



République Algérienne Démocratique et Populaire



Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

**Université des Sciences et de la Technologie Houari Boumediene**

**Faculté d'Informatique  
Département**

**Intelligence Artificielle et Sciences des Données**

Mémoire de Master

**Filière: Informatique**

**Spécialité : SII**

---

**Thème**

*Détection de plagiat cross- lingue.*

---

**Sujet Proposé par :**

**M (e) ALIANE Hassina :**

**Soutenu le :21/09/2022**

**Devant le jury composé de :**

**Présenté par :**

**ADMANE Hocine**

**TEBBANI Mohamed Walid**

**M ISLI Amar Président (e)**

**Me MOULAI Hadjer Membre**

Binôme N° : SII\_E\_019

# Remerciements

Tout d'abord, nous tenons à manifester notre gratitude à Dieu par aisance et excellence, qui nous a donné la force et la connaissance pour effectuer ce travail dur, qu'il nous guide dans le droit chemin inshallah.

Nous tenons à remercier tout particulièrement notre encadrante **Mme. ALIANE Hassina**, pour l'aide compétente qu'elle nous a apportée, pour sa patience et son encouragement. Son œil critique nous a été très précieux pour structurer le travail et pour améliorer la qualité des différentes sections.

Que les membres de jury trouvent, ici, l'expression de nos sincères remerciements pour l'honneur qu'ils nous font en prenant le temps de lire et d'évaluer ce travail.

Pour finir, nous souhaitons remercier toute personne ayant contribué de près ou de loin à la réalisation de ce travail.

***WALID & HOCINE***

# Résumé

Dès que nous sommes entrés dans la nouvelle ère de la communication numérique, la facilité de transfert et de partage d'informations par internet a encouragé la recherche en ligne. Cela provoque et augmente le risque potentiel du vol de propriété intellectuelle, comme le vol d'une publication en ligne ou le fait de l'utiliser comme une base pour produire une oeuvre sans citer l'auteur de la source. Ce type d'activité s'appelle le plagiat. Ce dernier est devenu un grand problème de nos jours qui évolue rapidement. Une nouvelle contre-vague s'est produite pour l'éliminer et minimiser ses cas, ce sont les outils de détection du plagiat qui utilisent des divers techniques, approches et algorithmes afin de le repérer et le prévenir.

L'un des phénomènes de plagiat qui est le plus observé récemment, c'est celui avec la traduction de la source. Ce type là est apparu dès que les outils de traduction automatiques ont été disponibles au public. De nombreuses études contribuent à la recherche des solutions afin de détecter ce phénomène que nous appelons le plagiat cross-lingue. Nous nous intéressons à l'utilisation de l'intelligence artificielle (**IA**) ainsi que le traitement automatique de langage naturel **TALN** pour développer un système de détection du plagiat cross-lingue.

---

**Mots clés :** Détection plagiat, **cross-lingue**, AI, Deep-learning, Traitement Automatique du Langage Naturel(**TALN**), Word **embeddings**.

---

# Abstract

As soon as we entered the new era of digital communication, the ease of transferring and sharing information using internet encouraged online literature research. This causes and increases the potential risk of intellectual property theft, such as stealing an online publication or using it as a basis for producing a work without citing the source. This type of activity which is called plagiarism has become a big problem these days and it is rapidly increasing day by day. With its evolution, a new counter-wave has arisen to eliminate it and minimize its cases, it is the plagiarism detection tools and softwares that use various techniques, approaches and algorithms to detect plagiarism and prevent it.

One of the most recent observed plagiarism phenomena is plagiarism using translation of the source language, this type of plagiarism appeared as soon as automatic translation tools were available to the public, numerous studies and research contributes to the search for prevention of this phenomenon that we call cross-lingual plagiarism. One of the approaches that interests us is the use of artificial intelligence (**IA**), so we will be using AI and also automatic natural language processing **NLP** to developed a new plagiarism prevention system on cross-lingual plagiarism that said it is the most populartype of plagiarism.

---

**Keywords :** detection plagiarism , cross-language, AI, Deep-learning, Automatic Natural Language Processing(**TALN**), Word embeddings.

---

## ملخص

بمجرد دخولنا العصر الجديد للاتصالات الرقمية ، شجعت سهولة نقل المعلومات ومشاركتها باستخدام الإنترنت البحث في الأدبي. يؤدي هذا إلى زيادة المخاطر المحتملة لسرقة الملكية الفكرية ، مثل سرقة منشور عبر الإنترنت أو استخدامه كأساس لإنتاج عمل دون ذكر المصدر. أصبح هذا النوع من النشاط الذي يسمى الانتحال مشكلة كبيرة هذه الأيام وهو يتزايد بسرعة يوماً بعد يوم. مع تطورها ، ظهرت موجة مضادة جديدة للقضاء عليها وتقليل حالاتها ، وهي أدوات وبرامج الكشف عن الانتحال التي تستخدم تقنيات وأساليب وخوارزميات مختلفة لاكتشاف الانتحال ومنعها.

من أحدث ظواهر السرقة الأدبية التي لوحظت هي الانتحال باستخدام ترجمة لغة المصدر ، ظهر هذا النوع من الانتحال بمجرد توفر أدوات الترجمة الآلية للجمهور ، وتساهم العديد من الدراسات والأبحاث في البحث عن الوقاية من هذه الظاهرة التي نسميها عبر اللغات الانتحال. أحد الأساليب التي تهمنا هو استخدام الذكاء الاصطناعي (IA) ، لذلك سنستخدم الذكاء الاصطناعي أيضاً المعالجة التلقائية للغة الطبيعية NLP لتطوير نظام جديد لمنع الانتحال على سرقة أدبية عبر اللغات.

---

### كلمات مفتاحية :

كشف السرقة الأدبية،الذكاء الاصطناعي ، تمثيل الكلمات.المعالجة التلقائية للغة الطبيعية

---

# Table des matières

Remerciements . . . . .	I
Résumé . . . . .	II
Introduction générale . . . . .	1
<b>1 État de l'art . . . . .</b>	<b>3</b>
1.1 Introduction . . . . .	4
1.2 Définition du plagiat . . . . .	4
1.3 Types de Plagiat . . . . .	4
1.3.1 Plagiat complet (Vol de travail) . . . . .	4
1.3.2 Plagiat direct (copier /coller) . . . . .	4
1.3.3 Paraphrase . . . . .	4
1.3.4 Auto plagiat . . . . .	4
1.3.5 Plagiat mosaïque (Patchwork Plagiarism) . . . . .	4
1.3.6 Plagiat basée sur une source externe . . . . .	5
1.3.7 Plagiat accidentelle . . . . .	5
1.3.8 Fabrication et modification des données . . . . .	5
1.3.9 Plagiat Structurelle . . . . .	5
1.3.10 Collaboration des auteurs . . . . .	5
1.3.11 Contribution d'un travail . . . . .	5
1.3.12 Plagiat par traduction . . . . .	5
1.4 Méthodes de détection du plagiat . . . . .	5
1.4.1 Méthodes extrinsèques . . . . .	6
1.4.1.1 Empreinte textuelle du document ( <i>Document fingerprinting</i> ) . . . . .	6
1.4.1.2 Modèle basée sur les caractères ( <i>String-Matching</i> ) . . . . .	6
1.4.1.3 Modèle basé sur l'espace vectoriel ( <i>Vector Space Model VSM</i> ) . . . . .	6
1.4.1.4 Modèle basée sur la citation . . . . .	7
1.4.1.5 Modèle basée sur la syntaxe ( <i>Grammaire</i> ) . . . . .	7
1.4.1.6 Modèle basé sur la sémantique . . . . .	7
1.4.1.7 Modèle basé sur la grammaire sémantique ( <i>Hybride</i> ) . . . . .	7
1.4.1.8 Modèle basé sur classification et clustering . . . . .	8
1.4.2 Méthodes intrinsèques . . . . .	8
1.4.2.1 la Stylométrie . . . . .	8
1.5 Les mesures de similarité sémantique . . . . .	8
1.6 Les approches pour la Détection du plagiat cross-langue . . . . .	9
1.6.1 Modèles basés sur le lexique et la syntaxe . . . . .	9
1.6.1.1 Approche modèle de longueur (Length) . . . . .	9
1.6.1.2 Approche vecteurs cross-langue de n-grammes (CL-CNG) . . . . .	9

1.6.1.3	Approche ascendance des mots (Cognateness) . . . . .	10
1.6.2	Modèles basés sur corpus parallèles . . . . .	10
1.6.2.1	Approche analyse des similarités sur les alignements (CL-ASA) . . . . .	10
1.6.2.2	Approche LSI (CL-LSI) . . . . .	11
1.6.2.3	Approche corrélation canonique de noyaux (CL-KCCA) . . . . .	11
1.6.3	Modèles basés sur corpus comparables . . . . .	11
1.6.3.1	Approche analyse sémantique explicite (CL-ESA) . . . . .	12
1.6.4	Modèles basés sur dictionnaires et thésaurus . . . . .	12
1.6.4.1	Approche espace vectoriel cross-lingue (CL-VSM) . . . . .	12
1.6.4.2	Approche similarité conceptuelle des thésaurus (CL-CTS) . . . . .	13
1.6.4.3	Approche graphe de connaissances (CL-KGA) . . . . .	13
1.6.5	Modèles basé sur la traduction automatique + analyse mono-lingue(T+MA) . . . . .	13
1.7	Modèles représentations distributionnelles continues de mots (Word embeddings) . . . . .	14
1.7.1	Word2Vec . . . . .	14
1.7.1.1	SkipGram . . . . .	14
1.7.1.2	Continuous bag of words (CBOW) . . . . .	15
1.7.2	Word embeddings cross-lingue . . . . .	16
1.7.2.1	Méthode basée sur les Mapping . . . . .	16
1.7.2.2	Méthodes basées sur des corpus pseudo-multilingues . . . . .	16
1.8	Conclusion . . . . .	16
<b>2</b>	<b>Apprentissage profond . . . . .</b>	<b>17</b>
2.1	Introduction . . . . .	18
2.2	Apprentissage automatique . . . . .	18
2.2.1	Définition . . . . .	18
2.2.2	Types d'algorithmes d'apprentissage . . . . .	18
2.2.2.1	Apprentissage supervisé . . . . .	18
2.2.2.2	Apprentissage non supervisé . . . . .	19
2.2.3	Les problèmes rencontrés durant l'apprentissage . . . . .	19
2.2.3.1	Manque des données . . . . .	19
2.2.3.2	Données de mauvaise qualité . . . . .	19
2.2.3.3	Sur-apprentissage (Overfitting) . . . . .	19
2.2.3.4	Sous-apprentissage (Underfitting) . . . . .	20
2.3	Apprentissage profond (Deep learning) . . . . .	20
2.3.1	Définition . . . . .	20
2.4	Structure des réseaux des neurones . . . . .	20
2.4.1	Architecture des réseaux des neurones . . . . .	20
2.4.2	Architecture du perceptron . . . . .	21
2.5	Les fonctions utilisées en réseaux des neurones . . . . .	22
2.5.1	La fonction d'activation . . . . .	22
2.5.2	La fonction coût . . . . .	22
2.5.3	Les fonctions d'optimisations . . . . .	22
2.5.3.1	Décence du gradient (GD) . . . . .	22
2.5.3.2	ADAM . . . . .	22
2.5.4	La propagation arrière (Back propagation) . . . . .	22
2.6	Les modèles de deep learning . . . . .	23
2.6.1	Les réseaux de neurones convolutifs . . . . .	23
2.6.2	Les réseaux de neurones récurrents . . . . .	23

2.6.2.1	Définition . . . . .	23
2.6.2.2	LSTM . . . . .	24
2.6.2.3	GRU . . . . .	25
2.6.2.4	bidirectionnel LSTM/GRU (Bi-LSTM / Bi-GRU) . . . . .	26
2.6.2.5	Encodeur-Décodeur . . . . .	27
2.6.3	Le modèle transformer . . . . .	28
2.6.4	Le modèle BERT . . . . .	29
2.6.4.1	Architecture de BERT . . . . .	29
2.6.4.2	Codage d'input . . . . .	30
2.6.4.3	L'entraînement de BERT . . . . .	31
2.7	Évaluation du modèle . . . . .	32
2.7.1	Les mesures d'évaluation . . . . .	32
2.7.1.1	Table confusion . . . . .	32
2.7.1.2	Le taux de succès(TS) . . . . .	32
2.7.1.3	Le taux d'erreur(TE) . . . . .	33
2.7.1.4	Rappel (Recall) . . . . .	33
2.7.1.5	Précision . . . . .	33
2.7.1.6	F1-Score . . . . .	33
2.8	Conclusion . . . . .	33
<b>3</b>	<b>Conception et Réalisation . . . . .</b>	<b>34</b>
3.1	Introduction . . . . .	35
3.2	Utilisation d'Arabic-English Cross-Lingual Word Embedding (ArbEngVec) . . . . .	35
3.2.1	Définition . . . . .	35
3.2.2	Pré-traitement et normalisation . . . . .	35
3.2.3	Les modèles de ArbEngVec . . . . .	35
3.2.4	L'espace commun multilingue . . . . .	36
3.2.5	Référence . . . . .	36
3.3	Conception du système . . . . .	36
3.4	Réalisation du système . . . . .	38
3.4.1	Dataset STS (Semantic Textual Similarity) . . . . .	38
3.4.1.1	Description du dataset STS . . . . .	38
3.4.1.2	Architecture du dataset STS . . . . .	39
3.4.2	Pré-traitement sur dataset . . . . .	39
3.4.3	Implémentation des modèles de deep learning . . . . .	42
3.4.3.1	Modèle LSTM . . . . .	42
3.4.3.2	Modèle GRU . . . . .	43
3.4.3.3	Les modèles LSTM/GRU bidirectionnels . . . . .	44
3.4.3.4	Modèle hybride LSTM/GRU Bidirectionnel . . . . .	45
3.4.3.5	Modèle BERT . . . . .	46
3.4.4	Optimisation des hyperparamètres . . . . .	47
3.4.5	Présentation de l'application . . . . .	48
3.4.5.1	Interface graphique . . . . .	48
3.4.5.2	Les bibliothèques utilisés . . . . .	49
3.4.5.3	l'environnement utilisé . . . . .	51
3.4.5.4	Diagramme de déploiement . . . . .	52
3.5	Tests et expérimentation . . . . .	54
3.5.1	Tests et interprétations . . . . .	54



3.5.2	Résultats finaux et évaluations . . . . .	56
3.6	Conclusion . . . . .	57
<b>Glossaire et Acronymes . . . . .</b>		<b>59</b>
<b>Annexes . . . . .</b>		<b>66</b>

# Table des figures

1.1	Architecture de modèle SkipGram [4]	15
1.2	Architecture de modèle CBOW [4]	15
2.1	Exemple d'une architecture de réseaux des neurones (Feed-Forward)	21
2.2	Exemple d'une architecture d'un perceptron	21
2.3	Architecture d'un réseau CNN [15]	23
2.4	Architecture d'un réseau RNN Basic	24
2.5	Architecture d'une cellule LSTM [22]	25
2.6	Architecture d'une cellule GRU [22]	26
2.7	Architecture bidirectionnel avec des cellules GRU(Bi-GRU). [17]	27
2.8	Architecture bidirectionnel avec des cellules LSTM(Bi-LSTM). [44]	27
2.9	Architecture du modèle Encoder-Décodeur avec utilisation des cellules LSTM[16]	28
2.10	Architecture du modèle transformer avec nombre des cellules égale à deux[18]	29
2.11	Architecture du modèle BERT-Base[21]	30
2.12	Architecture du vecteur d'entrée de BERT[28]	30
2.13	l'entraînement de BERT [38]	32
3.1	Architecture du notre première unité (phase Entraînement/Teste)	37
3.2	Architecture du notre deuxième unité (modèle utilisateur)	38
3.3	Architecture du modèle LSTM	43
3.4	Architecture du modèle GRU	43
3.5	Architecture du modèle Bidirectionnel LSTM	44
3.6	Architecture du modèle Bidirectionnel GRU	45
3.7	Architecture du modèle hybride bidirectionnel LSTM/GRU	46
3.8	Fenêtre de choix d'un modèle et d'une requête donné par l'utilisateur	48
3.9	Fenêtre pour les Résultats de la requête donné par utilisateur	48
3.10	Fenêtre de création d'un modèle personnalisé par utilisateur	49
3.11	Diagramme de déploiement de notre système	52
12	fenêtre pour connexion a flask	67
13	importer drive de compte connecté dans Google colab	67
14	l'emplacement des modèles entraînés	68
15	Distribution des fichiers HTML et CSS et JS dans les deux dossiers	68
16	Fenêtre montre le lien qui dirige vers notre système de anti-plagiat cross-lingue	69
17	La page principale de notre logiciel	69
18	Fenêtre de choix d'un modèle plus la sélection d'une requête de teste	70
19	Fenêtre résultat d'une requête entré par utilisateur	70
20	Fenêtre d'entrainement d'un modèle personnalisé	71

# Liste des tableaux

1.1	Mesure des similarité et leurs formules . . . . .	9
2.1	Structure de la table de confusion . . . . .	32
3.1	Préfixes dans la langue anglais et comment les supprimer . . . . .	40
3.2	Suffixe dans la langue anglais et comment les supprimer . . . . .	41
3.3	Préfixes dans la langue arabe . . . . .	41
3.4	Suffixes dans la langue arabe . . . . .	42
3.5	Résultats des tests sur le dataset utilisant LSTM . . . . .	54
3.6	Résultats des tests sur le dataset utilisant GRU . . . . .	55
3.7	Résultats des tests sur le dataset utilisant GRU et LSTM . . . . .	56
3.8	Résultat du modèle BERT . . . . .	56
3.9	Précision et F1-score du modèles . . . . .	57

# Introduction générale

De nos jours, l'évolution rapide des technologies et l'accès à l'information par différents moyens telles que les moteurs de recherche, a ouvert la voie au phénomène du plagiat qui est devenu un problème majeur qui touche plusieurs domaines tels que le journalisme, les universités (rapports d'études, thèses) et qui peut créer parfois des problèmes juridiques et éthiques pour les étudiants et les professeurs. Il est à savoir que le mot plagiat provient des racines latines : *plagiarius* qui veut dire « kidnappeur » et il s'agit d'une personne qui a volé les mots et les idées des autres. En 18ème siècle, le plagiat a été considéré comme de la malhonnêteté intellectuelle[9].

En effet, la facilité d'accès aux documents électroniques via le web a favorisé la généralisation de ce phénomène et constitue aujourd'hui une grande préoccupation pour la communauté universitaire.

Pour cela, les ingénieurs et les chercheurs sont parvenus à remédier à ce problème en mettant en place des logiciels anti-plagiat. On s'intéressera aux méthodes développées pour la détection du plagiat **cross-lingue** qui signifie son décèlement par la traduction traitant de l'identification et de l'extraction automatiques du plagiat dans un environnement multilingue. C'est pourquoi nous allons approfondir nos recherches afin de trouver des méthodes de détection pour ce type de plagiat ; Donc la question qui se pose est comment peut-on procéder pour détecter le plagiat **cross-lingue** en utilisant une approche deep-learning ?

## Objectifs

L'objectif de notre travail est l'étude de la détection du plagiat cross-lingue afin de parvenir à proposer une approche basée sur les modèles multilingues pour l'identification de plagiat entre les deux langues anglais-arabe.

## Organisation du mémoire

La suite de ce mémoire est organisé en quatre chapitres comme suit :

Le premier chapitre “**État de l'art**” se focalisera sur le phénomène de plagiat. Ensuite, on y citera ses différents types selon leurs fonctionnements et processus. Ce chapitre est constitué de deux catégories de méthodes de recherche et de détection de plagiat. Celles ci vont être comparées entre elles à travers un tableau. Par la suite, on mentionnera un autre tableau contenant des formules de mesure de similarité utilisées par les plupart de ces méthodes. Puis nous allons aborder le principe de la détection du plagiat **cross-lingue** et en particulier les travaux basés sur l'utilisation des modèles neuronaux multilingues.

Le deuxième chapitre “**Apprentissage profond**” se focalisera sur deux principes, le premier

consistera à définir l'apprentissage automatique (le machine learning) et décrire les différents types d'algorithme d'apprentissage du machine learning ainsi que tous les problèmes auquel on est susceptible de faire face durant cette opération. Le deuxième se concentrera sur l'un des outils de l'apprentissage automatique qui est le deep learning. Celui ci définira ce concept avec des citations de différents types, architectures et fonctions utilisées ainsi que la méthode d'évaluation des modèles de deep learning.

Dans la dernière phase de notre projet “**Conception et Réalisation**”, nous allons présenter le dataset utilisé pour l'entraînement des modèles et également définir le modèle OPEN source Word embeddings utilisé. Par la suite, nous introduirons les architectures que nous avons développé dans notre système de détection du plagiat. Et de faire des tests sur les performances des modèles ainsi que les comparaisons sur les résultats des modèles.

# Chapitre 1

## État de l'art

### 1.1 Introduction

Actuellement, le plagiat est un phénomène qui constitue un sérieux problème. ce chapitre présente le plagiat : la généralité sur le plagiat et les approches de détection du plagiat , par la suite nous intéressons aux méthode de la détection du plagiat cross-langue. on va le clôturer avec une définition et une présentation de la structure *word embeddings*.

### 1.2 Définition du plagiat

C'est la présentation du travail ou des idées de quelqu'un d'autre comme étant les nôtres, en les ajoutant dans notre travail sans mentionner la référence qui renvoie à la propriété du véritable auteur. Cette définition porte sur tous les documents qu'ils soient imprimés, manuscrits ou électroniques. Le plagiat peut être intentionnel ou pas mais reste quand même une faute disciplinaire[34].

### 1.3 Types de Plagiat

#### 1.3.1 Plagiat complet (Vol de travail)

C'est le type le plus grave. Un chercheur prend un travail d'un autre chercheur et il le soumet sous son nom.[30]

#### 1.3.2 Plagiat direct (copier /coller)

Ce type consiste à copier mot par mot sans modification, sans citation de la source et sans utilisation de guillemets pour dire que ce passage a été repris d'une autre source.[30]

#### 1.3.3 Paraphrase

C'est Le type le plus commun en plagiat. Il s'agit de l'utilisation du travail de quelqu'un d'autre avec modification de quelques phrases et mots pour ensuite le soumettre comme étant notre propre travail en gardant la même idée.[30]

#### 1.3.4 Auto plagiat

C'est utilisation de la copie d'un travail propre à nous et ainsi, beaucoup de gens pensent que ce plagiat est inoffensif car il ne s'agit pas d'un "vol".[30]

#### 1.3.5 Plagiat mosaïque (Patchwork Plagiarism)

C'est un type qui consiste à collecter, prendre et ajouter des phrases ou des mots de quelqu'un d'autre dans le travail, ce type est le plus difficile à détecter. [30]

### 1.3.6 Plagiat basée sur une source externe

C'est quand un chercheur fait référence à une source qui est incorrecte ou inconnue ou utilise des source secondaire de données avec la citation de la source principale d'informations.[41]

### 1.3.7 Plagiat accidentelle

Ce type de plagiat est produit lorsqu'une personne confond les auteurs des différentes sources utilisées et oublie de citer quelques auteurs. Le manque d'attention dans la partie de création de la bibliographie peut générer ce type de plagiat.[30]

### 1.3.8 Fabrication et modification des données

Il s'agit de modifier et/ou tricher sur les données d'une recherche quelconque. Ce type est dangereux car ces conséquences peuvent influencer négativement ou saboter le domaine de cette recherche.[27]

### 1.3.9 Plagiat Structurelle

C'est une copie du sommaire (squelette) d'un rapport ou article en apportant une légère modification au niveau du leur plan.[42]

### 1.3.10 Collaboration des auteurs

Lorsqu'un groupe de personnes/chercheurs participent à un projet commun. Parmi ces membres, il y a quelques individus qui ne participent pas, cela conduit à ce type de plagiat.[30]

### 1.3.11 Contribution d'un travail

C'est le contraire du plagiat précédent c'est-à-dire de ne pas inclure une personne ou plus dans la liste des auteurs. Mais cette personne a contribué au travail.[30]

### 1.3.12 Plagiat par traduction

C'est-à-dire qu'un individu prene et copie un autre travail qui a été écrit en une langue différente en la traduisant en sa langue. Ce type de plagiat cause des problèmes pour les logiciels de détection automatisés car la traduction peut être imprécise dû à de nombreuses traductions possibles pour un texte donné. Il est le plus populaire et le plus compliqué à détecter[26].

## 1.4 Méthodes de détection du plagiat

Les méthodes de détection du plagiat peuvent se catégoriser en deux s : les méthodes de détection extrinsèques et les méthodes de détection intrinsèques. Beaucoup de méthodes



externes visent à comparer un document suspect (qui peut être copié collé) avec un autre document externe comparable en utilisant l'une des formules de similarité (paragraphe 1.5).

### 1.4.1 Méthodes extrinsèques

Ces méthodes sont dites externes car elles ont besoin des références externes comme étant une collection des documents disponibles pour le public ou bien sur des sites web. La plupart de ces techniques comparent des phrases et des segments entre 2 textes en utilisant la mesure de similarité (paragraphe 1.5), la tâche donc à accomplir serait de calculer cette mesure entre le document suspect et documents références deux à deux afin de déduire s'il s'agit d'un plagiat ou pas[6].

