

UNIVERSITE LIBRE DES PAYS DES GRANDS LACS
FACULTE DES SCIENCES ET DES TECHNOLOGIES
APPLIQUEES

DEPARTEMENT DE GENIE ELECTRIQUE ET INFORMATIQUE



B.P 368 GOMA

www.ulpgl.net

**CONCEPTION ET REALISATION D'UN
SYSTEME BASE SUR L'INTELLIGENCE
ARTIFICIELLE POUR RESUMER
AUTOMATIQUEMENT LES TEXTES**

Par : **KRAME KADURHA DAVID**

Travail présenté en vue de l'obtention du Diplôme
d'ingénieur civil en génie électrique et informatique

Option : Génie informatique

Directeur : Prof BARAKA MUSHAGE Olivier

Encadreur : Msc KAMBALE WAMUHINDO

Abednego

ANNEE ACADEMIQUE 2021 – 2022

Dédicaces

A ma chère tante Catherine FEZA

Remerciements

Nous remercions en premier Dieu pour la guidance qu'il ne cesse de nous accorder. En second lieu, nos remerciements s'adressent à notre université, l'ULPGL, qui nous a encadré durant cette partie de notre quête des connaissances.

Nos remerciements les plus sincères alors à notre directeur, le Professeur BARAKA MUSHAGE Olivier, et notre encadreur, le Msc. KAMBALE WAMUHINDO Abednego, pour le suivi humble, patient et attentif qu'ils nous ont accordé durant ce travail.

Nous remercions aussi tous nos professeurs, en particulier ceux qui nous ont fortement inspiré, dont notre directeur ainsi que les professeurs Alain AKWIR, LOHALO Rostha et NKUMBI Mwamba.

Nos remerciements très reconnaissants à notre mère BALIBUNO NZIGIRE Charlotte et à notre oncle BALIBUNO Juvénal, pour le soutien tant moral, matériel qu'affectif qu'ils ne cessent de nous accorder.

Nous adressons également nos remerciements les plus sincères à la Sr Catherine FEZA, à Bijoux NAMWESI, BAKONDJO Pontien, Irène IRAGI, WILLY Miruho, BAENI Innocent, Labii KADURHA, Georgette KADURHA et nos oncles Marcellin BALIBUNO, Christian ACIZA CUBAKA, Akonkwa BALIBUNO, Bisimwa BALIBUNO, Mushagalusa ZAGABE, Aganze BALIBUNO et Bernardin BWIRABUCIZA pour le soutien.

Nous remercions aussi tous ceux dont l'existence contribue énormément à notre bien-être, pensant à Joëlle ANSIMA, Daniel KADURHA, Emmanuella MUSIMWA, Generose ABEKA, Florence CINAMA, Christelle MIRUHO, Patricia OLINABABO, Alohn SHETEBO, Carine SHETEBO, Christian RUBONEKA, Jehiel BABUYA, Jadhriel LUKOO, MUBARUZA Hugues, ILOMBE Bertrand, Wen KAMUNTU et René VUNINGA.

Nous remercions enfin nos amis et camarades, dont en particulier Styve EBASOMBA, Naomi MASASI, SOKI Claudette, KAMOLE Bruno, KAHASHI Yvan, Moise MUHESI, USHINDI Victoire, CIRHULWIRE Chrispin, Pierre MABILI, AMANI Lucien et tous les gens bienveillants que nous n'avons pas pu citer.

KRAME KADURHA DAVID

Résumé

A l'ère du numérique, le texte est l'un des principaux moyens de communication et surtout de transmission des savoirs. En 2018, c'était environ 80% des informations circulant sur le web. Mais, le temps étant une denrée rare, on voudrait avoir la possibilité d'accéder directement aux informations saillantes des textes, ou juste en avoir un aperçu global avant d'y consacrer du temps. D'où la nécessité d'un système de résumé automatique performant. Dans ce travail, notre objectif est de mettre en place un système de résumé automatique performant. Pour cela, nous avons considéré à la fois le résumé abstraktif et extractif, laissant le choix à l'utilisateur, selon ses besoins en information. Pour le résumé extractif, nous avons considéré deux approches qui ont consisté à utiliser un module python dénommé *gensim* pour la synthèse, puis un mélange de plusieurs algorithmes de synthèse que nous avons nommé *merging*. Pour le résumé abstraktif, nous avons opté pour les modèles du type *transformer* car ce sont eux qui ont atteint jusqu'à présent les performances les plus élevées pour diverses tâches de traitement du langage naturel. Néanmoins, nous avons servi les *transformers* utilisés pour la synthèse abstractive à travers un pipeline devant permettre l'amélioration des performances et l'augmentation du nombre de pages recevables. A l'issue de nos expérimentations, nous avons trouvé que l'algorithme *merging* était qualitativement plus performant que *gensim*. Nous avons également enregistré une nette amélioration des résultats fournis par les *transformers* une fois servis à travers le pipeline que nous avons proposé. Ainsi, nous avons obtenu un système capable de restituer les résumés abstratifs ou extractifs des textes. Pour cela, nous avons mis en place une interface de programmation des applications que les développeurs pourraient utiliser pour réaliser la synthèse automatique dans leurs systèmes respectifs, nous avons implémenté une application web qui exploite les services de cette interface mise au point et avons réalisé un modèle qui sera entrain d'être amélioré au fil de la collecte des couples texte-synthèse à travers notre système.

Mots-clés : Intelligence artificielle, Encodeur-décodeur, Transformer, résumé automatique des textes, Python, Flask, JavaScript, ReactJS, ExpressJS.

Abstract

In the digital age, text is one of the main means of communication and especially of transmitting knowledge. In 2018, it was about 80% of the information circulating on the web. But time being a scarce commodity, one would like to have the possibility to access directly the salient information of texts, or just to have an overview of it before spending time on it. Hence the need for a powerful automatic summarization system. In this work, our objective is to set up an efficient automatic summarization system. For this purpose, we have considered both abstractive and extractive summarization, leaving the choice to the user, according to his information needs. For extractive summarization, we considered two approaches which consisted in using a python module called *gensim* for synthesis, and then a mixture of several synthesis algorithms which we called *merging*. For the abstract summarisation, we opted for the *transformer* type models as they have so far achieved the highest performance for various natural language processing tasks. Nevertheless, we have served the *transformers* used for abstractive summarization through a pipeline to improve performance and increase the number of receivable pages. In our experiments, we found that the *merging* algorithm performed qualitatively better than *gensim*. We also recorded a clear improvement in the results provided by the *transformers* when served through the pipeline we proposed. Thus, we obtained a system capable of rendering abstractive or extractive summaries of texts. For this purpose, we have set up an application programming interface that developers could use to perform automatic summarization in their respective systems, we have implemented a web application that exploits the services of this developed interface and we have made a model that will be improved as we collect text-summary pairs through our system.

Keywords : Artificial intelligence, Encoder-decoder, Transformer, automatic text summarization, Python, Flask, JavaScript, ReactJS, ExpressJS.

Sigles et abréviations

ALBERT: A Lite BERT ,

ANN: Artificial Neural Network,

API: Application Programming Interface,

BART: Bidirectionnal and Auto-Regressive Transformer,

BBC : British BroodCasting Corporation,

BD : Base des Données,

BERT: Bidirectionnal Encoder Representations from Transformers,

CNN : Cable News Network,

CSS : Cascading Style Sheets,

DUC : Document Understanding Conference,

GPT : Generative Pretrained Transformer,

GRU : Gated Recurrent Unit,

HTML : HyperText Markup Language,

LCS : Long Common Sequence,

LDA : Latent Dirichlet Allocation,

LSA : Latent Semantic Analysis,

LSTM : Long Short-Term Memory,

MML : Masked Language Modelling,

NER : Named Entity Recognition,

NLP : Natural Language Processing,

NLU : Natural Language Understanding,

NSP : Next Sentence Prediction,

POS : Part Of Speech,

REST : REpresentationnal State Transfer,

RNN : Recurrent Neural Network,

ROBERTA: Robustly Optimized BERT Approach ,

ROUGE : Recall Oriented Understudy for Gisting Evaluation,

RST : Rhetorical Structure Theory,

SCU : Summary Content Unit,

SMS : Short Message Service,

SVD : Singular Value Decomposition,

T5 : Text-To-Text Transfer Transformer,

TAC : Text-Analysis Conference,

TF-IDF : Time Frequency Inverse Document Frequency,

ULPGL : Université Libre des Pays des Grands-Lacs,

XSum : Extreme Summarization,

mBART : multilingual BART,

mT5 : multilingual T5.

Table des matières

Dédicaces	i
Remerciements	ii
Résumé	iii
Abstract	iv
Sigles et abréviations	v
Introduction générale	1
0.1 Contexte	1
0.2 Identification et formulation du problème	1
0.3 Questions de recherche	2
0.4 Hypothèses de travail	2
0.5 Justification du choix du sujet et motivations	3
0.6 Objectifs de la recherche	4
0.6.1 Objectif général	4
0.6.2 Objectifs spécifiques	4
0.7 Méthodologie de recherche et délimitation du travail	5
0.8 Subdivision du travail	5
I Généralités sur le NLP	6
I.1 Introduction partielle	6
I.2 Présentation et définitions	6
I.3 Nécessité de l'approche par deep learning	8
I.4 Quelques techniques courantes de traitement des textes	9
I.4.1 La tokenisation (<i>tokenization</i>)	9
I.4.2 Les <i>stopwords</i> [1]	10

I.4.3	La racinisation (<i>stemming</i>)	10
I.4.4	La lemmatisation (<i>lemmatization</i>)	10
I.4.5	Reconnaissance d'entités nommées (<i>NER</i>) [1]	11
I.4.6	L'étiquetage morpho-syntaxique (<i>POS tagging</i>)	11
I.5	Approches du NLP	12
I.5.1	Les réseaux de neurones artificiels (<i>ANN</i>)	12
I.5.2	Les réseaux de neurones récurrents (<i>RNN</i>)	13
I.5.3	Mécanismes d'attention	20
I.5.4	Les transformers	22
I.6	Conclusion partielle	27
II	Résumé automatique et conception	29
II.1	Introduction partielle	29
II.2	Présentation et définitions	29
II.3	Catégorisation des résumés	30
II.3.1	Selon la fonction	31
II.3.2	Selon le nombre de documents sources	32
II.3.3	Selon le genre des documents	32
II.3.4	Selon le type de sortie (résumé obtenu)	33
II.3.5	Selon le type de résumeur	34
II.3.6	Selon le contexte	34
II.3.7	Selon le destinataire du résumé	35
II.4	Approches de résumé automatique	35
II.4.1	Techniques intuitives de résumé [2]	35
II.4.2	Algorithmes classiques de résumé automatique	38
II.5	Modèles Seq2Seq	45
II.5.1	Méthodes du Word-Embedding	45
II.5.2	Modèles séquence-à-séquence proprement dits	46
II.5.3	Modèle BART pour la synthèse abstractive	50
II.6	Conception de l'architecture globale de <i>Mon Résumeur</i>	51

II.6.1	Spécifications du système	52
II.6.2	Présentation des éléments du système	53
II.6.3	Architecture du module de synthèse extractive	55
II.6.4	Architecture du module de synthèse abstractive	58
II.6.5	Présentation des interfaces	59
II.7	Conclusion partielle	60
III	Conception détaillée, réalisation et tests	61
III.1	Introduction partielle	61
III.2	Conception détaillée	61
III.2.1	Conception de la base des données	62
III.2.2	Conception de l'API de synthèse	64
III.2.3	Conception des interfaces	67
III.3	Évaluation des résumés	70
III.3.1	Évaluation manuelle	71
III.3.2	Évaluation semi-automatique	72
III.3.3	Évaluation automatique	75
III.4	Description succincte de l'implémentation	75
III.4.1	API REST	76
III.4.2	Système client	78
III.5	Tests et interprétation des résultats	78
III.5.1	Datasets utilisés pour les tests	78
III.5.2	Modèles utilisés pour la synthèse abstractive	81
III.5.3	Tests	83
III.5.4	Résultats des tests	85
III.5.5	Interprétation des résultats	88
III.6	Justification des choix d'implémentation	90
III.7	Conclusion partielle	91
	Conclusion générale	93

Annexes	105
.1 Exemples des synthèses pour une évaluation qualitative du système	106
.2 Documentation sur les end-points utilisables de l'API	120
.3 Captures des interfaces d'authentification de l'API	127

Liste des tables

III.1 Résultats ROUGE des algorithmes de <i>merging</i> sur <i>CNN/DailyMail</i>	86
III.2 Poids des algorithmes constituant <i>merging</i>	87
III.3 Comparaison des algorithmes Gensim et <i>merging</i> sur <i>CNN/DailyMail</i>	87
III.4 Comparaison des algorithmes Gensim et <i>merging</i> sur <i>XSum</i>	87
III.5 Comparaison de BART avec BART optimisé sur <i>CNN/DailyMail</i>	87
III.6 Comparaison de BART avec BART optimisé sur <i>XSum</i>	88
III.7 Résultats de test de <i>BARTkrame-abstract</i> sur <i>for-ULPGL-Dissertation</i>	88
8 Tableau d'exemples de résumés	106

Liste des figures

I.1	Réseau de neurones artificiel simple [3]	13
I.2	Illustration de ce qu'est un RNN [4]	14
I.3	Comparaison entre cellules RNN classique et LSTM [5]	15
I.4	Vue fonctionnelle d'une cellule LSTM [5]	16
I.5	Cellule GRU [5]	19
I.6	Présentation des modèles encodeur-décodeur [6]	21
I.7	Architecture générique des <i>transformers</i> [7]	23
I.8	Vue éclatée d'un transformer [8]	27
II.1	Diagramme des fréquences des mots et le choix de Luhn [9]	40
II.2	Comparaison simplifiée entre BERT, GPT et BART [10]	50
II.3	Transformations de bruitage expérimentées pour BART [10]	51
II.4	Architecture globale de notre système	53
II.5	Pipeline de synthèse abstractive	54
II.6	Architecture globale du module de synthèse extractive	56
II.7	Architecture globale du système de synthèse abstractive	58
II.8	Architecture interne du modèle mentionné sur la figure II.7	58
II.9	Ébauche d'interface	59
III.1	Structure de la base des données	63
III.2	Interface générique du système	67
III.3	Interface durant le processus de synthèse	68
III.4	Interface après génération de la synthèse	68
III.5	Interface durant l'évaluation de la synthèse obtenue	69
III.6	Interface au survol du curseur de réglage de résumé	69
III.7	Interface réduite côté petits documents	70
III.8	Interface réduite côté gros documents	70

Introduction générale

0.1 Contexte

A l'ère du numérique, comme depuis l'invention de l'écriture, le texte est l'un des principaux moyens de communication et surtout, de transmission des connaissances.

Des livres aux SMS, en passant par diverses pages web, les données textuelles sont partout. En 2018, il s'agissait d'environ 80% de l'information qui circulait sur le web [11].

L'évolution de l'informatique continue à démontrer la possibilité de simplifier toujours grandement la vie de l'homme en automatisant de plus en plus l'accomplissement des tâches rébarbatives.

Certaines tâches comme celles liées explicitement à l'arithmétique semblent mieux se prêter à cette vague d'automatisation, les données numériques étant par essence celles prises en compte par les plateformes numériques.

