wahb

Table de matière:

1 – Introduction	2
2- Conception.....	3
2.1. NLP.....	3
2.3. Bert.....	8
2.4. Similitude des phrases.....	11
3- les outils	14
4- Entrainement du model :	15
1)-Checker la validité du training GPU	15
2) Importation des fichiers *.pkl pour le training	15
3)Le preprocessing:	16
4) Processed text Après le traitement :	16
5) Installation du sentence transformer (Hugging-Face)	17
6) Recherche sémantique des transformateurs de phrases BERT (encodeur)	17
7) L'Output sur Google Colab.	19

1 – Introduction

La recherche sémantique est une **technique de recherche de données** dans laquelle une requête de recherche vise non seulement à trouver des mots-clés, mais aussi à **déterminer l'intention et la signification contextuelle des mots** qu'une personne utilise pour une recherche.

La sémantique fait référence à l'étude philosophique du sens. Il est vrai que la philosophie rime rarement avec l'ingénierie logicielle, mais ce concept nous aide à parvenir à une définition. En effet, **la recherche sémantique est liée à la détermination de ce que votre utilisateur veut dire.**

La recherche sémantique vise à améliorer la précision de la recherche en comprenant le contenu de la requête de recherche. Contrairement aux moteurs de recherche traditionnels, qui ne trouvent que des documents basés sur des correspondances lexicales, la recherche sémantique peut également trouver des synonymes.

En fait, ce type de recherche rend la navigation plus complète en comprenant presque exactement ce que l'utilisateur essaie de demander, au lieu de simplement faire correspondre des mots-clés aux pages. *L'idée derrière la recherche sémantique est d'intégrer toutes les entrées de votre corpus, qui peuvent être des phrases, des paragraphes ou des documents, dans un espace vectoriel.*

Au moment de la recherche, *la requête est intégrée dans le même espace vectoriel et l'intégration la plus proche de votre corpus est trouvée*. Ces entrées devraient avoir un chevauchement sémantique élevé avec la requête.

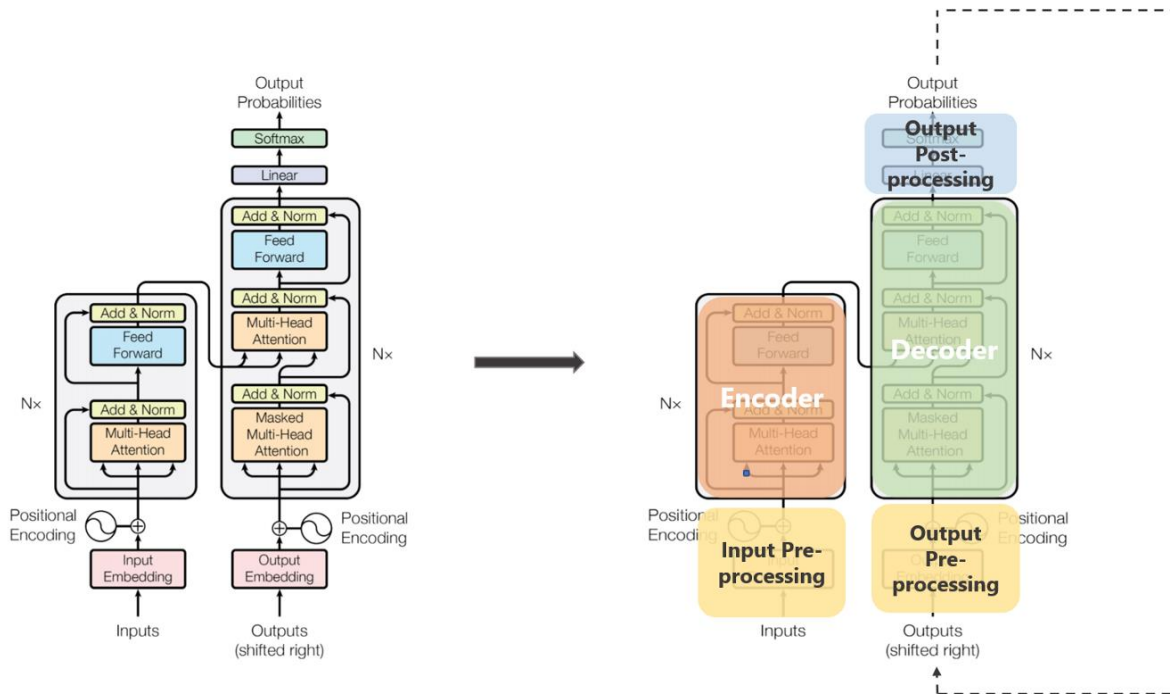
Le but principale de notre projet de développer une application intelligente de recherche et d'analyse de la littérature qui lit des articles scientifiques pertinents pour la NASA à l'aide du traitement du langage naturel et de l'exploration de texte, renvoie une analyse des connaissances et des découvertes scientifiques pertinentes, et révèle les lacunes dans les connaissances et les opportunités de recherche multidisciplinaire et déployer cette application dans une interface simple a utiliser .

2- Conception

2.1. NLP

Le traitement automatique de la langue naturelle ou traitement automatique des langues, plus couramment appelé NLP est un domaine multidisciplinaire impliquant la linguistique, l'informatique et l'intelligence artificielle, qui vise à créer des outils de traitement de la langue naturelle pour diverses applications.