Les avantages de ces techniques sont de trouver quelles sont les documents qui ont été plagiés est quel passage exact qui a été copié/reformulé. Mais le seul inconvénient majeur c'est que ses méthodes sont lourdes et demandent beaucoup de temps calculs des ressources pour la collecte des documents importants. Les méthodes les plus connues sont :

#### 1.4.1.1 Empreinte textuelle du document (*Document fingerprinting*)

Le principe du **fingerprinting** est que chaque document numérique ait son propre 'ID', ces **IDs** sont mappés en utilisant des algorithmes mathématiques complexes pour représenter tout le contenu d'un document sous format d'une courte chaîne de caractère[31]. La détection du plagiat se fait par la comparaison des **IDs** au lieu des documents eux même. Donc le plagiat peut être reconnu si la similarité des **IDs** entre deux documents est grande. Les fonctions de hachage qui sont appelées aussi les fonctions des fingerprinting sont les plus utilisées lors de la création des empreintes mais il existe toujours des algorithmes intelligents pour faire le même travail[24],[35]

#### 1.4.1.2 Modèle basée sur les caractères (*String-Matching*)

C'est la méthode la plus commune. Elle est aussi appelée la méthode naïve de recherche, elle utilise le calcul de similarité entre deux documents en utilisant la fonction de '**String-Matching**' qui consiste de trouver des occurrences d'une chaîne de motif dans une autre chaîne ou un corps de texte à comparer.[20] :

#### 1.4.1.3 Modèle basé sur l'espace vectoriel (*Vector Space Model VSM*)

Dans la recherche d'information (**RI**) traditionnelle, on utilise le modèle vectoriel pour représenter les phrases et les textes sous forme d'un vecteur. Cette représentation se caractérise du fait que chaque mot ait sa propre dimension dans le vecteur sans répétition en comptant la fréquence de chaque de ces mots.

Après la création des vecteurs (du texte suspect et du textes références externe), ce modèle utilise la mesure de similarité (paragraphe 1.5) pour calculer le taux de similarité entre le texte suspect et la référence.

### 1.4.1.4 Modèle basée sur la citation

D'après son nom, cette méthode est inutile lors des traitements des textes qui n'ont pas de références bibliographiques mais elle reste toujours indépendante des langages textuels. Elle appartient à la classe des méthodes sémantiques car elle vérifie/compare les motifs des références existantes dans les deux documents suspects et référence.

De plus elle vérifie l'ordre des citations dans le document suspect avec les documents références pour savoir si un motif peut être extrait et si c'est le cas, alors le taux de similarité augmente. Cet algorithme est efficace pour détecter les types de plagiat complexe tels que paraphrases et structurels.

### 1.4.1.5 Modèle basée sur la syntaxe (*Grammaire*)

Pour identifier le plagiat cette méthode fait une analyse syntaxique d'un texte pour extraire les caractéristiques syntaxiques en utilisant des structures simples comme fréquence des marqueurs **POS** (Partie de discours) ou marqueurs **POS**-n-gramme ou en utilisant des structures plus complexes comme des analyseurs syntaxiques complets ou partiels. Ces analyseurs peuvent contenir des règles syntaxiques des phrases (phrase nominale, phrase verbale), fréquence des mots dans une phrase ou nombre des mots dans une phrase.[5]

L'idée de base c'est que les documents similaires (copies exacts) ont presque même marqueur **POS** (ou même structure des phrases dans le cas d'utilisation des analyseurs syntaxiques). En effet, plus ces marqueurs sont utilisés, plus la similarité augmente. C'est-à-dire, les documents similaires ou ceux qui contiennent des fragments de textes des autres documents contiendront les mêmes structures [2]

### 1.4.1.6 Modèle basé sur la sémantique

Ces méthodes se basent sur la signification et le contenu des phrases plutôt que leurs propriétés syntaxiques. Elles permettent de calculer la similarité entre deux phrases même celles qui sont différemment écrites dû au fait que ces méthodes ne prennent pas en considération l'ordre des mots.

Deux phrases peuvent être sémantiquement identiques mais ne diffèrent dans leurs structures, ex : Voix active/passive, ou diffèrent dans le choix des mots, ex : Synonyme/Mots proches. Pas mal de plagiaires tentent de modifier leurs travaux lorsqu'ils prennent un texte en adaptant cette façon de changer les phrases et garder la même sémantique, ces actes des plagiats sont les plus communs (Le paraphrase). Alors ces méthodes de détection sont particulièrement utiles contre ce type de plagiat, mais restent coûteuses et difficiles à implémenter les connaissances sémantiques (les relations entre les mots).[2]

### 1.4.1.7 Modèle basé sur la grammaire sémantique (*Hybride*)

C'est la méthode la plus importante dans la détection du plagiat pour le traitement automatique du langage naturel **TALN**, elle permet de détecter les types de plagiat copier/coller, paraphrase et le ré-arrangement des mots, le changement de structure grammaticale des phrases.[12][13]  
[1]

### 1.4.1.8 Modèle basé sur classification et clustering

la technique de clusteriser les documents est l'une des techniques de recherche d'informations (**RI**). Dans la détection de plagiat, le clustering se consiste en le groupement des documents qui sont similaires dans un cluster et les documents qui sont pas similaires se trouvant dans des clusters différents. Il existe plusieurs algorithmes de clustering mais dans le cas d'une comparaison entre deux documents, le choix de la méthode requiert celle qui utilise le calcul de la similarité tel que (la distance, similarité cosinus (paragraphe 1.5)). [13]

### 1.4.2 Méthodes intrinsèques

Contrairement aux méthodes externes, ces méthodes n'ont pas besoin d'une référence externe car leurs principe est d'analyser le changement du style d'écriture dans le texte et aussi de prédire que les segments du texte n'ont pas été écrit par un même auteur. [6]

#### 1.4.2.1 la Stylométrie

La stylométrie (ou L'extraction des marques stylistiques d'un texte écrit) est une méthode basée sur l'examination statistique de caractéristique linguistique du texte. Elle essaye de déterminer la probabilité que ce document a été écrit par un auteur particulier basé sur les marqueurs de style. [11]

La stylométrie essaye de comparer le contenu du document avec lui-même sans utiliser des documents externes. Ces marqueurs sont des variables spéciales qui changent leurs valeurs suivant la richesse vocabulaire du texte, les différences des valeurs de ces variables entre l'état courant et précédent permet de détecter le plagiat ; ex :  $x=24$  après elle est changée en la valeur  $x=46$ .

Elle vise l'analyse de style de l'écriture et il existe plusieurs algorithmes qui permettent de détecter le changement du style. En théorie, cette méthode d'enquête peut déterminer les différences entre les auteurs, ce qui rend ce dernier unique. Naturellement, puisque la stylométrie est une méthode intrinsèque. Elle peut détecter le plagiat mais elle ne peut pas prédire quel texte à été volé.

## 1.5 Les mesures de similarité sémantique

soient  $X, Y$  deux vecteurs ou des chaînes à comparer,  $X$  et  $Y$  peuvent être des vecteurs qui représentent soit des mots, chaîne des caractères ou des documents. Ou bien ils peuvent être directement des chaînes, la distance entre deux caractères est la distance entre leur ordre alphabétique. Soit  $x_i, y_i$  deux éléments du vecteur/chaîne de caractères  $X$  et  $Y$  respectivement, la distance entre  $X$  et  $Y$  peut être calculée avec des formules de similarité, on liste quelque unes.

Nom	Formule	Nom	Formule
Distance Manhattan	$\sum  x_i - y_i $	Produit interne	$\sum x_i \cdot y_i$
Distance Euclidienne	$\sqrt{\sum  x_i - y_i ^2}$	Cosinus	$\frac{\sum x_i \cdot y_i}{\sqrt{\sum x_i^2} \cdot \sqrt{\sum y_i^2}}$
Distance Minkowski	$\sqrt[p]{\sum  x_i - y_i ^p}$	Jaccard	$\frac{\sum x_i \cdot y_i}{\sum x_i^2 + \sum y_i^2 - \sum x_i \cdot y_i}$
Coefficient de recouvrement	$\frac{\sum x_i \cdot y_i}{\min(\sum x_i^2, \sum y_i^2)}$	Dice	$\frac{2 * \sum x_i \cdot y_i}{\sum x_i^2 + \sum y_i^2}$

TAB. 1.1 : Mesure des similarité et leurs formules

## 1.6 Les approches pour la Détection du plagiat cross-lingue

### 1.6.1 Modèles basés sur le lexique et la syntaxe

Ces ensembles de modèles sont basés sur la comparaison lexicale ou syntaxique. Ces méthodes de traitement sont assez rapides en temps de déroulement à cause de l'utilisation de pré\_processing (suppression des mots vides les caractères diacritiques, des espaces,...etc.),et aussi elles ne nécessite pas la traduction du texte pour comparer. Ce modèle utilise plusieurs méthodes de détection du plagiat. [6]

#### 1.6.1.1 Approche modèle de longueur (Length)

Le modèle de longueur vise à comparer la taille de deux textes et à prédire s'ils indiquent la même idée. Il est beaucoup moins probable que deux textes T1 et T2 (et qui ont la même idée mais T1 est écrit dans une langue et T2 est écrit dans une autre) ont la même longueur D alors soit D1 la taille de texte T1 et D2 la taille de T2, ce qu'il y a été remarqué c'est qu'il y a à-peu-près toujours la même différence sur la taille des textes en suivant leurs langues. Cette différence est proportionnelle suivant les langues concernées et peut être engendrée par cette formule :[6]

$$Sim(d, d') = \exp\left(-0.5\left(\frac{|d'|}{|d|} - \mu\right)^2\right) \quad (1.1)$$

Avec  $\mu$  comme étant la moyenne de longueurs entre le document original  $d$  et sa traduction  $d'$ , et  $\sigma$  comme étant leurs écart-type

#### 1.6.1.2 Approche vecteurs cross-lingue de n-grammes (CL-CNG)

Le modèle ou (*Cross-Language Character N-Gram*) dans l'objectif de rendre deux documents écrits en deux langues différentes comparables d'une façon triviale et indépendante de leur langue d'origine, sous forme de vecteurs de **n-grammes**. C'est la méthode la plus simple pour la détection du plagiat mais elle est limitée à un niveau lexical. C'est à dire que les deux langues à comparer doivent se partager les mêmes alphabets, par exemple : on peut appliquer la méthode sur les langues (**Anglais/français**) contrairement aux langues (**Anglais/Arabic**).[6]

Les caractères **n-grammes** sont connus en étant une représentation textuelle puissante des mots et elles sont efficaces pour donner de bons résultats en recherche d'informations contrairement à utilisation des mots tel qu'ils sont. D'où la détection du plagiat devient possible grâce au calcul des similarités.

### 1.6.1.3 Approche ascendance des mots (Cognateness)

Le modèle **Cognateness** est ancien, il compare des cognâtes plutôt que les mots ou phrases eux mêmes, les cognâtes sont des termes spéciaux ou un ensemble de mots qui appartiennent à des langues différentes. La relation spéciale entre ces mots est qu'ils ont les mêmes ancêtres et parents, ex : Anglais : **Brother** , Allemand : **Bruder**, mais il n'ont pas nécessairement le même sens ex : Anglais : **to starve**, Allemand : **sterben = to die**. [6]

Le calcul du similarité entre les deux textes en se basant sur les cognâtes va être efficace car les mots des deux textes sont réduit en alphanumérique (suppression des ponctuations et des symboles) et ils sont réduit à leurs radical (suppression des préfixes et suffixes). Les mots résultants qui ont le nombre de lettres inférieur à un certain critère choisi ne sont pas considérés car ils s'agit de mots vides.

## 1.6.2 Modèles basés sur corpus parallèles

Afin de pouvoir faire une comparaison bilingue ou beaucoup plus plus spécifique **cross-lingue**, le corpus parallèle consiste à aligner deux ou plusieurs ensembles des documents de différentes langues où chaque document d'un ensemble est aligné avec un autre document dans l'autre ensemble qui est sa traduction. L'alignement c'est comme le fait de relier deux document qui ont une langue différente mais l'une est une traduction de l'autre. Les modèles basés sur ce type de corpus les utilise comme leur base de comparaison soit explicitement ou implicitement.

### 1.6.2.1 Approche analyse des similarités sur les alignements (CL-ASA)

Le principe du **CL-ASA** ou (*Cross-Language Alignment Similarity Analysis*) est de créer un dictionnaire statique entre deux langues (un dictionnaire bilingue) qui contient un ensemble des probabilités de traduction de chaque terme. Ce dictionnaire est obtenu à partir d'entraînement d'un corpus parallèle (Le modèle IBM-1. On dit qu'un terme (T1) est une traduction d'un terme (T2) avec une probabilité  $P(T1|T2)$ , pour calculer la probabilité, la règle de bayes est induite [3]

$$P(\mathbf{T1}|\mathbf{T2}) = P(\mathbf{T1}) \cdot \frac{P(\mathbf{T2}|\mathbf{T1})}{P(\mathbf{T2})} \quad (1.2)$$

Contrairement à essayer de traduire un ensemble de mots dans la nouvelle langue, l'objectif de cette méthode est de trouver des mots qui sont des traductions potentielles de la langue d'origine. Une amélioration possible pour cette méthode c'est d'utilisation d'un seuil afin de choisir des probabilités plus élevées pour avoir de bons résultats.

### 1.6.2.2 Approche LSI (CL-LSI)

La méthode **CL-LSI** ou (*Cross-Language Latent Semantic Indexing*) permet d'extraire des mots d'un thème donné depuis un document lui-même. Elle est construite à partir d'un corpus de documents et d'une matrice termes-documents (matrice **LSI**) qui permet d'avoir une vue de chaque terme dans chaque document du corpus, ainsi cette matrice permet de calculer une matrice du rang le plus faible afin de définir un espace conceptuel commun à tous les documents du corpus.

Dans le corpus parallèle, chaque paire de documents alignés est traitée comme un seul document dans la matrice **LSI**, donc la méthode **CL-LSI** construit une seule matrice pour plusieurs langues. C'est-à-dire elle fait une représentation d'un espace **cross-langue**. Une matrice dont les lignes correspondent aux termes et dont les colonnes correspondent aux documents. Soit la matrice où l'élément  $tf.idf(t_i, d_i)$  qui décrit la pondération  $tf.idf(t_i, d_i)$  du terme  $t_i$  (le  $i^{me}$  terme du corpus) dans le document  $d_j$  (le  $j^{me}$  document du corpus), d'où  $n$  est le nombre total de termes qui contiennent le corpus et  $m$  est le nombre total des documents qui forment le corpus. [6]

### 1.6.2.3 Approche corrélation canonique de noyaux (CL-KCCA)

La **CL-KCCA** ou (*Cross-Language Kernel Canonical Correlation Analysis*) est très peu utilisée dans le domaine de détection du plagiat dû à la complexité de ces formules mathématiques. Cette analyse vise à déterminer les corrélations entre les termes d'un corpus parallèle en associant chaque document avec son vecteur. Le corpus parallèle qui contient des documents en deux langues va être représenté sous forme de deux espaces ; un espace pour langue L1 et un autre pour L2 puis les corrélations se produiront à partir des noyaux de ces deux espaces calculés par le modèle.[6]

En utilisant ces deux espaces créés, la **CL-KCCA** projette ces vecteur sur un autre espace commun de manière sémantique ; les mots qui sont proches dans l'espace ont une relation sémantique entre eux comme des synonymes ou des relations de dépendance. Ce nouveau espace contient les vecteurs des documents (en fonction des mots).

Finalement, la **CL-KCCA** cherche seulement les projections où leurs valeurs sont hautement corrélées. Dès que la recherche est finie, des motifs apparaissent qui contiendront des ensembles de mots (en deux langues). Maintenant, cette analyse trouve ces motifs et utilise la mesure F-corrélation pour trouve le plus grand  $n$  de dimensions sémantiques similaire. La mesure F-corrélation canonique (en fonction de  $\rho$ ) calcule la similarité en projetant tous les mots du vecteur d'un document sur les  $n$  composants de la mesure F-corrélation afin de déterminer de quel vecteur le document est proche.[6]

## 1.6.3 Modèles basés sur corpus comparables

Le corpus comparable constitué de deux ensembles des documents ou textes sont écrit dans deux langues différentes, chaque document dans une langue est liée à un autre dans une autre langue. La liaison c'est que les deux documents traitent le même thème mais les deux ne sont pas nécessairement une traduction de l'un à l'autre.

### 1.6.3.1 Approche analyse sémantique explicite (CL-ESA)

Ce modèle est appelé **CL-ESA** ou (Cross-Language Explicite Similarity Analysis) et il s'intéresse au contenu d'un document, il détermine le thème du document à partir de la fréquence de ses termes puis il trouve la bonne collection des documents qui ont le même thème pour obtenir un vecteur spécial qui s'appelle vecteur caractéristique du document. En utilisant ce dernier, le modèle calcule la similarité entre le document et tout autre document de la collection choisie .[6]

Soit dans un corpus comparable deux collections des documents  $C_1$   $C_2$  de taille  $n$  qui sont écrits en deux langues différentes  $L_1$   $L_2$ , soit deux documents externes  $D_1$   $D_2$  à comparer entre eux et qui sont écrit aussi dans deux langue  $L_1$   $L_2$ , le calcul de similarité entre  $D_1$  et  $D_2$  revient à comparer la distance entre leurs vecteurs caractéristiques  $V_{D_1}^{C_1}$   $V_{D_2}^{C_2}$ ,

Un vecteur caractéristique  $V_D^C$  de document  $D$  avec la collection  $C$  est un vecteur de taille  $n$  où chaque composante  $i$  représente la similitude entre le document  $D$  avec chaque document  $d_i$  de la collection  $C$ , autrement dit tous les éléments  $i$  de  $V_D^C$  sont des distances entre le document  $D$  et le document  $d_i$  de la collection  $C$ . Les vecteurs  $V_1$   $V_2$  sont comparables car ils ont été créés à partir des collections comparables  $C_1$   $C_2$  qui viennent d'un même corpus comparable ; c'est-à-dire on peut calculer la similarité entre les deux documents externes  $D_1$  et  $D_2$  en utilisant le corpus comparable.

### 1.6.4 Modèles basés sur dictionnaires et thésaurus

Ces modèles exigent l'utilisation des ressources externes comme dictionnaire ou thésaurus et ils sont indépendants d'utilisation des corpus.

#### 1.6.4.1 Approche espace vectoriel cross-lingue (CL-VSM)

Le modèle **CL-VSM**, (*Cross-Language Vector Space Model*) représente les documents sous une représentation vectorielle en utilisant un thésaurus **cross-lingue**. Les vecteurs produit sont des vecteurs des descriptions pour le document (des libellés) et sont indépendants de la langue donc grâce à cette manière, il est facile de calculer la similarité en utilisant ces libellés qui représentent les documents à comparer au lieu du contenu des documents eux-mêmes.

Lors de création du modèle **CL-VSM**, la ressource externe qui a été utilisée avec c'est EuroVoc. EuroVoc est un thésaurus **cross-lingue** et multidisciplinaire de l'union européenne. Il contient des mots-clés (des concepts) répartis dans 21 domaines et 127 sous-domaines, qui servent à décrire le contenu des documents disponibles[25]. Ce thésaurus contient pour l'instant 24 langues officielles.

Le modèle **CL-VSM** rencontre des termes spéciaux lorsqu'il fait une analyse d'un document. ils sont dits "concepts" ce qui donne une idée sur le thème du document et son domaine. Le modèle utilise ces concepts pour la création des vecteurs. Pour faire ce traitement, les concepts déduits à partir du document sont comparés avec une liste de descriptions qui est associée à la structure EuroVoc. Les éléments de la liste de description EuroVoc sont des libellées assignées automatiquement au document en utilisant le calcul de similarité entre le contenu du document (les concepts) et les éléments de la liste des description EuroVoc. Cette opération dépend beaucoup sur la complétude de la liste des description EuroVoc et aussi de l'exactitude du domaine de document.

A la fin de l'opération, les éléments qui décrivent le document dans le vecteur créé sont ordonnés et triés par leur pertinence. Ensuite pour faire le calcul de similarité entre deux vecteurs, les formules de la recherche d'information habituelle sont utilisées. La formule du *Cosinus* (paragraphe 1.5) est la plus utilisée généralement.

### 1.6.4.2 Approche similarité conceptuelle des thésaurus (CL-CTS)

Le modèle **CL-CTS**, (*Cross-Language Conceptual Thesaurus-Based Similarity*) vise à représenter les documents à Comparer en tant que vecteurs de concept et mesurer leur similarité sémantique sous cette forme. **CL-CTS** n'implique également aucune traduction explicite sur les documents, elle sera implicite à l'aide des correspondances internes au thésaurus utilisé. Cette méthode construit des vecteurs correspondants à ces textes en pondérant les termes selon leurs fréquence d'apparition dans ces textes en projetant ces textes dans l'espace des domaines spécifiques présents dans l'EuroVoc. Par la suite, la formule qui est utilisée pour la comparaison de deux vecteurs fait intervenir une similarité cosinus(paragraphe 1.5).[6]

afin de construire les vecteurs pondérés. **CL-CTS** découpe les textes en phrases pour éliminer les mots vides et permet d'avoir la rapidité d'indexation et aussi la plupart des mots vides (prépositions,pronoms,.. *etc.*) ne disposent pas de traduction claire. Puis elle construit un sac de mots pour chaque phrase en traduisant les lemmes restant de chaque phrase à partir d'un dictionnaire de traduction. Après en fusionnant tous les sacs de mots, il en résulte des représentations dans la même langue des deux phrases d'origine. Pour calculer la similarité des deux phrases d'origine, on effectue un calcul basé sur l'intersection des deux sacs les représentant.

### 1.6.4.3 Approche graphe de connaissances (CL-KGA)

Le modèle **CL-KGA**, (*Cross-language knowledge graph analysis*) utilise la comparaison des graphes de connaissances qui ont été créés en utilisant un réseau sémantique **cross-lingue** pour représenter un document sous forme d'un graphe. La similarité entre deux documents peut être déduite en calculant la similarité entre leurs graphes des connaissances. Ce modèle a été introduit avec utilisation d'une structure appelé BabelNet. [6]

BabelNet est un réseau sémantique **cross-lingue** et une ontologie lexicalisée. BabelNet a été créé en intégrant automatiquement la plus grande encyclopédie **cross-lingue** -Wikipédia-. L'intégration a été réalisée par correspondance automatique. Les entrées manquantes en d'autres langues ont été obtenues par des techniques de traduction automatique. Le résultat est un "dictionnaire encyclopédique" qui fournit des entrées lexicalisées dans la plupart des langues, reliées entre elles par une grande quantité de relations sémantiques[45].

Dans BabelNet, les pages de l'encyclopédie Wikipédia forment des noeuds et les relations sémantiques ainsi que les liens entre ces pages forment des arêtes. L'avantage de cette structure réside dans l'élimination des ambiguïtés des entre mots et la clarification de la relation entre les pages.

### 1.6.5 Modèles basé sur la traduction automatique + analyse mono-lingue(T+MA)

Le modèle (**T+MA**), (*translate followed by **mono-lingue** analysis*) effectue une traduction propre au document contrairement aux autres méthodes qui ont été présentées précédemment



qui utilisent des seulement les ressources lexicale. Cependant la méthode de traduction automatique ramène les deux textes à comparer sous forme d'un seul texte de même langue afin de pouvoir utiliser des approches **mono-lingues**.

Les deux documents sont traduits vers une troisième langue que l'on appelle langue pivot. La plupart des chercheurs favorisent le choix d'utilisation de l'anglais comme langue pivot parce que n'importe quelle langue est facile pour la traduire en anglais. En plus, la plupart des contenus disponibles sur Internet sont en anglais. Aussi, la raison la plus évidente c'est la disponibilité massive d'outils de traitement en anglais : listes de mots vides, tokenisateurs, *etc.* Ces outils ne sont pas tout le temps disponibles dans les langues moins communes, les mots restants sont mis à leurs radicaux. Ensuite, les textes sont découpés en segments afin de constituer des empreintes. La méthode utilisée pour la comparaison est un croisement des empreintes de chaque segment. Puis ces deux documents peuvent alors être comparés par des techniques de comparaison monolingues. Pour cela, il est préférable d'utiliser des méthodes telles que la détection du changement du style d'écriture ou l'utilisation des sacs de mots (avec pondération si possible) qui donnent de meilleurs résultats sur la comparaison des textes **mono-lingues**. Finalement, leur similarité est calculée avec la formule de similarité commune comme la formule du *cosinus* (paragraphe 1.5). [6]

## 1.7 Modèles représentations distributionnelles continues de mots (Word embeddings)

Le word **embeddings** désigne un ensemble de techniques de machine learning qui visent à représenter les mots ou les phrases d'un texte par des vecteurs de nombres réels, décrits dans un modèle vectoriel (ou Vector Space Model). Ces nouvelles représentations de données textuelles ont permis d'améliorer les performances des méthodes de traitement automatique des langues (ou Natural Language Processing), comme le Topic Modeling ou le Sentiment Analysis [36].

### 1.7.1 Word2Vec

word2vec est une classe de modèles qui représente un mot dans un grand corpus de texte sous la forme d'un vecteur dans un espace à  $n$  dimensions (ou espace de caractéristiques à  $n$  dimensions) rapprochant les mots similaires les uns des autres. Un de ces modèles est le modèle SkipGram [36].

#### 1.7.1.1 SkipGram

L'idée de cette méthode est d'utiliser un seul mot pour prédire ses voisins (ces mots sont appelés des "contextes"). Elle se concentre sur le calcul des probabilités des mots contextes en connaissant le mot central. Le réseau de neurones dans cette architecture contient deux couches. Il prend dans la première couche en entrée un seul mot qui est le mot central (qui se trouve au milieu) et il prédit dans la couche de sortie des vecteurs pour chaque mot voisin (mots contextes) avec l'utilisation de la fonction d'activation "SOFTMAX" comme fonction d'activation. L'algorithme SkipGram est meilleur que CBOW car il représente mieux les mots dans le cas où le nombre des mots dans le vocabulaire et le nombre des documents à entraîner sont petits [4].