Néanmoins, des transformations adéquates permettent de prendre en compte tout type de donnée, et le texte n'est pas exclu. C'est ainsi que, des avancées récentes en traitement automatique du langage naturel ont prouvé que le traitement du texte par l'ordinateur peut être raffiné autant qu'on veut, dans les limites du possible.

Cela est en fait une bonne nouvelle car, il s'avère que des nombreux sujets restent fermés à la majorité des gens suite au manque de temps, au regard de la quantité d'informations à consulter pour espérer avoir ne fût qu'une lueur d'idée du domaine ou du sujet qu'on veut rapidement explorer.

C'est en ce sens que la mise au point des technologies pouvant faciliter l'exploration des connaissances présentées sous forme textuelle est salvatrice.

0.2 Identification et formulation du problème

Comme présenté dans la section précédente, la voie la plus privilégiée pour transmettre les connaissances est l'**écriture**. Mais, admettons que souvent, dans un long texte, la

quantité d'information pertinente est moindre par rapport à la longueur du texte entier. Comment faire donc pour identifier cette partie utile et gagner ainsi en temps ?

Il est souvent inintéressant de passer du temps à lire des textes très longs, surtout quand on veut juste avoir une compréhension suffisante en peu de temps de ce qui est écrit, ou quand le sujet traité ne fait pas partie de notre domaine de prédilection. **Il est donc intéressant de mettre au point un système qui pourra assister l'homme dans la tâche de synthèse des connaissances** afin de promouvoir par là-même un échange entre disciplines, ce qui est souvent très enrichissant. Le système que nous mettrons au point durant ce travail, pour résoudre le problème ici présenté, sera nommé *Mon Résumeur*.

0.3 Questions de recherche

Vu le problème que nous venons de présenter, une question se pose :

Est-il possible de mettre au point un système informatique capable de synthétiser les textes avec une performance de niveau humain ?

La précédente question nous amène aussi à nous demander ceci :

- Un traitement purement linguistique ne pourrait-il pas nous permettre de générer des synthèses suffisamment bonnes pour atteindre notre objectif ?
- L'inclusion des traitements basés sur l'intelligence artificielle dans les modules de synthèse est-elle obligatoire pour atteindre des bonnes performances ?
- Quelle est l'architecture globale la plus adaptée pour réaliser un système de synthèse automatique performant ?

0.4 Hypothèses de travail

A la suite des questions que nous venons de soulever, nous postulons que :

- Vu la complexité du langage naturel, un traitement purement linguistique ne nous permettrait pas de mettre au point un système de niveau humain en synthèse des textes;

- Étant donné que, par définition, le langage naturel est difficile à formaliser complètement, on ne pourrait pas se passer de l'intelligence artificielle pour parvenir à réaliser un système performant;
- Une architecture basée essentiellement sur des modèles du type *transformer*, joint à l'utilisation de quelques règles inspirées de la linguistique permettrait d'avoir un système de synthèse performant.

0.5 Justification du choix du sujet et motivations

Pour synthétiser un texte, il faut l'avoir au moins lu! Et pourtant, pour lire un texte, il faut du temps, une denrée souvent rare.

Certains textes sont souvent fournis, accompagnés des synthèses qui sont parfois très bonnes, parfois incomplètes et parfois même très polarisées ou tout simplement mauvaises. Toutefois, avoir une synthèse à la demande serait mieux que de ne trouver que des synthèses de certains textes, sans d'ailleurs en avoir le plus souvent besoin. Nombreux sont des textes (livres, articles, pages web et autres documents) dont on voudrait avoir des bonnes synthèses, qu'on ne trouve que très rarement si on ne s'est pas découragé avant.

C'est la raison pour laquelle, **nous nous sommes fixé comme objectif de répondre à ce besoin précis en mettant au point une application web de synthèse des textes.**

Beaucoup de chercheurs en linguistique et en traitement automatique du langage naturel principalement se sont penchés sur ce sujet [11, 12, 13, 14, 15].

Des solutions ont été proposées mais ne sont pas toujours à la hauteur de nos attentes (mettre au point un système de performance presque humaine en synthèse automatique des textes). Les plus prometteuses de ces solutions se limitent à des tailles bien réduites de texte, ce qui est déjà un grand pas mais pas suffisant évidemment. C'est pour cette raison qu'il nous semble pertinent d'étudier cette question en profondeur et de **mettre au point un système complet et utilisable en dehors du monde de la recherche.**

Socialement, la mise au point de ce système sera d'une très grande importance. Cela dans plusieurs axes dont principalement :

- Pour les chercheurs, car il pourra faciliter le survol rapide des connaissances provenant

des filières liées à leurs domaines, sans être obligés de consulter à l'avance un tas de documents issus de ces domaines connexes;

- Pour tout le monde alors, le système pourra permettre un gain de temps considérable chaque fois qu'il donnera la possibilité d'avoir accès à une synthèse de bonne qualité à la demande, en un temps raisonnable.

0.6 Objectifs de la recherche

0.6.1 Objectif général

Cette recherche a pour objectif principal de concevoir et réaliser un système (une application web) qui permettra la génération automatique des résumés des documents.

0.6.2 Objectifs spécifiques

Pour arriver à bout de notre projet nous comptons :

- Évaluer les failles et limites des techniques de synthèse automatique existantes;
- Corriger les failles ou compléter les techniques de synthèse automatique existantes;
- Établir des architectures logiques optimales pour obtenir des synthèses de qualité;
- Élaborer une interface de programmation d'applications devant faciliter l'accès au service de synthèse automatique;
- Mettre au point une base de données pour stocker les synthèses les mieux cotées par les usagers, en prévision d'une amélioration future du système;
- Réaliser une interface web de qualité pour permettre l'accès au service par divers utilisateurs.

0.7 Méthodologie de recherche et délimitation du travail

Pour la mise au point du système, nous comptons utiliser les méthodes d'analyse moyennant les techniques expérimentale (pour vérifier l'adéquation du fonctionnement de l'application mise sur pied avec le problème posé), et documentaire (pour une vision approfondie des techniques couramment utilisées et d'éventuelles améliorations nécessaires).

Ce travail se focalise sur la synthèse des documents du type informationnel (livres historiques, discours, articles de presse, lettres, nouvelles, romans et tout autre type de document ayant une faible densité d'expressions mathématiques) et il s'agira d'une synthèse mono-document **en langue française**.

0.8 Subdivision du travail

Excepté l'introduction et la conclusion générales, ce travail est ainsi constitué :

1. Au premier chapitre, *Généralités sur le traitement automatique du langage naturel*, nous passons en revue toute la théorie nécessaire à la compréhension de notre travail.
2. Au second chapitre, *Présentation du résumé automatique et conception de l'architecture du système*, nous y présentons les aspects du résumé automatique essentiels à notre travail et y concevons pas à pas le système de synthèse automatique des textes dans tous ses aspects (pas uniquement le côté synthèse).
3. Au troisième chapitre : *Conception finale, réalisation et tests*, nous y finalisons la conception et expliquons les points importants de l'implémentation en nous basant sur la conception faite, puis nous présentons les résultats des tests que nous avons effectués.

Chapitre I

Généralités sur le traitement automatique du langage naturel

I.1 Introduction partielle

Dans ce chapitre, nous allons présenter brièvement le traitement automatique du langage naturel, ainsi que les techniques de traitement qui seront utiles pour la réalisation de l'objectif principal de ce travail. Nous allons donc y présenter une vue d'ensemble des architectures généralement utilisées, en nous focalisant essentiellement sur l'aspect intelligence artificielle du *NLP* (*Natural Language Processing*).

Dans un premier temps, nous y présentons quelques techniques, souvent incontournables lorsqu'on veut réaliser une tâche de traitement du langage. Après cela, nous parcourons divers modèles qui nous permettront d'aborder le modèle le plus adapté à la tâche de synthèse automatique des textes, qui est l'objectif de ce travail.

I.2 Présentation et définitions

Le NLP est une discipline rattachée à l'intelligence artificielle et ayant pour principal objectif, l'étude des possibilités du traitement du langage humain par des machines. La raison pour laquelle la discipline s'inscrit comme faisant partie du domaine d'intelligence artificielle est que le langage est considéré comme étant une aptitude centrale de l'intelligence humaine, étant donné que l'usage d'un langage si complexe est l'un des éléments distinctifs principaux entre humains et autres animaux.

Le NLP inclut l'ensemble d'algorithmes, des tâches et des problèmes prenant en entrée

des textes produits par des humains, pour finalement ressortir des informations pertinentes à propos de ces derniers ou alors du texte modifié de façon appropriée selon l'objectif poursuivi. C'est ainsi que des tâches comme la traduction automatique, la génération automatique des textes ou aussi la synthèse automatique qui va nous intéresser dans ce travail, produisent directement du texte en sortie.

Mais, dans tous les cas, la sortie est soit immédiatement utilisable, soit alors elle est prise comme entrée d'un autre système dans la chaîne de traitement du texte.

On peut toutefois se demander la raison pour laquelle on parle de traitement automatique du "langage naturel" (quitte à se demander ce qui distinguerait un langage naturel des autres langages).

Pour établir clairement cette différence, il est nécessaire de donner une définition de ce qu'est un langage formel. Pour caricaturer, un langage formel est celui pour lequel il existe un mécanisme fini, et explicite, permettant d'en faire une analyse, quand bien même il serait constitué d'un nombre infini de mots. Donc, c'est un ensemble de mots analysable par un automate (au sens mathématique du terme) [16].

On peut donc comprendre directement que le mot "naturel" est ici utilisé pour faire une distinction avec *les langages formels*. C'est donc dans ce sens que toutes les langues parlées peuvent être vues comme des langages naturels. Les langages formels ont une syntaxe précise et sont spécifiquement conçus pour des objectifs bien cernés (penser à tous les langages de programmation par exemple). Ils sont donc très précis tant au point de vue syntaxique que sémantique.

Concernant les langues humaines usuellement utilisées, on ne peut pas dire, sans être démenti, qu'elles sont dénuées d'imprécisions. Elles regorgent en générale une grande richesse, ce qui a pour conséquence d'introduire très souvent une grande ambiguïté. Pour s'en convaincre, il suffirait par exemple de considérer la phrase suivante :

Je le vois avec mes jumelles.

Très vite on remarque que cette phrase peut s'interpréter selon le contexte. On ne sait pas, en effet, si le sujet affirme voir quelqu'un avec ses jumelles d'observation, se promenant avec ses enfants jumelles, ou si le sujet voit quelque chose en utilisant ses jumelles en tant

qu'instrument.

Ceci n'est qu'un exemple particulier pour illustrer cette dichotomie inhérente à l'emploi de la langue quelle qu'elle soit, mais cela suffit pour qu'on s'aperçoive que le problème est bel et bien réel.

Ce n'est d'ailleurs pas juste au niveau des interprétations qu'on peut identifier ce problème. Il s'observe même quand on considère les règles de grammaire. Certaines règles sont ainsi admises par certains linguistes mais rejetées ou trouvées superflues par d'autres [17].

C'est tout ce qui précède qui rend le langage humain à la fois riche et challengeant quand il s'agit de doter les machines de cette aptitude. D'où la raison d'être d'une discipline à part entière dédiée à la mise au point des règles de traitement du langage naturel, le NLP [18].

I.3 Nécessité de l'approche par deep learning

Avant l'avènement du *deep learning*, des techniques traditionnelles du NLP étaient utilisées pour des tâches comme la détection des spams, l'analyse des sentiments et le POS (Part Of Speech tagging). Ces approches utilisaient essentiellement des caractéristiques statistiques des séquences comme, la fréquence des mots et les co-occurrences par exemple. Néanmoins, le principal désavantage de ces techniques était qu'elles ne parvenaient pas à capturer une grande partie de la complexité linguistique du langage humain, comme par exemple le contexte.

Ainsi, les développements, récents d'ailleurs, des réseaux de neurone et du *deep learning* ont donné des nouveaux outils, pour approcher dans une large mesure les performances humaines en terme de traitement de langage. A notre avis, ces techniques sont les plus adaptées car, tout d'abord elles se rapprochent beaucoup plus des méthodes de traitement d'information par le cerveau humain, et ensuite, il serait autrement très coûteux, voir impossible, d'élaborer des modèles capables d'embrasser toute la complexité du langage humain.

Le *deep learning* pour le NLP est axé grosso-modo sur la représentation d'entités textu-

elles et le traitement élaboré sur ces représentations, de manière à en tirer des informations pertinentes ou à réaliser des transformations appropriées. Cette représentation constitue d'ailleurs un problème fondamental car c'est d'elle que dépend toute la chaîne de traitement des systèmes de NLP [19].

I.4 Quelques techniques courantes de traitement des textes

Dans cette partie, nous allons présenter diverses techniques intervenant dans le traitement des données de langage naturel. Ces traitements seront présentés de manière à dégager un *pattern* presque récurrent en terme de structure de traitement pour divers systèmes de NLP. Pour cela, nous allons d'abord présenter certaines manipulations réalisées sur les données en guise de pré-traitement. Puis, nous évoquerons deux techniques utiles aux tâches relevant du *NLU* (*Natural Language Understanding*).

I.4.1 La tokenisation (*tokenization*)

Manipuler des longues chaînes de caractères ne serait pas envisageable. Mais en informatique on est habitué à traiter des structures en terme de listes, de tableaux, de vecteurs,... Le tout étant représenté numériquement. C'est pour cela que l'opération consistant à réduire un corpus de texte en ses *tokens* est centrale.

Dans notre contexte, la tokenisation est une opération qui consiste à décomposer un texte (une suite de phrases) en ses phrases constitutives ou une phrase en ses mots constitutifs. Cela est une première étape pour diminuer la difficulté inhérente au traitement des textes. En considérant la décomposition en mots, pour diminuer au maximum les difficultés de traitement et l'ambiguïté, on ajoute à la tokenisation d'autres traitements qui sont en général : **la désaccentuation**, **le passage aux minuscules**, **la suppression des *stopwords***, **la racinisation** et **la lemmatisation** appliqués aux tokens obtenus [20].

I.4.2 Les *stopwords* [1]

Les *stopwords* sont, pour une langue donnée, des mots qui permettent de réaliser des phrases correctes mais qui n'apportent pas directement d'information significative sur l'ensemble (du point de vu traitement). Il s'agit par exemple en français de mots comme *de, la, le,...* ce qui correspond, en gros, aux prépositions, aux articles, aux conjonctions,... Il faut néanmoins préciser qu'on peut très bien décider de ne pas supprimer certains *stopwords*.

I.4.3 La racinisation (*stemming*)

La racinisation ou *stemming* en anglais consiste à découper le token de manière à n'en conserver qu'une partie qui semble rendre mieux compte de ce dont dérive ledit token. Seulement, ceci est fait sans se fier à ce que le résultat obtenu en tant que racine fasse partie du dictionnaire de la langue considérée [1, 20].

Cela permet juste de maximiser la probabilité de confondre des mots semblables qui sont présentés différemment dans diverses phrases. C'est à des fins de comparaison de phrases et de réduction d'ambiguïté. Pour illustration, on voudrait par exemple que si on retrouve les éléments "manger", "mange", "mangeable", "mangeons" dans un corpus, qu'ils soient transformés en un seul terme "mange". Cela se fait en découpant tous les mots qui ajoutent d'autres affixes au terme. C'est cela en bref le *stemming* et, contrairement à ce que le nom suggère, il ne s'agit pas exactement de trouver la racine des mots (les mots dont ils dérivent). L'opération consiste essentiellement à réaliser un découpage des mots de manière à en supprimer les affixes.