2.2. Transformer



le transformateur est un modèle d'apprentissage profond proposé en 2017, qui est essentiellement utilisé dans les tâches de PNL. Si vous travaillez sur des tâches de traitement du langage naturel comme le résumé du texte, la traduction ou la prédiction des émotions, alors vous rencontrerez ce terme très souvent.

RNN a souffert du **problème du gradient de disparition** qui provoque une perte de mémoire à long terme. RNN fait le traitement du texte de manière séquentielle, ce qui signifie que s'il y a une longue phrase comme - "XYZ est allé en France en 2019 alors qu'il n'y avait pas de cas de covid et qu'il y a rencontré le président de ce pays". Maintenant, si nous demandons que ici, « ce pays » fait référence à quel endroit ? RNN ne pourra pas se rappeler que le pays était « France » parce qu'il a rencontré le mot « France » bien avant. La nature séquentielle du traitement signifie que le modèle a été formé au niveau des mots, et non au niveau de la phrase. Les gradients portent les informations utilisées dans la mise à jour des paramètres RNN et lorsque le gradient devient plus petit, aucun apprentissage réel n'est terminé. En ajoutant quelques cellules de mémoire supplémentaires et en résolvant le problème des gradients en voie de disparition, le problème de perte de mémoire à long terme a été résolu dans une certaine mesure. Mais le problème avec le traitement séquentiel est resté parce que RNN n'a pas

été en mesure de traiter la phrase intacte à la fois. Plutôt que de traiter en parallèle, il traite les mots un par un. Ce problème ne peut pas être résolu dans les LSTM en raison de leur conception séquentielle. Dans les LSTM, nous utilisons la méthode d'intégration statique, ce qui suggère que sans connaître le contexte d'un mot, nous l'intégrons dans un vecteur n-dimensionnel. Mais si le contexte change, le sens change aussi.

Ex : Il y a un mot - « **Point** », et nous l'utilisons dans deux contextes différents donnés ci-dessous

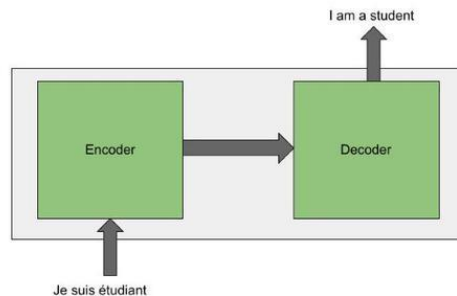
- L'aiguille a une pointe pointue.
- Il n'est pas poli de pointer du doigt les gens.

Ici, le mot « Point » a deux contextes différents dans les deux phrases, mais lorsque l'intégration est effectuée, le contexte n'est pas pris en considération. Par conséquent, il y avait un besoin d'une architecture différente - Transformer. Le transformateur a été proposé dans le document Attention is All You Need.

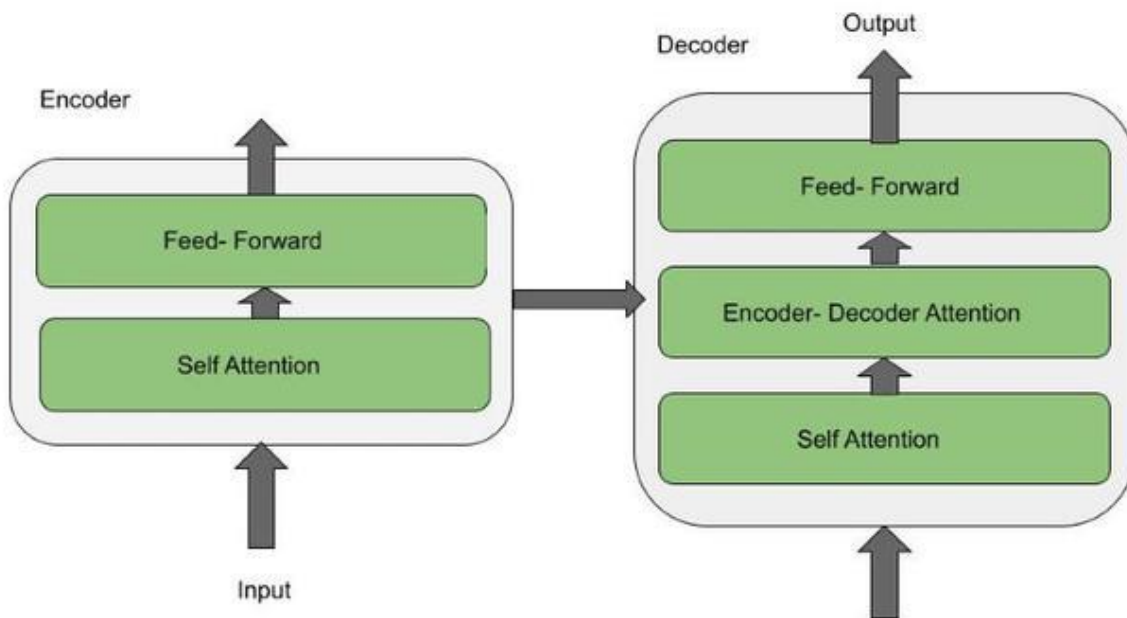
Les réseaux neuronaux ont été classés principalement en deux groupes, à savoir - Feedforward et Feedback. Les transformateurs sont basés sur des réseaux feedforward, ce qui signifie que l'information se déplace de l'entrée à la sortie et qu'elle ne contient pas de boucle de rétroaction. Contrairement à cela, les LSTM utilisent des réseaux de rétroaction, ce qui signifie que l'information peut passer dans les deux sens et qu'elle consiste en un chemin de rétroaction, c'est-à-dire que nous pouvons à nouveau utiliser la mémoire pour de nouvelles prédictions.