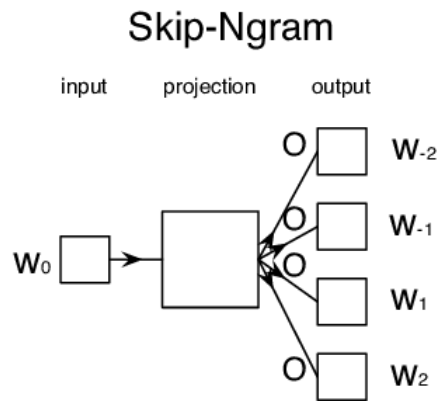


FIG. 1.1 : Architecture de modèle SkipGram [4]

### 1.7.1.2 Continuous bag of words (CBOW)

L'architecture "Continuous Bag of Words (**CBOW**)" est l'opposé du SkipGram. Elle prédit le mot central à partir de ses mots (son contexte). Ce modèle maximise la probabilité qu'un mot se trouve dans un contexte spécifique. Pour entraîner le modèle, des mots sont représentés en utilisant le one-hot encoding qui consiste à représenter une variable catégorielle à  $n$  catégories par  $n$  variables binaires, la  $i$ -ème variable binaire représentant la  $i$ -ème catégorie) puis les contents sont en entrée du réseau de neurones qui prend en entrée (mot- $k$ , ..., mot-1, mot+1, ..., mot+ $k$ ) afin de trouver le mot central, avec  $k$  comme taille de la fenêtre à prendre.

Dans le réseau des neurones entraîné pour ce modèle, la représentation et le nombre de mots en entrée ont un effet sur le nombre du neurone dans le réseaux. En tout, ce modèle contient deux couche. Le nombre de neurones dans la première couche est égal à la taille du vecteur qui représente un mot, elle n'utilise aucune fonction d'activation et aucun poids. La deuxième couche utilise la fonction d'activation "SOFTMAX" pour traduire les nombres réels des vecteurs des mots en un seul vecteur qui contient de distribution des probabilités sur le mot cible. Le modèle CBOW est obtenu en minimisant la perte d'information dans le vecteur obtenu et le vecteur du mot cible (le mot cible est encodé en utilisant le one-hot encoding). Le choix de  $K$  mots contexte et la taille du vecteur d'un mot sont des valeurs paramétrables qu'on peut changer[4].

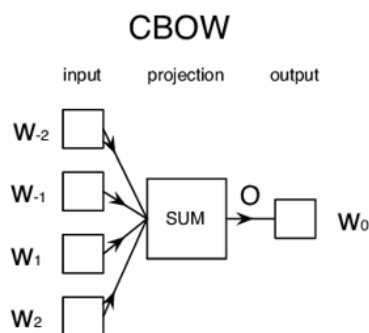


FIG. 1.2 : Architecture de modèle CBOW [4]

### 1.7.2 Word embeddings cross-lingue

Les **embeddings cross-lingue** contiennent des vecteurs des mots (qui ont été pris à partir de plusieurs langues) distribués dans un espace des vecteurs commun. Les **embeddings cross-lingue** peuvent comparer les significations des mots d'une langue à une autre langue, cela signifie qu'ils ne demandent pas beaucoup de ressources puisque les vecteurs sont sur un espace partagé, c'est-à-dire qu'il n'y aura pas de calculs comme des projections des vecteurs, création des copies des vecteurs, les fonctions de traitements et d'alignements ... *etc.*

Lors d'apprentissage d'un modèle **embeddings mono-lingue**, il est possible de faire un transfert de connaissance d'un modèle à un autre, ceci est particulièrement pratique pour la création d'un modèle **embeddings cross-lingue**, les approches de création des modèles **cross-lingue** se différencient, et elles sont :

#### 1.7.2.1 Méthode basée sur les Mapping

Elle transforme des espaces vectoriels **mono-lingue** à plusieurs langues vers un espace vectoriel **cross-lingue**. Dans ce groupe d'approches, la plupart des modèles d'**embeddings cross-lingue** sont créés et étendus avec des modèles d'**embeddings mono-lingue**. Tout d'abord le modèle **cross-lingue** cherche à apprendre une matrice de transformation qui mappe les représentations **embeddings** dans une langue aux représentations **embeddings** dans l'autre langue. Le modèle apprend cette transformation à partir des alignements des mots ou d'un dictionnaire bilingue/multilingue.[7]

#### 1.7.2.2 Méthodes basées sur des corpus pseudo-multilingues

En faisant la concaténation des différents mots qui se trouvent dans un corpus comparable ou parallèle à plusieurs langues puis faire construire les **embeddings** en choisissant un mot aléatoirement à partir d'une langue source puis le faire traduire vers une langue cible, les méthodes basées sur les corpus pseudo-multilingues apprennent simultanément ces corpus, ensuite ils créent un espace vectoriel commun de plusieurs langues où les **embeddings** sont déjà des **embeddings cross-lingue**. [7]

## 1.8 Conclusion

Ce chapitre a été dédié à la définition du phénomène de plagiat et les méthodes de détection qui se répartissent en deux familles : la détection du plagiat extrinsèque et la détection du plagiat intrinsèque, avec la création d'un tableau qui compare entre eux, suivi par une citation de quelques mesures de similarité. À la fin nous avons cité les différentes méthodes de la détection du plagiat cross-lingue qui se basent généralement sur la comparaison entre deux langues ou plus par des approches différentes, mais ce que nous intéressons c'est l'approche de détection de plagiat **cross-lingue** en utilisant le **word embeddings cross-lingue** vu précédemment avec une nouvelle approche qui est le deep-learning. Pour le prochain chapitre, nous allons voir c'est quoi le machine learning et comment fonctionne l'apprentissage automatique (**deep learning**) afin de pouvoir l'adapter dans le projet de détection de plagiat **cross-lingue**.

# Chapitre 2

## Apprentissage profond

### 2.1 Introduction

L'intelligence artificielle (**IA**) a pour but de créer des entités intelligentes. Dans ce sens, il y a une poursuite de cette capacité dont l'humain est maître, à résoudre des problèmes. "La capacité à résoudre des problèmes" est d'ailleurs une définition de l'intelligence que l'on retrouve régulièrement. Il est important de noter qu'il y a une différence forte entre l'**IA** et l'apprentissage automatique (machine learning). L'apprentissage automatique est une forme d'**IA** qui consiste en un système qui s'améliore grâce à l'expérience alors que l'**IA** peut être un simple ensemble de règles et d'heuristiques.

En nous basant sur le **deep learning** et ses modèles, nous devons concevoir et réaliser un système qui analyse et détecte la similarité textuelle entre les langues. Avant de commencer le travail, une étude sur l'apprentissage profond est nécessaire. Dans ce troisième chapitre, nous allons d'abord introduire la notion de la machine learning avant de passer au **deep learning** avec la présentation de différentes architectures des réseaux de neurones de chaque modèle comme **LSTM**, **GRU** et **BERT**.

### 2.2 Apprentissage automatique

#### 2.2.1 Définition

L'apprentissage automatique (machine-learning en anglais) est une discipline scientifique qui est aussi l'un des champs d'étude de l'intelligence artificielle. Il fait référence au développement, à l'analyse et à l'implémentation de méthodes qui permettent à une machine d'évoluer grâce à un processus d'apprentissage et ainsi de remplir des tâches qui peuvent s'avérer difficiles ou impossibles à assurer par des moyens algorithmiques plus classiques. Des systèmes complexes peuvent être analysés y compris pour des données associées à des valeurs symboliques ou un ensemble de modalités possibles sur un attribut de valeur (numérique) ou catégoriel. L'analyse peut même concerner des données présentées sous forme de graphes ou d'arbres ou encore de courbes[40].

#### 2.2.2 Types d'algorithmes d'apprentissage

De manière générale, les algorithmes d'apprentissage automatique existent en plusieurs catégories selon le type d'apprentissage et on cite quelques types : apprentissage supervisé, apprentissage non supervisé.

##### 2.2.2.1 Apprentissage supervisé

Le but de l'apprentissage supervisé est d'entraîner et de trouver des corrélations entre les données d'entrées (variables explicatives) et sorties (variables à prédire) en utilisant une base de données d'apprentissage qui contient des exemples de cas réels traités et validés et par la suite, déduire la connaissance extraite sur des entrées avec des sorties inconnues. En apprentissage supervisé, on distingue deux types de tâches :

1. La classification : Quand la variable à prédire est discrète c'est-à-dire une variable qui ne peut prendre des valeurs que dans un ensemble d'éléments afin d'attribuer une classe ou

une étiquette pour chaque entrée. Exemple : **ID3** , Gini index (**IBM**), naïve bayes *..etc.*

2. La régression : Ce type est utilisé quand la variable à prédire est quantitative continue c'est-à-dire des valeurs réels  $\in \mathbf{R}$ . Exemple : prédire le prix futur en dollars de l'actif en question.

### 2.2.2.2 Apprentissage non supervisé

Ce type d'apprentissage permet de découvrir de façon autonome la structure en fonction des données c'est-à-dire que la base de données d'apprentissage ne contient pas de variable cible contrairement au apprentissage supervisé qui a seulement un ensemble de données collectées en entrée. Souvent, on y utilise la distance comme mesure de similarité entre les groupes. Ils existent plusieurs algorithmes pour ce type d'apprentissage tel que : clustering, réduction de dimensionnalité et la détection des anomalies.

### 2.2.3 Les problèmes rencontrés durant l'apprentissage

Les problèmes peuvent être rencontrés durant l'entraînement d'un système du machine learning sont apparus que ce soit avec le dataset utilisé pour l'entraînement ou le système d'apprentissage utilisé. Il s'agit de :

#### 2.2.3.1 Manque des données

La plupart des algorithmes d'apprentissage automatique exigent une énorme quantité de données afin d'obtenir de bons résultats même pour des problèmes très simples tels que la reconnaissance d'image ou de la parole.

#### 2.2.3.2 Données de mauvaise qualité

C'est essentiel d'avoir un bon dataset pour que le système d'apprentissage soit successivement bien entraîné. Mais lorsque le dataset est de mauvaise qualité, le système d'apprentissage fera de moins bonnes prédictions et dans certains cas, il sera inutilisable. On en déduira donc que les données sont de mauvaise qualité lorsque le dataset contient des erreurs, des valeurs absentes et des bruits (par exemple : trop d'outlayer ou beaucoup de vide dans les instances du dataset). Une solution disponible pour régler le dataset afin que le système d'apprentissage puisse l'apprendre et se familiariser avec relève de l'utilisation de l'une de ces méthode :

- Faire une correction sur les données qui semble bizarre ou qui ont des valeurs inhabituelles, si la correction ne peut pas se faire, alors il est favorable de les instances qui les contient.
- Ignorer les instances où l'une ou plusieurs valeurs sont inconnues (vides). On peut aussi se tourner vers une autre solution possible qui est de compléter ces valeurs par exemple la moyenne de son attribut ou la remplir avec une nouvelle classe considérée "non définie"

#### 2.2.3.3 Sur-apprentissage (Overfitting)

un modèle d'apprentissage automatique est dite au cours de sur-apprentissage lorsqu'il n'assure pas de bonnes prédictions sur les données de test. Parmi les causes de sur-apprentissage

on trouve : l'utilisation d'un dataset qui contient un nombre énorme d'instances de données contenant à leurs tour des bruits (outlayer) et l'utilisation des méthodes non paramétriques et non linéaires car ces types d'algorithmes peuvent construire des modèles inapplicables.

Afin d'éviter le sur-apprentissage, il faut fixer les paramètres du modèle d'apprentissage à des valeurs raisonnables par rapport au dataset pour ne pas compliquer le modèle.

### 2.2.3.4 Sous-apprentissage (Underfitting)

un modèle d'apprentissage automatique est dit au cours de sous-apprentissage lorsqu'il ne peut pas trouver une corrélation entre les données : c'est-à-dire il fonctionne mal sur les données de test malgré son bon fonctionnement sur les données d'apprentissage. Par conséquent, le sous-apprentissage se produit lorsque le dataset ne contient pas assez d'instances de données. Afin de l'éviter, il est préférable d'utiliser plus de données car le modèle entraîné sur un dataset léger produira des règles d'apprentissage simples et insuffisantes pour faire la prédiction.

## 2.3 Apprentissage profond (Deep learning)

### 2.3.1 Définition

Le Deep learning ou apprentissage profond est l'une des technologies principales du Machine learning. Le Deep Learning est dit profond car il se base plus spécifiquement sur la notion de réseaux des neurones artificiels. C'est un ensemble d'algorithmes capables de simuler les actions du cerveau humain grâce à des réseaux de neurones artificiels. Les réseaux sont composés de centaines de nombres d'ensembles d'unités d'exécution d'information (représentant les neurones) superposées en couches et liés entre eux via des connecteurs (les synapses), chacune recevra et interprétera les informations de la couche précédente[23].

## 2.4 Structure des réseaux des neurones

Le réseau de neurones contient un ensemble de neurones appelés "Perceptrons". Ces perceptrons sont liés entre eux grâce à des connecteurs. Ils envoient des information entre chaque couches afin de faire un apprentissage sur le dataset d'entrée, l'architecture du réseaux des neurones et du perceptron se présentent comme suit :

### 2.4.1 Architecture des réseaux des neurones

Le réseau des neurones est un système informatique composé de nombreuses unités interconnectées appelées neurones artificiels. Ceux-ci sont inspirés du système nerveux de l'être humain. La connexion entre les neurones artificiels peut transmettre un signal d'un neurone à un autre. Ainsi, il existe de multiples possibilités pour les connecter en fonction de chaque architecture donnée. Dans chaque architecture du réseau des neurones, les neurones sont organisés dans des couches avec la présence obligatoire de deux couche : une couche d'entrée au une de sortie. Le nombre de couches intermédiaires peut varier de zéro à infini (théoriquement) mais le nombre maximal des couches est limité suivant et dépendant de la taille de la mémoire de la machine qui l'exécute[43].

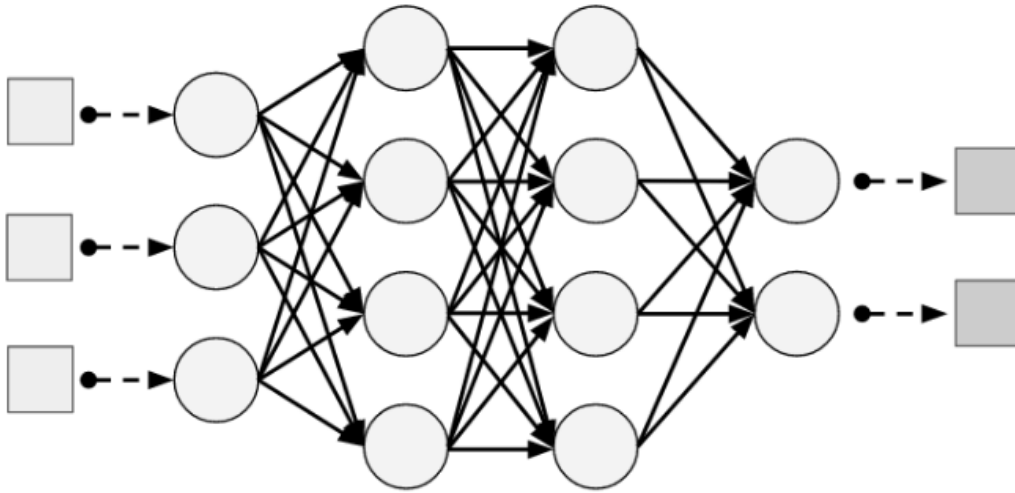


FIG. 2.1 : Exemple d'une architecture de réseaux des neurones (Feed-Forward)

### 2.4.2 Architecture du perceptron

Dans l'architecture du réseau des neurones, un perceptron reçoit un ou plusieurs signaux à partir de ces prédécesseurs qui lui sont connectés et chaque signal reçu est associé à un poids  $W_i$ . Le perceptron déclenche et envoie son signal à ses successeurs en utilisant sa fonction d'activation  $f(x)$ .

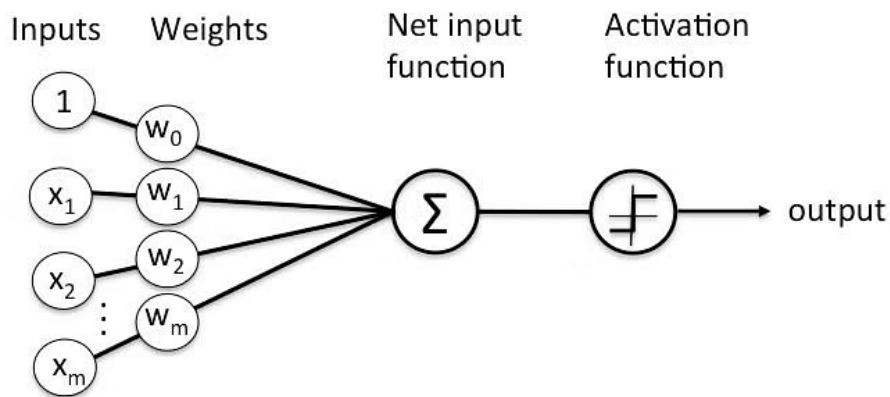


FIG. 2.2 : Exemple d'une architecture d'un perceptron

Le signal sortant du perceptron (Output) est calculé de manière général avec cette formule :

$$output = f(W_0 + \sum_{i=1}^m W_i * x_i) \quad (2.1)$$



## 2.5 Les fonctions utilisées en réseaux des neurones

### 2.5.1 La fonction d'activation

Une fonction d'activation est appliquée après que le neurone ait effectué le produit entre ses entrées et ses poids. C'est une fonction non linéaire, elle permet de décider si le neurone est activé ou non par le calcul de la somme pondérée des entrées et ajoute le biais. Après la transformation, cette sortie est envoyée à la couche suivante. La fonction d'activation c'est une chose très importante dans les réseaux de neurones et sans elle, un réseau de neurones devenait un simple modèle linéaire. Il existe plusieurs types de ces fonctions d'activation : Sigmoid, Tanh, ReLu et Softmax.

### 2.5.2 La fonction coût

Le but de cette fonction est de mesurer l'erreur commise par le modèle lors de la prédiction sur les données d'apprentissage. Plus précisément, elle calcule la différence entre les résultats prédits par le modèle et les résultats réels.

### 2.5.3 Les fonctions d'optimisations

#### 2.5.3.1 Décence du gradient (GD)

La décente du gradient est l'algorithme d'optimisation le plus basique qui est énormément utilisé dans la régression linéaire et dans la classification. La décente du gradient est un optimiseur de premier ordre qui dépend de la première dérivée de la fonction coût. Cet algorithme permet de trouver l'optimum global mais son inconvénient réside dans la perte de temps par rapport aux autres fonctions d'optimisations. De plus que la fonction puisse diverger si le taux d'apprentissage  $\alpha$  n'est pas bien choisi.

#### 2.5.3.2 ADAM

**ADAM** est un optimiseur efficace lors du traitement d'un lourd dataset qui contient un grand nombre d'instances ou d'attributs. La méthode est une combinaison des deux algorithmes **SGD** et le momentum. Elle hérite les points forts des deux méthodes comme l'utilisation des estimations du premier et second moment du gradient afin d'adapter le taux d'apprentissage[32][33].

### 2.5.4 La propagation arrière (Back propagation)

La propagation arrière est l'algorithme le plus important en réseau de neurones. C'est la base de son entraînement réseau et elle permet de régler les poids afin de réduire les taux d'erreur et rendre le modèle robuste en augmentant sa généralisation.

## 2.6 Les modèles de deep learning

### 2.6.1 Les réseaux de neurones convolutifs

Les CNNs sont largement utilisés dans la reconnaissance et classification des images et des objets. Ils jouent un grand rôle dans diverses tâches comme les traitements d'images, visions par ordinateur, analyse des vidéos ou la reconnaissance vocale dans le traitement du langage naturel.

L'algorithme CNN prend en entrée une image apprenable et qu'on peut différencier d'une autre. Le pré-traitement est nécessaire pour réduire la dimension de l'image. L'algorithme applique d'abord une série des filtres de convolution (qui fait apparaître les caractéristique de l'image) ce qui donne en sortie un ensemble d'images filtrées, ensuite cet ensemble est rectifié en utilisant la fonction **ReLU** qui a la possibilité de rendre les valeurs négatives à zéro pour permettre de faire un apprentissage plus rapide et plus efficace. Finalement ce nouveau ensemble est suivi par une simplification (réduction du dimensionalité) jusqu'à obtenir un ensemble d'images filtrées de taille réduite, on refait ce processus jusqu'à obtenir un ensemble d'images désiré. A la fin du pré-traitement, les images passent au réseau des neurones **feed-forward** pour assurer l'apprentissage afin de faire le classement sur l'objet qui se trouve dans l'image[15].

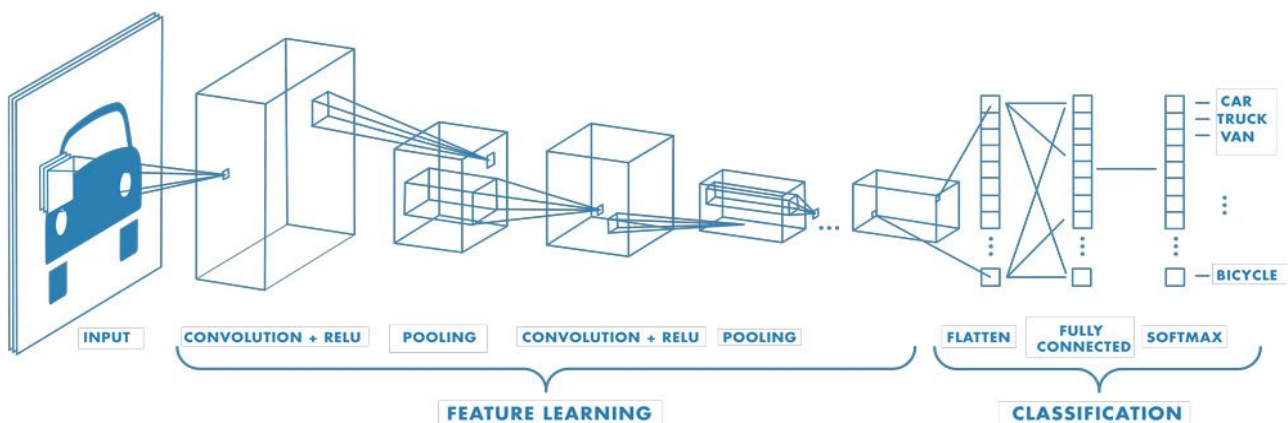


FIG. 2.3 : Architecture d'un réseau CNN [15]

### 2.6.2 Les réseaux de neurones récurrents

#### 2.6.2.1 Définition

Un RNN est un réseau des neurones très répandu en deep learning. Ressemblant grandement au réseau de neurones artificiels. Ils sont conçus de manière à reconnaître les caractéristiques séquentielles et les modèles d'utilisation des données requises pour prédire le scénario suivant le plus probable[14].

Les RNNs se distinguent des autres types de réseaux de neurones artificiels dans le sens qu'ils utilisent des neurones acycliques qui contiennent des boucles de rétroaction pour traiter une séquence de données. Ces boucles de rétroaction permettent aux informations de persister. Les RNNs les utilisent dans des modèles linguistiques visant à identifier la prochaine lettre d'un mot ou le prochain mot d'une phrase d'après les données qui les précèdent[14].

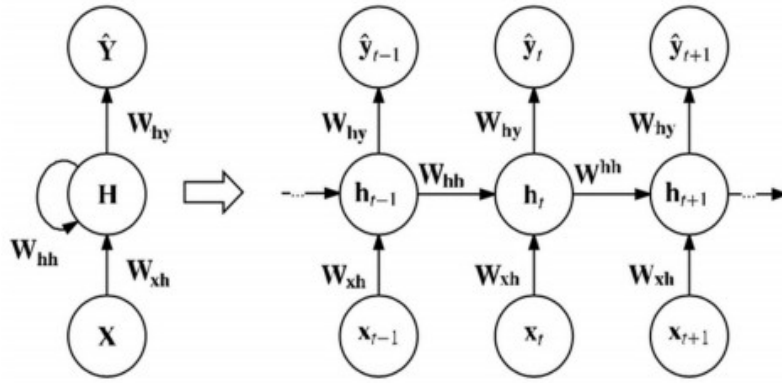


FIG. 2.4 : Architecture d'un réseau RNN Basic

Les **RNNs** sont particulièrement utiles pour les tâches qui contiennent des séquences. Par exemple, une traduction automatique où les séquences sont des phrases ou des mots. Les modèles **LSTMs** sont utilisés beaucoup plus contrairement aux **RNNs** standards car les réseaux **RNNs** souffrent d'un problème qui se produit lors de l'utilisation de longues séquences. Il est donc très compliqué pour un RNN basique de générer un long texte cohérent.

### 2.6.2.2 LSTM

Comme expliqué précédemment, la cellule récurrente utilise le résultat de l'état précédent pendant le calcul de la sortie de l'état actuel mais dans certains cas il est nécessaire d'avoir des informations provenant des états précédents qui sont loins dans la séquence. Les cellules **LSTM** appartiennent à une architecture qui a été proposée comme étant une solution à ce problème. C'est une cellule composée de trois « portes » : ce sont des zones de calculs qui contrôlent le flot d'informations. On a également deux types de sorties (nommées états). En effet, les données qui sont stockées dans la mémoire du réseau sont en fait un vecteur noté  $c_t$  et les entrées de chaque porte sont pondérées par les poids liés aux portes et un biais avec une taille qui dépend des dimensions des vecteurs  $h_{t-1}$  et  $x_t$ [37].

#### ❖ Forget gate (porte d'oubli) :

Cette porte a pour but de garder ou débarrasser l'information de l'état caché précédent, elle utilise la fonction sigmoïde pour normaliser les valeurs entre 0 et 1. L'information est oubliée si la valeur de sigmoïde est proche de 0 et si elle est proche de 1 alors il faut la garder pour la suite[37].

#### ❖ Input gate (porte d'entrée) :

Elle a un rôle de retirer l'information de la donnée courante, afin de contrôler la quantité d'information qui doit être ajoutée à la mémoire à long terme. En parallèle, elle applique une sigmoïde aux données concaténées pour voir si cette dernière est importante (si renvoie une valeur proche de 0) ou non (si la valeur est proche de 1) et une tanh pour normaliser les valeurs entre -1 et 1. Le résultat de ces deux dernières fonctions sont multipliés, c'est-à-dire les informations inutiles sont mises à 0[37].