I.4.4 La lemmatisation (*lemmatization*)

La lemmatisation quant à elle est une opération plus soignée mais plus coûteuse en terme d'implémentation [1, 20]. Elle réalise en fait ce qui n'est pas réalisé par le *stemming* en ce sens que lemmatiser un token consiste à le transformer en sa racine, et cette dernière doit être présente dans le dictionnaire. Par exemple, pour un mot au pluriel, il s'agira de le remplacer par son singulier, un verbe conjugué, par son infinitif,... Pour illustration, la

lemmatisation consisterait à transformer par exemple "va", "allions", "irons" et "allé" par "aller" et "une", "des" par "un"... Cette tâche est grandement facilitée par des techniques de deep learning.

L'obtention des tokens peut également conduire à des tâches plus élaborées comme la **détection des entités nommées** et **l'étiquetage morpho-syntaxique**. Il s'agit des tâches très importantes que nous devons nécessairement mentionner.

I.4.5 Reconnaissance d'entités nommées (NER) [1]

La détection des entités nommées (*Named Entity Recognition* ou **NER**) consiste à repérer tout ce qui correspond à des noms de personnes, des noms d'organisations ou d'entreprises, des noms de lieux, des quantités, des distances, des valeurs, des dates ou tout autre élément qui constitue une nomination d'une entité existante précise dans un texte donné. Cette tâche est visiblement très importante dans la phase d'interprétation des données textuelles et il s'agit d'un simple problème de classification.

I.4.6 L'étiquetage morpho-syntaxique (*POS tagging*)

Le *Part-Of-Speech tagging* est une tâche consistant en gros, à associer aux éléments des textes, des informations grammaticales. En général, il s'agit d'associer aux termes des textes, leur nature grammaticale. Cela consisterait à dire que tel élément est un nom, tel autre un verbe,...[1, 20]

Cette tâche n'est pas une fin en soi. En effet, c'est une première étape dans l'analyse structurelle des textes, permettant de déduire diverses dépendances du point de vue linguistique. Elle est fortement facilitée par des approches basées sur le *deep learning* comme c'est le cas aussi pour la reconnaissance d'entités nommées.

Nous allons passer sous silence certains autres concepts du NLP comme le **sacs de mots** et le *word embeddings* dont nous parlerons dans la partie qui va suivre et qui présentera le résumé automatique, en tant que tâche du NLP.

I.5 Approches du NLP

Comme cela a été maintes fois mentionné, deux approches majeures sont d'usage pour traiter automatiquement les données de langage naturel. Il s'agit de l'approche numérique et de l'approche symbolique ou linguistique. Mais les deux approches sont dans la majorité des cas complétées par certaines heuristiques [21].

En ce qui nous concerne, l'approche sera essentiellement numérique avec un penchant prononcé pour les techniques du deep learning. D'ailleurs, concernant ces dernières techniques, les modèles de l'état de l'art les plus adaptés sont les *transformers* et leur présentation exige une revue chronologique car en effet, pour y arriver, des modèles classiques basés sur des réseaux de neurones récurrents (*RNN*) ont été utilisés car plus adaptés aux données séquentielles que sont les textes. Ensuite, le constat de leur mémoire limitée a fait à ce qu'on les modifie pour obtenir des unités à mémoire plus large dont les *LSTM*(*Long Short-Term Memory*) et les *GRU*(*Gated Recurrent Unit*). Furent ensuite introduits les mécanismes d'attention qui améliorèrent les techniques, aboutissant finalement aux modèles dits *transformers*, plus adaptés à des tâches de NLP élaborées.

I.5.1 Les réseaux de neurones artificiels (*ANN*)

Les réseaux de neurones artificiels (*Artificial Neural Network* ou *ANN*) sont un ensemble de neurones (artificiels) assemblés pour résoudre des tâches considérées comme requérant une certaine intelligence. Le neurone artificiel est un algorithme élaboré en s'inspirant du modèle théorique simplifié d'un neurone naturel. Il s'agit essentiellement d'une fonction d'agrégation ayant pour rôle de réaliser une somme pondérée des entrées qui lui sont présentées et d'une fonction d'activation qui formate la sortie de la fonction d'agrégation selon les valeurs attendues en sortie [22].

Les neurones sont généralement assemblés par couche comme présenté sur la figure qui suit :

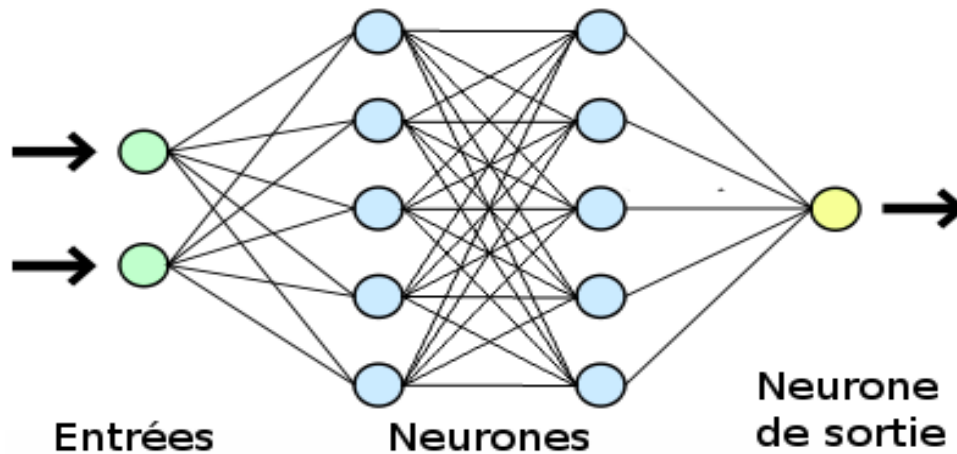


Figure I.1: Réseau de neurones artificiel simple [3]

Ce qui vient d'être présenté est suffisant pour avoir une idée globale de ce qu'est réellement un réseau de neurones artificiel. Néanmoins, nous pousserons plus loin pour toucher le plus vite possible aux modèles qui nous intéressent dans ce travail.

I.5.2 Les réseaux de neurones récurrents (RNN)

Un *RNN* (*Recurrent Neural Network*) est un type de réseaux de neurones conçu en principe pour traiter les données séquentielles, comme les données textuelles,...

La principale différence structurelle entre les *ANN simples* et les *RNN* est l'existence des connexions de récurrence dans ces derniers. Il s'agit des boucles permettant la prise en compte des sorties passées dans le traitement final des données [6].

Pour l'illustrer, rien de mieux qu'une image représentant la structure fonctionnelle des réseaux de neurones récurrents :

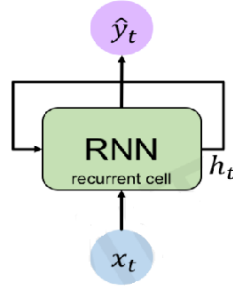


Figure I.2: Illustration de ce qu'est un RNN [4]

Où x_t , h_t et \hat{y}_t (à nommer juste y_t) représentent respectivement les entrées, les états internes qui en résultent et les sorties (c'est-à-dire x , h et y à chaque pas temporel t).

Pour une meilleure compréhension, une présentation formelle serait plus commode :

Soient W_x la matrice des poids associée au vecteur d'entrée x , W_y une matrice associée au vecteur de sortie y et W_h celle associée au vecteur représentant les états cachés du réseau, avec b_h et b_y respectivement les vecteurs des biais des neurones pour l'état caché et pour la sortie. On aura alors [23] :

$$\begin{cases} h_t = f_{act}(W_x x_t + W_h h_{t-1} + b_h) \\ y_t = g_{act}(W_y h_t + b_y) \end{cases} \quad (I.1)$$

On voit très bien que la sortie du système dépend non seulement de l'entrée, mais aussi de l'état du système (h).

Les fonctions d'activation f_{act} et g_{act} qui sont mentionnées dans les équations I.1 représentent respectivement la *tangente hyperbolique* \tanh et la fonction dite *softmax* [23].

L'entraînement des réseaux de neurones récurrents se fait de la même façon que pour les réseaux de neurones simples (avec uniquement une différence due au fait que pour le RNN on prend en compte le temps). On n'entrera pas dans le détail, vu que ce n'est pas exactement le sujet du travail mais, pour entamer la partie qui suit, il nous faut préciser que, comme pour les réseaux de neurones simples, l'entraînement exige d'appliquer une fonction de différentiation sur l'erreur produite par le système. Il s'agit de la fonction gradient. Mais, comme ici le gradient tient compte des grandeurs précédentes dans le temps, il y a un certain nombre de termes multiplicatifs qui peuvent amener le modèle

à ne jamais converger ou au contraire, à la saturation. C'est le problème classique d'évanouissement (disparition) des gradients ou d'explosion des gradients [23].

En réponse au problème de disparition des gradients, les cellules *LSTM* (*Long Short-Term Memory*) sont utilisées en lieu et place des cellules *RNN* normales.

Les cellules LSTM

Les cellules *LSTM* (pour *Long Short-Term Memory*) sont utilisées en lieu et place des cellules *RNN classiques* (dites *vanilla*) pour permettre au réseau de traiter des séquences de plus en plus longues sans perte rapide d'information [6]. Pour cela, des éléments de contrôle de la mémoire de la cellule sont ajoutés. Pour illustrer nos propos, voici une image qui nous permettra de différencier une cellule *RNN* classique d'une cellule *LSTM* :

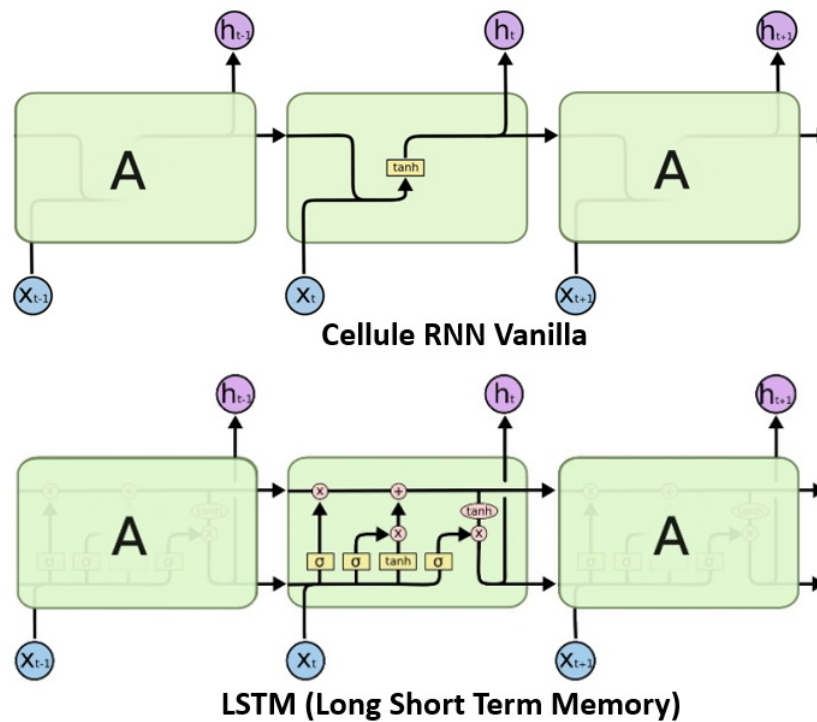


Figure I.3: Comparaison entre cellules *RNN* classique et *LSTM* [5]

Présentée comme cela, la cellule *LSTM* semble superflue mais si on présentait les équations associées à un réseau fait de ces cellules, on se rendra compte que c'est plutôt intuitif. Pour aborder les équations associées, considérons l'image suivante :

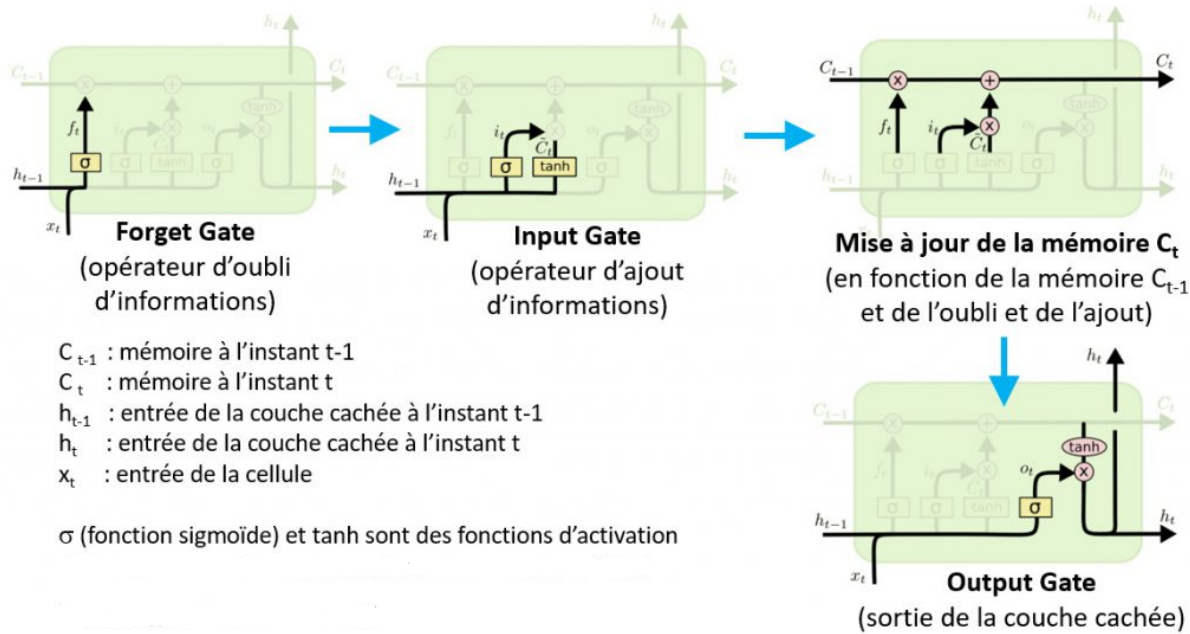


Figure I.4: Vue fonctionnelle d'une cellule LSTM [5]

Une cellule *LSTM* se comprend en la considérant comme constituée d'un ensemble de portes avec des fonctions bien particulières. Il s'agit d'une **porte d'entrée**, une **porte d'oubli** et une **porte de sortie**.

Il est évident que, pour chacune de ces portes que nous nommerons, à un instant t donné respectivement par I_t , F_t et O_t , le système doit apprendre ses paramètres en fonction de l'entrée et de l'état interne. Mais on doit aussi remarquer que, l'état est défini par deux paramètres au lieu d'un seul comme pour les RNN simples. Il s'agit, à un instant t donné, de h_t (considéré comme état à court terme) et de c_t (qui est un état à long terme mais dont le contenu est contrôlé, au vu de l'architecture de la cellule).

De ce que nous venons de dire, nous pouvons conclure que F_t , I_t et O_t sont des fonctions de X_t et de h_{t-1} **aux poids près**. On sait aussi que, si on veut une mémoire à long terme contrôlée, la valeur finale de c_t doit être mise à jour en repérant ce qui doit être oublié parmi les éléments qui étaient précédemment dans la mémoire, pour y ajouter ensuite ce qui est sélectionné comme pertinent à l'entrée. Cela revient à utiliser F_t et I_t comme des portes de contrôle (ou de sélection). Et de cela on peut conclure que c'est plus intéressant d'avoir F_t et I_t qui prennent des valeurs entre 0 et 1 (pour modéliser la sélection) et c_t devra dépendre de ces deux éléments, avec aussi l'état précédent de la mémoire à long

terme.