Maintenant, en venant à l'architecture du Transformer. L'encodeur et le décodeur sont des éléments constitutifs d'un transformateur. Le bloc encodeur transforme la séquence de mots d'entrée en vecteur et un décodeur convertit un vecteur en séquence. Par exemple : Un texte en français traité dans son équivalent anglais peut être :

Je suis étudiant → Je suis étudiant.



L'architecture de l'encodeur comporte deux couches : **Self Attention** et **Feed Forward**. Les entrées de l'encodeur passent d'abord par une couche d'auto-attention, puis les sorties de la couche d'auto-attention sont alimentées à un réseau neuronal de feed-forward. Les données séquentielles ont des caractéristiques temporelles. Cela signifie que chaque mot occupe une certaine position concernant l'autre. Par exemple, prenons une phrase - « Le chat n'a pas chassé la souris, parce qu'elle n'avait pas faim ». Ici, nous pouvons facilement dire que « c'est » fait référence au chat, mais ce n'est pas aussi simple pour un algorithme. Lorsque le modèle traite le mot « il », l'auto-attention lui permet d'associer « il » à « chat ». L'auto-attention est la méthode pour reformuler la représentation en fonction de tous les autres mots de la phrase.



L'architecture du décodeur comporte trois couches : Self Attention, Encoder-decoder attention et Feed Forward. Le décodeur a à la fois la couche d'auto-attention et de feed-forward qui est également présente dans l'encodeur, mais entre eux se trouve une couche d'attention qui aide le décodeur à se concentrer sur les parties pertinentes de la phrase d'entrée.

Il existe six couches d'encodeurs et de décodeurs dans l'architecture Transformer. Dans l'encodeur inférieur, les incorporations de mots sont effectuées et chaque mot est transformé en un vecteur de taille 512. L'entrée vers les autres encodeurs serait la sortie de l'encodeur qui se trouve directement en dessous. Les différentes couches de l'encodeur sont pour découvrir le pipeline de la PNL. Comme - la première couche est utilisée pour une partie des balises vocales, la deuxième couche pour les constituants, la troisième couche pour les dépendances, la quatrième couche pour les rôles

sémantiques, la cinquième pour la coréférence et la sixième pour les relations.

La toute dernière couche est la couche Softmax qui attribue une probabilité à chaque mot du vocabulaire et toute cette probabilité totalisent jusqu'à 1.

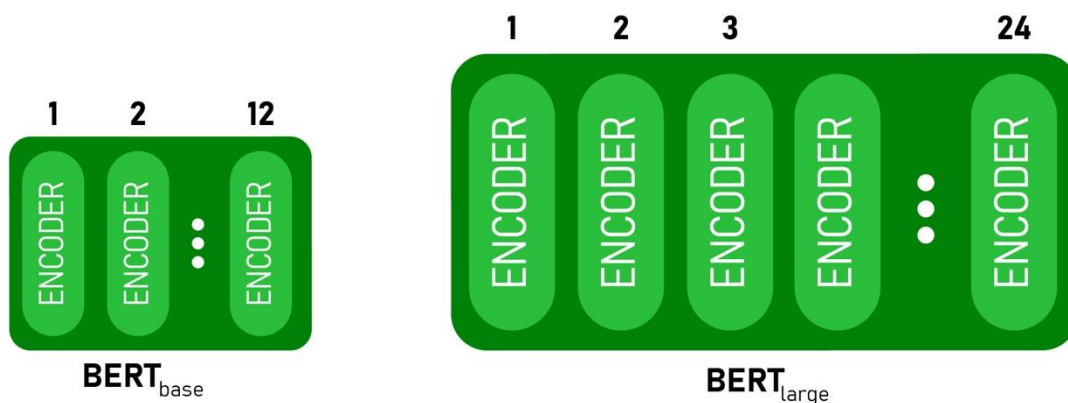
2.3. Bert

BERT (Bidirectional Encoder Representations from Transformers) est un modèle de traitement du langage naturel proposé par les chercheurs de Google Research en 2018. Lorsqu'il a été proposé, il atteignait une précision de pointe sur de nombreuses tâches de la PNL et de la NLU telles que :

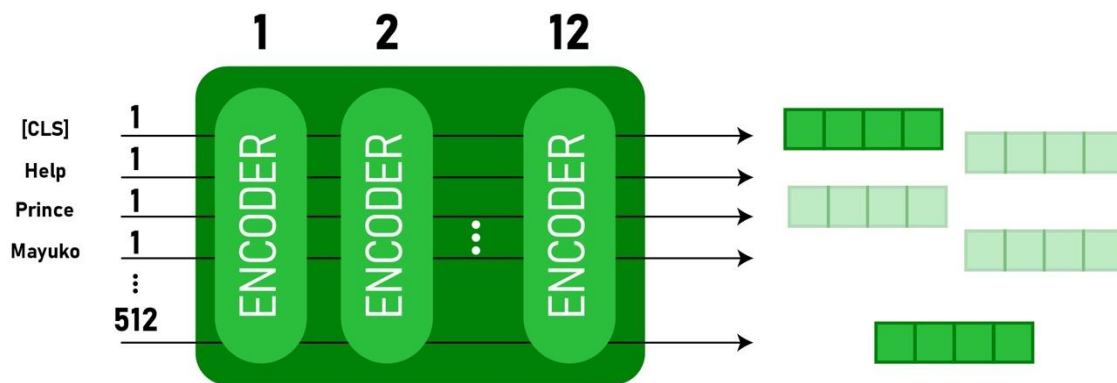
- Évaluation générale de la compréhension linguistique
- Stanford Q/A dataset SQuAD v1.1 et v2.0
- Situation Avec Les Générations Opposées