#### ❖ Cell state (état de la cellule) :

L'état de cellule est très importante pour la porte de sortie. Elle est calculée par une multiplication de la sortie d'oubli avec l'ancien état de la cellule pour oublier quelques

informations de l'état précédent qui n'ont pas un effet pour la nouvelle prédiction. Par la suite, on additionne le résultat précédent avec la sortie de la porte d'entrée afin de garder l'état de la cellule LSTM[37].

❖ **Output gate (porte de sortie) :**

Cette porte doit décider quelle est la quantité d'information stockée dans la mémoire à long terme qui sera considérée dans le calcul de la sortie de l'état courant, grâce à un calcul qui est fait par la fonction tanh et sigmoïde[37].

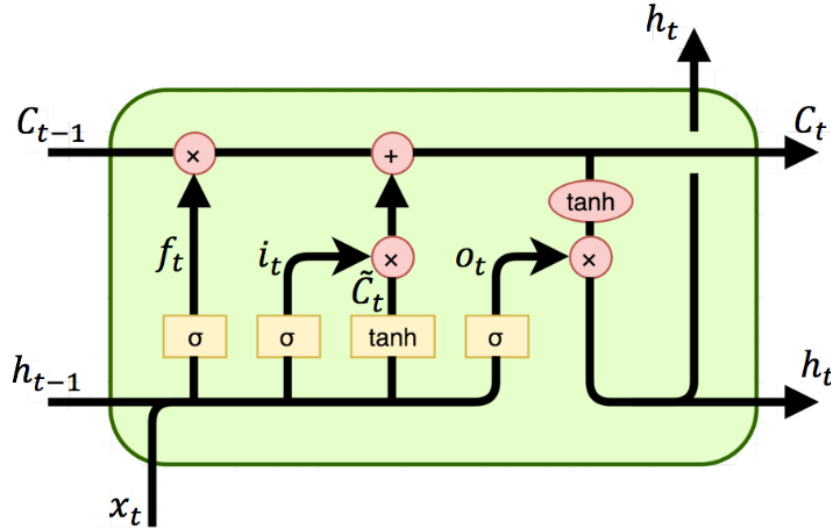


FIG. 2.5 : Architecture d'une cellule LSTM [22]

Avec la cellule a une entrée normale " $x_t$ " qui vient des cellules des couches inférieures à l'état actuel et une autre entrée récurrente qui vient de l'état précédent " $h_{t-1}$ ".

$$\begin{aligned}
 \text{Input gate :} \quad & i_t = \sigma(W^{(ii)}\bar{x}_t + W^{(hi)}h_{t-1}) \\
 \text{Forget gate :} \quad & f_t = \sigma(W^{(if)}\bar{x}_t + W^{(hf)}h_{t-1}) \\
 \text{Output gate :} \quad & o_t = \sigma(W^{(io)}\bar{x}_t + W^{(ho)}h_{t-1}) \\
 \text{Process input :} \quad & \tilde{C}_t = \tanh(W^{(i\tilde{c})}\bar{x}_t + W^{(h\tilde{c})}h_{t-1}) \\
 \text{Cell update :} \quad & C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \\
 \text{Output :} \quad & y_t = h_t = o_t * \tanh(C_t)
 \end{aligned} \tag{2.2}$$

### 2.6.2.3 GRU

**GRU** est une autre type de cellule récurrente, et une forme plus simplifiée des cellules **LSTM**, ou les deux vecteurs  $c_t$  et  $h_t$  ont été combinés dans un seul vecteur " $h_t$ ", elle dispose deux portes et un état en sortie. Les calculs opérés par le **GRU** sont plus rapides et plus simples[37].

❖ **Reset gate (porte de reset) :**

Cette port Permet de conserver que les informations importantes de l'état précédent. L'état caché précédent, concaténé avec les données d'entrée, passe par une fonction sigmoïde, puis est multiplié par l'ancien état caché[37].

❖ **Update gate (porte de mise à jour) :**

Cette porte agit exactement de la même manière que les portes d'oublis et d'entrées du LSTM : elle décide les informations à conserver et de celles à oublier, en passant ses informations (les données d'entrées et l'ancienne) par une fonction sigmoïde. C'est-à-dire elle permet de contrôler dans quelle mesure le nouvel état n'est qu'une copie de l'ancien état[37].

❖ **Cell state (état de la cellule) :**

Après avoir éliminé toutes les informations inutiles pour la prédiction grâce à la porte de mise à jour, par la suite ces informations sont normalisé par une tanh entre -1 et 1, enfin on y ajoute les coordonnées de l'état caché précédent inutiles pour avoir cette fois toutes les coordonnées pertinentes[37].

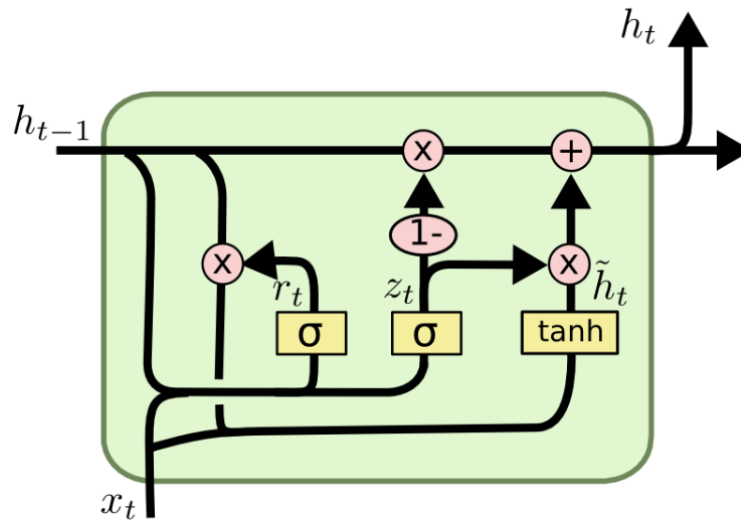


FIG. 2.6 : Architecture d'une cellule GRU [22]

Avec la cellule a une entrée normale " $x_t$ " qui vient des cellules des couches inférieures à l'état actuel et une autre entrée récurrente qui vient de l'état précédent " $h_{t-1}$ ".

$$\begin{aligned}
 \text{Reset gate :} & \quad r_t = \sigma(W^{(ir)}\bar{x}_t + W^{(hr)}h_{t-1}) \\
 \text{Update gate :} & \quad z_t = \sigma(W^{(iz)}\bar{x}_t + W^{(hz)}h_{t-1}) \\
 \text{Process gate :} & \quad \tilde{h}_t = \tanh(W^{(ih)}\bar{x}_t + W^{(hh)}h_{t-1}) \\
 \text{Hidden state update :} & \quad h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \\
 \text{Output :} & \quad y_t = h_t
 \end{aligned} \tag{2.3}$$

#### 2.6.2.4 bidirectionnel LSTM/GRU (Bi-LSTM / Bi-GRU)

Généralement, dans les réseaux LSTM et GRU normaux, nous relevons la sortie directement mais dans les réseaux LSTM bidirectionnel et GRU bidirectionnel, on entraîne deux LSTM ou deux GRU ou GRU et LSTM en même temps au lieu d'un seul sur la séquence d'entrée. Le premier sur la séquence d'entrée reste telle qu'elle est et le second sera sur une copie inversée de la séquence d'entrée c'est-à-dire que la sortie de la couche avant et arrière à chaque étape est donnée à la couche d'activation qui est un réseau neuronal et la sortie de cette couche

d'activation est prise en compte. Cette sortie contient également les informations sur la relation entre le mot passé et le futur[29].

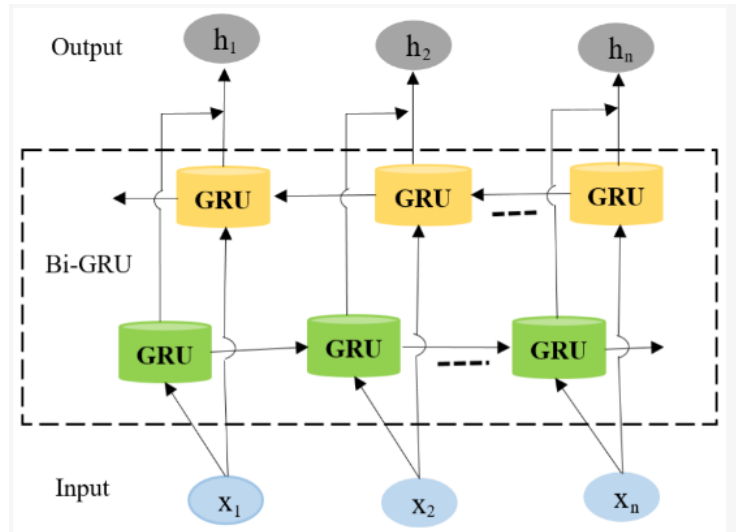


FIG. 2.7 : Architecture bidirectionnel avec des cellules GRU(Bi-GRU). [17]

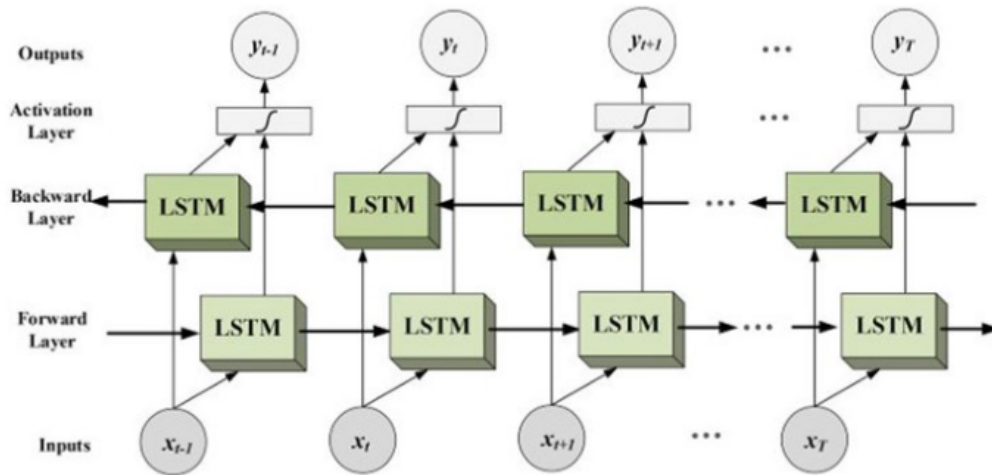


FIG. 2.8 : Architecture bidirectionnel avec des cellules LSTM(Bi-LSTM). [44]

### 2.6.2.5 Encodeur-Décodeur

Le modèle encodeur-décodeur est une utilisation du réseau **RNN** pour faire des prédictions séquence-à-séquence. Il est initialement développé pour les problèmes de traduction automatique bien qu'il ait prouvé son succès dans des tâches de prédictions comme la création des résumés de textes et l'action de répondre aux questions[14].

Cette approche nécessite deux réseaux de neurones récurrents (Ils peuvent être soit des **RNNs**, **LSTMs** ou **GRUs**), une pour encoder la séquence d'entrée (input) et l'autre pour décoder la séquence encodée.

- ❖ **Encoder** : C'est un réseau de neurones récursif qui prend en entrée une séquence de mots où chaque mot est représenté sous forme de vecteur ayant une taille fixe. Celui-ci est appelé "le vecteur d'encodage"[14].

- ❖ **Décoder** : c'est un réseau de neurones récurrent qui prend en entrée le vecteur d'encodage créé précédemment et le transforme en une autre séquence des mots[14].

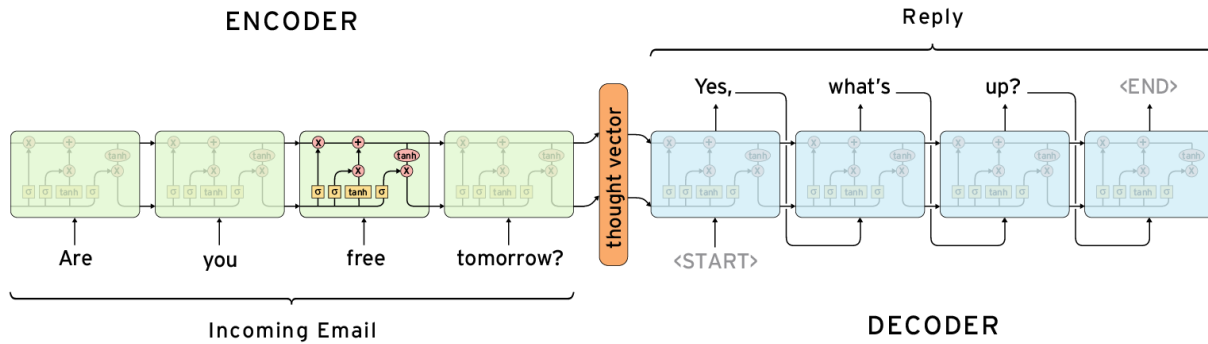


FIG. 2.9 : Architecture du modèle Encoder-Décodeur avec utilisation des cellules LSTM[16]

### 2.6.3 Le modèle transformer

Une nouvelle architecture présentée par **Google Brain** est une variante du modèle Encoder-Décodeur qui n'utilise pas les réseaux de neurones récurrent dans les deux parties "encoder et decoder" mais elle se sert plutôt d'une nouvelle structure qui est le mécanisme *self-attention*.

Ce mécanisme fournit un contexte pour n'importe quelle position dans la séquence d'entrée. C'est à dire qu'il peut traiter plus qu'un seul mot à la fois dans la séquence d'entrée. Ce processus permet le traitement en parallélisme de la séquence contrairement au **RNN**, cela permet d'assurer l'entraînement sur de grands datasets et fait gagner en temps dans l'apprentissage du modèle. Ce modèle contient quatre composantes principales[8].

- ❖ **Encoder** : Il prend en entrée la séquence de mots. Un encodeur contient deux sous-composantes, un *self-attention* et un réseau basique. Le modèle transformer contient six boîtes Encodeurs
- ❖ **Décodeur** : Il contient les mêmes sous-composantes de l'encodeur et en plus il a une troisième composante appelée *Encoder-Décodeur self-attention* qui fait attention à la sortie du décodeur.
- ❖ **Couche linéaire finale** : La sortie du décodeur est un vecteur des valeurs flottantes. Cette couche transforme ce vecteur en vecteur de probabilités pour les termes (mots) grâce à une transformation linéaire. La couche linéaire finale fait ce travail en utilisant à un réseau de neurones *feed-forward*.
- ❖ **Couche Softmax** : La couche Softmax transforme ces scores vers des probabilités normalisées où leur somme est égale à un pour ensuite choisir l'indice de la plus grande probabilité et le mot associé avec cet indice est choisi et envoyé à l'output (la sortie).

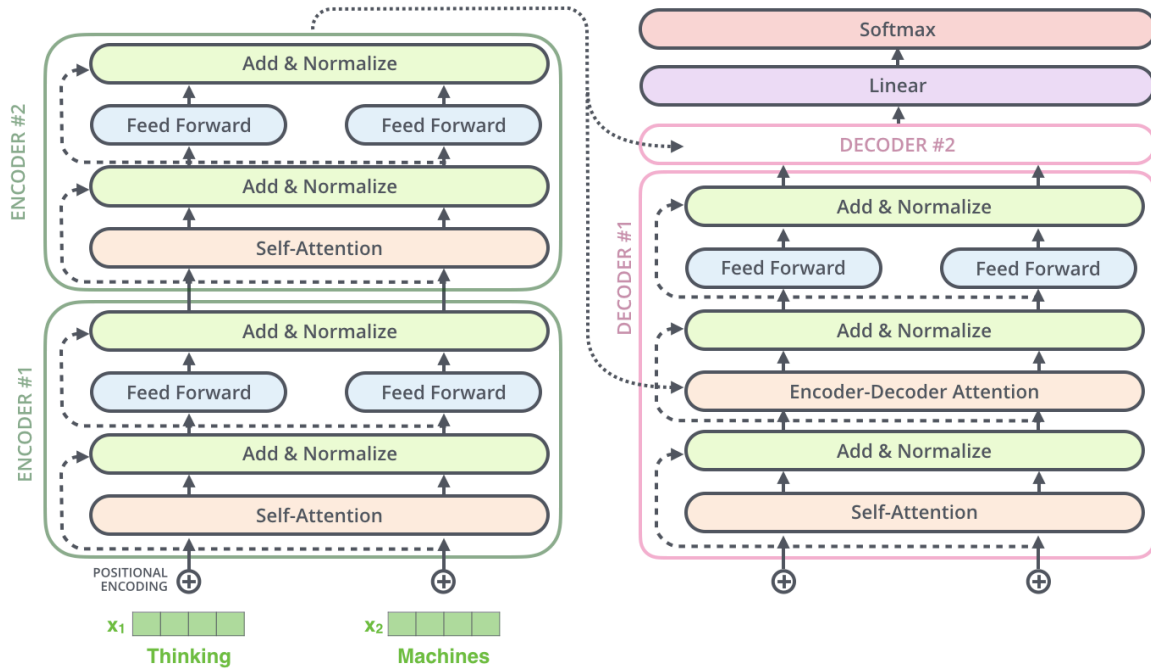


FIG. 2.10 : Architecture du modèle transformer avec nombre des cellules égale à deux[18]

## 2.6.4 Le modèle BERT

**BERT** signifie Bidirectional Encoder Representations from Transformers, est un modèle d'apprentissage profond basé sur Transformers proposé en 2018 par Google et qui permet de générer des représentations des textes écrits en langage naturel sous forme vectorielle[14].

Les modèles de langage ne pouvaient lire la saisie de texte que de manière séquentielle mais **BERT** est différent car il a la capacité de lire dans les deux sens à la fois ce qui est connu sous le nom de bidirectionnalité.

BERT est contextuelle, c'est-à-dire un mot n'est pas représenté de façon statique comme dans un modèle de plongements des mots classiques mais cette représentation change en fonction du sens du mot dans le contexte du texte[14].

**BERT** est actuellement utilisé chez Google pour optimiser l'interprétation des requêtes de recherche des utilisateurs. Il assure plusieurs fonctions telles que : la réponse aux questions, prédiction des phrases et la classification des sentiments *etc...* .

### 2.6.4.1 Architecture de BERT

**BERT** réutilise l'architecture des Transformers mais le modèle **BERT** ne contient que la partie de l'encodeur du Transformer. Il prend une séquence de mots en entrée qui remonte à la pile. Chaque couche applique l'auto-attention et transmet ses résultats à un réseau **feed-forward** puis les transmet à l'encodeur suivant[14].



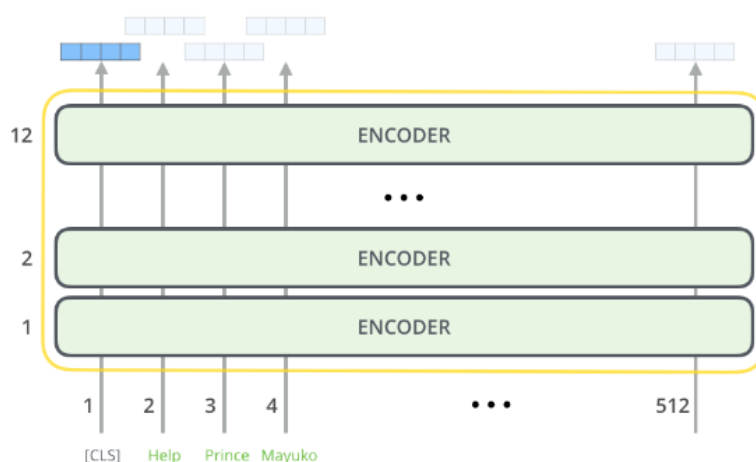


FIG. 2.11 : Architecture du modèle BERT-Base[21]

La version représentée ci-dessus est la version dite « Base » de **BERT**. Elle contient 12 encodeurs superposés. Il y a une autre version plus grande dite « Large » qui a 24 encodeurs, bien sur cette version est la plus performante par contre, elle exige trop en ressource machine.

#### 2.6.4.2 Codage d'input

Le modèle BERT accepte une entrée qui a une structure différente aux autres modèles, cette structure d'entrée est un vecteur d'**embeddings** composé de deux phrases, le modèle BERT utilise ces deux phrases pour qu'il puisse apprendre à prédire si la deuxième phrase de la paire est la phrase suivante dans le document d'origine.

Le modèle BERT fait tokeniser les deux phrases A et B, B comme étant la phrase suivante de A. Afin de savoir le repère de début, un token **[CLS]** est inséré au début de la première phrase A. Un autre token **[SEP]** est inséré en milieu des deux phrases pour les séparer, et il est aussi placé à la fin de la phrase B [28]. Par la suite, les tokens sont transformés en **embeddings**, chaque mot **embeddings** est concaténé avec autre **embeddings** qui contient le numéro de sa phrase, le résultat est concaténé aussi avec un autre **embeddings** qui contient la position de chaque mot **embeddings** dans sa phrase, à la fin on aura une structure d'**embeddings** composé qui peut être représenté dans la figure 2.12.

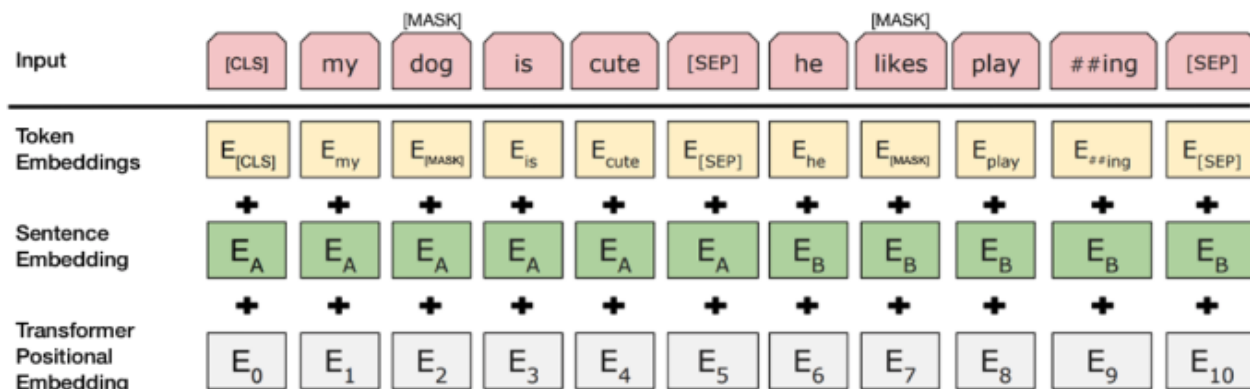


FIG. 2.12 : Architecture du vecteur d'entrée de BERT[28]

### ☆ embeddings des tokens :

Ce sont les **embeddings** que représentent les mots eux-mêmes tokenisé

### ☆ embeddings de appartenance :

Ils représentent à quelle phrase le mot lui appartient.

### ☆ embeddings de position :

Ils permettent savoir quelle est la position du mot **embeddings** dans la phrase.

les deux derniers **embeddings** concaténés aident le modèle **BERT** à savoir la position de chaque mot dans la phrase. Grâce à ça, il peut connaître le contexte de chaque mot où chaque contexte peut être différent selon l'endroit du mot dans la phrase.

### 2.6.4.3 L'entraînement de BERT

l'entraînement d'un modèle ayant une structure complexe comme **BERT** nécessite une grande quantité de données, sachant que pour certaines tâches supervisées, la construction d'un tel dataset est très difficile, pour cela le modèle **BERT** est procéder en deux phases :

#### ❖ Pre-Training :

Premièrement, on effectue un pré-entraînement de l'algorithme en apprenant à représenter le texte à l'aide d'une tâche non-supervisée. Cette étape contient deux tâches de pré-entraînement [19] :

##### – la détection du mot masqué :

Dans cette partie le modèle est entraîné pour détecter le mot masqué, (15% des mots sont masqués aléatoirement). L'algorithme masque un mot en le remplace par le Token [MASK] avec 80% des cas, sinon par un autre mot aléatoirement dans 10% des cas sinon il garde le mot tel qu'il est dans 10% des cas.

##### – la détection de la prochaine phrase :

Dans cette tâche, **BERT** apprend à reconnaître si deux phrases sont consécutives ou non.

#### ❖ Fine-Tuning :

Deuxièmement, dans cette phase l'algorithme est spécialisé pour une tâche précise qui est la phase de spécialisation du modèle, **BERT** est entraîné pour une tâche supervisée spécifique sur un dataset. Cette phase n'exige pas une très grande quantité de données car le modèle est déjà pré-entraîné.