Il est aussi vraisemblable que, l'état à court terme doit provenir de la mémoire à long terme (ça correspondra à une sélection de ce qui doit être pris en compte directement dans la mémoire à long terme). Cet état h_t doit par conséquent dépendre de c_t (il faut néanmoins noter qu'une autre approche serait possible ici, mais celle-ci est déjà pertinente). Finalement, on sait que la sortie finale doit nécessairement dépendre de l'état interne de la cellule. Il va ici s'agir de h_t vu que la cellule est développée par analogie avec le processus de mémorisation des systèmes naturels (mémoire à court terme correspondant à la mémoire de travail).

De ce qu'on vient de dire on peut tirer que, fondamentalement on doit avoir :

$$\begin{cases} F_t &= \mathcal{F}(X_t, h_{t-1}) \\ I_t &= \mathcal{G}(X_t, h_{t-1}) \\ O_t &= \mathcal{J}(X_t, h_{t-1}) \\ c_t &= \mathcal{K}(c_{t-1}, X_t, h_{t-1}) \\ h_t &= \mathcal{L}(c_t) \\ y_t &= \mathcal{M}(h_t) \end{cases} \quad (\text{I.2})$$

Avec $\mathcal{F}, \mathcal{G}, \mathcal{J}, \mathcal{K}, \mathcal{L}, \mathcal{M}$ des fonctions dépendant des coefficients considérés (poids et/ou éléments de sélection qui sont les diverses portes définies).

Une implémentation classique de ce raisonnement se présente comme suit [6, 23] :

$$\begin{cases} F_t &= \sigma(W_{fx}X_t + W_{fh}h_{t-1} + b_f) \\ I_t &= \sigma(W_{ix}X_t + W_{ih}h_{t-1} + b_i) \\ O_t &= \sigma(W_{ox}X_t + W_{oh}h_{t-1} + b_o) \\ c_t &= F_t \circ c_{t-1} + I_t \circ \tanh(W_{cx}X_t + W_{ch}h_{t-1} + b_c) \\ h_t &= O_t \circ \tanh(c_t) \\ y_t &= W_{yh}h_t + b_y \end{cases} \quad (\text{I.3})$$

Il faut remarquer qu'on a utilisé la fonction sigmoïde σ pour restreindre les valeurs des sélecteurs (portes) entre 0 et 1, puis on a utilisé le *produit de Hadamard* (produit terme à terme des matrices) pour réaliser effectivement la sélection grâce aux portes, en diminuant les termes dont les valeurs correspondantes des portes sont proches de 0 et en essayant de conserver ceux dont les valeurs correspondantes des portes sont proches de 1.

Cette implémentation peut être modifiée, surtout en ce qui concerne les fonctions d'activation utilisées (σ et \tanh), et en particulier la fonction d'activation de finalisation \tanh ici, mais c'est l'une des plus optimales.

Le seul problème qui demeure est que le nombre de termes à apprendre est très grand. Cela a fait à ce qu'on puisse essayer de le diminuer en implémentant le **GRU** (*Gated Recurrent Unit*) poussant un peu plus loin l'abstraction des portes pour diminuer le nombre de paramètres.

Les cellules GRU

Les cellules *GRU* (*Gated Recurrent Unit*) sont une autre implémentation des cellules des réseaux de neurones récurrents comme les *LSTM* à la différence près que, bien que partant de la même idée fondamentale évoquée précédemment, les *GRU* apparaissent comme une simplification des *LSTM*.

Elles possèdent néanmoins des performances comparables en ce qui concerne la prédiction des séries temporelles,... Les simplifications sont réalisées au niveau des états cachés et des portes. On conserve un seul état caché h (quitte à le contrôler à l'interne pour implémenter la mémorisation à long terme et à court terme). Et pour les portes, on fusionne les portes de sélection des entrées avec celle des éléments à oublier (donc les portes I et F) pour former une porte dite de mise à jour (porte qui sera appelée *update* ou U). La porte de sélection des éléments de sortie quant à elle, est transformée en porte de réinitialisation.

Ces deux portes (de mise à jour et de réinitialisation) sont en fait implémentées de façon identique que celles des cellules *LSTM*. La particularité des *GRU* se situe principalement au niveau de la gestion de la mémoire (l'implémentation du processus de mémorisation) car, ayant supprimé la distinction long-terme/court-terme, il fallait bien trouver un mécanisme devant permettre de bien gérer les deux aspects de la mémoire

avec un seul état interne conservé.

C'est ainsi que, la porte de mise à jour (porte U) est introduite dans le calcul de l'état h pour assurer la sélection du type de mise à jour à effectuer. Il s'agit de faire en sorte que, selon l'état interne et l'entrée, tout l'état interne précédent soit considéré mais que certains éléments soient complètement modifiés, selon le besoin, et d'autres presque conservés.

Ainsi donc, h devient une combinaison d'éléments provenant de l'état interne précédent avec ceux provenant des nouveaux calculs effectués par la cellule (en fonction de l'entrée et de l'état interne précédent). Le comportement est alors le suivant :

Quand le vecteur de mise à jour a un terme proche de 1, cet état interne est presque conservé. Par conséquent, sa mise à jour est presque ignorée. Quand c'est plutôt 0, l'état interne précédent est presque ignoré et une mise à jour complète de cet état est effectuée.

La formulation mathématique permet de mieux en saisir le fonctionnement [6, 23] :

$$\begin{cases} U_t &= \sigma(W_{ux}X_t + W_{uh}h_{t-1} + b_u) \\ R_t &= \sigma(W_{rx}X_t + W_{rh}h_{t-1} + b_r) \\ h_t &= U_t \circ h_{t-1} + (1 - U_t) \circ \tanh(W_{hx}X_t + W_{hr}(R_t h_{t-1}) + b_c) \\ y_t &= W_{yh}h_t + b_y \end{cases} \quad (\text{I.4})$$

Et pour illustration, on peut considérer l'image suivante :

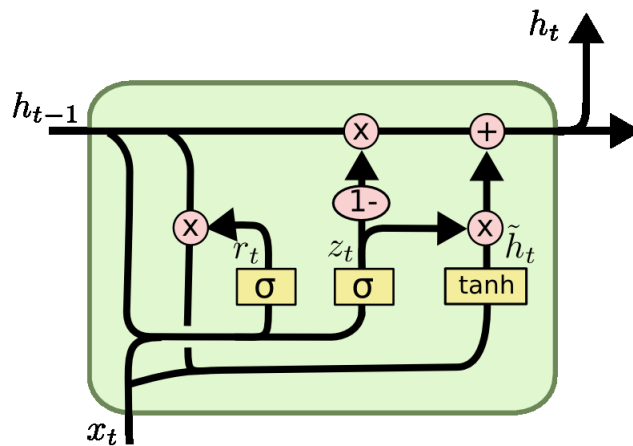


Figure I.5: Cellule GRU [5]

Il faut noter que sur cette image (figure I.5), l'implémentation de la mise à jour est l'inverse de celle que nous avons décrit par les équations I.4. C'est-à-dire que les termes U_t et

$(1 - U_t)$ sont permutés. Mais aussi, ici Z_t représente U_t .

Ces modèles fonctionnent très bien et certaines implémentations permettent d'améliorer encore leurs performances. Ils sont néanmoins lents à entraîner, surtout à cause de l'aspect séquentiel. Parmi les techniques d'amélioration des performances, une peut être considérée car elle a un rapport direct avec notre travail. Il s'agit des **mécanismes d'attention** [24].

I.5.3 Mécanismes d'attention

Les mécanismes d'attention sont en bref des techniques permettant de lutter contre la perte de mémoire qu'on constate par exemple dans les cellules récurrentes ci-haut décrites, en se focalisant sur des éléments les plus importants à chaque traitement. Le travail consiste donc à repérer, pour chaque entrée, les éléments sur lesquels se focaliser. C'est là qu'interviennent donc ces mécanismes.

L'une des implémentations les plus commodes est l'**attention globale** [25].

Pour l'expliquer, nous allons considérer une architecture jusque là passée sous silence, mais qui permet aux modèles introduits là haut de s'utiliser efficacement pour les tâches courantes du *NLP* en particulier. Il s'agit des modèles dits *encodeur-décodeur*.

En effet, lorsqu'on a un modèle à séquence fonctionnel, les objectifs peuvent être multiples. On peut vouloir :

- 1°) fournir une série d'éléments en entrée et ressortir une autre série (utile pour la prédiction de la valeur des actions par exemple,...);
- 2°) fournir un série en entrée mais faire ressortir un seul élément ou vecteur (utile pour la classification des textes, l'analyse des sentiments,...);
- 3°) fournir un vecteur plusieurs fois en entrée et produire une série (pour la génération des légendes pour des images par exemple,...);
- 4°) on peut aussi avoir un réseau série-vers-vecteur, appelé encodeur, suivi d'un réseau

vecteur-vers-série, appelé décodeur (très utile pour la traduction et la synthèse automatique par exemple,...). Il s'agit du modèle **encodeur-décodeur**.

Une illustration par image sera suffisante :

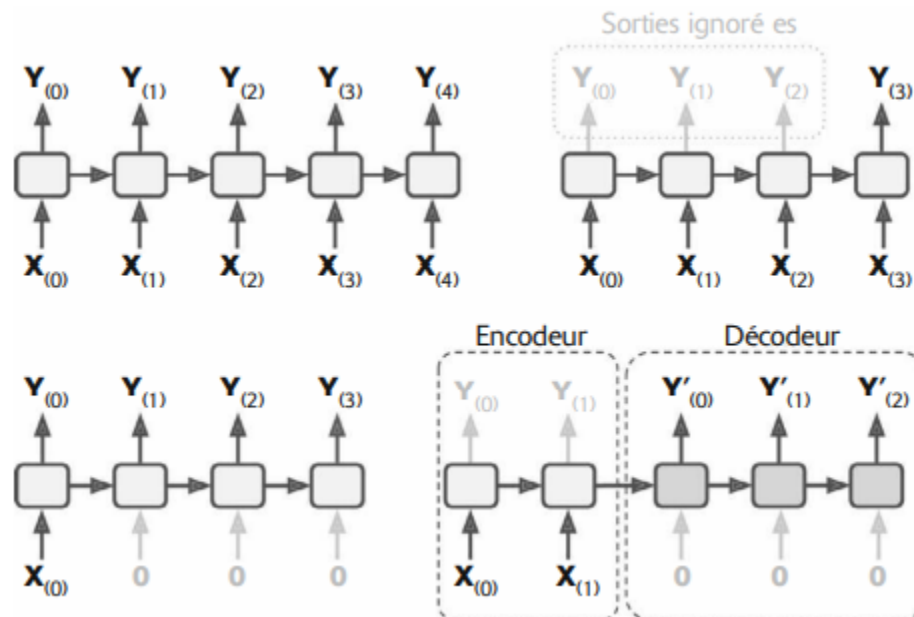


Figure I.6: Réseaux série-vers-série (en haut à gauche), série-vers-vecteur (en haut à droite), vecteur-vers-série (en bas à gauche) et encodeur-décodeur (en bas à droite) [6]

L'élément (le vecteur d'état) passé entre l'encodeur et le décodeur est dit **vecteur de contexte**. Il représente en quelques sortes un condensé des informations passés à l'entrée de l'encodeur. Toutefois, plus la séquence d'entrée est longue, plus le risque que la mémoire de certaines séquences puisse s'étioler devient grand. Ainsi, si par exemple on est entrain de vouloir traduire une longue phrase, on peut finir par transmettre un vecteur de contexte qui a perdu toute information sur les premiers éléments de la séquence passée en entrée. C'est pour cela qu'au lieu de passer un vecteur de contexte général, les mécanismes d'attention permettraient ici de ne se focaliser que sur certaines informations lors du traitement d'un élément particulier de la séquence (en ayant évidemment passé tous les états internes passés au décodeur). Pour le réaliser concrètement, le mécanisme d'attention global consiste à formater le vecteur de contexte en fonction des éléments de l'encodeur à prendre en compte lors du traitement par le décodeur.

Considérons que Ω , dont les termes sont représentés par w_{ij} , est la matrice des poids d'attention normalisés par une fonction *softmax* pour chaque ligne. Et que Π , dont les termes sont représentés par α_{ij} , est la matrice des poids d'attention générée par le mécanisme avant normalisation. Si les éléments c_i représentent à chaque fois le vecteur contexte final à l'étape i de décodage et les h_j sont les vecteurs d'état interne de l'encodeur, l'attention globale revient à réaliser la manipulation suivante, pour formater le vecteur de contexte à prendre en compte pour l'élément en cours de traitement [25] :

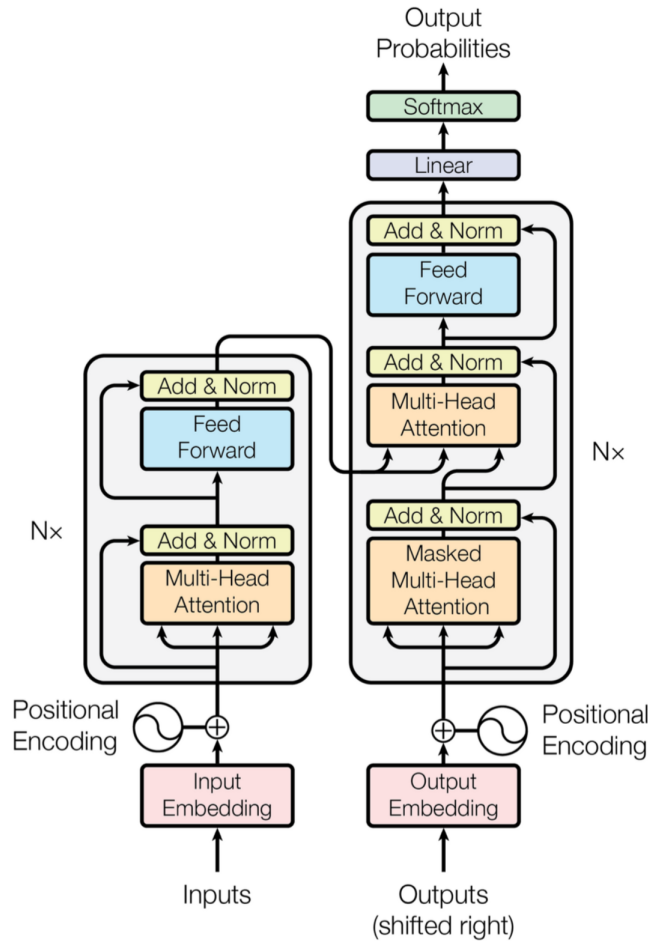
$$\begin{cases} w_{ij} &= \text{softmax}(\alpha_{ij}) = \frac{e^{\alpha_{ij}}}{\sum_k e^{\alpha_{ik}}} \\ c_i &= \sum_j w_{ij} h_j \end{cases} \quad (\text{I.5})$$

La dernière relation du système I.5 revient à réaliser une somme pondérée des vecteurs d'état internes passés de l'encodeur, selon l'importance de chaque état pour le traitement en cours. De ces équations il faut aussi remarquer que la notation des sommations n'est pas rigoureuse. Cela est volontaire car c'est intuitif (on réalise des sommations sur tous les éléments).