Peu après quelques jours de sortie, le code ouvert publié a publié deux versions du modèle pré-entraîné BERTBASE et BERTLARGE qui sont formés sur un ensemble de données massif. BERT utilise également de nombreux algorithmes et architectures de PNL antérieurs tels que la formation semi-supervisée, les transformateurs OpenAI, les intégrations ELMo, ULMFit, les transformateurs. Architecture du modèle BERT : BERT est disponible en deux tailles BERTBASE et BERTLARGE. Le modèle BASE est utilisé pour mesurer les performances de l'architecture comparable à une autre architecture et le modèle LARGE produit des résultats de pointe qui ont été rapportés dans le document de recherche. Apprentissage semi-supervisé : L'une des principales raisons de la bonne performance de BERT sur différentes tâches de PNL était l'utilisation de l'apprentissage semi-supervisé. Cela signifie que le modèle est formé pour une tâche spécifique qui lui permet de comprendre les modèles de la langue. Après la formation, le modèle (BERT) dispose de capacités de traitement du langage qui peuvent être utilisées pour responsabiliser d'autres modèles que nous construisons et formons à l'aide de l'apprentissage supervisé.

BERT est essentiellement une pile d'encodeurs d'architecture de transformer. Une architecture de transformer est un réseau encodeur-décodeur qui utilise l'auto-attention du côté de l'encodeur et l'attention du côté du décodeur. BERTBASE a 12 couches dans la pile d'encodeurs tandis que BERTLARGE a 24 couches dans la pile d'encodeurs. Il s'agit de plus que l'architecture Transformer décrite dans l'article original (6 couches d'encodeur). Les architectures BERT (BASE et LARGE) ont également des réseaux de flux plus grands (768 et 1024 unités cachées respectivement), et plus de têtes d'attention (12 et 16 respectivement) que l'architecture Transformer suggérée dans l'article original. Il contient 512 unités cachées et 8 têtes d'attention. BERTBASE contient 110 millions de paramètres tandis que BERTLARGE a 340 millions de paramètres.

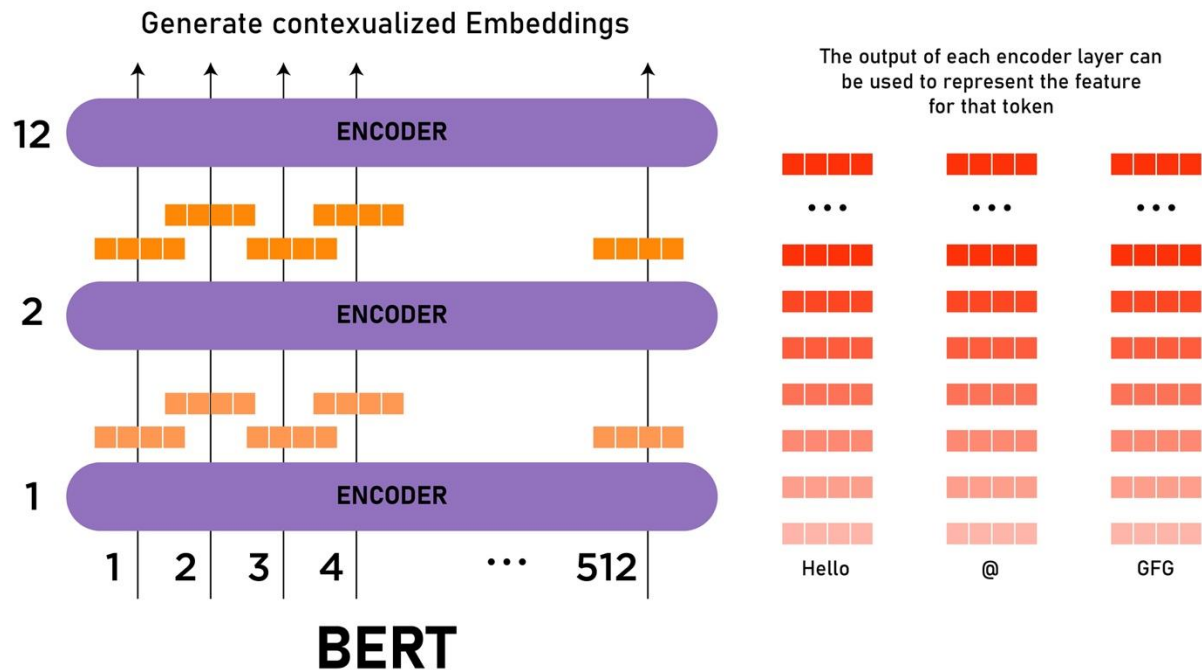


Ce modèle prend d'abord le jeton **CLS** comme entrée, puis il est suivi d'une séquence de mots comme entrée. Ici, CLS est un jeton de classification. Il transmet ensuite l'entrée aux couches ci-dessus. Chaque couche applique l'auto-attention, passe le résultat à travers un réseau de feedforward, puis il passe à l'encodeur suivant. Le modèle produit un vecteur de taille cachée (768 pour BERT BASE). Si nous voulons produire un classificateur à partir de ce modèle, nous pouvons prendre la sortie correspondant au jeton CLS.