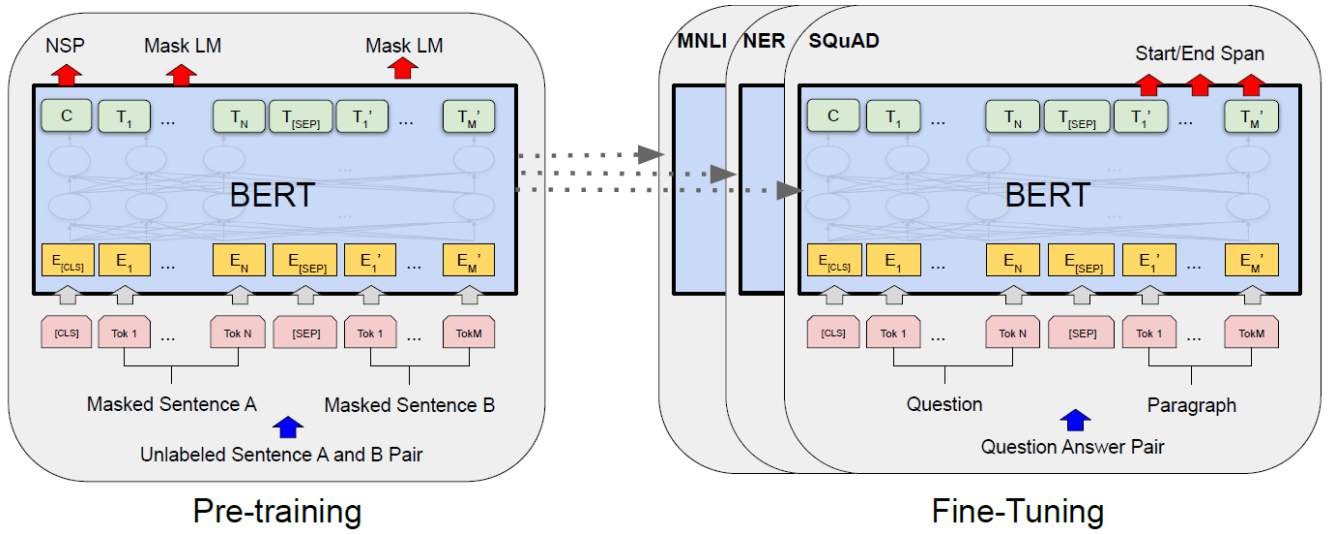


FIG. 2.13 : l'entraînement de BERT [38]

## 2.7 Évaluation du modèle

### 2.7.1 Les mesures d'évaluation

Il existe plusieurs mesures d'évaluation dans la littérature. Ces mesures sont basées sur rappel et précisions avec une matrice de confusion pour chaque classe qui fournit quatre informations essentielles :

#### 2.7.1.1 Table confusion

La matrice de confusion est l'une des mesures les plus intuitives et des plus simples à utiliser pour trouver l'exactitude et la précision du modèle. Elle est utilisée pour le problème de classification où la sortie peut être deux ou plusieurs classes. Une matrice de confusion n'est rien d'autre qu'un tableau à deux dimensions où les valeurs "Réelles" sont en colonnes et les valeurs "Prévues" en lignes. En outre, les cases représentent "Vrais Positifs (TP)", "Vrais Négatifs (TN)", "Faux Positifs (FP)", "Faux Négatifs (FN)" comme indiqué ci-dessous :

		Réal	
		<i>True(1)</i>	<i>False(0)</i>
Prévue	<i>True(1)</i>	True Positive (TP)	False Positive (FP)
	<i>False(0)</i>	False Negative (FN)	True Negative (TN)

TAB. 2.1 : Structure de la table de confusion

#### 2.7.1.2 Le taux de succès(TS)

Le **TS** appelé **Accuracy** en anglais, il permet de calculer l'exactitude de chaque algorithme, elle représente la proportion des instances qui sont correctement classifiées.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \quad (2.4)$$

### 2.7.1.3 Le taux d'erreur(TE)

le TE représente le pourcentage des instances incorrectement classées par le système c'est à dire l'erreur globale de la classification. Il existe deux formules qui permettent de calculer le taux d'erreur

$$\mathbf{TE} = \frac{TP + TN}{TP + FP + FN + TN} \quad (2.5)$$

$$\mathbf{TE} = 1 - Accuracy \quad (2.6)$$

### 2.7.1.4 Rappel (Recall)

C'est le nombre de vrais positifs sur le nombre d'instances positives (vrai positive + faux négative), il représente le nombre d'instances positives que l'algorithme a réussi à bien classer.

$$R = \frac{\mathbf{TP}}{\mathbf{FP} + \mathbf{TN}} \quad (2.7)$$

### 2.7.1.5 Précision

C'est le nombre de vrais positifs sur le nombre total de prédictions positives (vrai positive + faux positive). Il représente le nombre d'instances positives que l'algorithme a réussi à bien classer.

$$P = \frac{\mathbf{TP}}{\mathbf{TP} + \mathbf{FN}} \quad (2.8)$$

### 2.7.1.6 F1-Score

Il s'agit de la moyenne harmonique de la précision et le rappel. Il prend des valeurs entre 0 et 1. Cette mesure est utilisée quand on cherche une balance entre la précision et le rappel. Plus la valeur de cette mesure s'approche de 1, meilleure sera la décision prise par notre modèle et le contraire est vrai, quand elle s'approche de zéro.

$$F1 = \frac{2(\mathbf{P} * \mathbf{R})}{\mathbf{P} + \mathbf{R}} \quad (2.9)$$

## 2.8 Conclusion

Dans ce troisième chapitre, nous avons persisté dans notre étude en présentant la notion de machine learning et ses types d'apprentissage automatique qui existent avec. Puis, dans la deuxième partie, nous avons présenté l'apprentissage profond et ses modèles de réseaux des neurones avec leurs architectures et à la fin nous avons parlé des mesures d'évaluations du modèle.

Dans le prochain chapitre, nous allons exposer quelques travaux de réalisation de la détection du plagiat **cross-lingue** basés sur les différents modèles de deep learning avec une comparaison des résultats des différentes architectures pour chaque modèle.

## Chapitre 3

### Conception et Réalisation

## 3.1 Introduction

Dans ce chapitre, nous allons nous intéresser aux différents choix techniques que nous avons fait en termes de bibliothèques, langages et outils choisis. Puis, nous présenterons l'architecture technique du système ainsi que les expérimentations et résultats d'évaluation.

## 3.2 Utilisation d'Arabic-English Cross-Lingual Word Embedding (ArbEngVec)

### 3.2.1 Définition

Nous avons utilisé ce modèle pour faire la codification word embeddings dans notre projet qui s'appelle Arabic-English Cross-Lingual Word Embedding ou **ArbEngVec**, L'ArbEngVec est un projet open-source qui fournit plusieurs modèles de word embeddings multilingues Arabe-Anglais. Pour former ces modèles bilingues, les développeurs d'**ArbEngVec** ont utilisé un grand ensemble des datasets qui contiennent plus de 93 millions de paires de phrases parallèles Arabe-Anglais. Les datasets utilisés pour créer ces différents modèles sont extraits à partir de *Open Parallel Corpus Project* ou **OPUS**. Ce corpus dataset contient 2.9 milliards de phrases parallèles pour plus de 90 langues. Ce corpus se compose de données provenant de plusieurs sources, comme : [10]

*MultiUN Corpus, OpenSubtitles, Tanzil, NewsCommentary, United Nations (UN) , Wikipedia, TED, GNOME, Tatoeba, Global Voices, KDE4 et Ubuntu corpus.*

### 3.2.2 Pré-traitement et normalisation

Les pré-traitements appliqués sur le dataset utilisé par **ArbEngVec** consiste à éliminer les signes de ponctuation, les liens URLs, les emojis et les émoticônes sur les deux langues Arabe et Anglais. La normalisation est appliquée surtout sur la langue Arabe, c'est une standardisation des lettres : les lettres أ , إ , آ sont remplacées par ا et ة par ه , ou une reconversion des mots allongés vers leurs forme d'origine : ورووووووود est remplacé par ورود , ensuite la suppression des mots vides dans les deux langues est nécessaire.[10]

### 3.2.3 Les modèles de ArbEngVec

Le projet **ArbEngVec** utilise deux architectures SkipGram et CBOW (Défini dans chapitre 2) et pour chacune de ces architectures, l'apprentissage sur les datasets utilisés se diffère d'un modèle à un autre :[10]

☆ **Mode alignement parallèle (PA :*Parallel alignment*)** : Le premier modèle va prendre les deux phrases de chaque langue tel comme elle est :

$$\begin{aligned} S_{en} &= \text{"men going to cafe"} \\ S_{ar} &= \text{"لرجال يذهبون إلى المقهى"} \end{aligned} \quad (3.1)$$

Dans ce mode, la paire  $(S_{ar}, S_{en})$  est passée comme input de cette manière dans le modèle 1 comme étant : ("men going cafe" , "الرجال يذهبون إلى المقهى" )

- ☆ **Mode alignement mot par mot (WbW : *Word by Word*)** : Le second modèle prend chaque mot des phrases et il les arrange une par une, si on se base sur l'exemple précédent, la paire  $(S_{ar}, S_{en})$  est passée comme paramètre d'entrée sous la forme ("men", "الرجال", "going", "يذهبون", "going", "المقهى") pour entraîner le modèle 2.
- ☆ **Mode aléatoire (RS : *Random shuffle*)** : Le dernier modèle prend les deux phrases et place les mots qui forment celles-ci dans une liste puis il les mélange aléatoirement. La paire  $(S_{ar}, S_{en})$  est passée comme paramètre d'entrée ("المقهى", "يذهبون", "men", "الرجال", "going", "going").

Ces trois types de modèles sont utilisés pour chacune des architectures SkipGram et CBOW donc à la fin on aura six modèles de Word2Vec à utiliser dans notre projet : *SkipGram parallel alignement*, *skipGram word by word*, *skipgram random shuffle*, *CBOW parallel alignement*, *CBOW word by word* et *CBOW random shuffle*

### 3.2.4 L'espace commun multilingue

Dans **ArbEngVec**, chaque terme peut être soit écrit en arabe ou en anglais et est représenté par un vecteur de taille 300 (300 étant donné le nombre de dimensions) dans l'espace commun. Un vecteur est dit l'embeddings du terme. Dès la fin de l'apprentissage sur un corpus, l'**ArbEngVec** crée un dictionnaire de relation clé-valeur qui contient tout les termes des deux langues avec leurs embeddings (vecteurs) associés. Deux vecteurs sont comparables en utilisant la distance *Cosinus* puisqu'ils sont créés dans un espace commun du **ArbEngVec**, alors le calcul de similarité entre les termes cross-langue est possible grâce à ça.

En plus que les vecteurs créés sont tellement proches que les termes se ressemblent et sont même similaires mais n'ont pas nécessairement le même sens. Nous illustrons ça par l'exemple qui s'agit de trouver les cinq termes les plus proches du terme "Weapons" dans le modèle *skipgram random shuffle* qui sont ("أسلحة", "دمار", "war", "mass" et "indiscriminat)e". On remarque bien que "أسلحة" est une traduction de mot "weapons" alors que les termes "war", "mass" et "دمار" ne s'agissent pas d'une traduction mais ce sont plutôt des termes qui sont reliés au même sujet et thème. Ceci va nous permettre de faire une comparaison entre les textes de manière sémantique et pas seulement de les comparer de façon textuelle ou avec leur traduction.

### 3.2.5 Référence

Le projet **ArbEngVec** a été créé par Raki Lachraf, El Moatez Billah Nagoudi, Youcef Aya-chi, Ahmed Abdelali et Didier Schwab et c'est project open-source disponible à tout le monde, le projet contient les six modèles pré entraîné discuté précédemment et ils sont disponible aussi et ils ont été créé avec la bibliothèque Gensim avec le langage de programmation Python

## 3.3 Conception du système

Nous planifions que notre système sera constitué de deux unités, la première unité va tester les requêtes données par l'utilisateur vers nos modèles de deep learning prédéfinis et déjà entraînés. Ces modèles sont entraînés sur le dataset STS. Ce dernier va être nettoyé et transformé en embeddings pour pouvoir faire de l'apprentissage dans le premier lieu. L'utilisateur choisit

une requête de test, avec le modèle de deep learning souhaité, le système charge le modèle choisi puis il transforme la requête en embêtions en utilisant soit Word2Vec (pour LSTM ou GRU) ou bien Sentence2Vec (pour BERT). Enfin, modèle sélectionné essaye de prédire les embeddings résultats à partir des embeddings d'entrée. La mesure de similarité entre les embeddings sera utilisée pour comparer la distance entre les vecteurs. Nous comparons la requête avec l'ensemble du dataset pour trouver les  $n$  phrases les plus similaires à la requête de l'utilisateur. Il est à noter que le paramètre  $n$  est aussi choisi par l'utilisateur. La figure si dessous donne une idée sur cette architecture :

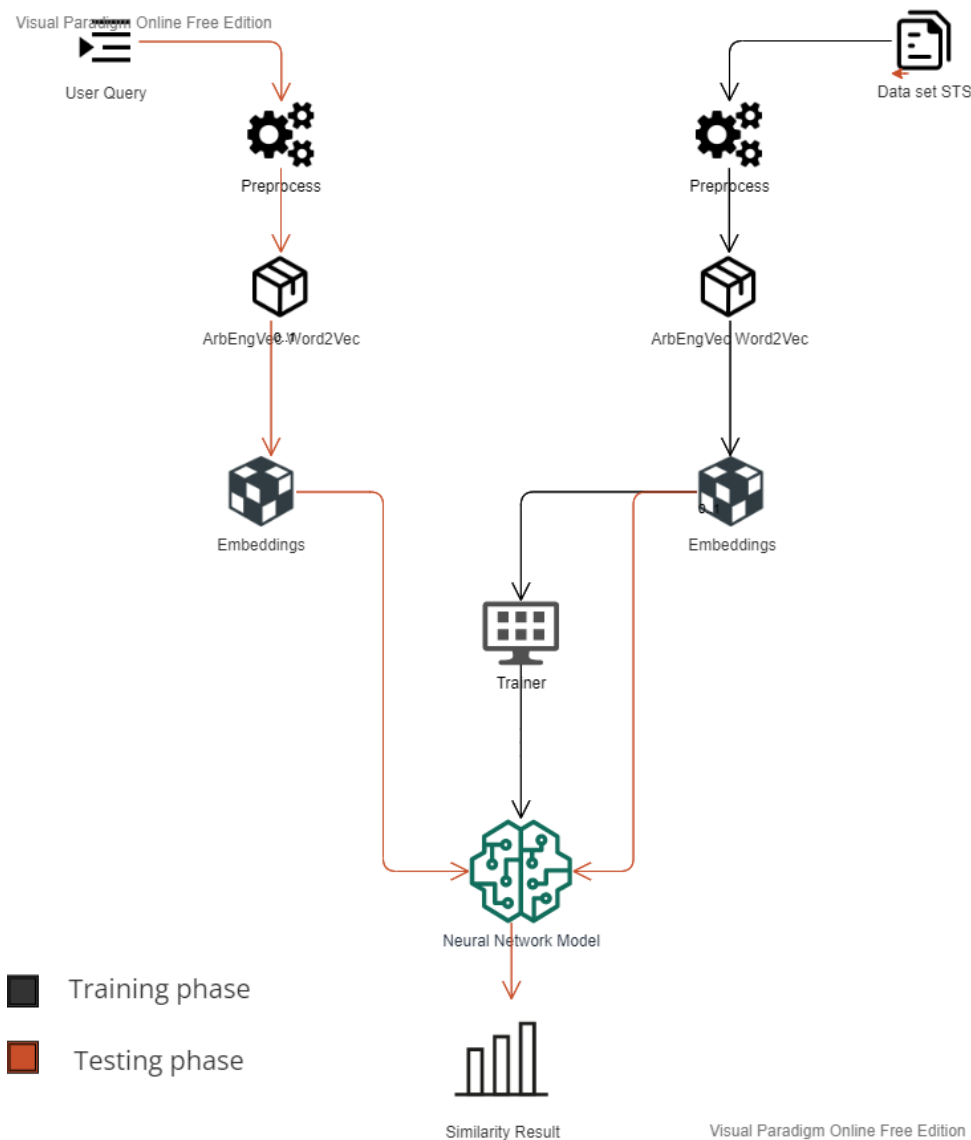


FIG. 3.1 : Architecture du notre première unité (phase Entraînement/Teste)

La deuxième unité de l'architecture donnera la main à l'utilisateur afin de créer son propre modèle personnalisé : définir son type de modèle préféré, les différents hyper-paramètres qui lui sont associés, les choix des différentes fonctions utilisées par les modèles et le pourcentage de la division entre les données de l'entraînement et tests. Dès que le modèle personnalisé est choisi et entraîné, l'utilisateur a la possibilité de lui faire un test de ses requêtes. Voici un schéma pour montrer comment ça marche :



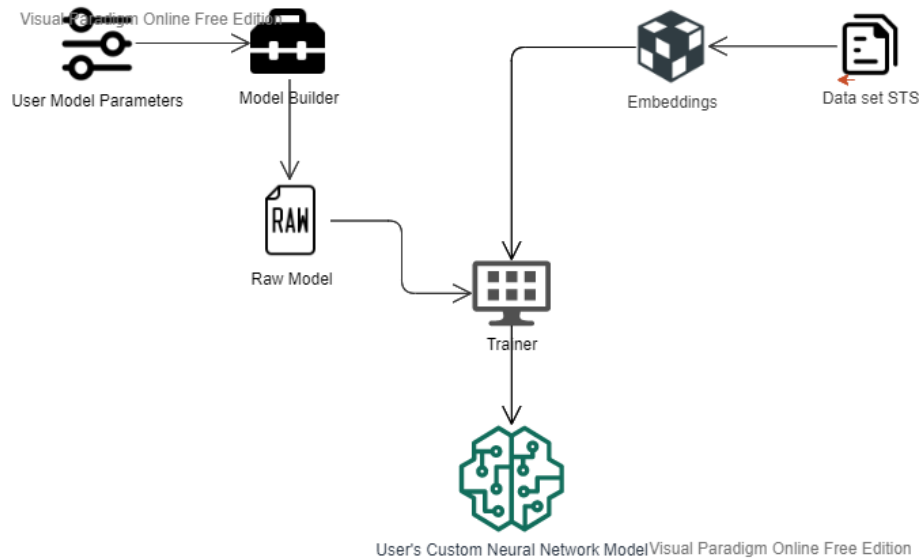


FIG. 3.2 : Architecture du notre deuxième unité (modèle utilisateur)

## 3.4 Réalisation du système

Après avoir donné une conception de notre système, nous passons à le réaliser, pour faire ça, nous entrons à la phase de réalisation où nous commençons par faire un pré traitement sur le dataset STS suivi par une implémentation et création des différents modèles de l'apprentissage profond et à la fin nous terminons par une présentation sur l'application développée avec une citation de tout les outils utilisées.

### 3.4.1 Dataset STS (Semantic Textual Similarity)

#### 3.4.1.1 Description du dataset STS

Pour faire l'apprentissage, nous avons utilisé le dataset **STS**, l'acronyme **STS** signifie la similarité sémantique textuelle (*Semantic Textual Similarity*) et c'est l'une des tâches fondamentales en **NLP** qui mesure les degrés de similarité entre les paires de phrases qui se trouvent dans les datasets qui leurs sont associés. L'utilité de cette compétition **STS** est de trouver des algorithmes ou des modèles informatiques qui imitent les performances de la compréhension du langage au niveau humain. Celle-ci est un problème difficile et profond relié au langage naturel **NLP**.

Pour stimuler la recherche dans le domaine **NLP** et encourager le développement de nouvelles approches créatives de modélisation au niveau sémantique de la phrase. La compétition **STS** se déroule annuellement depuis 2012. Chaque année, celle-ci réunit de nombreuses équipes et chercheurs qui participent à la création des approches diverses et aux améliorations continues sur les méthodes qui existent jusqu'à présent.

La STS-2017 évalue la capacité des systèmes à déterminer le degré de similitude sémantique entre les phrases monolingues et multilingues, les langues concernées dans cette compétition sont l'arabe, l'anglais et Espagnol. Cette compétition a deux tâches : une tâche primaire et une secondaire. La tâche primaire se compose seulement de fournir tous les résultats des sous-tâches secondaires, la tâche secondaire est organisée en un ensemble de sous-tâches. Chaque

sous-tâche consiste à fournir des scores de similarité pour les paires de phrases monolingues dans une langue particulière ou bien pour des paires de phrases multi-lingues à partir de la combinaison de deux langues particulières. La liste complète des tâches peut se résumer en de-sous :[39]

- ☆ **Tâche primaire** : des paires mono-lingues et cross-lingues de langue Arabe-Anglais, Espagnol-Anglais, Arabe-Arabe, Anglais-Anglais and Espagnol-Espagnol
- ☆ **Sous-tâche secondaire 1** : des paires mono-lingues de langue Arabe
- ☆ **Sous-tâche secondaire 2** : des paires cross-lingues de langue Arabe-Anglais
- ☆ **Sous-tâche secondaire 3** : des paires mono-lingues de langue Espagnol
- ☆ **Sous-tâche secondaire 4** : des paires cross-lingues de langue Anglais-Espagnol
- ☆ **Sous-tâche secondaire 5** : des paires mono-lingues de langue Anglais
- ☆ **Sous-tâche secondaire 6** : Une tâche surprise annoncée durant la période d'évaluation des modèles.

### 3.4.1.2 Architecture du dataset STS

STS-2017 sous-tâche 2 contient des paires de phrases cross-lingues des langues Anglais-Arabe. Le dataset de cette sous-tâche qui contient 250 paires phrases sera utilisé pour chaque langue alignées deux à deux dans une seule ligne. Ce dataset contient 1169 mots en anglais et 1289 en arabe. Les données d'entraînement en anglais contiennent 935 mots et celles en arabe contiennent 1031 mots. Concernant les données de test, elles contiennent 234 mots en anglais et 257 mots en arabe.

### 3.4.2 Pré-traitement sur dataset

C'est une tâche nécessaire pour n'importe quel projet de traitement automatique de langage naturel **TALN**. IL permet de nettoyer et de segmenter le dataset et parfois même de le rectifier pour le rendre apprenable par des modèles de deep learning. Le pré-traitement se compose des tâches suivantes :

- ❖ **Nettoyage** : Cette tâche s'agit d'enlever les caractères qui ne sont pas pris en considération tels que les symboles, les ponctuations, des chiffres et des dates. La suppression de ces caractères ne change rien dans l'apprentissage.
- ❖ **Suppression des diacritiques** : La langue anglaise ne contient aucune marque ou accent sur ses lettres. En revanche, la langue arabe en contient. La suppression des diacritiques rassure que la représentation en embeddings aura le nombre minimal de termes.
- ❖ **Re formulation des mots et réécriture** : Surtout dans la langue Arabe, une seule lettre peut avoir jusqu'à cinq ou six variantes tel que la phase de réécriture est une standardisation des lettres qui permet de rendre les mots plus clairs et compréhensibles. Un exemple de réécriture est le suivant : اثار و رؤية va être transformée en اثار و رويه.

- ❖ **Tokenisation** : Son objectif réside la division des phrases en unités lexicales de texte consécutive qui ne contient pas d'espaces. Ces unités sont appelées des tokens. Dans notre cas, les tokens qu'on retire sont des mots.
- ❖ **Suppression des mots vides** : Cette étape est exécutée pendant la tokenisation. Si un token ou un mot est un mot vide, on le supprime de la liste des tokens. Le mot vide est un mot commun qui ne contient aucune information et sa suppression ne change rien du point de vue sens du phrase. On le supprime pour déminer le nombres des mots à embedder afin de ne pas encombrer les modèles de deep learning.
- ❖ **Codification en Word2Vec + Racinisation des mots** : C'est la dernière étape du pré-traitement qui s'agit de codifier les unités de texte obtenues en utilisant notre modèle Word2Vec **ArbEngVec** (ou Sentence2Vec dans le cas d'utilisation du BERT). On sait que la structure **ArbEngVec** est de relation clé-valeur dont la clé représente un terme et la valeur représente le vecteur embeddings associé. Notre objectif est de codifier toutes les unités du texte. Le problème qui se pose est l'absence des clés pour à-peu-près de 30 35% le nombre des unités à coder de la langue anglaise et 60 65% de la langue arabe. Pour résoudre ça, on passe au processus de racination (ou radicalisation) qui est un processus itératif qui supprime les préfixes et/ou suffixes d'une unité jusqu'à ce que cette unité trouve son vecteur. On procède par la racination afin de modifier ces unités et diminuer leur nombres pour pouvoir trouver leur vecteur associé. Si le processus s'arrête et que l'unité n'a pas encore trouvé son vecteur c'est-à-dire qu'on ne soit pas parvenus à la co-difier, alors cette unité est automatiquement ignorée et supprimée. Le tableau suivant résume tous les préfixes/suffixes concernés à supprimer ou modifier :

Pour la langue Anglais on a :

Préfixe		
Préfixe	Remplacement	Exemple
dis-	Chaine Vide	dissatisfied = satisfied
is-	Chaine Vide	misspell = spell
un-	Chaine Vide	unacceptable = acceptable
re-	Chaine Vide	reelection = election
inter-	Chaine Vide	interrelated = related
pre-	Chaine Vide	prepay = pay
non-	Chaine Vide	nonsense = sense
super-	Chaine Vide	superscript = script
sub-	Chaine Vide	submerge = merge
anti-	Chaine Vide	antibacterial= bacterial

TAB. 3.1 : Préfixes dans la langue anglais et comment les supprimer

Suffixe		
Suffixe	Remplacement	Exemple
-ness	Chaine Vide	darkness = dark
-iness	y	happiness = happy
-ly	Chaine Vide	scholarly = scholar
-ily	y	readily = ready
-ing	Chaine Vide, pas toujours valide	caring = cate
-able	Chaine Vide	replaceable = replace
-ful	Chaine Vide	careful = care
-less	Chaine Vide	careless = careful
-ment	Chaine Vide	argument = argue
-er	Chaine Vide, pas toujours valide	dancer = dance
-ier	y	sunnier = sunny

TAB. 3.2 : Suffixe dans la langue anglais et comment les supprimer

- On doit noter que les règle de remplacement des suffixe ne sont pas toujours valable et qu'il y a des cas exceptionnelles comme :

\* **tanning** = tan, le mot (**tan**) contient une seul syllabe.

\* **regretting** = regret, le mot (**regretting**) se termine par une seule voyelle suivie d'une seule consonne.

- et on doit aussi noter qu'un mot peut avoir un préfixe et suffixe en même temps, comme mot : **uncomfortable** = **un-confort-able**

Et pour la langue arabe on a :

ا	أ	ت	ث	ن	ن	ي	ي	و	ف
س	ك	ل	ل	ب					

TAB. 3.3 : Préfixes dans la langue arabe

- Dans la langue arabe, ces préfixes peuvent être totalement supprimées, il peuvent aussi être connectés avec au autre préfixe (ال) (alif wa lam) et ce dernier peut être aussi supprimé. Exemple (بالبركة = البركة = بركة).