Plusieurs techniques arrivant à réaliser l'attention existent. En général, comme on peut d'ailleurs le déduire des relations de l'attention globale, ces mécanismes étaient utilisés dans le cadre des réseaux récurrents. Une question s'est toutefois naturellement posée : *ne pourrait-on pas se passer des RNN pour mettre au point des réseaux complètement basés sur l'attention ?*. La réponse est oui, avec des ajustements adéquats pour résoudre les faiblesses des modèles classiques dans le traitement des données séquentielles. C'est cela qui a conduit aux modèles dits *transformers* [7].

I.5.4 Les transformers

Il s'agit des modèles dont l'architecture générique se présente comme suit :

Figure I.7: Architecture générique des *transformers* [7]

Les *transformers* sont des modèles du type *encodeur-décodeur* comme on peut le constater sur la figure ci-dessus (bien que certaines implémentations n'en utilisent qu'une partie selon la tâche). Ils sont essentiellement basés sur les mécanismes d'attention, se passant de la récurrence [6, 23].

Nous donnerons une explication succincte de chacun des modules présents dans l'image I.7. En effet, présentons les modules selon l'ordre dans lequel les données traversent le modèle :

- 1°) **Module d'embedding** : Nous savons que les données textuelles doivent être présentées au modèle sous forme numérique. Elles doivent donc être transformées avant de les passer aux parties suivantes. Néanmoins, vu que la représentation des entrées a

un impact significatif sur les performances d'un modèle, cette représentation doit être bien choisie. Un choix intuitif, et qui s'avère être performant, est de tout faire pour que si deux termes ont des sens proches, ils aient aussi des représentations vectorielles proches. Cela est réalisé par différentes techniques que nous présenterons dans le chapitre suivant, mais c'est là le rôle de la couche d'enchâssement (*embedding*).

2°) **L'encodage positionnel (*positional encoding*)** : Ce module ajoute l'information sur la position relative de chacun des éléments placés en entrée par rapport aux autres. Cela pallie au problème de perte d'information sur la position des mots quand on utilise un réseau non séquentiel comme les réseaux récurrents. Donc, la position de chaque terme de la séquence placée en entrée est encodée dans un vecteur puis ajoutée à l'encodage global du terme. L'un des encodages les plus utilisés est celui basé sur les fonctions trigonométriques tel qu'introduit dans [7].

3°) **Module d'auto-attention** : La couche d'attention, présentée en première position dans la boîte de l'encodeur, est en fait une couche dite de *self-attention* car elle opère sur la même séquence d'entrée. L'opération est réalisée pour permettre au modèle d'avoir une représentation de l'importance des termes dans la séquence d'entrée, les uns par rapport aux autres.

Pour illustration, considérons la phrase suivante : *Walter est malade, il préfère se reposer.*

Dans cette phrase, l'un des constats qu'on peut faire est que, le nom "Walter" est beaucoup plus lié au pronom "il" qu'au verbe "préférer". C'est à l'établissement des tels liens dans les représentations que sert le module d'auto-attention ici présenté. Il est important que ce lien soit implicitement présent dans les représentations, pour que le traitement soit efficace comme on l'a mentionné lors de la présentation des mécanismes d'attention. Donc cette couche est en fait un prolongement de celle d'*embedding*. Ici, le mécanisme d'attention utilisé est différent de celui qui a été présenté là-haut (attention globale). Il s'agit ici d'un mécanisme plutôt basé sur le produit scalaire mis à l'échelle (*scaled dot-product*). En effet, très brièvement, l'idée du *scaled dot-product attention* consiste à opérer une recherche des termes

sur lesquels focaliser l'attention de la même façon qu'on réalise la recherche de la signification d'un mot dans un dictionnaire. Supposons qu'on veuille avoir la signification d'un mot dont on ne connaît pas l'orthographe exacte. Pour retrouver ce dernier dans un dictionnaire, il suffit de rechercher le mot qui ressemble le plus à l'orthographe que nous estimons être la plus vraisemblable. Mathématiquement, cette recherche de similitude correspond à un produit scalaire.

Similairement, le *scaled dot-product* consiste à générer trois éléments qui sont la clé ou *key* k , la valeur ou *value* v et la requête ou *query* q . La *requête* correspond au mot qu'on cherche (orthographié selon ce que nous pensons), la *clé* correspond au mot présent dans le dictionnaire et la *valeur* correspond à la signification associée. Si on supposait qu'il existe plusieurs termes du dictionnaire qui s'orthographient presque de la même façon que le mot qu'on cherche, on devra passer par une mesure de similarité avant de se décider sur le sens le plus probable. Cela correspond à réaliser le produit de tous les k par les q présents, puis à normaliser l'ensemble des résultats de manière à ce qu'ils représentent des mesures de probabilité, et finir par choisir le sens v le plus probable.

Pour aller plus vite, on implémente ce processus en considérant tous les k , q et v au même moment de manière à réaliser le calcul une fois pour toutes. Cela revient à regrouper tous les k , q et v dans des matrices K , Q et V . Ce qui donne la relation qui définit l'attention par produit scalaire mis à l'échelle [7] :

$$Attention(Q, K, V) = softmax\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) \cdot V \quad (I.6)$$

Dans cette relation, expression I.6, le terme $\sqrt{d_k}$ permet de mettre à l'échelle le résultat du produit scalaire de Q par K , c'est-à-dire $Q \cdot K^T$. Il faut noter que d_k est la dimension d'une clé, et que cette normalisation permet d'améliorer les performances du modèle mais elle n'est pas la seule envisageable.

Il est aussi important de remarquer que la couche d'attention utilise trois termes pour arriver à bout du problème. Ces trois termes sont obtenus par une transformation linéaire dont les poids sont appris à travers un réseau de neurones simple.

Il faut aussi noter que l'on utilise parallèlement plusieurs modules d'attention pour capture toutes les caractéristiques des séquences (on parle de *multi-head attention*). Pour une plus ample illustration, voir la figure [I.8](#).

- 4°) **Le module *feed-forward*** : Il s'agit en fait d'un réseau de neurones de propagation avant classique (réseau à couches ajoutées de façon séquentielle). Il permet de réaliser le traitement qui fait suite à l'attention.
- 5°) **Couche d'attention encodeur-décodeur** : Il s'agit de la couche qui reçoit les données en provenance de l'encodeur. Il s'agit ici d'une *couche d'attention et non d'auto-attention* comme c'était le cas pour la première couche de l'encodeur. En effet, contrairement à la couche de *self-attention*, pour laquelle tous les trois paramètres sont calculés à partir de la même séquence, la couche d'attention ici prend les clés K et valeurs V provenant de l'encodeur mais une requête Q provenant du décodeur. Une autre couche *feed-forward* suit celle-ci et a le même rôle que celle de l'encodeur.
- 6°) **Module d'attention masquée** : Il s'agit de la première couche du décodeur. C'est aussi un module de *self-attention* auquel on ajoute le masquage. Ce module est dit masqué suite au fait que, comme le décodeur est un module de génération, on ne regarde que les termes précédemment générés, en masquant les termes qui seront probablement générés aux pas d'après. Cela est réalisé en rendant juste leurs probabilités nulles.
- 7°) **Module linéaire final** : Il s'agit d'un réseau de neurones classique pour réaliser la déduction finale, le tout étant passé à la fin à travers une opération **softmax** qui permet de transformer les résultats en probabilité d'éléments générés (cela permet de choisir le terme le plus vraisemblable à générer comme sortie).

Cette explication simplifiée se comprend mieux si on y joint la vue éclatée suivante :

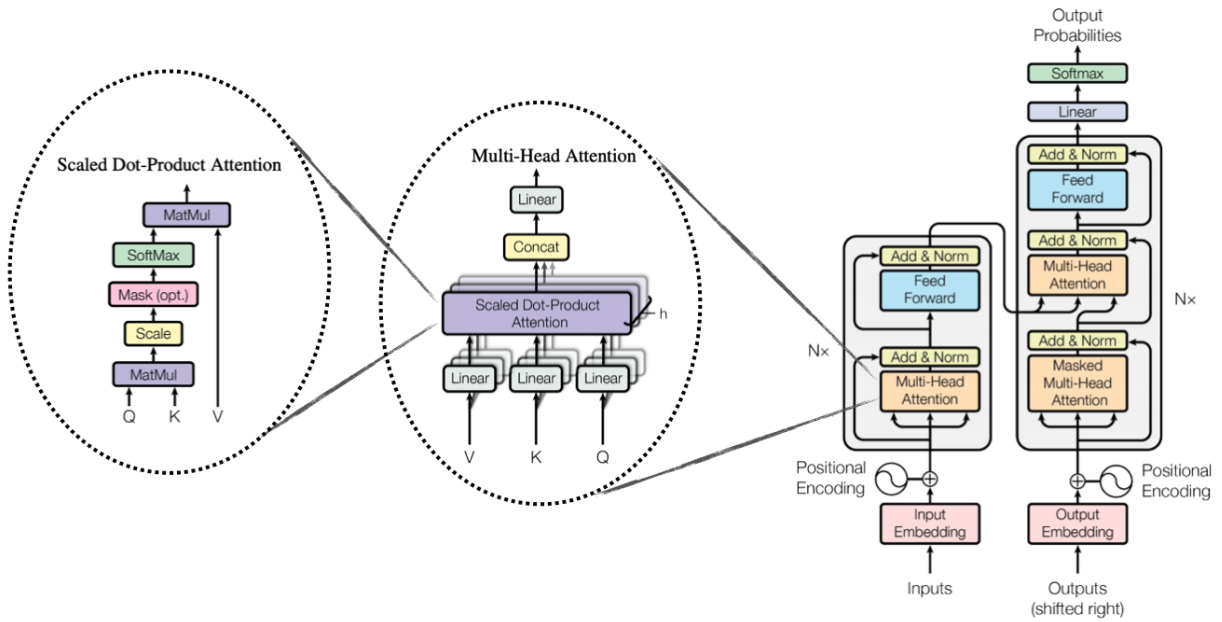


Figure I.8: Vue éclatée d'un transformer [8]

Les *transformers*, ici succinctement présentés, sont un modèle très adapté aux tâches de traitement automatique du langage naturel. C'est un modèle incontournable vu aussi que ses traitements peuvent être facilement parallélisés. Cela est rendu possible par le fait que l'architecture des *transformers* est parallèle par essence.

I.6 Conclusion partielle

Nous venons de réaliser une vue d'ensemble du domaine de traitement automatique du langage naturel, ainsi que diverses techniques couramment utilisées. Pour cela, nous avons tout d'abord justifié la préséance des modèles basés sur le *deep learning* pour diverses tâches du NLP.

Ensuite, nous avons évoqué les techniques de pré-traitement des textes, souvent incontournables, comme la réduction des séquences en leurs tokens constitutifs, la suppression des mots fréquents mais n'apportant pas assez d'informations et la réduction des mots en leurs racines respectives. Nous y avons aussi joint quelques techniques utiles à la compréhension du langage humain comme le *POS tagging* et la reconnaissance d'entités nommées. Ce qui précède nous a finalement conduit à présenter les modèles courants

du NLP basés sur les RNNs et, nous avons terminé par la présentation de l'architecture *transformer*, modèle que nous utiliserons pour ce travail (les précisions sur les modèles particuliers de *transformer* à utiliser seront données au chapitre suivant).

Les transformers constituent un type de modèle qui s'avère être le plus adapté (pour le moment) au résumé automatique des textes et, dans le chapitre suivant, nous commencerons par présenter les diverses spécificités du résumé automatique comme tâche du NLP, pour finir par présenter l'architecture globale du système que nous comptons élaborer.

Chapitre II

Présentation du résumé automatique et conception du système *Mon Résumeur*

II.1 Introduction partielle

Le résumé automatique étant le sujet principal de ce mémoire, dans cette partie nous le présentons alors en détail en tant que discipline et tâche du *NLP*. Nous allons ici présenter les théories sur la synthèse automatique des textes, en classifiant les diverses méthodes utilisées, afin de pouvoir situer notre système dans l'ensemble des travaux jusque-là menés sur ce sujet.

Ensuite, nous présenterons les diverses approches utilisées pour le résumé automatique, sans oublier d'approfondir notre présentation des modèles de type *transformer* adaptés à cette tâche, pour finalement mentionner le modèle que nous estimons le plus adapté concernant l'approche basée sur le *deep-learning* pour la synthèse automatique. Enfin, nous allons réaliser une conception rapide mais suffisante de l'architecture globale de notre système, que nous appellerons *Mon Résumeur*, tout en précisant le rôle et le fonctionnement de chaque partie.

II.2 Présentation et définitions

Selon Le Petit Robert, résumer c'est reprendre en plus court un discours, le présenter brièvement en conservant l'essentiel. En d'autres termes, c'est l'abréger, l'écourter, le réduire. De même, en tant qu'exercice intellectuel, le résumé, consiste à réduire un texte tout en lui restant fidèle. Il exige donc de restituer les idées en un nombre déterminé de

mots, en évitant au mieux de recopier le texte à résumer. Il faut alors composer un texte plus court qui contienne l'essentiel du message initial [26].

De cela on tire que le résumé devient automatique s'il est généré par un logiciel ou un système informatique.

Cette définition est en fait correcte bien qu'elle ne soit pas assez précise pour notre contexte. Il nous faut une définition assez générale et précise, embrassant au mieux l'aspect automatique, ou mieux, l'aspect informatique, qui nous intéresse dans ce mémoire. Une définition assez valable est celle de Torres-Moreno Juan-Manuel qui dit qu'*un résumé automatique est un texte généré par un logiciel, cohérent et contenant une partie importante des informations pertinentes de la source, et dont le taux de compression est inférieur au tiers de la taille du(des) document(s) source(s)* [12].

L'introduction du taux de compression dans la définition n'est pas anodine car, on s'est très vite rendu compte que la performance d'un système de résumé automatique dépendait fortement du taux de compression. En effet, les études de [27] montrent que les meilleures performances des systèmes de résumé automatique sont généralement atteintes pour des taux de compression compris entre 15 et 30% [12].

Nous allons adopter, dans ce travail, la définition de Torres-Moreno Juan-Manuel ci-haut présentée.

Toutefois, on ne doit pas manquer de signaler que la génération automatique des résumés est un problème complexe en soi, tout comme l'évaluation des résultats. Le résumé est en effet une tâche cognitive requérant la compréhension du texte considéré et, les humains n'étant pas toujours bons dans les tâches de synthèse, le manque d'étalon explique qu'il y ait également une difficulté d'automatisation du processus.

II.3 Catégorisation des résumés

Les résumés peuvent être classifiés selon différents critères tels que leur fonction, le nombre de documents source, le genre de document, le type de résumé, le type de résumeur, le contexte,...

Parcourons de manière succincte ces différents critères de classification [2, 21, 28, 29, 30, 12] :

II.3.1 Selon la fonction

Selon leur fonction, on classifie les résumés en deux groupes qui sont le *résumé indicatif* et le *résumé informatif*.