Maintenant, ce vecteur entraîné peut être utilisé pour effectuer un certain nombre de tâches telles que la classification, la traduction, etc. Par exemple, le document obtient d'excellents résultats simplement en utilisant un NN à une seule couche sur le modèle BERT dans la tâche de classification.

Fondamentalement, Word Embeddings pour un mot est la projection d'un mot vers un vecteur de valeurs numériques en fonction de sa signification. Il existe de nombreux mots populaires Embedding tels que Word2vec, GloVe, etc. ELMo était différent de ces incorporations parce qu'il donne l'intégration à un mot en fonction de son contexte, c'est-à-dire des incorporations de mots contextualisées. Pour générer l'intégration d'un mot, ELMo examine la phrase entière au lieu d'une incorporation fixe pour un mot. Elmo utilise un LSTM bidirectionnel formé pour la tâche spécifique afin de pouvoir créer ces incorporations. Ce modèle est formé sur un ensemble de données massif dans le langage de notre ensemble de données, puis nous pouvons l'utiliser comme composant dans d'autres architectures qui sont nécessaires pour effectuer des tâches linguistiques spécifiques.

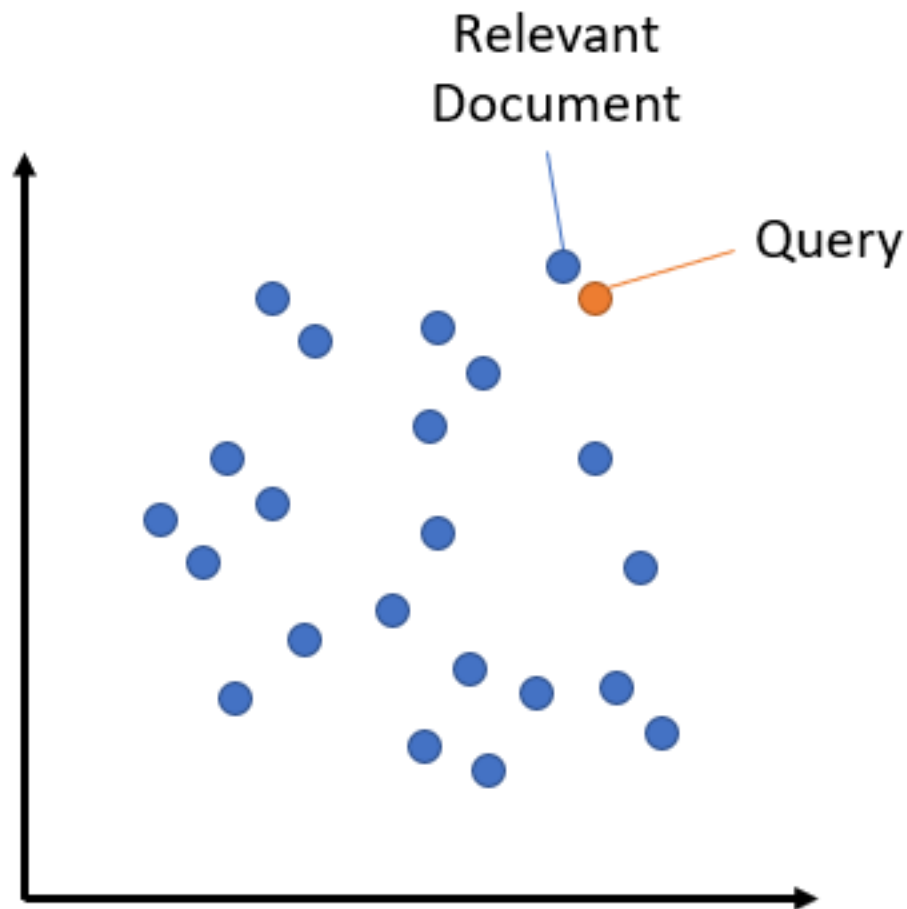


2.4. Similitude des phrases

La similitude des phrases est la tâche de déterminer à quel point deux textes sont similaires. Les modèles de similitude des phrases convertissent les textes d'entrée en vecteurs (embeddings) qui capturent les informations sémantiques et calculent à quel point elles sont proches (similaires) entre eux. Cette tâche est particulièrement utile pour la récupération et le regroupement/le regroupement d'informations. Alors cette tâche est la principale étape dans notre de recherche afin d'effectuer les recherches. Vous pouvez extraire des informations à partir de documents à l'aide de modèles de similarité de phrase. La première étape consiste à classer les documents à l'aide de modèles de classement des passages. Vous pouvez ensuite vous rendre dans le document le mieux classé et le rechercher avec les modèles de similitude des phrases en sélectionnant la phrase qui a le plus de similitude avec la requête d'entrée.

La bibliothèque Sentence Transformers est très puissante pour calculer les incorporations de phrases, de paragraphes et de documents entiers. Une incorporation n'est qu'une représentation vectorielle d'un texte et est utile pour trouver à quel point deux textes sont similaires.