ا	ت	ي	ك	و	ة	ه	ات
كم	ما	ون	تم	نا	ني	ين	وه
يه	ته	ها	هم	هن	يا	ان	ية
وا	يات	تان	ونن	اته	ونه	تنا	وها
يها	تها	وهم	يون	اها	تين	يين	تهم
هما	انكم	يتنا	اتنا	يتها	اتها	اتية	اتهم
تموها							

TAB. 3.4 : Suffixes dans la langue arabe

- Les suffixe défini ci-dessus sont des terminaison de la conjugaison dans la langue arabe et ils peuvent être supprimés sans aucun changement ou modification.

### 3.4.3 Implémentation des modèles de deep learning

Pour la réalisation des différentes architectures expérimentée nous avons utilisé la bibliothèque « Tensorflow » qui permet de générer les types d'architectures neuronales avec les différentes configurations possibles à travers son api fonctionnelle.

Afin de choisir les architectures neuronales que nous allons utiliser pour la détection du plagiat, nous allons construire plusieurs modèles en exploitant différentes architectures neuronales dans un premier temps, ensuite, nous allons tester et voir les performances obtenues par chaque architecture. Enfin, nous allons développer le modèle final avec celle qui donne la meilleure précision, voici les architectures que nous allons utiliser :

Pour les architectures utilisées nous définons la couche de sortie comme une couche entièrement connectée qui contient 300 neurones, car la sortie des architectures se sont une suite des vecteurs d'embeddings de taille 300 éléments *vecteur de 300 gale un mot d'embeddings*.

#### 3.4.3.1 Modèle LSTM

Les LSTM sont des réseaux de neurones récurrents adaptés aux données textuelles. Comme nous avons vu dans l'étude de l'existant. L'architecture détaillée des cellules LSTM est décrite dans (paragraphe 2.6.2.2).

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 20, 500)	1602000
dense (Dense)	(None, 20, 300)	150300
Total params: 1,752,300		
Trainable params: 1,752,300		
Non-trainable params: 0		

FIG. 3.3 : Architecture du modèle LSTM

Dans cette architecture nous avons utilisé une couche LSTM avec 500 réseaux de neurones, une fonction d'activation "sigmoïde" et un drop-out qui est à 50

### 3.4.3.2 Modèle GRU

C'est une autre variante des réseaux récurrents, elle peut prendre en considération un contexte large dans les données séquentielles. L'architecture détaillée des cellules GRU est décrite dans (paragraphe 2.6.2.3).

Layer (type)	Output Shape	Param #
gru (GRU)	(None, 20, 400)	842400
dense_1 (Dense)	(None, 20, 40)	16040
gru_1 (GRU)	(None, 20, 250)	219000
dense_2 (Dense)	(None, 20, 200)	50200
gru_2 (GRU)	(None, 20, 150)	158400
dense_3 (Dense)	(None, 20, 300)	45300
Total params: 1,331,340		
Trainable params: 1,331,340		
Non-trainable params: 0		

FIG. 3.4 : Architecture du modèle GRU

Dans cette partie nous avons utilisé trois couche de GRU ( gru, gru-1, gru-2 ) avec des paramètres différentes suivantes : 400 neurones dans la premier couche avec 25% de la valeur de drop-out, 250 neurones dans la 2ème couche et 150 neurones pour la 3 ème couche.

Une fonction d'activation 'tanh' pour toutes les couches.avec une couche dense connectés avec toutes les couches par 40 neurones et 200 neurones pour la 2 ème couche de dense(Fully connected layers).

### 3.4.3.3 Les modèles LSTM/GRU bidirectionnels

Cette architecture est très utilisée pour améliorer les résultats des architectures précédentes. Elle peut fournir un contexte supplémentaire au réseau et entraîner un apprentissage plus rapide et encore plus complet du problème. (paragraphe 2.6.2.4).

Layer (type)	Output Shape	Param #
bidirectional_4 (Bidirectional)	(None, 20, 680)	1743520
dropout (Dropout)	(None, 20, 680)	0
dense_1 (Dense)	(None, 20, 20)	13620
bidirectional_5 (Bidirectional)	(None, 20, 680)	981920
dense_2 (Dense)	(None, 20, 300)	204300
Total params: 2,943,360		
Trainable params: 2,943,360		
Non-trainable params: 0		

FIG. 3.5 : Architecture du modèle Bidirectionnel LSTM

détails sur l'architecture montré dans la figure précédente :

- **Bidirectionnel layers** : le modèle contient une série de couches LSTM, le nombre de couches et le nombre de neurones par couche sont déterminés pendant les expérimentations. Nous utilisons la fonction "tanh" comme fonction d'activation, car c'est la fonction la plus utilisée dans les couches bidirectionnelles. Voici la liste des couches bidirectionnelles :
  - **première couche(bidirectional-4)** :cette couche contient 340 cellules LSTM et un taux de drop-out de 30%.
  - **dernière couche(bidirectional-5)** :cette couche contient 340 cellules LSTM et un taux de drop-out de 20%.
- **Fully connected layers** : nous ajoutons des couches entièrement connectées au-dessus des couches bidirectionnelles. Le nombre de couches est déterminé d'une manière expérimentale. Voici la liste des couches entièrement connectées :
  - **dense-1**:cette couche contient 20 neurones entièrement connectés qui utilisent la fonction d'activation "tanh".
  - **dense-2**:cette couche contient 300 neurones entièrement connectés qui utilisent la fonction d'activation "tanh".

Layer (type)	Output Shape	Param #
bidirectional (Bidirectional)	(None, 20, 800)	1684800
dense (Dense)	(None, 20, 20)	16020
bidirectional_1 (Bidirectional)	(None, 20, 800)	1012800
dense_1 (Dense)	(None, 20, 10)	8010
bidirectional_2 (Bidirectional)	(None, 20, 800)	988800
dense_2 (Dense)	(None, 20, 300)	240300
=====		
Total params: 3,950,730		
Trainable params: 3,950,730		
Non-trainable params: 0		

FIG. 3.6 : Architecture du modèle Bidirectionnel GRU

la figure précédente montre :

- **Bidirectionnel layers** : le modèle contient une série de couches GRU, le nombre de couches et le nombre de neurones par couche sont déterminés pendant les expérimentations. Nous utilisons la fonction “tanh” comme fonction d’activation. Voici la liste des couches bidirectionnelles :
  - **première couche(bidirectional)** :cette couche contient 400 cellules GRU et un taux de drop-out de 10%. avec une fonction d’activation qui est ”tanh”.
  - **2ème couche(bidirectional-1)** :cette couche contient 340 cellules GRU et un taux de drop-out de 20%. avec une fonction d’activation qui est ”tanh”.
  - **dernière couche(bidirectional-2)** :cette couche contient 400 cellules GRU et un taux de drop-out de 30%. avec une fonction d’activation qui est ”tanh”.
- **Fully connected layers** : nous ajoutons des couches entièrement connectées au-dessus des couches bi-directionnelles. Le nombre de couches est déterminé d’une manière expérimentale. On s’intéresse à la fonction d’activation “tanh” car c’est la la plus utilisée. Voici la liste des couches entièrement connectées :
  - **dense** :cette couche contient 20 neurones entièrement connectés qui utilisent la fonction d’activation “tanh”.
  - **dense-1**:cette couche contient 10 neurones entièrement connectés qui utilisent la fonction d’activation “tanh”.
  - **dense-2**:cette couche contient 300 neurones entièrement connectés qui utilisent la fonction d’activation “tanh”.

#### 3.4.3.4 Modèle hybride LSTM/GRU Bidirectionnel

Nous avons utilisé cette architecture qui est un mélange entre LSTM et GRU bidirectionnel, qui permet d’entraîner des textes très longs et avoir des résultats très intéressantes a interpréter. Cette architecture a un fonctionnement spécial (paragraphe 2.6.2.4).



Layer (type)	Output Shape	Param #
bidirectional_6 (Bidirectional)	(None, 20, 600)	1083600
dense_6 (Dense)	(None, 20, 40)	24040
bidirectional_7 (Bidirectional)	(None, 20, 500)	582000
dense_7 (Dense)	(None, 20, 300)	150300
=====		
Total params: 1,839,940		
Trainable params: 1,839,940		
Non-trainable params: 0		

FIG. 3.7 : Architecture du modèle hybride bidirectionnel LSTM/GRU

la figure précédente montre :

- **Bi-directional layers** : le modèle contient une série de couches GRU et LSTM, le nombre de couches et le nombre de neurones par couche sont déterminés pendant les expérimentations. La fonction “sigmoïde” sert de fonction d’activation. Voici la liste des couches bi-directionnelles :
  - **première couche(bi-directional)** :cette couche contient 300 cellules GRU et un taux de drop-out de 35%. avec une fonction d’activation qui est ”sigmoïde”.
  - **2ème couche(bi-directional-1)** :cette couche contient 250 cellules LSTM et un taux de drop-out de 15%. avec une fonction d’activation qui est ”sigmoïde”.
- **Fully connected layers** : nous ajoutons des couches entièrement connectées au-dessus des couches bi-directionnelles. Le nombre de couches est déterminé d’une manière expérimentale. Nous utilisons la fonction d’activation “tanh”, car c’est la fonction la plus utilisée. Voici la liste des couches entièrement connectées :
  - **dense** :cette couche contient 40 neurones entièrement connectés qui utilisent la fonction d’activation “tanh”.
  - **dense-2**:cette couche contient 300 neurones entièrement connectés qui utilisent la fonction d’activation “tanh”.

Pour l’entraînement du modèle de n’importe quelle architecture dont nous avons parlé précédemment, nous utilisons la fonction d’optimisation “Adam” (nous avons testé les fonctions Adam, SGD et RMSprop qui ont donné des résultats similaires donc nous avons décidé d’utiliser Adam, car c’est la plus rapide) ainsi que plusieurs valeurs pour le batch {1 , 20 , 40 , 100} et aussi la fonction ”mse” comme une fonction de loss.

### 3.4.3.5 Modèle BERT

Pour introduire le modèle BERT dans notre système, on utilisera un *Sentence transformers* qui contient des modèles multilingues pré-entraînés. Un modèle particulier qui nous intéresse

dans notre recherche appelé *distiluse-base-multilingual-cased-v1*. Celui-ci est entraîné actuellement sur quinze langues et parmi celles-ci, on a la langue anglaise et arabe. Le choix de ce modèle parmi tant d'autres est qu'il soit le seul à contenir les deux langues avec un nombre de langues minimal ; on minimise le nombre des langues pour améliorer la performance car on doit être mis en garde du fait que si un modèle contient un grand nombre de langues, il sera largement moins satisfaisant que celui qu'on a choisi.

Comme étudié précédemment (dans paragraphe 2.6.4), un modèle BERT pré-entraîné ne peut suffire pour répondre à notre problématique. Il va falloir l'entraîner sur le dataset STS et pour ce, nous allons nous intéresser à la phase de *fine tuning* afin de pouvoir faire des prédictions et répondre à des requêtes.

Dans la partie de fine tuning, l'entraînement du modèle BERT s'est fait trois fois avec différents paramètres comme suit : (10 itérations , 20 pour batch), (50 itérations , 40 pour batch), (100 itérations avec 30 pour le batch). On relèvera par la suite les meilleurs paramètres pour la phase de test.

### 3.4.4 Optimisation des hyperparamètres

C'est un paramètre qui contrôle le processus d'entraînement du modèle, a pour but de trouver la valeur optimale des hyperparamètres qui maximisent les performances du modèle. Dans le cas de l'apprentissage profond, il y'a plusieurs paramètres que nous devons optimiser :

- ❖ **Taux d'apprentissage** : c'est un paramètre crucial pour l'optimisation du temps d'entraînement et assurer la convergence du modèle. Il permet aussi de savoir si le taux d'apprentissage du modèle est plus petit que les valeurs optimales afin d'assurer plus d'itérations dans le but d'avoir un bon résultat. Cependant, si le taux d'apprentissage est beaucoup trop plus élevé que la valeur optimale, l'algorithme pourrait ne pas converger.
- ❖ **Le nombre de couches cachées** : Le nombre de couches cachées influencent beaucoup sur les résultats de taux d'apprentissage. En plus l'utilisation d'un grand nombre de couche produit une exécution plus complexe mais elle permet d'augmenter la capacité de prédiction malgré son effet mais il peut causer un sur-apprentissage du modèle.
- ❖ **Nombre de neurones** : Dans ce cas, le nombre de neurones dans les couches est très important pour assurer un bon apprentissage mais ça reste un souci car ça peut quand même causer un sur-apprentissage. Aussi, l'utilisation d'un grand nombre de neurones implique un temps énorme d'exécution.
- ❖ **Le paramètre de régularisation** : C'est la partie la plus importante. Elle permet d'ajouter une contrainte supplémentaire au modèle pendant la phase d'entraînement qui va minimiser le pourcentage et le risque d'avoir un sur-apprentissage ce qui assure une performance plus efficace du modèle. Parmi les méthodes de régularisation les plus utilisées, on cite le drop-out qui consiste à ignorer au hasard un certain nombre de sorties de neurones pendant l'entraînement avec une probabilité "p".

### 3.4.5 Présentation de l'application

Dans cette partie, après avoir créé nos modèles de deep learning afin de le rendre fonctionnel et accessible aux les utilisateurs, nous créons une interface homme-machine facile à utiliser. Alors, nous illustrerons d'abord quelques captures de notre application et de son principe de fonctionnement. Ensuite nous citerons toutes les bibliothèques et les environnements utilisés pour clôturer en donnant un diagramme qui montre la façon dont notre application est déployée au niveau du serveur publique.

#### 3.4.5.1 Interface graphique

Pour faire une illustration sur notre projet, nous données quelque captures d'ecrants

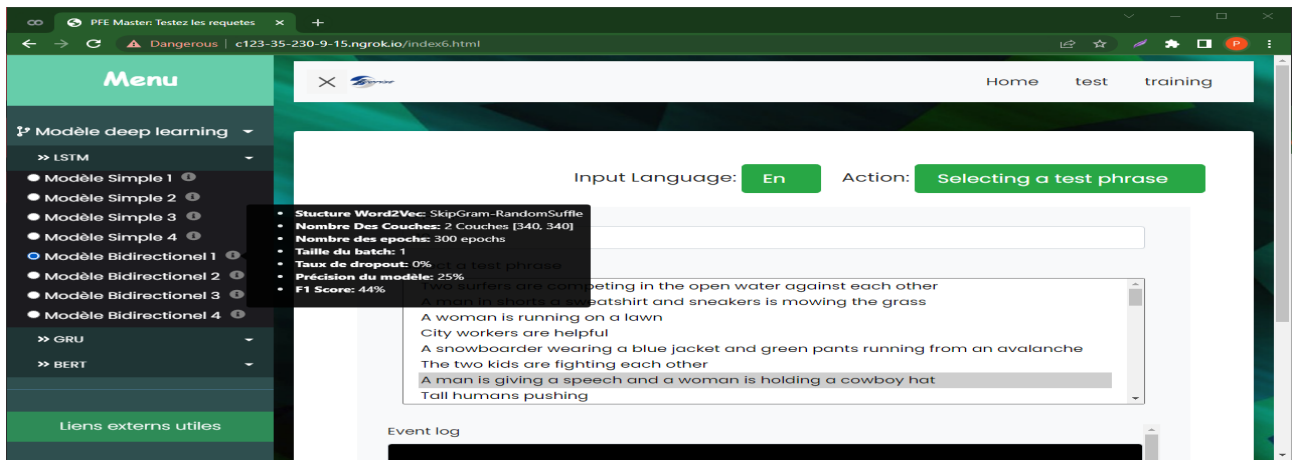


FIG. 3.8 : Fenêtre de choix d'un modèle et d'une requête donné par l'utilisateur

L'utilisateur choisit l'un des modèle de deep learning, soit LSTM, GRU ou BERT, ensuite il peut maintenant choisir une requête de teste (à partir des données de testes), par la suite il doit attendre quelque moments pour afficher le résultat de sa requête.

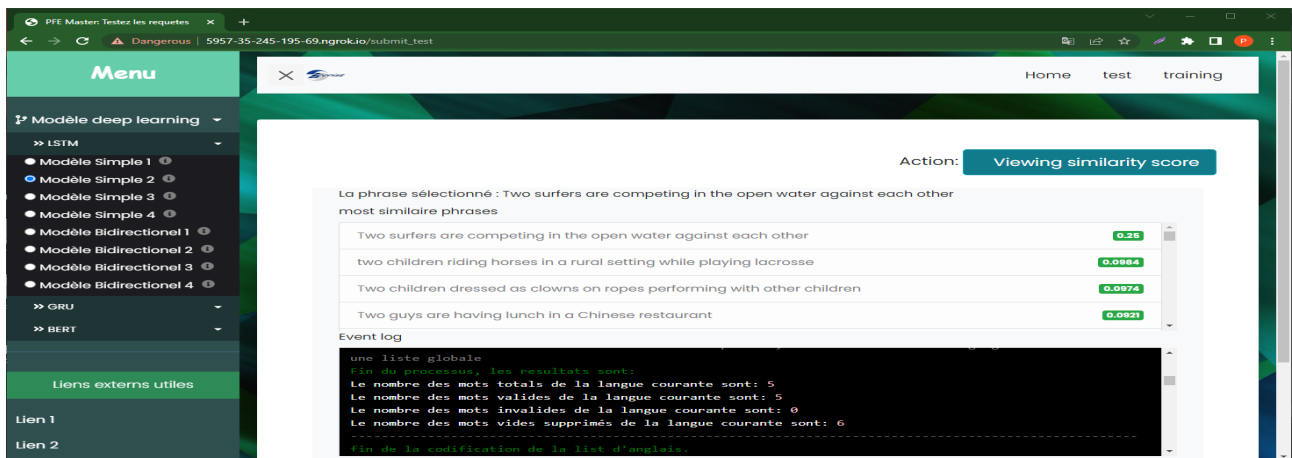


FIG. 3.9 : Fenêtre pour les Résultats de la requête donné par utilisateur

Après avoir attendu un moment, l'application retourne le résultat du test sur la requête d'utilisateur et affichera une liste des  $n$  phrases les plus proche pour l'utilisateur.

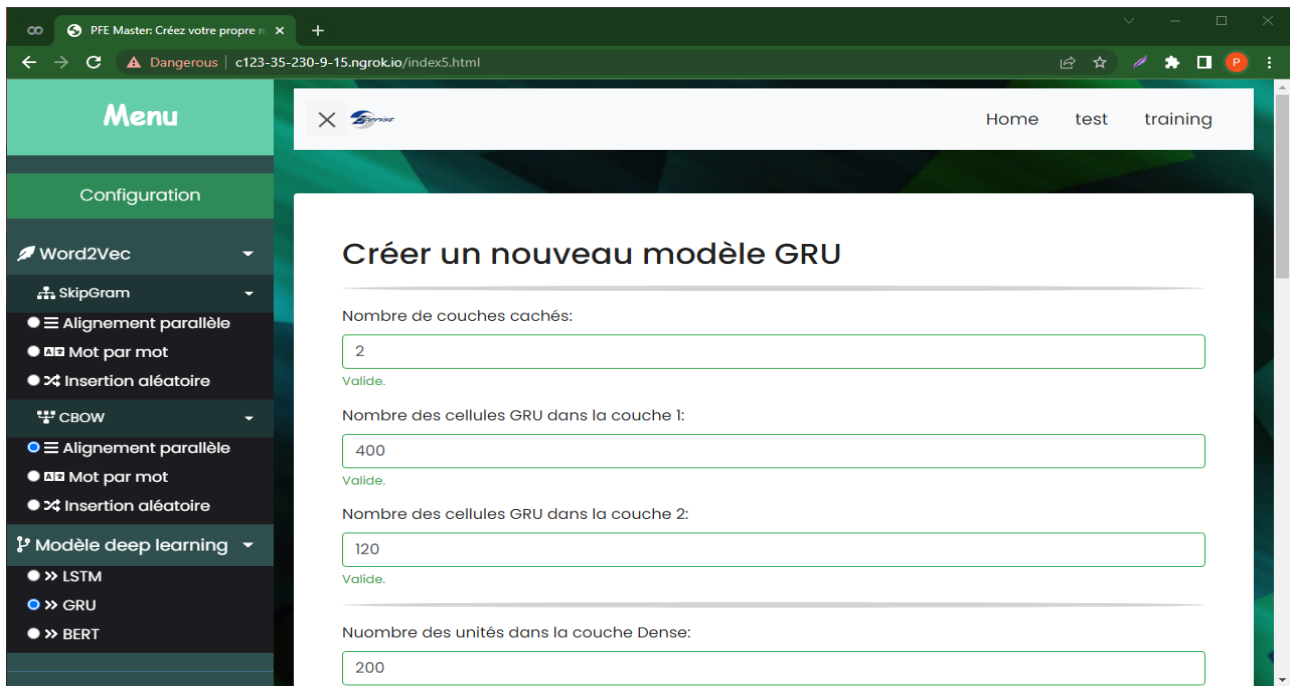


FIG. 3.10 : Fenêtre de création d'un modèle personnalisé par utilisateur

Une autre interface qu'on a développée, c'est pour la création des modèles personnalisés au choix d'utilisateur, le modèle est entraîné ensuite en ligne pour pouvoir tester les requêtes et de voir son performance.

### 3.4.5.2 Les bibliothèques utilisées

#### ❖ Tensorflow

C'est une bibliothèque développée par Google spécialisée dans l'entraînement et le développement des modèles d'apprentissage profond. Tensorflow propose un API qui permet d'implémenter des architectures profondes très complexes d'une manière simple et rapide. Nous utilisons tensorflow pour implémenter et tester les différentes configurations du modèle d'apprentissage profond.



#### ❖ Bootstrap

Bootstrap est un ensemble d'outils fiables à la création du design (graphisme, animation .. etc de sites et d'applications. Bootstrap permet d'utiliser des codes **HTML** et **CSS** ainsi que des extensions **JS** en option. C'est l'un des projets les plus utilisés par les développeurs. Nous nous servons de Bootstrap pour faire le codage et design de l'interface.



### ❖ Transformers

c'est une bibliothèque Python qui permet d'utiliser des architectures neuronales pré-entraînées sur des grands datasets sur des différentes tâches comme l'extraction d'informations, traductions automatiques..*etc.* Elles offrent une API simple et facile à utiliser qui permet de télécharger et d'adapter les différentes architectures au jeu de données disponibles. De plus, il est facilement intégrable avec les bibliothèques populaires d'apprentissage profond. Nous nous servons de transformers pour télécharger et entraîner le modèle BERT multilingue.



### ❖ Gensim

C'est une bibliothèque Python qui a pour but de manipuler des documents textes, d'implémenter et appliquer des algorithmes d'apprentissage automatique non supervisés et en particulier les algorithmes de génération des word-embeddings comme word2vec et fast-text. Nous avons utilisé Gensim pour la génération et la manipulation des plongements des mots.



### ❖ Flask

C'est un Framework Python qui permet de générer des applications Web simples en écrivant le moins de code. Nous allons utiliser Flask pour le développement de l'API de détection du plagiat.



### ❖ NLTK

C'est une bibliothèque dont on se sert pour le traitement automatique du langage naturel(**TALN**). Celle-ci assure le traitement facile des textes à travers les différents modules pour faire la tokenization par exemple. L'utilité de cette bibliothèque est le pré-traitement des documents.



### ❖ Keras

Keras est le Framework le plus utilisé, il s'agit d'une bibliothèque open source saisie sur Python qui permet de construire facilement et rapidement des réseaux de neurones et des modèles d'apprentissage automatique en se basant sur les principaux Frameworks (Tensorflow, Pytorch). Keras fut requis pour l'implémentation du modèle de prédiction **LSTM**, **GRU** et le pré-traitement des données.



#### 3.4.5.3 l'environnement utilisé

Nous avons développé cette application avec langage python en nous servant plus précisément de IDE Visual studio et google colab comme outil.

### ❖ Langage Python

Nous avons choisi de travailler avec python (version 3.9) car c'est le langage de programmation open source le plus utilisé par les informaticiens. Parmi ses qualités, Python permet notamment aux développeurs de se concentrer sur ce qu'ils font plutôt que sur la manière dont ils le font.

Ce langage s'est propulsé en tête de la gestion d'infrastructure, d'analyse de données et dans le domaine du développement de logiciels. Il a facilité la tâche et a libéré les développeurs des contraintes de formes avec les langages plus anciens qui ont tendance à être plus lents. Ainsi, saisir et développer un code avec Python est connu pour être plus rapide en temps d'exécution par rapport à d'autres langages.

### ❖ Visual Studio Code :

C'est un éditeur de code développé par Microsoft qui est libre , extensible par l'utilisateur et gratuit. Il nous a permis de réaliser et créer des pages HTML avec leurs CSS.

### ❖ Google-colab :

C'est Web-IDE, qui est créer par Google, est un environnement gratuit pour exécuter les notebooks jupyter, de plus les notebooks crée peux être modifier par un groupe d'utilisateurs simultanément. son grand avantage c'est d'utilisation le langage Python c'est-à-dire il a accès a plusieurs bibliothèques Les spécifications matérielles de l'environnement sont :

### ❖ Environnement Matériel

Nous avons utilisés deux environnement le premier c'est celui de Google-colab pour exécuté le code python.

- \* RAM :12 Géra extensible jusqu'à 25 Géra octect.
- \* Disque :358.27 Gb.
- \* GPU :1xNvidia Tesla T4.
- \* CPU :1xIntel(R) Xeon(R) CPU @ 2.30GHz.

le deuxième environnement pour créer les pages Webs.

- \* RAM :4 Géra.
- \* Disque :525 Gb.
- \* CPU :1xIntel(R) i5 CPU @2.30GHz.