Résumé indicatif

Tel une table des matières, un résumé indicatif renseigne le lecteur sur les thèmes abordés dans un document. Il liste donc les sujets les plus importants évoqués par le texte. Certains systèmes de résumé guidé génèrent un résumé indicatif du texte comme étape initiale, l'utilisateur choisit alors parmi les sujets proposés par le résumé ceux qui l'intéressent et le système produit enfin un résumé informatif du texte, guidé par la requête de l'utilisateur. La requête dans ce cas est l'ensemble des sujets sélectionnés à partir du résumé indicatif.

Résumé informatif

Il s'agit d'un modèle rétréci du texte d'origine, relatant le plus largement possible les informations contenues dans celui-ci. Ce type de résumé répond souvent à une attente en résumant de plus le contenu. La problématique ici est donc double : comprendre ce qui n'est pas information dans un texte et connaître le besoin de l'utilisateur final.

Néanmoins, si on n'a pas de requête spécifique de la part de l'utilisateur, le résumé informatif est réalisé en veillant à ce que l'ensemble des principaux sujets du texte d'origine soit rapporté. Ainsi, les sujets principaux qui sont rappelés dans le résumé sont répartis de manière fidèle par rapport à l'organisation initiale afin de donner un juste aperçu du texte source.

II.3.2 Selon le nombre de documents sources

Selon le nombre de documents sources on a les résumés mono-document et multi-document.

Résumé mono-document

Il consiste à résumer un document isolé. Le corpus de documents source est donc ici constitué d'un seul et unique document.

Résumé multi-documents

Il s'agit d'un résumé de plusieurs documents (un groupe de documents), très souvent liés thématiquement, en faisant attention à ne pas insérer des informations déjà évoquées.

II.3.3 Selon le genre des documents

Résumé des documents journalistiques

Il s'agit de résumer les documents du type article de presse (sachant qu'ils ont une structure particulière). En effet, on sait par exemple que dans le domaine journalistique, les informations les plus importantes sont souvent mentionnées au début du texte.[\[2\]](#)

Résumé des documents spécialisés

Il s'agit de résumer des documents en provenance d'un domaine précis (géologie, médecine, mathématique,...), fortement spécialisé.

Résumé des documents littéraires

C'est le résumé de documents du type narratif, des textes littéraires, des textes argumentatifs, ...

Résumé des documents encyclopédiques

Ici il s'agit de résumer des documents de type encyclopédique (en général multi-thématiques de toute évidence) à l'exemple de Wikipédia...

II.3.4 Selon le type de sortie (résumé obtenu)

Cette classification est très importante et très utilisée. Il s'agit des :

Résumé extractif (*extractive summarization*)

Le résumé extrait est formé de segments de texte extraits du(des) document(s) source(s). Ces segments peuvent être des phrases, des propositions ou n'importe quelle unité textuelle présent dans le(s) document(s) à résumer. Le problème consiste donc à repérer les segments de texte qui semblent être les plus pertinents pour faire partie du résumé final. Les éléments obtenus à la fin sont donc explicitement présents dans le(s) document(s) source(s).

Résumé abstraktif (*abstractive summarization*)

Les méthodes de résumé abstractives imitent, jusqu'à un certain degré, le processus naturel accompli par l'homme pour résumer un document. Par conséquent, elles produisent des résumés plus similaires aux résumés manuels (humains). Ce processus peut être décrit par deux étapes majeures : la compréhension du texte source et la génération du résumé. La première étape vise à analyser sémantiquement le contenu du texte et à identifier les parties à exprimer dans le résumé. C'est en quelques sortes une tâche d'extraction d'information liée au domaine abordé ou de regroupement des phrases du texte source. Vient ensuite la génération du texte.

Bref, on produit un résumé rapportant le contenu du(des) texte(s) source(s) en utilisant un vocabulaire souvent différent et plus concis.

Il existe aussi des résumés dits *semi-extractifs*, et même aussi des résumés dits *par compression* [12] mais nous estimons inutile de les décrire ici étant donné que la distinction abstraktif-extractif suffit pour notre contexte.

II.3.5 Selon le type de résumeur

Le résumeur est le système qui réalise le résumé. Il peut s'agir d'une entité naturelle (un humain) ou artificielle (un logiciel). On a donc essentiellement les deux cas suivants :

Résumé humain (manuel)

Il s'agit d'un résumé réalisé par un humain. Il peut être fait par l'auteur même du document (on parle souvent de *résumé d'auteur*), par un expert du domaine traité (on parle souvent de *résumé d'expert*) ou par un professionnel de résumé (on parle de *résumé professionnel*).

Résumé automatique

Il s'agit, comme on l'a maintes fois mentionné, d'un résumé fait par un système informatique.

II.3.6 Selon le contexte

Résumé générique

Ici on résume le document sans prendre en compte les besoins d'information de l'utilisateur. On produit juste un résumé complet et le plus mieux fait possible.

Résumé guidé

Pour ces types de résumé, l'utilisateur commande la génération du résumé en précisant les types d'information dont il a besoin.

Résumé mis à jour

Il s'agit d'un résumé de type dynamique par essence. Ici, un ensemble de documents sources est résumé en veillant minutieusement à ce que le document dont le résumé est ajouté à la suite d'un précédent résumé ne puisse pas créer une répétition d'information. Il y a donc un contrôle de nouveauté.

II.3.7 Selon le destinataire du résumé

On peut aussi classer un résumé selon le public auquel il est destiné.

Résumé sans profil

Il s'agit d'un résumé qui ne tient pas compte d'un quelconque profil utilisateur. Le résumé est donc généré sans tenir compte de la personnalité des utilisateurs.

Résumé avec profil

Il s'agit d'un résumé dont l'un des éléments guides (requête) est le profil des individus auxquels le résumé est destiné.

En ce qui concerne notre système, nous implémenterons à la fois un résumeur abstraktif et un résumeur extractif et ce sera mono-document. En plus de cela, le résumé ne sera pas guidé, il s'agira de produire des résumés génériques, pour des documents de type littéraire (documents du type narratif, des textes littéraires, des textes argumentatifs,...).

II.4 Approches de résumé automatique

Nous allons présenter ici diverses approches algorithmiques pour résumer les documents textuels. Les approches seront abordées en supposant que les résumés sont principalement classés en *abstraktif* et *extractif*.

II.4.1 Techniques intuitives de résumé [2]

Avec des **critères centrés sur le contenu des textes**, il existe un grand nombre d'algorithmes assez triviaux de résumé, qui sont basés entre autres sur :

- La fréquence d'occurrence des mots et
- L'annotation en rôle sémantique.

Ces critères mettent l'accent sur le contenu du texte et le message qu'il communique.

Fréquence d'occurrence des mots

L'idée majeure des techniques qui utilisent ce critère consiste à considérer que les mots les plus fréquents sont les plus liés au sujet principal du texte à résumer. Cette approche assez simpliste mais fonctionnelle fut introduite en 1958 par **Luhn** [9], une première tentative de résumé automatique.

On affecte des scores aux phrases présentes dans le texte, en additionnant chaque fois les poids des mots les constituant (on attribue ce poids en fonction de la fréquence d'apparition du mot considéré dans le texte entier). Et, à la fin, le résumé est constitué avec les phrases extraites du texte source, et dont le score dépasse un certain seuil dépendant de la taille maximale imposée pour le résumé. Le tout est finalement réarrangé selon l'ordre d'apparition (des phrases sélectionnées) dans le texte d'origine.

L'annotation en rôle sémantique

Ici, l'idée est simple. En utilisant des techniques de repérage d'entités nommées (voir le chapitre précédent), on identifie les entités présentes dans le document. Après cela, l'entité la plus fréquente est identifiée et considérée comme entité principale. Par la suite, les phrases contenant cette entité sont sélectionnées. Enfin, seules les phrases où l'entité principale possède un rôle sémantique fondamental (non auxiliaire) sont gardées pour le résumé.

L'un des moyens les plus simples pour repérer les entités nommées est de passer par l'apprentissage profond comme on l'a précédemment mentionné.

Il existe tout de même des **techniques qui ne se fient qu'à la forme et à la structure du texte**, sans en considérer le contenu. L'intuition derrière cette approche est basée sur le constat que dans un texte, les éléments ne sont pas présentés de façon arbitraire. De manière usuelle, les techniques utilisées se basent sur :

- La position des phrases;

- La similarité avec le titre
- La longueur des phrases ou sinon,
- Les mots indices (*cue word*)

La position des phrases

Cette approche est à appliquer en fonction de la nature du document et de son genre. Pour certains types de documents (documents journalistiques par exemple), les phrases se trouvant au début sont généralement plus informatives et décrivent le sujet principal du document. De plus, les phrases situées au début de chaque paragraphe tendent à apporter plus d'informations pertinentes. Le résumé des articles scientifiques par contre, peut essentiellement se former en se basant sur les contenus des parties **résumé** et **introduction** (sous l'hypothèse que ces dernières parties sont bien faites). En revanche, dans le cas des revues intégratives (critique et comparaison des études), les phrases les mieux notées sont celles des parties **résultats et discussion** et **conclusion**. Ces exemples suffisent pour illustrer dans quelle mesure cette approche peut s'appliquer.

La similarité avec le titre

Cette approche part du principe selon lequel un bon titre doit informer de manière brève du contenu principal du texte qu'il encadre. Cela permet alors de fixer comme mesure de pertinence des phrases, leur similarité avec les titres. Toute la problématique se réduit donc à la construction d'algorithmes capables de capturer efficacement la similarité.

La longueur des phrases

L'approche consistant à se baser sur la longueur des phrases est assez naïve mais fonctionnelle.

En effet, la longueur moyenne d'une phrase dans un texte dépend de son genre. Généralement, les phrases très courtes sont considérées comme peu informatives alors que les phrases très longues sont présumées favoriser la redondance. Cette caractéristique est

exploitée en fixant un intervalle de longueur (entre 15 et 30 mots). Une phrase ayant une longueur en dehors de cet intervalle est pénalisée [31].

Les mots indices

Ici, on considère une liste de mots, constituée manuellement, et qui a comme rôle de permettre de se décider si une phrase doit être prise dans le résumé ou rejetée, selon qu'elle contient ou non un(des) mot(s) de la liste qualifié(s) inhibiteur(s) ou valorisant(s). Comme exemple des mots ou groupes de mots inhibiteurs on trouve : *par exemple, accessoirement, ...* Et pour les mots valorisants on peut citer : *notez bien, ...*

Nous devons quand même préciser encore une fois que tout dépend de celui qui écrit la liste.

Les méthodes que nous venons de présenter sont assez intuitives mais constituent la base des processus de synthèse. En effet, synthétiser un texte revient au fond à implémenter un certain nombre de règles, dont font parties évidemment celles que nous venons de mentionner. Néanmoins, ce que nous venons de présenter est décrit en se basant sur le concept de résumé extractif. Nous devons toutefois signaler que les résumés abstractifs se basent au fond sur les mêmes principes, soit en partant des résumés extractifs pour ensuite réaliser des paraphrases, insérer des connecteurs appropriés et éliminer les références anaphoriques dans les résumés, soit en implémentant indirectement toutes ces techniques à travers un modèle d'*apprentissage automatique* ou un *modèle basé sur les graphes* capables de capturer d'un seul coup tous ces aspects (ou une grande partie d'entre-eux). Les techniques intuitives ci-haut présentées ne sont pas les seules. Il en existe également d'autres, basées essentiellement sur les théories linguistiques. Entre autres **les méthodes d'analyse du discours** (par exemple la **RST** [21] ou *Rhetorical Structure Theory*)...

II.4.2 Algorithmes classiques de résumé automatique

Comme nous venons de l'introduire dans la section précédente, le résumé automatique est abordé essentiellement selon deux approches qui sont [21] :

- 1°) Les *approches numériques*, fondées sur les techniques à base des scores (poids), et
- 2°) Les *approches symboliques* fondées sur les techniques purement linguistiques, basées en premier sur une étude sémantique.

Il faut noter qu'on peut considérer aussi des *approches basées sur la théorie des graphes* comme intégrant les idées de ces deux approches de façon implicite, tout comme celles basées sur *l'apprentissage automatique*. Mais, dans tous les cas, une vue sur quelques heuristiques (méthodes basées sur le bon sens) est toujours à considérer (surtout en amont et en aval du processus de synthèse).

Ici, nous allons présenter les approches essentiellement numériques (on va y inclure celles basées sur l'apprentissage automatique et celles basées sur la théorie des graphes).

Algorithme de Luhn [9]

Il s'agit d'une méthode semi-heuristique pour la synthèse des documents. C'est la plus ancienne méthode de résumé automatique (au sens moderne du terme).

Cette approche n'est pas considérée comme très bien formalisée. Elle exécute implicitement l'approche du *Tf-Idf* que nous allons décrire dans la sous-section qui suit celle-ci (sous-section [II.4.2](#)).

La sélection (des mots ici) se fait en considérant les hypothèses qui suivent :

- la synthèse consiste à supprimer certains mots pour n'en conserver que les plus importants;
- les mots se trouvant au début sont probablement importants;
- les autres mots utiles respectent une certaine distribution. La figure [II.1](#) montre, selon *Luhn*, comment choisir ces mots importants (partie hachurée de la courbe).

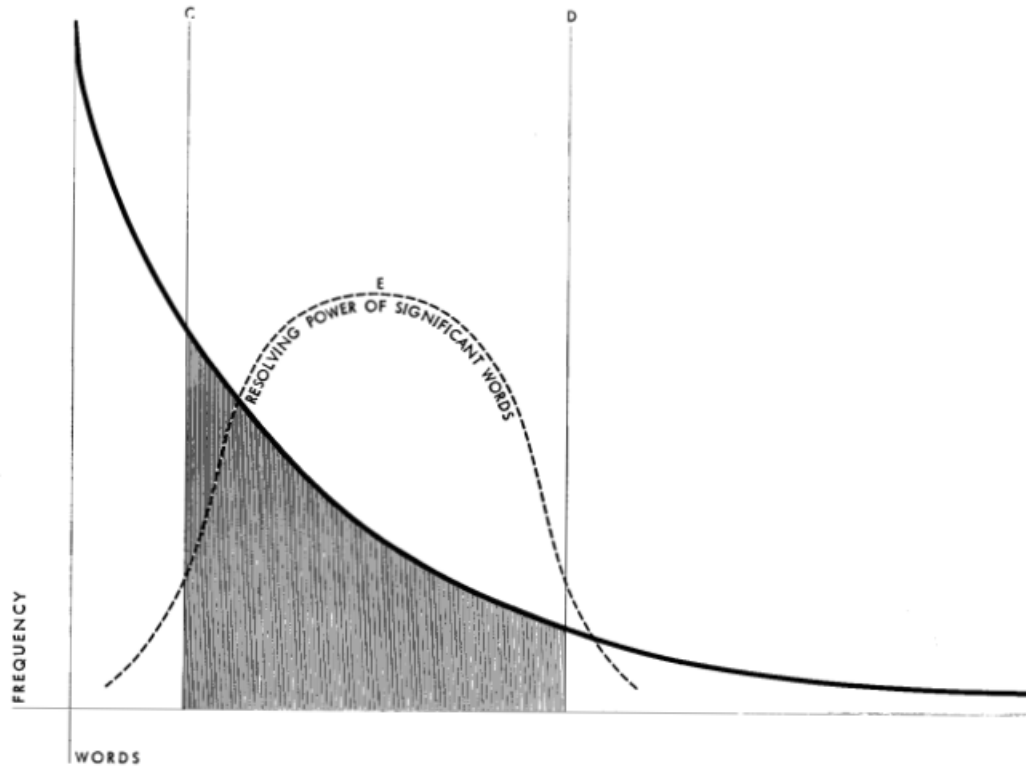


Figure II.1: Diagramme des fréquences des mots et le choix de Luhn [9]

Cette approche, comme on l'a mentionné au début, est assez moins précise et empirique, mais elle sous-tend les idées fondamentales appliquées plus tard.