Pour obtenir cette similitude de deux vecteurs de phrase, nous utilisons la similitude du cosinus ($1 - \text{distance du cosinus}$). C'est parce que la direction des vecteurs joue un rôle énorme dans leur signification sémantique. Par exemple, les vecteurs du « café » et du « thé » peuvent pointer dans une direction qui correspond à peu près aux « boissons du matin ». Maintenant, disons que nous avons le vecteur $a=[1,1,-1]$ et le $b=2a=[2,2,-2]$. Ces deux vecteurs ont des magnitudes différentes mais la même direction. Leur distance cosinus est de 0, mais leur distance euclidienne, qui ne tient pas compte de la direction, est de 4,58 et il serait très facile de trouver un autre vecteur avec moins de distance, dans une direction complètement différente.



SentenceTransformers est un cadre Python pour les incorporations de phrases, de texte et d'images de pointe. Le travail initial est décrit dans une article scientifique [Sentence-BERT : Sentence Embeddings using Siamese BERT-Networks](#) , publiée en 2019 .

Nous avons utilisé ce cadre pour calculer les incorporations de phrases , Cela est utile dans notre recherche textuelle sémantique similaire, la recherche sémantique ou l'exploitation minière par paraphrase.

Le cadre est basé sur PyTorch (bibliothèque de Python) et Transformers et offre une grande collection de modèles pré-entraînés accordés pour diverses tâches. De plus, il est facile de peaufiner vos propres modèles.

3- les outils

Streamlit	Streamlit est un framework open-source Python spécialement conçu pour les ingénieurs en machine learning et les Data scientists. Ce framework permet de créer des applications web qui pourront intégrer aisément des modèles de machine learning et des outils de visualisation de données.
Python	Python est un langage de programmation interprété, multiparadigme et multiplateformes. Il favorise la programmation impérative structurée, fonctionnelle et orientée objet
Pickle	Pickle est un outil Python utile qui vous permet de sauvegarder vos modèles ML, de minimiser les longs réentraînements et de partager, valider et recharger des modèles d'apprentissage automatique pré-formés. La plupart des data scientists travaillant en ML utiliseront Pickle ou Joblib pour enregistrer leur modèle ML pour une utilisation future.
Pytorch	PyTorch est une bibliothèque logicielle Python open source d'apprentissage machine qui s'appuie sur Torch développée par Meta. PyTorch est gouverné par la PyTorch Foundation. PyTorch permet d'effectuer les calculs tensoriels nécessaires notamment pour l'apprentissage profond.

4- Entraînement du model :

1)-Checker la validité du training GPU

```
# GPU Setup

import torch

# If there's a GPU available...
if torch.cuda.is_available():

    # Tell PyTorch to use the GPU.
    device = torch.device("cuda")

    print('There are %d GPU(s) available.' % torch.cuda.device_count())

    print('We will use the GPU:', torch.cuda.get_device_name(0))

# If not...
else:
    print('No GPU available, using the CPU instead.')
    device = torch.device("cpu")
```

There are 1 GPU(s) available.
We will use the GPU: Tesla T4

2) Importation des fichiers *.pkl pour le training

```
#Creating PyDrive instance to load in data from PeTaL shared drive, follow the steps to authenticate
!pip install -U -q PyDrive

from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
import pandas as pd

# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```


3)Le preprocessing:

```
import pandas as pd

import unicodedata

#content/arc-code-ti-publications.pkl
# Import this into the Colab via the Files section
#with open(data, 'rb') as f:
#    pubs = pd.read_pickle(f)

pubs = data
import re
TAG_RE = re.compile(r'<[>]+>')

def remove_tags(text):
    return TAG_RE.sub('', text)

def preprocess_text(sen):

    sentence = str(sen)

    # Removing html tags
    sentence = remove_tags(sentence)

    # Remove hyphenation if at the end of a line
    sentence = sentence.replace('-\n', '')

    # Fix ligatures
    sentence = unicodedata.normalize("NFKD", sentence)

    # Remove punctuations and numbers
    sentence = re.sub('[^a-zA-Z]', ' ', sentence)

    # Single character removal
    sentence = re.sub(r"\s+[a-zA-Z]\s+", ' ', sentence)

    # Removing multiple spaces
    sentence = re.sub(r'\s+', ' ', sentence)

    return sentence

# Not really needed any more but will leave in and just comment out

# full_texts = []
# sentences = list(pubs['Text'])
# for sen in sentences:
#     full_texts.append(preprocess_text(str(sen)))

pubs.drop(pubs[pubs['Text'] == 'PDF error occurred'].index, inplace = True)
pubs.drop_duplicates(subset=['Text'])

pubs['Text Processed'] = pubs.apply(lambda row: preprocess_text(row['Text']), axis=1)
pubs['Word Count'] = pubs.apply(lambda row: len(row['Text Processed'].split()), axis=1)

text_df = pubs[['Text Processed',]].copy()
print(text_df)
```

4) Processed text Après le traitement :

	Text Processed
Index	
0	Adaptive Stress Testing of Trajectory Predicti...
1	Capturing Analyzing Requirements with FRET Dim...
3	The Ten Lockheed Martin Cyber Physical Challen...
4	Generation of Formal Requirements from Structu...
5	Formal Requirements Elicitation with FRET Dimi...
...	...
669	A Flexible Evolvable Architecture for Constell...
670	Extended Abstract General Purpose Data Driven ...
671	PARAMETRIC ANALYSIS OF HOVER TEST VEHICLE USI...
672	Bringing Web to Government Research Case Stud...
673	Online Detection and Modeling of Safety Bounda...
[666 rows x 1 columns]	

5) Installation du sentence transformer (Hugging-Face)