#### 3.4.5.4 Diagramme de déploiement

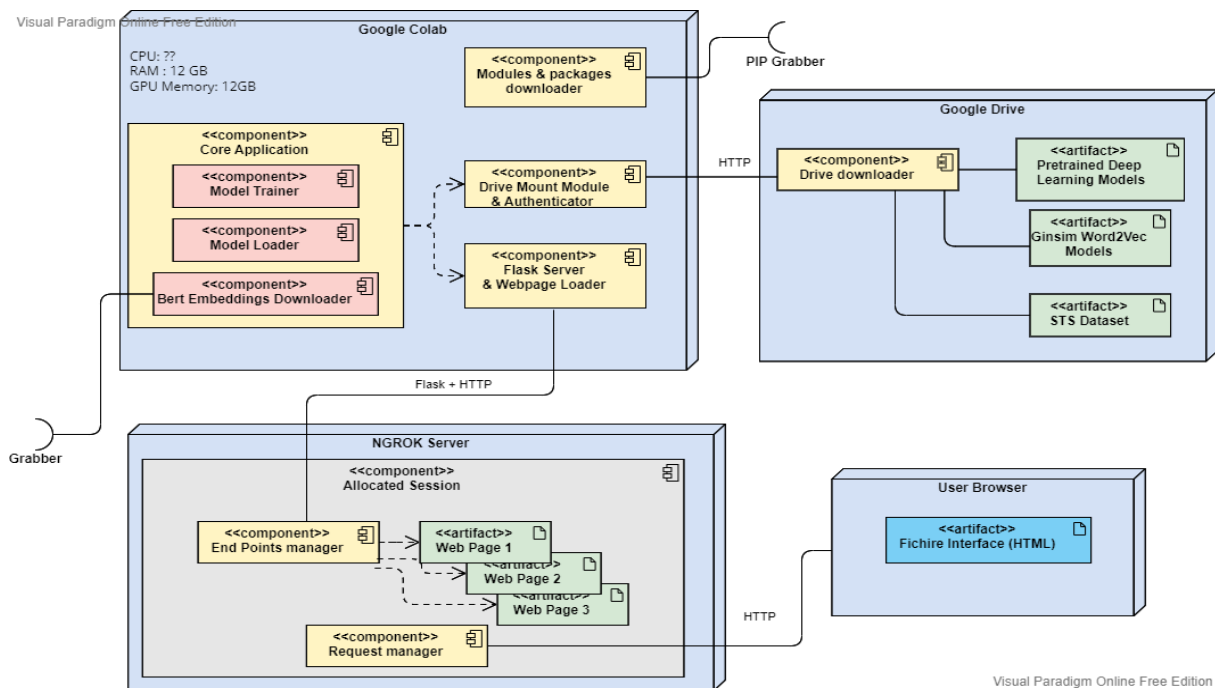


FIG. 3.11 : Diagramme de déploiement de notre système

En résumant, Nous montrons l'utilisation de ces bibliothèques dans notre application dans l'environnement Web-IDE "Google Colab". Notre application est déployé sur un serveur *ngrok*,

ce serveur permet d'exécuter l'application sur sa plateforme en lui fournissant le code d'exécution avec les fichiers interfaces **HTML**

Pour commencer, notre application est exécuté depuis *Google Colab* qui lance le module *Flask*, le module *Flask* va réserver une session pour nous dans le serveur *ngrok* et il cominique avec lui en utilisant des routes (des *end-points*) ces routes sont connecté directement avec le corps de l'application, nous procédons de cette manière :

- ☆ Tout d'abord, on doit motionner que les modèles de *deep learning* entraînés (LSTM, GRU et BERT) avec les *Word Embeddings ArbEngVec Word2Vec* que nous utilise sont sauvegardé dans le *Google Drive*.
- ☆ Note application charge les pages Web à afficher et les envoi au serveur *ngrok*, ce dernier va affichera aussi au n'importe quel utilisateur qui a le lien public de notre session sur le serveur *ngrok*.
- ☆ L'utilisateur communique avec Le serveur *ngrok* en lui fournissant tous les détailles/requêtes choisi, le serveur va envoyer ces dernier sous une méthode de **POST**.
- ☆ En suivant le choix du l'utilisateur, L'application peut charger des différent modèle de *deep learning* et des modèle de *Word2Vec* à partir du *Google Drive* qui est connecté avec l'application.
- ☆ Le *Google Drive* peut nous envoyé des modèles de *deep learning* ou *Word2Vec* à la demande.
- ☆ L'application peut utiliser ces modèle pour faire des prédiction et répondre à la demande de l'utilisateur.
- ☆ L'application envoie des résultats d'évaluations en utilisant *Flask* au serveur *ngrok* qui affichera ensuite sur son navigateur.
- ☆ Dans le cas de création du modèle personnalisé de *deep learning*, l'application a un module de construction et fabrication des modèle qui utiliser les hyper-paramètres et paramètres de définition du modèle et elle lui donne la possibilité a entraîner ce modèle et de luit de tester sur les différents requêtes de tests.
- ☆ Si l'utilisateur souhaite de créé un modèle *BERT*, l'application doit télécharger alors les *word embeddings* de ce modèle, les *word embeddings* de BERT diffère du *Word2Vec ArbEngVec* car il sont de type *Sentence2Ve* spécial pour *BERT*.



## 3.5 Tests et expérimentation

### 3.5.1 Tests et interprétations

Nous utilisons 70% du dataset pour la phase d'entraînement du modèle, 10% pour la validation et 20% pour la phase des tests, avec des plongements de mots multilingues générés en utilisant l'algorithme "RandomShuffle-Skip".

Modèle	WbW Cbow		WbW SkipGram		RS Cbow		RS SkipGram		PA Cbow		PA SkipGram	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
LSTM 1C20 50EP	4.24%	1.03%	3.68%	2.3%	5.04%	2.4%	6.3%	1.6%	3.5%	0.9%	2.56%	1.01%
LSTM 1C500 50EP	9%	2.93%	8%	3.01%	11.4%	3.48%	11.05%	3.01%	10%	1.24%	12%	2.4%
LSTM 3C500 50EP	7.05%	0.49%	4.53%	1.41%	9.02%	2.41%	10.13%	3.3%	7.3%	2.10%	5.42%	0.34%
LSTM 3C500 500EP	12%	1.99%	11.3%	2.5%	14%	2.3%	13.94%	1.85%	11.2%	1.4%	7.02%	0.95%
Bi LSTM 1C30 10EP	1.04%	0.15%	0.94%	0.13%	2.05%	1%	1.94%	0.9%	0.46%	0.1%	0.94%	0.52%
Bi LSTM 1C500 300EP	19.3%	3.1%	18.20%	1.2%	20.2%	4.5%	21.4%	3.1%	16.3%	2.09%	17.01%	1.95%
Bi LSTM 2C 400 250EP	23%	3.5%	21%	2.5%	21.01%	2.22%	22.95%	2.28%	19.10%	1.76%	20.01%	2.4%
Bi LSTM 2C340 300EP	18.9%	3.01%	17.09%	2.3%	23.54%	3.09%	22%	2.99%	17.03%	2.04%	16.93%	2.51%

TAB. 3.5 : Résultats des tests sur le dataset utilisant LSTM

**Légende :**

- "**<numéro1>C<numéro2>**" : <numéro1> couche avec contenant <numéro2> neurones. Exemple 2C100
- "**<numéro>EP**" : numéro Epoch

**Interprétations :**

Tout d'abord nous remarquons le changement de l'architecture neuronale ainsi que la méthode de présentation du document influence la performance du modèle.

D'où notre curiosité à vouloir utiliser un autre type d'architecture de LSTM qui est le bidirectionnel pour distinguer ses performances et voir la qualité des résultats obtenus et on observe que dans ce cas là, il y a une relation proportionnelle entre le nombre d'itérations et le taux de précision sur le dataset d'entraînement tel que plus on augmente les nombres de neurones dans les couches, meilleurs seront les résultats obtenus.

Il est à noter qu'il soit primordial d'augmenter le nombre d'itérations en même temps que le nombre des neurones sinon on risque de causer un sous-apprentissage d'où un taux de validation moins bon.

Dans notre cas, on a obtenu un taux de précision sur le dataset d'entraînement 23.54% On remarque que le modèle souffre de sur-apprentissage car il existe une grande différence entre les résultats obtenus et le taux de validation donc on doit ajouter un mécanisme de régularisation afin de résoudre ce problème.

Pour le modèle Bi-LSTM avec 2 couches de 340 neurones et 300 itérations, on remarque qu'il atteint une précision sur dataset d'entraînement de 23.54% et un taux de précision de 3.09% sur le dataset de validation .

Le modèle basé sur l'algorithme RScbow Skipgram a donné une meilleure performance avec une précision 23.54% sur le dataset d'entraînement.

Modèle	WbW Cbow		WbW SkipGram		RS Cbow		RS SkipGram		PA Cbow		PA SkipGram	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
GRU 1C20 50EP	7.34%	0.89%	5.43%	0.97%	9.03%	2.3%	7.3%	2.04%	2.65%	1.96%	3.59%	1.43%
GRU 1C500 50EP	11.03%	0.94%	7.49%	1.30%	11.45%	1.31%	11.34%	1.24%	9.67%	0.98%	7.89%	1.43%
GRU 1C500 250EP	14.02%	2.49%	13.1%	1.38%	17.03%	3.01%	18%	2.04%	12.32%	2.98%	11.13%	1.34%
GRU 3C340 750EP	14%	2.98%	14.3%	1.29%	19.1%	1.87%	17.23%	0.96%	10.20%	0.46%	11.13%	1.01%
Bi GRU 1C30 10EP	2.31%	0.31%	1.94%	0.72%	1.05%	1.89%	2.94%	0.93%	1.46%	1.15%	0.94%	0.52%
Bi GRU 1C500 300EP	20.3%	3.1%	18.20%	1.2%	20.2%	4.5%	21.4%	3.1%	16.3%	2.09%	17.01%	1.95%
Bi GRU 2C 400 250EP	19%	3.5%	21%	2.5%	21.01%	2.22%	22.95%	2.28%	19.10%	1.76%	20.01%	2.4%
Bi GRU 3C500 250EP	17%	3.01%	17.09%	2.3%	21%	4.09%	23%	3.99%	17.03%	2.04%	22.1%	2.51%

TAB. 3.6 : Résultats des tests sur le dataset utilisant GRU

### Interprétation

On remarque que la précision sur le dataset d'entraînement du GRU donne de meilleurs résultats par rapport aux résultats de l'architecture LSTM mais ça reste une légère différence et c'est pour ça qu'on a fait recours au Bi-LSTM qui a donné de meilleurs résultats.

Et il est à noter aussi que le Bi-GRU assure une légère amélioration par rapport aux résultats de GRU.

Présence du sur-apprentissage quelque soit l'architecture utilisée.

La précision sur le dataset d'entraînement du GRU a atteint 19%. La précision sur le dataset

d'entraînement du Bi-GRU a atteint 22.95%.

A partir de nos observations et analyses du tableau, on déduit que l'algorithme RS est le meilleur en performances quelque soit l'architecture appliquée.

Modèle	WbW Cbow		WbW SkipGram		RS Cbow		RS SkipGram		PA Cbow		PA SkipGram	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
bi-GRU & bi-LSTM 2C100 50EP	5.34%	0.45%	3.43%	0.39%	7.13%	1.22%	8.22%	2.22%	2.68%	0.13%	5.21%	0.02%
bi-GRU & bi-LSTM 1C300, 1C250 250EP	10.03%	2.01%	8.39%	0.89%	11.52%	2.94%	11.04%	1.02%	10.67%	2.98%	9.89%	1.38%

TAB. 3.7 : Résultats des tests sur le dataset utilisant GRU et LSTM

### Interprétations :

Le tableau représente une architecture hybride du GRU et LSTM ainsi que le Bi-GRU et Bi-LSTM et on observe que la précision du dataset d'entraînement ne dépasse pas 11.52% quelque soit le nombre de couches et le nombre d'itération et l'algorithme utilisé.

Quant à la précision du dataset de validation, on remarque la présence d'un sur-apprentissage.

D'où dans ce cas là ; on en déduit aucune amélioration observée quelque soit l'algorithme appliqué.

**Bert-Multilingues** : le tableau suivant résume les résultats obtenus par le modèle basé sur BERT-Multilingue.

	Train-acc	Test-acc
Bert-Multilingue 10 EP batch=20	60.41%	55.05%
Bert-Multilingue 50 EP Batch=40	70.41%	65.01%
Bert-Multilingue 100 EP Batch=30	79.11%	65.87%

TAB. 3.8 : Résultat du modèle BERT

### Interprétation :

Nous remarquons que le modèle BERT donne de bons résultats pour la phase d'entraînement et test quelque-soit les paramètres utilisés dans l'entraînement, en comparant avec les modèles basés sur le word embedding.

### 3.5.2 Résultats finaux et évaluations

pour l'évaluation des modèles nous allons nous baser sur un ensemble de métriques de performances :

- ❖ **accuracy** : c'est la proportion des instances correctement classifiées par le modèle, (la formule de l'accuracy est présenté dans l'étude bibliographique, 2.7.1.2).
- ❖ **Le F1-Score** : c'est un compromis entre la précision et le recall. (Les formules du f1-score et de la précision et recall sont présentés dans l'étude bibliographique, 2.7.1.6).

Pour la phase d'évaluation nous avons choisi le meilleur résultat obtenu pour chaque tableau des résultats du test précédent.

Les modèles à base de réseaux bidirectionnel LSTM et bidirectionnel GRU souffrent d'un sur-apprentissage (Overfitting) d'où l'importance d'ajouter des mécanismes de régularisation et c'est ce que nous allons faire dans la partie optimisation des hyperparamètres.

Le modèle BERT est plus performant que les modèles basés sur les words embedding en terme de précision avec un pourcentage de 79.41%.

**Résultats sur le dataset total** : Le tableau suivant présente chacune de la précision et le F-score des modèles sur le dataset de test et de validation.

Modèle	Train		Validation		Test	
	Accuracy	F1_Score	Accuracy	F1_Score	Accuracy	F1_Score
Bi-LSTM	20%	46.32%	11.01%	36.54%	12.03%	37.51%%
Bi-GRU	15%	56.14%	9.03%	35.73%	10.35%	29.13%
BERT	79.41%	78%	70.19%	62.43%	59.52%	60.29%

TAB. 3.9 : Précision et F1-score du modèles

### Interprétation des résultats finales

D'après les résultats du tableau en-dessus, nous observons que les résultats ne sont pas satisfaisants pour l'architecture Bi-LSTM et Bi-GRU malgré toute la modification apportée, l'optimisation des hyperparamètres et l'utilisation des mécanismes de régularisation. Il existe une différence entre Accuracy et F1-Score pour les architectures Bi-GRU et Bi-LSTM d'où une influence directe sur les résultats de validation. Par contre L'architecture BERT donne un résultat plutôt satisfaisant. On en conclut que le modèle BERT est plus préformant par rapport aux autres architectures.

## 3.6 Conclusion

Dans ce dernier chapitre, nous nous sommes particulièrement intéressé aux différents outils et technologies utilisées pour réaliser notre système de détection du plagiat ainsi que l'architecture des quatre modèles de deep learning que nous avons construit. Il s'agit de l'architecture LSTM, GRU, bidirectionnel LSTM/GRU basée sur l'utilisation des word-embedding multilingues générés par l'algorithme "Mode aléatoire (*Random shuffle*)" et l'architecture Bert-Multilingue basée sur l'utilisation de *sentence transformers*. Puis nous avons présenté notre application développée avec les bibliothèques utilisées. Par conséquent, nous avons obtenu les résultats des différentes expérimentations et tests réalisés sachant que nous avons testé plusieurs configurations de différentes architectures (LSTM ,GRU, Bi-LSTM , Bi-GRU) avec plusieurs modèles word embeddings ArbEngVec qu'on a accompagné de nos observations, commentaires et interprétations pour enfin faire une comparaison entre les performances des Accuracy et F1-score pour les architectures dans la partie des tests finaux.

# Conclusion générale

L'élaboration de ce projet s'est révélée bien profitable sur plusieurs points, elle nous a permis d'acquérir une expérience enrichissante en mettant en pratique ce que nous avons acquis.

En effet, la technologie et internet se développent énormément au fil des générations d'où l'apparition du phénomène du plagiat qui est devenu très courant et il peut s'avérer compliqué de le détecter et c'est justement ce qui a poussé les développeurs à mettre en place des algorithmes en deep learning qui ont contribué à la création et le développement de logiciels d'anti-plagiat afin de protéger autrui d'une violation d'une propriété intellectuelle.

Durant la réalisation de notre projet, nous avons travaillé à répondre à la problématique précédemment citée en aboutissant à la réalisation du logiciel d'anti-plagiat Cross-langue. On en a déduit que les résultats obtenus ne sont pas optimaux pour les deux types d'architectures LSTM et GRU et ce, dû au fait que ces dernières ont un problème en mémorisation de longues séquences ce qui va quand même nous pousser à vouloir les améliorer. Cependant, l'architecture BERT, elle, fournit de meilleurs résultats grâce à son bon fonctionnement vu qu'elle a la capacité d'analyser en entrée la phrase entière sans qu'on ait à la décomposer pour étudier chaque mot séparément. Cette opportunité nous a quand même permis de nous familiariser avec les types d'architectures utilisées en traitement du langage automatique(TALN) surtout avec le développement sans cesse croissant du contenu textuel sur le web. Nous avons également pris connaissance des dernières tendances et techniques utilisées dans ces domaines là.

L'application reste bien évidemment ouverte à toute amélioration future et enrichie par des fonctionnalités plus avancée. Nous comptons par la suite optimiser notre projet en modifiant et améliorant l'algorithme saisi sur le logiciel "Gensim" et en ouvrant notre champ de perception en appliquant un entraînement des modèles conçus précédemment en utilisant d'autres Corpus de plus grande taille contenant plus d'instances d'application. A titre d'exemple : MultiUNV1, UbuntuV4.10 *etc...*

Le but que nous nous étions assigné étant atteint, nous avons achevé ce projet en faisant de notre mieux pour présenter un travail aussi cohérent que possible avec les moyens et connaissances que nous avons acquis durant tout notre cursus universitaire.

# Glossaire et Acronymes

- ADAM** Adaptive moment estimation. 22
- ArbEngVec** Arabic Anglais Word2Vec open source. 35, 36, 40
- BERT** Bidirectional Encoder Representations from Transformers. 18, 29–31
- CBOW** Continuous Bag of Words Model. 15
- CL-ASA** Cross-Language Alignment Similarity Analysis. 10
- CL-CTS** Cross-Language Conceptual Thesaurus-Based Similarity. 13
- CL-ESA** Cross-Language Explicite Similarity Analysis. 12
- CL-KCCA** Cross-Language Kernel Canonical Correlation Analysis. 11
- CL-KGA** Cross-language knowledge graph analysis. 13
- CL-LSI** Cross-Language Latent Semantic Indexing. 11
- CL-VSM** Cross-Language Vector Space Model. 12
- CLS** Sentence-level classification. 30
- CNN** Réseau des neurones convolutif. 23
- CSS** Cascading Style Sheets. 49
- Cognateness** Un modèle cross-lingue qui compare les origines des mots. 10
- FN** False Negative. 32, 33
- FP** False Postive. 32, 33
- GRU** Gated Recurrent Unit. 18, 25, 27, 51
- Google Brain** Un projet de recherche d'apprentissage profond conduit par Google.. 28
- HTML** hypertext markup language. 49, 53
- IA** Intelligence artificiel. 18, II–IV
- IBM** International Business Machines Corporation. 19
- ID3** Algorithme arbre de décision, ID3 est prédécesseur de C4-5. 19
- ID** Identité. 6

**JS** Java scripte. 49

**LSI** Latent Semantic Indexing. 11

**LSTM** Long short-term memory. 18, 24, 25, 27, 51

**NLP** Natural language processing. 38, III, IV

**OPUS** cOrPUS Open source parallèle corpus. 35

**POS** Part of Speech, Partie de discours. 7

**P** Précision. 33

**RI** Recherche Information. 6, 8

**RNN** Les réseaux de neurones récurrents. 23, 24, 27, 28

**ReLU** Rectified Linear Unit. 23

**R** Rappel. 33

**SEP** Indique la séparation entre les phrases ou la fin de la phrase. 30

**SGD** Descente du gradient stochastique. 22

**STS** Semantic Textual Similarity.. 38

**String-Matching** Technique de recherche de chaînes. 6

**T+MA** Translate followed by mono-lingue analysis. 13

**TALN** Traitement automatique du langage naturel. 7, 39, 51, II, III

**TE** Taux d'erreur. 33

**TN** True Negative. 32, 33

**TP** True Positive. 32, 33

**TS** Taux de succès. 32

**cross-lingue** Entre les termes et entre vocabulaire de deux ou plusieurs langues. 1, 10–13, 16, 33, II

**deep learning** Apprentissage profond. 16, 18

**embeddings** Ou Word embeddings, une représentation de texte où les mots qui ont le même sens sont similaires. 4, 14, 16, 30, 31, II

**feed-forward** Un type de réseau de neurone. 23, 28, 29

**fingerprinting** Empreinte digitale du document. 6

**mono-lingue** Contient des termes d'une seule langue. 13, 14, 16

**n-grammes** une séquence continue de n lettres dans un mot donné. 9, 10

# Bibliographie

- [1] Asim ALI, Hussam ABDULLA et Vaclav SNASEL. “Overview and Comparison of Plagiarism Detection Tools.” In : jan. 2011, p. 161-172.
- [2] Salha ALZAHIRANI, Naomie SALIM et Ajith ABRAHAM. “Understanding Plagiarism Linguistic Patterns, Textual Features, and Detection Methods”. In : *Systems, Man, and Cybernetics, Part C : Applications and Reviews, IEEE Transactions on* 42 (avr. 2012), p. 133-149. DOI : [10.1109/TSMCC.2011.2134847](https://doi.org/10.1109/TSMCC.2011.2134847).
- [3] *Cross-Language Plagiarism Detection Methods*. 2013.
- [4] Wang LING et al. “Two/Too Simple Adaptations of Word2Vec for Syntax Problems”. In : *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies*. Denver, Colorado : Association for Computational Linguistics, mai 2015, p. 1299-1304. DOI : [10.3115/v1/N15-1142](https://doi.org/10.3115/v1/N15-1142). URL : <https://aclanthology.org/N15-1142>.
- [5] Magdalena JANKOWSKA. “Author Style Analysis in Text Documents Based on Character and Word N-Grams”. In : 2017.
- [6] Ferrero JÉRÉMY. “Similarités Textuelles Sémantiques Translingues : vers la Détection Automatique du Plagiat par Traduction”. Thèse de doct. Déc. 2017.
- [7] Sebastian RUDER, Ivan VULIĆ et Anders SØGAARD. “A Survey Of Cross-lingual Word Embedding Models”. In : (2017). DOI : [10.48550/ARXIV.1706.04902](https://doi.org/10.48550/ARXIV.1706.04902). URL : <https://arxiv.org/abs/1706.04902>.
- [8] Ashish VASWANI et al. “Attention is All you Need”. In : *ArXiv abs/1706.03762* (2017).
- [9] Bendenia HANANE. “Détection des différents types de plagiat (copier-coller, plagiat des idées, plagiat des images, paraphrasé, plagiat avec synonymie, plagiat avec traduction)”. In : (2019).
- [10] Raki LACHRAFI et al. “ArbEngVec : Arabic-English Cross-Lingual Word Embedding Model”. In : *Proceedings of the Fourth Arabic Natural Language Processing Workshop*. Florence, Italy : Association for Computational Linguistics, août 2019, p. 40-48. DOI : [10.18653/v1/W19-4605](https://doi.org/10.18653/v1/W19-4605). URL : <https://www.aclweb.org/anthology/W19-4605>.
- [11] David C. ISON. “Detection of Online Contract Cheating Through Stylometry : A Pilot Study”. In : *Online Learning* (2020).
- [12] Farah KHILED et Mohammed AL-TAMIMI. “Plagiarism Detection Methods and Tools : An Overview”. In : *Iraqi Journal of Science* 62 (août 2021), p. 2771-2783.
- [13] Sagar KULKARNI, Sharvari GOVILKAR et Dhiraj AMIN. “Analysis of Plagiarism Detection Tools and Methods”. In : *SSRN Electronic Journal* (jan. 2021). DOI : [10.2139/ssrn.3869091](https://doi.org/10.2139/ssrn.3869091).



- [14] J. NILSSON. “Punctuation restoration in Swedish through fine-tuned KB-BERT”. In : *ArXiv* abs/2202.06769 (2022).

# Webographie

- [15] URL : <https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html>. (accessed : 15.07.2022).
- [16] Sachin ABEYWARDANA. *Sequence to sequence tutorial*. URL : <https://towardsdatascience.com/sequence-to-sequence-tutorial-4fde3ee798d8>. (accessed : 20.07.2022).
- [17] *Advanced Fusion-Based Speech Emotion Recognition System Using a Dual-Attention Mechanism with Conv-Caps and Bi-GRU Features*. URL : <https://www.mdpi.com/2079-9292/11/9/1328/htm>. (accessed : 09.09.2022).
- [18] Jay ALAMMAR. *The Illustrated Transformer*. URL : <https://jalammar.github.io/illustrated-transformer/>. (accessed : 20.07.2022).
- [19] Florian Arthur ANTOINE SIMOULIN. *Le BERT NLP de Google AI sur le banc de test!* URL : <https://www.quantmetry.com/blog/bert-google-ai-banc-de-test/>. (accessed : 19.07.2022).
- [20] Paul E. BLACK. *Dictionary of Algorithms and Data Structures*. URL : <https://xlinux.nist.gov/dads/>. (accessed : 23.04.2022).
- [21] Loïck BOURDOIS. *NLP - Illustration du modèle BERT de Devlin et al. et de ses dérivés, notamment français : CamemBERT et FlauBERT*. URL : <https://lbourdois.github.io/blog/nlp/BERT/>. (accessed : 19.07.2022).
- [22] Bernardo BRADTKE. *A step-by-step tutorial on developing LSTM, GRU and BiLSTM models for multi-step forecasting of water consumption*. URL : <https://morioh.com/p/34ad430cb59b>. (accessed : 19.07.2022).
- [23] Antoine CROCHET-DAMAIS. *Deep learning : définition et principes de l'apprentissage profond*. URL : <https://www.journaldunet.fr/web-tech/guide-de-l-intelligence-artificielle/1501333-deep-learning-definition-et-principes-de-l-apprentissage-profond/>. (accessed : 11.07.2022).
- [24] DEVOPEDIA. *Fingerprinting Algorithms*. URL : <https://devopedia.org/fingerprinting-algorithms>. (accessed : 21.04.2022).
- [25] Euro-Lex : un site web officiel de l'Union EUROPÉENNE. *EuroVoc*. URL : <https://eur-lex.europa.eu/browse/eurovoc.html?locale=fr>. (accessed : 19.06.2022).
- [26] Brayden FOX. *Translation Plagiarism : burning issue in modern plagiarism detection*. URL : <https://medium.com/@braydenfox/translation-plagiarism-burning-issue-in-modern-plagiarism-detection-318703dbb9ec>. (accessed : 18.04.2022).
- [27] NIH Central Resource for GRANTS et Funding Information (GOV). *Research Misconduct - Definitions*. URL : [https://grants.nih.gov/policy/research\\_integrity/definitions.htm](https://grants.nih.gov/policy/research_integrity/definitions.htm). (accessed : 25.03.2022).