Algorithme TF-IDF

Le *tf-idf* (*time-frequency inverse document frequency*) est une approche essentiellement utilisée pour le résumé extractif. Il s'agit d'une correction de l'approche naïve consistant à poser que plus un mot est répété dans un corpus de texte, plus il y est important.

Soit donc un corpus constitué de D documents et N_j le nombre total de mots (termes) présents dans un document j donné du corpus. Nommons $Freq(i, j)$ le nombre de fois qu'un terme i apparaît dans le document j .

On définit classiquement la fréquence d'apparition par :

$$TF(i, j) = \frac{Freq(i, j)}{N_j} \quad (II.1)$$

L'approche qui se base naïvement sur la fréquence d'apparition des mots dans les textes pour juger de leur importance relative, accorde à chaque mot un poids égal à $TF(i, j)$.

La grande faiblesse de cette approche est d'inclure ainsi des termes sans grande pertinence informationnelle comme des prépositions, des articles,... très présents au sein des documents.

Pour corriger cette faiblesse, on pose l'hypothèse que les termes importants apparaissent plusieurs fois dans un document (ou juste dans peu de documents du corpus) et non pas dans plusieurs documents, puisque dans le cas où c'est dans plusieurs documents, il est souvent question des éléments communs du langage, sans grande utilité informationnelle. Ceci constitue en fait *la loi de Zipt* [32] et c'est le fondement de l'approche du *tf-idf*.

A cet effet, on définit DF_i comme étant le nombre de documents dans le corpus, qui contiennent le terme numéro i . Cela permet d'affecter alors le poids selon la formule [33] :

$$TFIDF(i, j) = \log(1 + TF(i, j)) \cdot \log\left(\frac{D}{DF_i}\right) \quad (\text{II.2})$$

Dans l'expression II.2, en supposant que N est le dictionnaire des termes présents dans l'ensemble des documents et D le nombre de documents du corpus, il faut noter que : $i \in \{1, \dots, N\}$ et $j \in \{1, \dots, D\}$.

D'où finalement, le poids d'un terme i dans un document j est donné par :

$$w_{ij} = TFIDF(i, j) \quad (\text{II.3})$$

Pour notre cas, l'application de cette approche consiste à décomposer un long texte en ses phrases et de considérer que chacune de ces phrases est un document et que le texte entier constitue le corpus.

Plusieurs définitions des éléments $TF(ij)$ et IDF_i formant l'expression II.2 sont toutefois possibles selon les besoins en terme de performance.

Mais, dans l'ensemble, l'idée de base demeure la même car il ne s'agit en général que de changement des types de normalisation [32].

L'application de cette méthode pour le résumé consiste finalement à calculer le poids de chaque phrase en additionnant les poids des termes la constituant, puis à normaliser le résultat en fonction de la taille de la phrase considérée. Après tout, on définit un seuil qui permet de soutirer les phrases selon leur pertinence ainsi évaluée (en considérant évidemment plus pertinente une phrase dont le résultat de la sommation des poids est élevé).

Algorithme TextRank

TextRank est un algorithme de résumé extractif, basé sur la théorie des graphes et qui s'inspire de l'algorithme *PageRank* de Google [34, 35].

A la base, on considère un ensemble de N phrases donné, et on calcule les coefficients de liaison de chaque phrase aux $N-1$ autres. A la fin, on peut obtenir une matrice M de taille $N \times N$ dont chaque terme M_{ij} représente le degré de liaison entre la phrase numéro i et la numéro j . Il s'agit en fait d'une *matrice d'adjacence* dans laquelle on pose au préalable que $M_{ii} = 0$, pour tout i (c'est la même idée pour l'algorithme *PageRank* étant donné qu'il est logique de considérer qu'une page ne peut s'auto-référencer).

Soit donc $i \in \{1, \dots, N\}$. Appelons Phr_i la phrase numéro i du corpus. Cela veut dire qu'on peut écrire :

$$\text{Liaison } Phr_i \leftrightarrow Phr_j = M_{ij} = M_{ji} \quad (\text{II.4})$$

Les valeurs de M_{ij} sont calculées au choix, selon le programmeur. Ce dernier implémente en effet une mesure de similarité selon sa définition de la liaison entre phrases et les besoins en performance.

C'est ainsi qu'on peut utiliser par exemple une mesure de similarité classique nommée *similarité cosinus* en la basant par exemple sur *TFIDF* [36].

Pour représenter les mots à comparer, on utilise les méthodes classiques de vectorisation des mots (*word embedding*). Nous esquisserons ces méthodes dans les sections qui vont suivre, parlant du *word embedding* (II.5).

Le rang des phrases sont alors calculés de manière itérative en s’inspirant de la formule [37] :

$$TextRank(Phr_i) = (1 - K) + K \cdot \sum_{\substack{j=1 \\ j \neq i}}^N [TextRank(Phr_j)] \cdot M_{ij} \quad (II.5)$$

Dans cette formule, K est une constante comprise entre 0 et 1.

Initialement, on prend en général une valeur identique de $TextRank(Phr_i)$ pour toutes les phrases (souvent $TextRank(Phr_i) = 1$), mais la valeur initiale prise n’affecte pas les valeurs finales, mais elle affecte le temps de convergence [37].

La formule II.5 n’est pas arbitraire, elle est d’ailleurs triviale si on s’inspire de l’algorithme de *PageRank* la plus simple. Pour cet algorithme (*PageRank*), on avait pris à l’origine $K = 0.85$ [34].

Justification de la formule Le principe de *PageRank* consiste à se dire que, si une page Pag_i contient N_i références vers d’autres pages, la probabilité qu’on aille vers l’une de ces pages référencées est de $\frac{1}{N_i}$ (avec l’hypothèse que les références ne sont pas répétées et que la distribution de leur importance est uniforme). On sait tout de même que plus une page est référencée, plus on doit lui donner de l’importance.

Si alors on pose que l’importance de la page Pag_i est connue, le calcul de l’importance d’une page Pag_j vers laquelle elle pointe se calculera logiquement par :

$$Importance(Pag_j) = \sum_i Importance(Pag_i) \cdot \frac{1}{N_i} \quad (II.6)$$

Avec i appartenant à l'ensemble des pages qui mentionnent la page Pag_j en leur sein.

Malheureusement, pour les phrases non référencées (pages dites isolées), on trouve une importance nulle. Pour lutter contre cela, la formule II.6 est un peu modifiée en y introduisant adéquatement une constante non nulle K .

Ce qui donne l'expression [34] :

$$Importance(Pag_j) = (1 - K) + K \cdot \sum_i Importance(Pag_i) \cdot \frac{1}{N_i} \quad (\text{II.7})$$

On voit alors qu'il s'agit belle et bien de la formule utilisée pour *TextRank* (formule II.5).

Après initialisation des rangs de chaque phrase du texte (les $TextRank(Phr_i)$) et après calcul de la matrice d'adjacence M . On applique la formule II.5 itérativement et à la convergence, on choisit les phrases qui vont former le résumé selon leur importance (valeurs des $TextRank(Phr_i)$ pour toute valeur de i).

A la fin, les phrases sélectionnées sont réarrangées pour former un résumé extrait plus ou moins cohérent.

Il existe également un algorithme nommé **LexRank** [38] qui est assez similaire à *TextRank* ici décrit, à la différence près que :

- Il prend essentiellement en compte les métriques de similarité robustes;
- Il considère la position et la longueur des phrases dans le calcul de leur pertinence;
- Il est optimisé pour le résumé multi-document.

Plusieurs autres algorithmes populaires existent, par exemple les algorithmes **LSA** (*Latent Semantic Analysis* ou Analyse Sémantique Latente) et **LDA** (*Latent Dirichlet Allocation* ou Allocation Latente de Dirichlet) [33].

Le premier, la *LSA*, est un algorithme statistique, basé sur l'algorithme **SVD** (*Singular Value Decomposition* ou décomposition en valeurs singulières). Seulement, cette technique est très gourmande en ressources suite à la complexité de l'algorithme qui implémente le

SVD. Le second, la *LDA*, basé sur la détection des thématiques, peut aussi être utilisé.

Toutefois, il faut remarquer que les algorithmes ici présentés sont essentiellement adaptés à la synthèse extractive. Même si ces traitements peuvent être mélangés avec les **techniques de résolution d'anaphores** et les **paraphrases** pour obtenir des synthèses qui tendent vers la synthèse abstractive, nous devons souligner que les techniques jusque là les plus performantes pour la synthèse abstractive sont essentiellement basées sur le *deep learning* [2]. Le *deep learning* peut également être utilisé pour la synthèse extractive, permettant ainsi la génération des synthèses extraites plus cohérentes (avec résolution d'anaphores). Ainsi donc, nous abordons les méthodes de *deep learning* utilisées pour cet effet dans les parties qui suivent.

II.5 Modèles Seq2Seq

II.5.1 Methodes du Word-Embedding

Tout traitement commence par une représentation numérique des termes (des mots ici) pour qu'ils soient assimilables par le modèle. Une approche naïve consisterait à regrouper tous les mots de notre vocabulaire dans une liste (un dictionnaire) et de les représenter chacun par un nombre unique (un identifiant). Une autre approche, plus classique, consiste à représenter chaque mot par un vecteur de dimension égale à la taille du dictionnaire et dont tous les termes sont nuls, sauf à la position, dans le dictionnaire, du mot qu'on est entrain de vouloir représenter (on parle du *one-hot encoding*).

Ces représentations, et toutes celles qui s'y apparentent, ont la grande faiblesse d'être peu informatives (au point de vu sémantique). Étant artificiellement construites, sans tenir compte du sens des mots, ni de leur contexte, ces méthodes de représentation rendent la tâche de découverte des caractéristiques par les systèmes de *machine learning* encore plus difficile. D'ailleurs, l'une des faiblesses de la seconde méthode décrite (le *one-hot encoding*) est que les vecteurs sont creux (une majorité de valeurs nulles) et de dimension inutilement très grande. On pourrait directement songer à une représentation plus judi-

cieuse pour éviter ces deux soucis, et qui consisterait à réaliser une représentation binaire des termes mais, le problème de la sémantique sera toujours là.

On recourt donc à des méthodes de représentation plus élaborées, partant du principe selon lequel le contexte d'un mot suffit pour en appréhender le sens. Ainsi, tout mot est représenté en réalisant une statistique (implicitement bien sûr) sur les divers mots qui l'accompagnent souvent, de telle sorte que les mots aux sens proches aient aussi des vecteurs très proches. Bref, on en arrive à réaliser la proposition : "Similarité sémantique implique similarité de représentation". Ce sont les méthodes classiques du *word embedding* (ou plongement lexical). Il s'agit par exemple des méthodes comme le **Word2Vec** [39, 40], **Glove** [41], **fastText** [42]...

II.5.2 Modèles séquence-à-séquence proprement dits

S'agissant des modèles séquence-à-séquence (*Seq2Seq*), ils ont été présentés dans la section I.5.3 (voir particulièrement la figure I.6). Il s'agit bel et bien des modèles adaptés aux tâches de synthèse, vu qu'en entrée on reçoit une séquence pour ressortir une autre séquence en sortie.

Comme nous l'avons déjà bien mentionné au précédent chapitre, nous n'allons parler que des modèles *Seq2Seq* de type *transformer* car actuellement, ils sont les plus adaptés à la tâche que nous voulons réaliser (celle de synthèse automatique). Les *transformers* (voir la figure I.7) sont un modèle très avantageux car en fait, au-delà de leurs performances et autres avantages, ils facilitent encore plus la recherche en *NLP* en rendant effectif le *transfer learning* (apprentissage par transfert) dans ce domaine.

L'entraînement des *transformers* est *semi-supervisé*. Il se fait en deux crans (nous les décrivons dans le cadre du *NLP*) :

- 1°) **Pré-entraînement** : il s'agit d'un *apprentissage non supervisé*, qui consiste à donner au modèle une masse colossale de données textuelles, non étiquetées, pour qu'il développe une compréhension statistique du langage qu'on veut qu'il puisse assimiler.

Au final, on obtient un modèle pré-entraîné.

2°) ***Affinage de l'apprentissage*** (*fine-tuning*) : Ça consiste à finaliser l'apprentissage du modèle pré-entraîné *de manière supervisée* pour qu'il soit en mesure de réaliser une tâche donnée du *NLP* (il s'agit du *transfer learning* en fait). Cette spécialisation, requiert une très faible quantité de données car le modèle aura déjà une représentation assez bonne de la langue. Cela pallie à la fois au problème de manque des données labellisées en *NLP* et de la consommation en terme de ressource énergétique des gros modèles lors de leur entraînement.

Les méthodes de pré-entraînement sont très déterminantes pour les performances finales du modèle. Ce premier entraînement du modèle a pour rôle de l'amener à construire un ***modèle de langage*** [10].

Il existe ainsi plusieurs objectifs de de pré-entraînement (pour construire le modèle de langue). On peut par exemple entraîner le modèle à :

- Prédire le mot suivant : donc, lors de cet entraînement non supervisé, on fournit chaque fois au modèle une séquence de mots en lui demandant de prédire le suivant. Il s'agit d'un objectif d'entraînement dit *NSP* (*Next Sentence Prediction*) visant à transformer implicitement le *transformer* en un modèle de langue [43];
- Deviner le mot caché (masqué) : on fournit au modèle du texte dont certaines parties (mots ou suite de mots) sont cachées. L'objectif assigné au modèle est alors de retrouver les mots masqués. On parle du *MML* (*Masked Language Modelling*) [43].

Ainsi, au fur et à mesure, les paramètres du modèle s'affinent, le transformant en un modèle de langue performant. Mais, à part les deux que nous venons de mentionner, il existe d'autres *objectifs de pré-entraînement* [10, 44] selon les variantes de *transformers* et les objectifs finaux de spécialisation du modèle.

Bien que la forme classique des *transformers* est bel et bien celle de la figure I.7, il existe 3 types d'implémentation selon les types de tâche visées en dernier lieu :

- 1°) Modèles à encodeur seul : on supprime la partie décodeur. Ces modèles sont très bons pour les tâches de compréhension du langage comme la classification par exemple.
- 2°) Modèles à décodeur seul : on supprime alors la partie décodeur du modèle. Ils sont bons pour les tâches de génération de texte.
- 3°) Modèles encodeur-décodeur : ou encore modèles *seq2seq* proprement-dits. Ils sont bons pour les tâches demandant à la fois la compréhension et la génération des textes.

Pour illustrer ce fait, on va considérer donc 3 types de *transformers* [45, 46] :

- 1°) *Like-BERT* : semblables au transformer dénommé *BERT* (*Bidirectional Encoder Representations from Transformers*). Ce sont des modèles du type encodeur seul. Ils sont également bidirectionnels. Donc, les phrases sont lues dans les deux sens pour mieux saisir tout le contexte.
- 2°) *Like-GPT* : donc semblables au *transformer* dénommé *GPT* (*Generative Pre-trained Transformer*) qui n'ont que la partie décodeur et sont dits *auto-regressifs* car, seules les parties précédant le mot en cours de traitement sont connues du modèle et il y a chaque fois réinjection des sorties à l'entrée.
- 3°) *Like-BART/T5* : semblables à *BART* (*Bidirectional and Auto-Regressive Transformers*) ou à *T5* (*Text-To-Text Transfer Transformer*). C'est donc ceux du type encodeur-décodeur.