```
!pip install -U sentence-transformers
```

6) Recherche sémantique des transformateurs de phrases BERT (encodeur)

Il s'agit d'une application simple pour les incorporations de phrases :
 recherche sémantique donnée de la phrase de requête, cela trouve la phrase
 la plus similaire dans ce corpus script sorties pour diverses requêtes les 5
 publications les plus similaires dans le corpus

```

from sentence_transformers import SentenceTransformer
import scipy.spatial
import pickle as pickle
embedder = SentenceTransformer('bert-base-nli-mean-tokens')

sentences = list(text_df['Text Processed'])

# Example query sentences
queries = ['How to evolve architecture for constellations and simulation', 'Build behavior of complex aerospace and modeling of safety']
query_embeddings = embedder.encode(queries, show_progress_bar=True)
text_embeddings = embedder.encode(sentences, show_progress_bar=True)

# Find the closest 5 sentences of the corpus for each query sentence based on cosine similarity
closest_n = 5
print("\nTop 5 most similar sentences in corpus:")
for query, query_embedding in zip(queries, query_embeddings):
    distances = scipy.spatial.distance.cdist([query_embedding], text_embeddings, "cosine")[0]

    results = zip(range(len(distances)), distances)
    results = sorted(results, key=lambda x: x[1])

    print("-----User Query: -----")
    print("---", query, "---")
    print("-----")

# Print out all information for the publications related to user query and a relevancy score
for idx, distance in results[0:closest_n]:
    print("Relevancy Score: ", "(Score: %.0f%%)" % ((1-distance) * 100.0), "\n")
    row_dict = pubs.iloc[idx].to_dict() # pubs.loc[pubs.index== sentences[idx]].to_dict()
    #print(row_dict)
    print("Title: ", row_dict["Title"], "\n")
    print("Authors: ", row_dict["Authors"], "\n")
    print("Date: ", row_dict["Date"], "\n")
    print("Link: ", row_dict["Link"], "\n")
    print("Abstract Length: ", row_dict["Abstract Length"], "\n")
    print("Abstract: ", row_dict["Abstract"], "\n")
    print("-----")

```

7) L'Output sur Google Colab.

```
100%|██████████| 405M/405M [00:15<00:00, 26.2MB/s]
Batches: 100% ██████████ 1/1 [00:01<00:00, 1.50s/it]

Batches: 100% ██████████ 21/21 [01:30<00:00, 4.31s/it]

Top 5 most similar sentences in corpus:
-----User Query: -----
-- How to evolve architecture for constellations and simulation --
-----
Relevancy Score:    (Score: 68%)

Title:   Fault Diagnostics and Prognostics for Large Segmented SRM's
Authors: Dimitry Luchinsky,Vadim Smelyanskiy,Viatcheslav Osipov,Dogan Timucin
Date:    03/07/09
Link:    http://ti.arc.nasa.gov/publications/245/download/
Abstract Length:  1544
Abstract:

Abstract-Prognostics has taken center stage in Condition
Based Maintenance (CBM) where it is desired to estimate
Remaining Useful Life (RUL) of a system so that remedial
measures may be taken in advance to avoid catastrophic
events or unwanted downtimes. Validation of such
predictions is an important but difficult proposition and a
lack of appropriate evaluation methods renders prognostics
meaningless. Evaluation methods currently used in the
research community are not standardized and in many cases
do not sufficiently assess key performance aspects expected
out of a prognostics algorithm. In this paper we introduce
several new evaluation metrics tailored for prognostics and
show that they can effectively evaluate various algorithms as
compared to other conventional metrics. Four prognostic
algorithms, Relevance Vector Machine (RVM), Gaussian
Process Regression (GPR), Artificial Neural Network
```

1)Le cosinus Score:

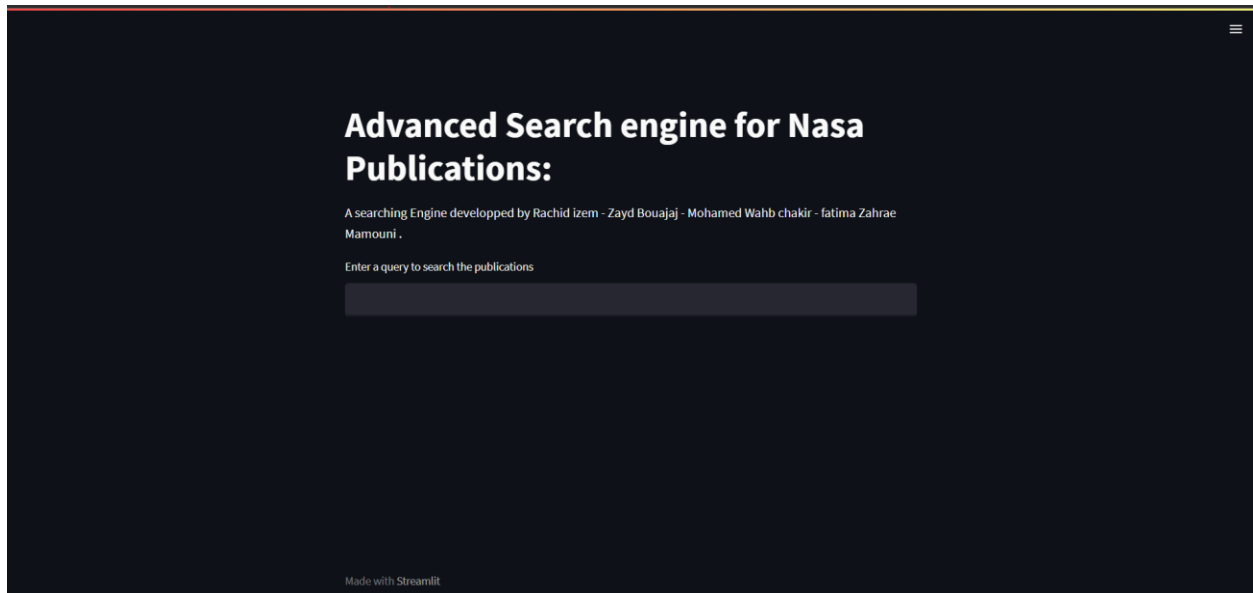
il est utilisé comme métrique d'évaluation de distance entre deux points du plan.