- [28] Rani HOREV. *BERT expliqué : modèle de langage de pointe pour le TAL*. URL : <https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270>. (accessed : 4.08.2022).
- [29] Xinyu Liu Yongjun Wang Xishuo Wang Hui Xu Chao LI et Xiangjun XIN. *Bi-directional gated recurrent unit neural network based nonlinear equalizer for coherent optical communication system*. URL : <https://opg.optica.org/oe/fulltext.cfm>. (accessed : 1.09.2022).
- [30] Katie Murrell LIBRARY. *How to Avoid Plagiarism : Types of Plagiarism*. URL : <https://libguides.lindsey.edu/plagiarism/types>. (accessed : 16.03.2022).
- [31] NATE LORD. *What is File Fingerprinting ?* URL : <https://digitalguardian.com/blog/what-file-fingerprinting>. (accessed : 21.04.2022).
- [32] *Optimizers in Deep Learning*. URL : <https://medium.com/mlearning-ai/optimizers-in-deep-learning-7bf81fed78a0>. (accessed : 07.08.2022).
- [33] *Optimizers in Deep Learning*. URL : <https://runder.io/optimizing-gradient-descent/index.html#nesterovacceleratedgradient>. (accessed : 07.08.2022).
- [34] University of OXFORD. *Plagiarism defenition*. URL : <https://www.ox.ac.uk/students/academic/guidance/skills/plagiarism>. (accessed : 07.03.2022).
- [35] Tutorials POINT. *Cryptography Hash functions*. URL : [https://www.tutorialspoint.com/cryptography/cryptography\\_hash\\_functions.htm](https://www.tutorialspoint.com/cryptography/cryptography_hash_functions.htm). (accessed : 21.04.2022).
- [36] Qata Analytics POST. *word embedding*. URL : <https://dataanalyticspost.com/Lexique/word-embedding/>. (accessed : 01.05.2022).
- [37] Lambert R. *Comprendre le fonctionnement d'un LSTM et d'un GRU en schémas*. URL : <https://penseeartificielle.fr/comprendre-lstm-gru-fonctionnement-schema/>. (accessed : 19.07.2022).
- [38] Kaushik RANGADURAI. *BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding*. URL : <https://www.weak-learner.com/blog/2019/08/16/bert/>. (accessed : 19.07.2022).
- [39] *SemEval-2017 Task 1*. URL : <https://alt.qcri.org/semeval2017/task1/>. (accessed : 28.08.2022).
- [40] TECHNO-SCIENCE.NET. *Apprentissage automatique - Définition et Explications*. URL : <https://www.techno-science.net/glossaire-definition/Apprentissage-automatique.html>. (accessed : 07.07.2022).
- [41] UNIKLLIB. *10 Types of Plagiarism in Research*. URL : <https://unikllib.wordpress.com/2016/03/21/10-types-of-plagiarism-in-research/>. (accessed : 20.03.2022).
- [42] Dominican UNIVERSITY. *Plagiarism of Structure*. URL : [https://jicsweb1.dom.edu/ICS/Resources/Student\\_Services/Learning\\_Resources/Writing\\_Resources/Plagiarism.jnz?portlet=Plagiarism\\_of\\_Structure](https://jicsweb1.dom.edu/ICS/Resources/Student_Services/Learning_Resources/Writing_Resources/Plagiarism.jnz?portlet=Plagiarism_of_Structure). (accessed : 25.03.2022).
- [43] VERSATILE1990. *Architecture et processus d'apprentissage dans un réseau de neurones*. URL : <https://fr.acervolima.com/architecture-et-processus-d-apprentissage-en-reseau-de-neurones/>. (accessed : 15.07.2022).
- [44] *What does it mean by Bidirectional LSTM ?* URL : <https://medium.com/analytics-vidhya/what-does-it-mean-by-bidirectional-lstm-63d6838e34d9>. (accessed : 09.09.2022).

- [45] WIKIPEDIA. *word embedding*. URL : <https://fr.wikipedia.org/wiki/BabelNet>. (accessed : 06.05.2022).

# Annexes

# Tutoriel : Comment exécuté notre application sur *Google Colab*

Afin de lancé notre logiciel d'anti-plagiat voici les étapes à suivre, nous avons utilisé *Google Colab* pour exécuter la page Web qui utilise *FLASK*. Pour cela, on suive les étape suivantes :

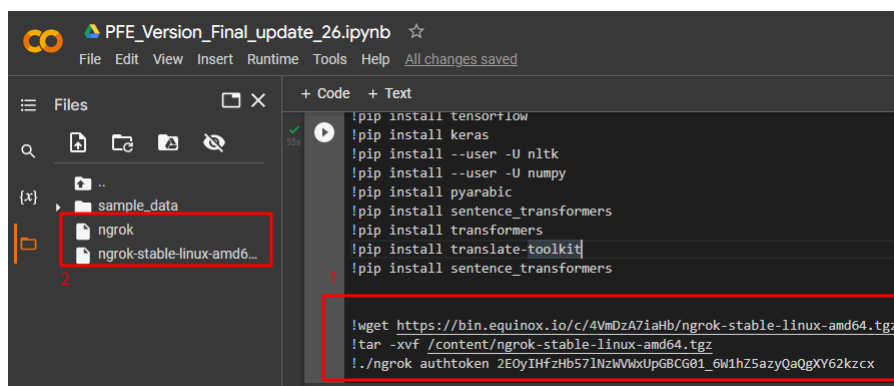


FIG. 12 : fenêtre pour connexion a flask

D'abord on exécute la première cellule du notebook, cette cellule contient des commandes *pip* pour installer les packages nécessaire, ce qui nous intéresse dans cette cellule, c'est le carte n°1, ce cadre contient les instructions pour installer les fichiers, packages et une clé d'authentification pour créer et gérer une session sur le serveur *ngrok*, son résultat c'est la génération des deux fichiers apparu dans cadre n°2, ces fichiers sont nécessaire dans l'application.

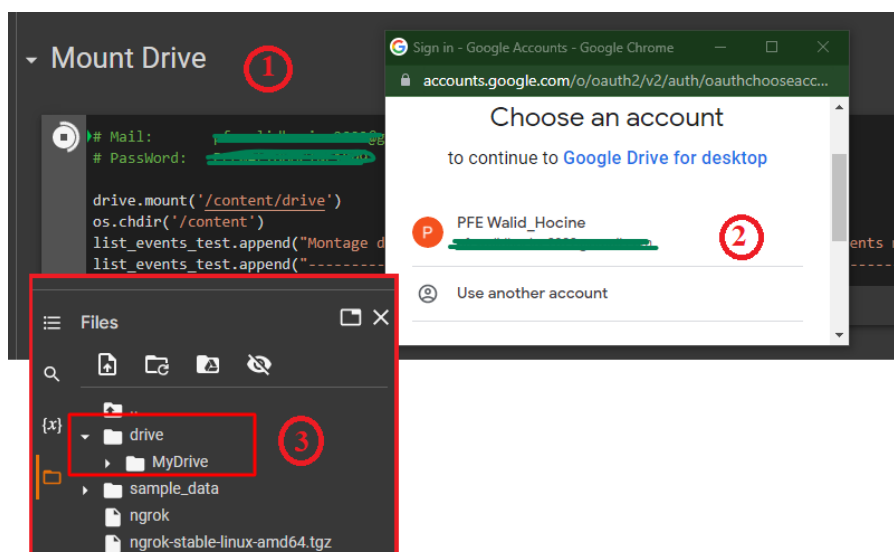


FIG. 13 : importer drive de compte connecté dans Google colab

En exécutant la cellule de connexion à *drive* une nouvelle fenêtre pop-up va apparaître et demande de confirmation que ce notebook veut se connecter au *drive*, en lui donnant l'adresse e-mail et le mot de passe en peut se connecter à *drive*, le résultat de l'exécution c'est apparition des nouvelle dossiers dans l'onglet gauche du *Colab*. Pour des raison de sécurité, on masque l'e-mail et le mot de passe dans ce mémoire.

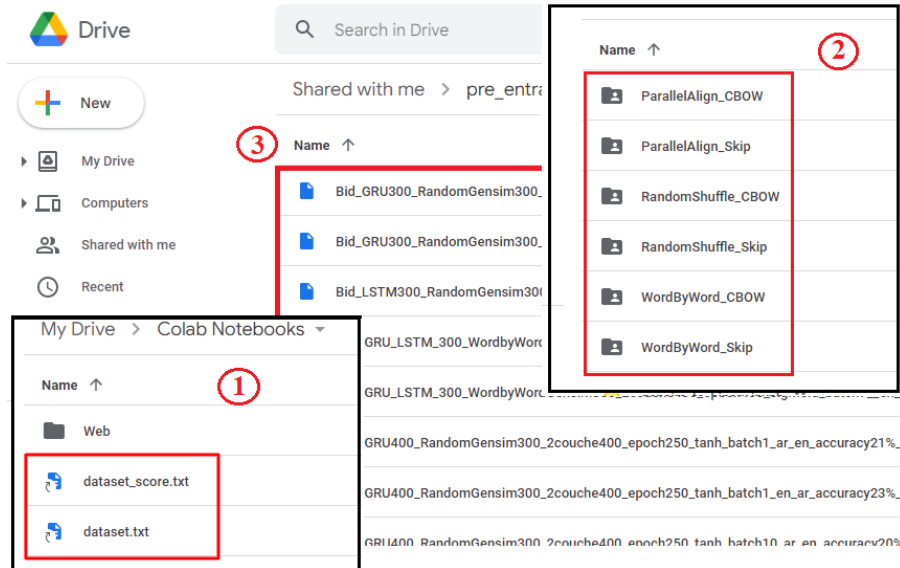


FIG. 14 : l'emplacement des modèles entraînés

Ici nous montrons le contenu de *drive* que nous avons utiliser, le cadre numéro 1 montre les fichiers de dataset STS, le numéro 2 montre les d'efférents architecture Word2Vec de *ArbEngVec*, et le numéro 3 montre les modèles de *deep learning* utilisés. La connexion du *drive* avec notre code de *Colab* va permettre un accès rapide à ces fichiers.

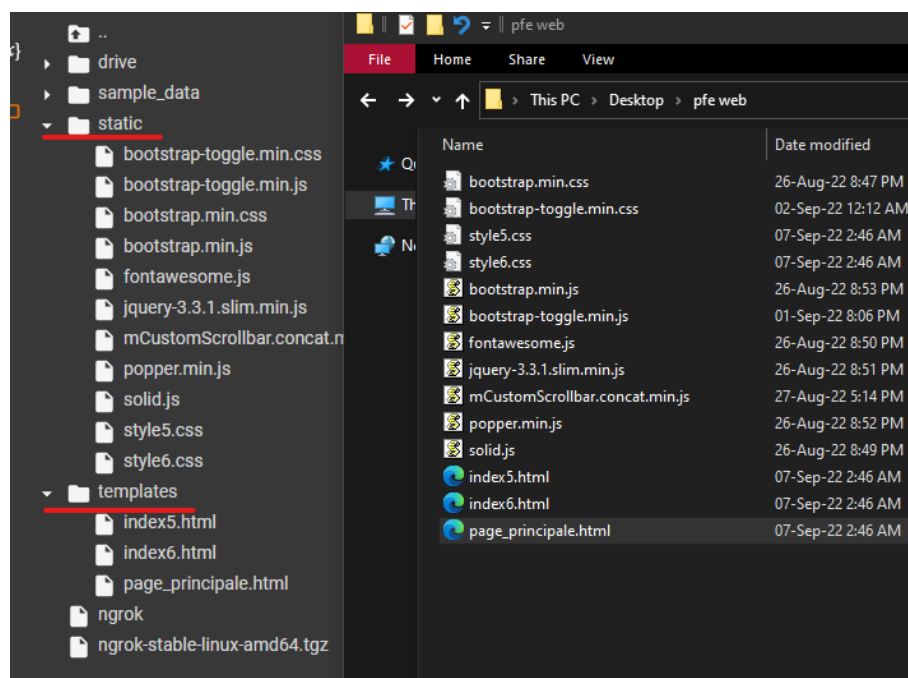
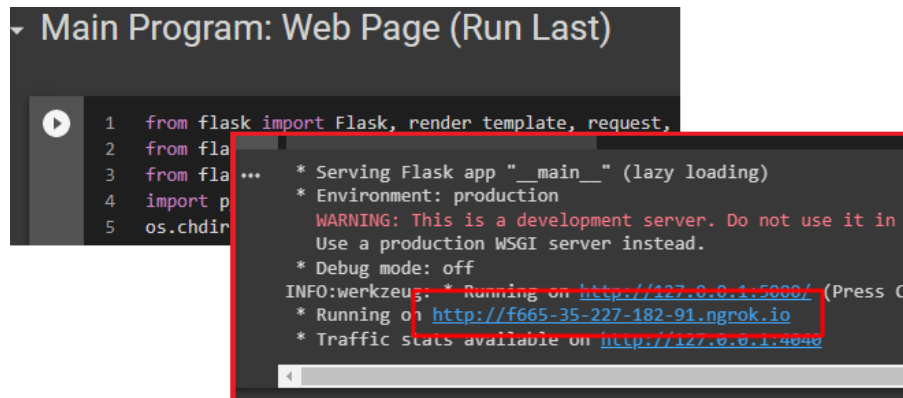


FIG. 15 : Distribution des fichiers HTML et *CSS* et *JS* dans les deux dossiers

Ensuite, c'est le temps de charger les fichiers associé avec ce projet, on doit créer deux dossiers nécessaire dans *Colab*, le premier dossier **static** qui contient le fichier 'static' qui ne change pas durant l'exécution, comme des fichiers *java script .js*, Les fichiers de styles *.css* ou des autres formats comme des images audio vidéo ...etc, le deuxième dossier s'appel **templates** ce dossier doit contenir des modèles de la page *HTML* qu'on exécute, dans la figure on montre comment nos fichiers sont distribués dans les deux dossiers.



```

Main Program: Web Page (Run Last)
1 from flask import Flask, render template, request,
2 from fla
3 from fla ...
4 import p
5 os.chdir

* Serving Flask app "__main__" (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in
Use a production WSGI server instead.
* Debug mode: off
INFO:werkzeug: * Running on http://127.0.0.1:5000/ (Press C
* Running on http://f665-35-227-182-91.ngrok.io
* Traffic Stats available on http://127.0.0.1:4040

```

FIG. 16 : Fenêtre montre le lien qui dirige vers notre système de anti-plagiat cross-langue

Après avoir exécuté tous les cellule de *Colab*, on se retrouve à la dernière, celle ci nous donne a son exécution un lien internet, ce lien (Qui est marqué dans le cadre en rouge) va permettre d'accéder directement à l'interface de notre application.

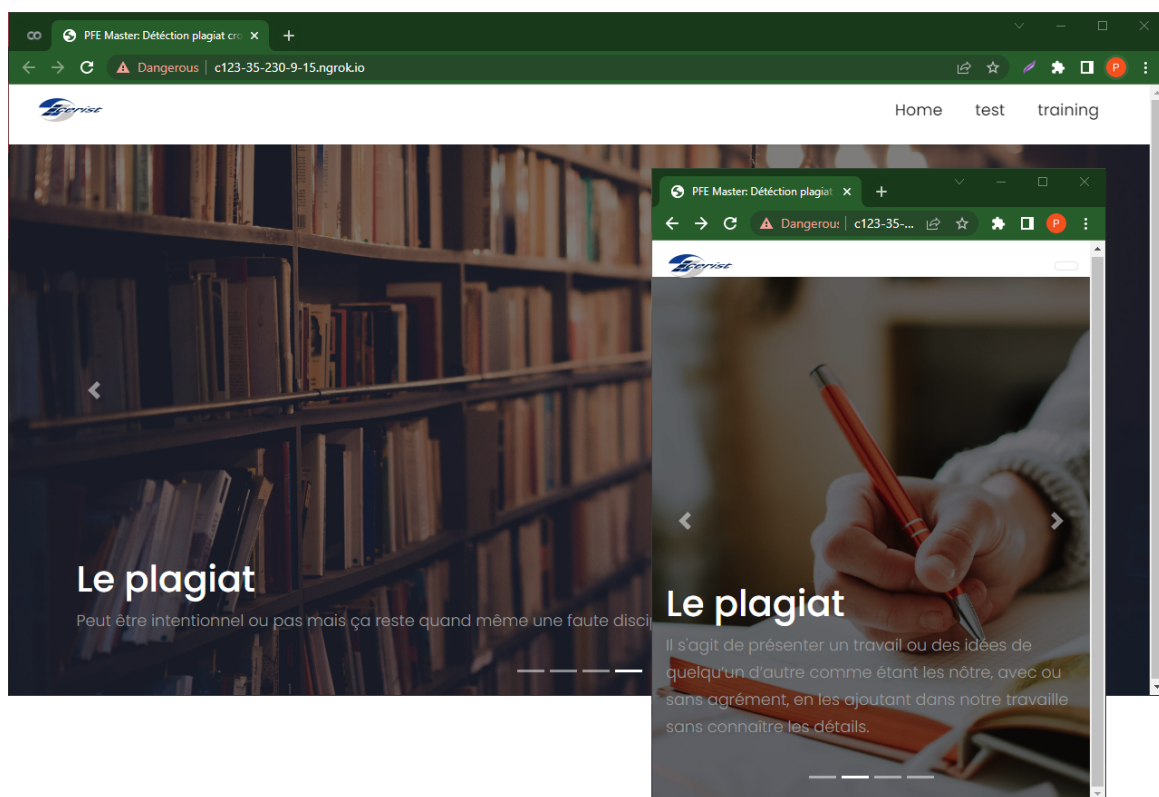


FIG. 17 : La page principale de notre logiciel

Voici notre application finale, lancé depuis le lien discuté, l'interface est facile a utiliser et contient des éléments de *UI* qui sont extensibles avec agrandissement de la fenêtre, c'est à dire



que cette interface fonctionne bien sur les smart-phones tablettes ou PCs, ce site est accessible par tous les utilisateurs.

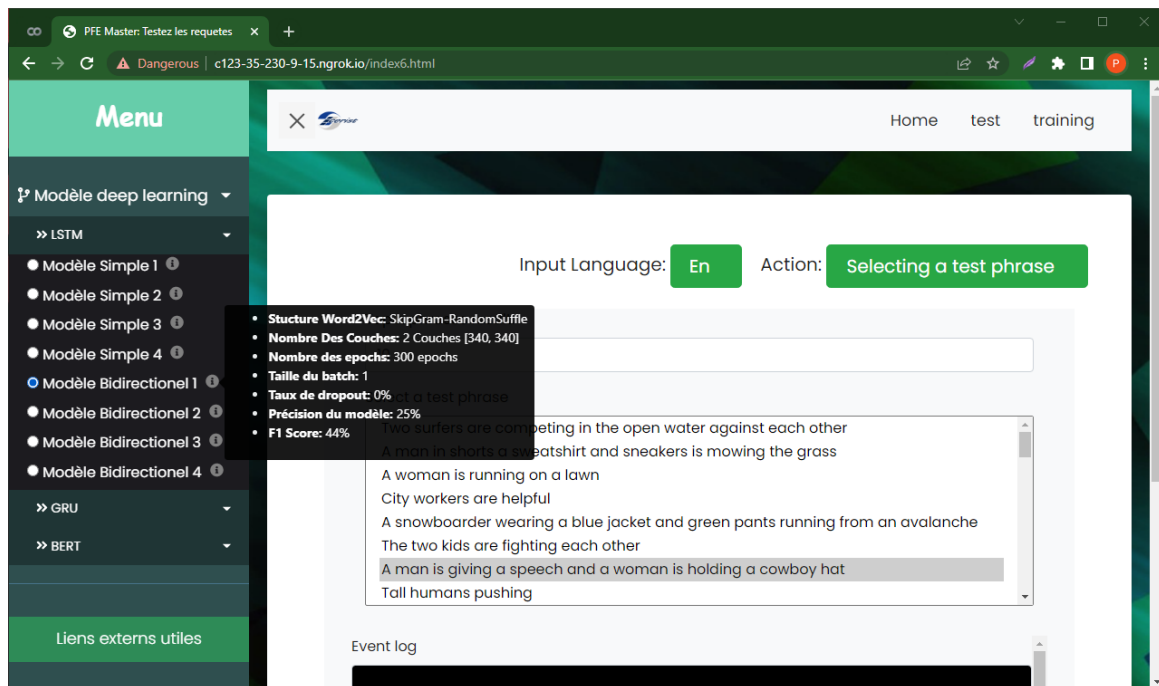


FIG. 18 : Fenêtre de choix d'un modèle plus la sélection d'une requête de teste

Ici on peut tester les différentes requêtes des testes, l'utilisateur doit choisir d'abord l'un des modèles déjà entraîné qui se trouve dans le volet gauche (On fournisse une description du modèle lorsqu'on point avec la souris on dessus). Ensuite, il choisit le nombre  $n$  souhaité pour retourner les  $n$  phrases les plus similaire à la requête du test, le  $n$  peut varie de 1 jusqu'à 500 (taille de dataset), Il a le droit de choisir une requête soit en langue Arabe ou en langue Anglais.

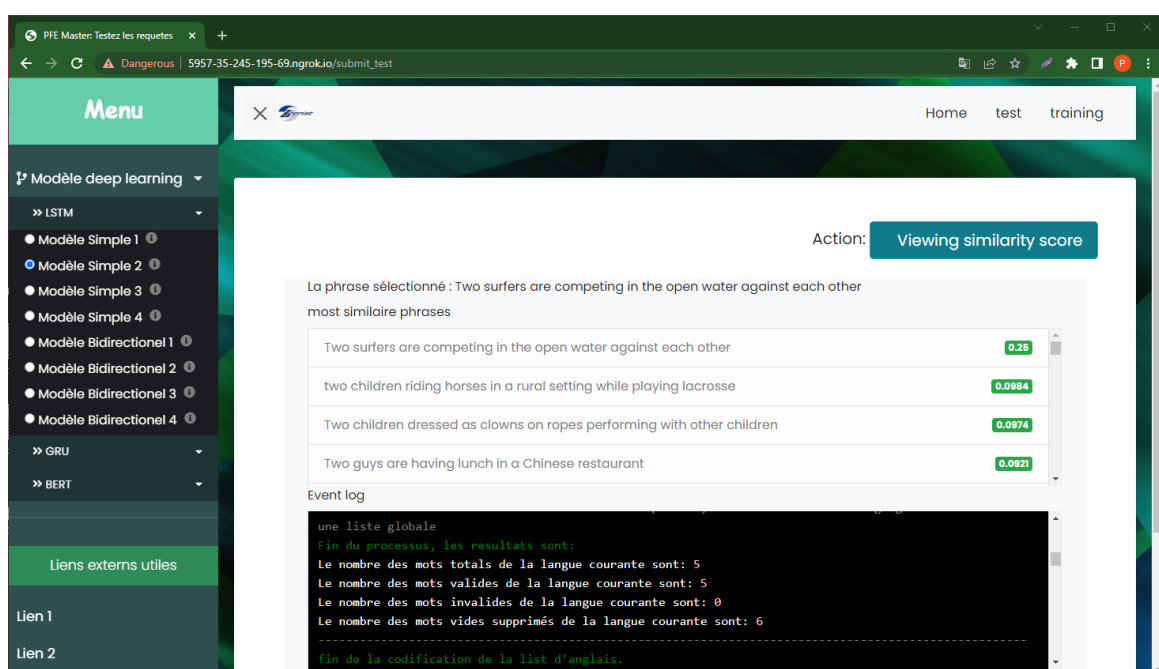


FIG. 19 : Fenêtre résultat d'une requête entré par utilisateur

Voici les résultats d'exécution d'une requête du test, les phrases retournés avec leur similarité sont triés à l'ordre décroissant suivant la valeur de similarité, on donne également une console qui contient tous les détails d'exécution, de pré-traitement à prédiction et jusqu'à la phase d'évaluation.

FIG. 20 : Fenêtre d'entraînement d'un modèle personnalisé

On donne ici une interface facile à utiliser, elle s'agit de remplir un formulaire qui contient des paramètres de construction de la structure du modèle et les hyper-paramètres qui caractérisent l'entraînement du modèle. L'utilisateur peut choisir l'un des modèles de deep learning *LSTM*, *GRU* ou *BERT* et l'un des modèles Word2Vec d'*ArbEngVec*, on note que les modèles Word2Vec fonctionnent uniquement avec *LSTM/GRU*, le modèle *BERT* a son propre **embeddings** qui est *Sentence2Vec*. Après la phase d'entraînement de ce nouveau modèle, l'utilisateur a le droit de tester sur les requêtes de tests (C'est comme dans l'interface précédente).