```
-----
Relevancy Score:    (Score: 67%)
```

1) Implementation sur Streamlit:

```
# Streamlit app
st.title('Advanced Search engine for Nasa Publications:')
st.write('A searching Engine developped by Rachid izem - Zayd Bouajaj - Mohamed Wahb chakir - fatima Zahrae Mamouni .')
query = st.text_input('Enter a query to search the publications')
if query:
    results = get_search_result(embedder, text_embeddings, query, closest_n=5)
    st.write('Results:')
    for result in results:
        st.write(result['title'])
        st.write(result['abstract'])
        st.write('Score:', result['score'])
        st.write('Word Count:', result['word_count'])
        st.write('Abstract Length:', result['abstract_length'])
        st.write('---')
```

2) Interface Graphique avec Streamlit:



3) Training apres chaque requette de recherche :

```

2022-11-30 21:00:59.252 Load pretrained SentenceTransformer: sentence-transformers/bert-base-nli-mean-tokens
2022-11-30 21:01:01.397 Use pytorch device: cuda
2022-11-30 21:01:01.469 Use pytorch device: cuda
Batches: 100% | 22/22 [00:48:00:00, 2.21s/it]
Batches: 100% | 22/22 [00:48:00:00, 2.21s/it]
Batches: 100% | 1/1 [00:00:00:00, 4.38it/s]
2022-11-30 21:06:52.467 Load pretrained SentenceTransformer: sentence-transformers/bert-base-nli-mean-tokens
2022-11-30 21:06:52.581 Load pretrained SentenceTransformer: sentence-transformers/bert-base-nli-mean-tokens
2022-11-30 21:06:56.582 Use pytorch device: cuda
2022-11-30 21:06:58.653 Use pytorch device: cuda
Batches: 100% | 22/22 [01:50:00:00, 5.00s/it]
Batches: 100% | 22/22 [01:50:00:00, 5.02s/it]
Batches: 100% | 1/1 [00:00:00:00, 6.81it/s]
2022-11-30 21:51:46.353 Load pretrained SentenceTransformer: sentence-transformers/bert-base-nli-mean-tokens
2022-11-30 21:51:47.709 Use pytorch device: cuda
Batches: 77% | 17/22 [00:23:00:05, 1.19s/it]

```

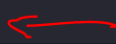
Torch-Cuda : C'est pour L'utilisation du GPU dans le training Deep learning.

4) Test :

Advanced Search engine for Nasa Publications:

A searching Engine developped by Rachid izem - Zayd Bouajaj - Mohamed Wahb chakir - fatima Zahrae Mamouni .

Enter a query to search the publications

the nearest blackholes of our planet 

Results:

[Adaptive Stress Testing of Trajectory Predictions in Flight Management Systems](#)

A primary component of flight management systems is to provide guidance in the form of navigational waypoints between source and destination airports. A trajectory predictor is the subsystem that provides this guidance directly to the pilots or autopilot. Failures within the trajectory prediction system can occur if the provided waypoints are infeasible to fly given the physical limitations of the aircraft. Adaptive stress testing is a method which uses reinforcement learning to search for rare failure events in sequential decision-making systems. This work applies adaptive stress testing of trajectory prediction systems to systematically find failure events and their likelihoods. The trajectory prediction system is treated as a black-box simulator and the adaptive stress testing approach controls the seed for the random number generator that affects the sampling of assembled waypoints and other environmental input parameters. Monte Carlo tree search with action progressive widening is used to explore the possible trajectories and a notion of "miss distance" to a failure event is used to help guide the search. Transition probabilities between states are used to find the most likely failures. Experiments will be run to generate likely failure

Le moteur affiche 5 résultats des articles scientifiques pertinentes

Avec :

Score : (COSINUS SIMILARITY SCORE SCORE)

Word Count:

Abstract Length:

Score: 0.6116993414519832

Word Count: 235

Abstract Length: 1412

Capturing & Analyzing Requirements with FRET

§ Exceeding sensor limits shall latch an autopilot pullup when the pilot is not in control (not standby) and the system is supported without failures (not apfail). § The autopilot shall change states from TRANSITION to STANDBY when the every time these conditions hold or only when they become true?

§ The autopilot shall change states from TRANSITION to NOMINAL when the

Score: 0.5746247970004706

Word Count: 636

Abstract Length: 378