Modèles encodeurs (*encoder-model*) : Comme on l'a dit, pour ces modèles, on n'implémente que la partie encodeur du *transformer* d'origine (celui d'origine étant dans [7]). En plus de cela, ces modèles ont une couche d'attention bidirectionnelle et sont généralement appelés *auto-encodeurs* (*auto-encoding model*). Ces modèles sont principalement bons pour les tâches de *NLU* (*Natural Language Understanding*) comme la classification, le *NER* (*Name Entity Recognition*), l'*extractive question-answering*,...

Dans ce groupe, les modèles les plus connus sont :

- ALBERT [47],

- BERT [43],
- DistilBERT [48],
- RoBERTA [49],
- Etc.

Modèles décodeurs (*decoder-models*) : Utilisent seulement la partie décodeur, sont auto-regressifs et par conséquent les têtes de *self-attention* n'accèdent qu'aux mots précédant l'étape à laquelle elles sont (pas de regard dans le futur) comme on l'a déjà un peu mentionné. Ces modèles sont particulièrement bons pour les tâches liées fortement au *NLG* (*Natural Language Generation*).

Dans ce groupe, les modèles les plus connus sont :

- Les GPT (1, 2 et 3) [50],
- TransformerXL [51],
- Etc.

Modèles encodeur-décodeur (*sequence-to-sequence models*) : Ces modèles utilisent l'intégralité de l'architecture des *transformers* et sont ainsi bons pour les tâches demandant à la fois du *NLU* et du *NLG* comme la synthèse automatique abstractive, le *generative question-answering* et la traduction automatique.

Ici nous pouvons particulièrement mentionner les modèles comme :

- BART [10],
- mBART [52],
- BARThez [53],
- T5 [54],
- mT5 [55],

- PEGASUS [44],
- Etc.

II.5.3 Modèle BART pour la synthèse abstractive

Le modèle *BART* est comme une combinaison de *BERT* [43] et de *GPT-2* [56, 50] en terme d'architecture et d'objectif de pré-entraînement, avec quelques optimisations supplémentaires [10]. Pour illustration, voici une image de comparaison :

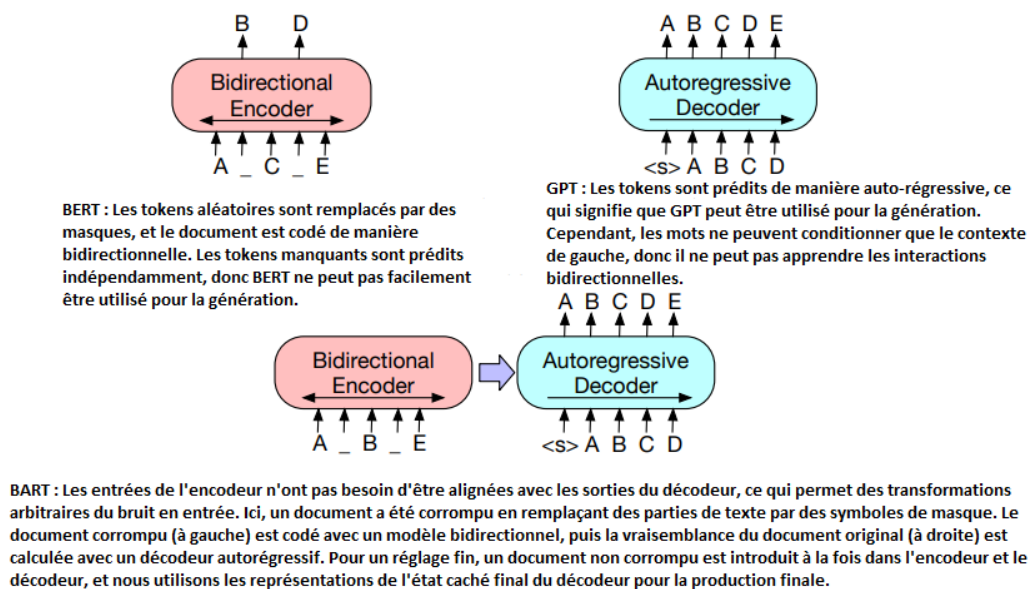


Figure II.2: Comparaison simplifiée entre BERT, GPT et BART [10]

L'image II.2 étant claire, nous pouvons illustrer les diverses corruptions que peuvent subir les données pour le pré-entraînement. L'image ci-dessous l'illustre :

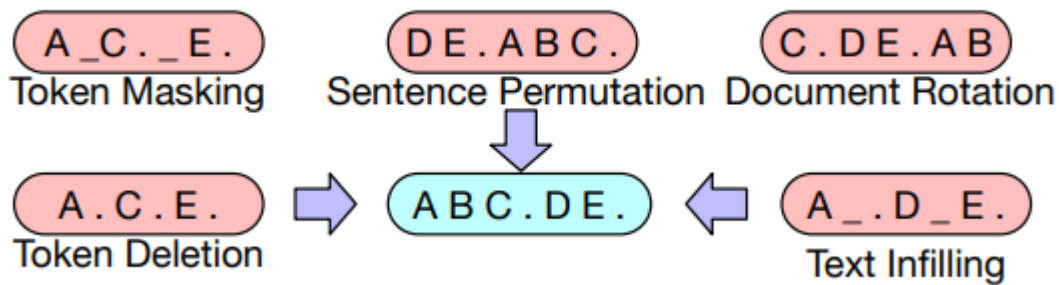


Figure II.3: Transformations de bruitage expérimentées pour BART [10]

Le modèle *BART* est bien adapté à la tâche de synthèse abstractive. C'est celui que nous allons utiliser (les modèles dérivés de *BART* principalement) pour réaliser cette tâche dans notre système.

Justification du choix de BART

Le choix de *BART* est dû au fait que c'est le modèle que nous avons trouvé réalisant un bon compromis poids-performances. Aussi, après quelques tests, ses résultats nous ont paru être plus intéressants. En outre, l'objectif d'entraînement utilisé pour *BART* nous paraît assez général pour construire un modèle de langage performant. Nous justifierons plus précisément ce choix dans le chapitre qui suit, en présentant également quelques résultats des tests.

II.6 Conception de l'architecture globale de *Mon Résumeur*

Il existe un large éventail des méthodes de développement des systèmes informatiques mais, en règle générale, toutes suivent les étapes suivantes [57] :

- 1°) Spécifications : on définit avec précision ce que fera le système (à quoi est-il destiné?);
- 2°) Conception et mise en oeuvre : on conçoit et on réalise le système;
- 3°) Validation : on teste le système pour voir s'il correspond aux objectifs précisés dans les spécifications;

- 4°) Évolution : ça correspond à tout ce qui vient après la livraison du produit (*versionning*, maintenances,...).

Ici, on ne va pas utiliser une méthode de conception particulière. Pour pouvoir tout de même y aller méthodiquement, nous nous inspirerons de ces étapes classiquement suivies lors de la conception des systèmes informatiques.

Dans ce second chapitre, nous ne présenterons que les spécifications du système ainsi qu'une ébauche de conception avec une présentation de l'architecture globale. La suite sera traitée dans le chapitre suivant.

II.6.1 Spécifications du système

Le système devra pouvoir permettre de réaliser ce qui suit :

- Synthétiser les textes qui lui sont fournis en entrée (saisis directement ou importés dans fichiers *.pdf* non scannés, des fichiers *.docx* et *.txt*);
- Servir les synthèses directement ou à travers un fichier *.pdf* à télécharger;
- Obtenir des synthèses produites par plusieurs algorithmes et les évaluer;
- Stocker les couples document-synthèse;
- Permettre l'affinage d'un modèle de synthèse automatique (ici nous réaliserons le *fine-tuning* du modèle *mBART* ou du modèle *mT5* selon celui qui se prêtera mieux à cet affinage).

C'est cela le minimum de besoins que le système devra être capable de combler.

Exigences fonctionnelles

Pour fonctionner, nous allons considérer que les fichiers chargés (pour en obtenir le résumé) seront au format *pdf*, *txt* ou *docx* et qu'ils seront en français.

Exigences non fonctionnelles

Comme exigence non fonctionnelle, on doit noter qu'un processus d'authentification sera utile, songeant à une rentabilisation future du système.

II.6.2 Présentation des éléments du système

L'architecture globale de notre système est un trois-tiers classique. Elle se présentera comme sur la figure II.4:

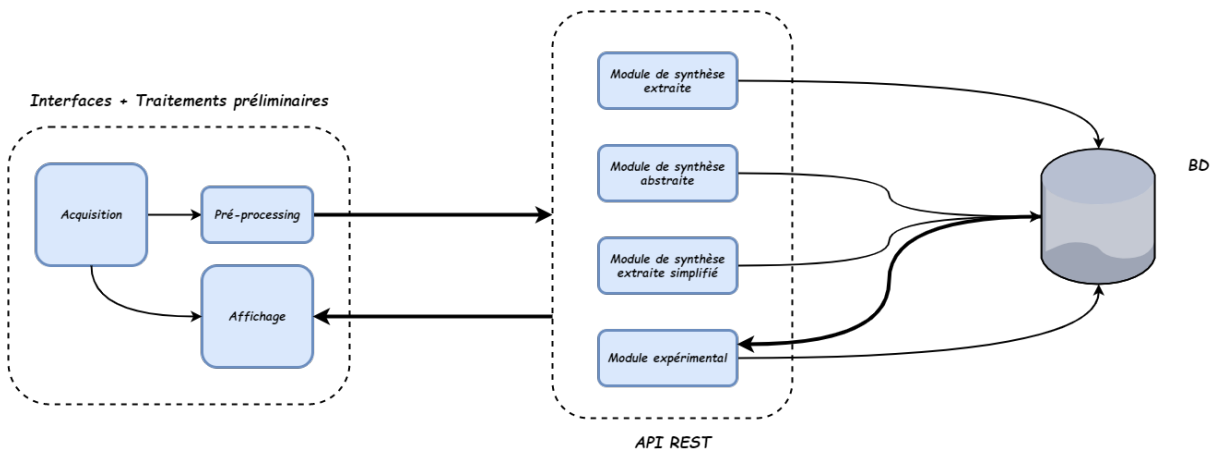


Figure II.4: Architecture globale de notre système

La figure II.4 présente l'architecture du système qui est une architecture 3-*tiers* classique. Il y a toutefois une partie qui n'est pas ici représentée car nous voulons nous donner une grande liberté de conception à son sujet. Il s'agit en fait de l'interface d'accès à l'API (*Application Programming Interface*), qui permettra aux développeurs de s'authentifier et générer éventuellement un *token* à utiliser pour implémenter leurs propres interfaces devant permettre d'utiliser les services de cette API. Il s'agit donc d'une API *privée*. Cette interface permettra aussi de voir toute la documentation de l'API (pour les développeurs) pour mieux utiliser ses services.

Quant au bloc interface que nous venons de présenter sur la figure II.4, c'est en nous mettant à la place d'un développeur lambda qui exploite les services de l'API.

Notre API quant à elle, est une API REST (*REpresentationnal State Transfer*) qui aura 4 *end-points* principaux dédiés à la synthèse automatique (selon les besoins d'implémentation, on pourra en insérer d'autres mais qui ne concerneront probablement pas la synthèse).

- *Module de synthèse extraite* : ce module réalisera une synthèse en combinant divers résultats d'algorithmes de synthèse extraite. Nous prévoyons, dans un premier temps, ne l'utiliser que pour des petits documents (la taille optimale sera déterminée avec les expérimentations au chapitre suivant).
- *Module de synthèse abstraite* : ce module donnera une synthèse abstraite en utilisant l'un des *transformers* affinés pour la synthèse ou bien par le module qui sera en train d'être amélioré au cours de l'utilisation du système (on l'a nommé *experimental* sur les interfaces, voir la figure II.9). Comme les *transformers* réalisent des synthèses de documents de taille généralement limitée à environ une page, nous mettrons au point, dans cette partie, un pipeline qui nous permettra d'augmenter le nombre de pages (nous pensons à 100 pages mais les expérimentations nous permettront de choisir une taille optimale, tenant compte surtout de la rapidité). Le pipeline en question peut se résumer par la figure II.5 qui suit :

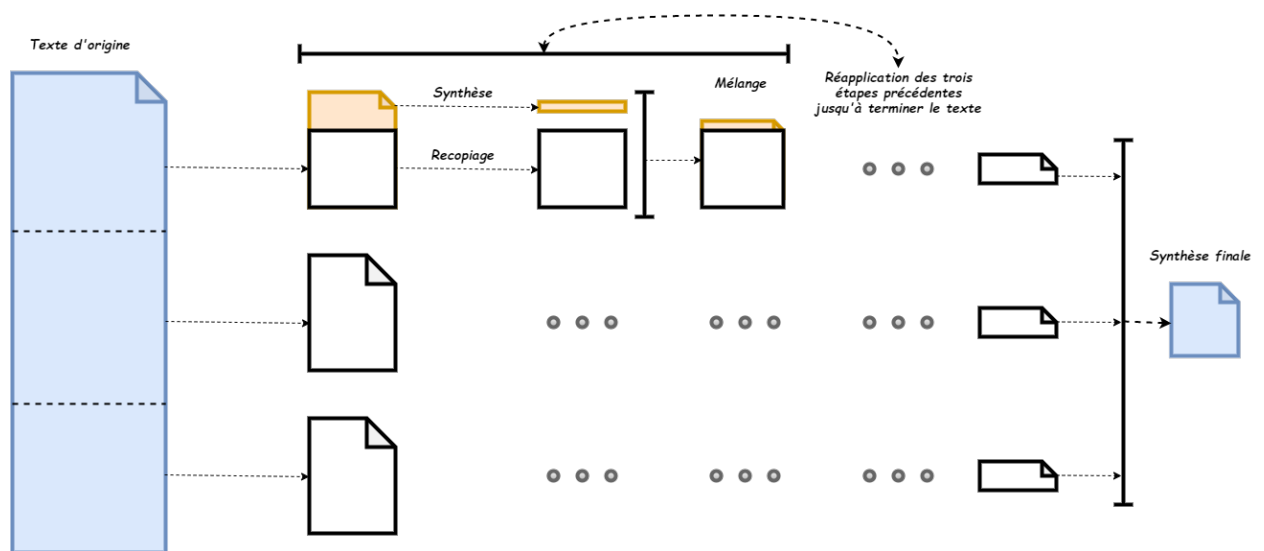


Figure II.5: Pipeline de synthèse abstraite

Sur la figure II.5, la particularité réside au fait que dans l'implémentation, les divisions se feront en découpant au préalable tout texte en ses phrases constitutives, puis, en reliant un certain nombre de phrases se présentant en tête de chaque partie jusqu'à atteindre une longueur inférieure à la limite admissible par le modèle utilisé.

Finalement, la synthèse obtenue pour cette sous-partie sera ajoutée à la partie globale et le processus se répétera comme l'illustre la figure. Nous avons testé cette méthode et elle nous a permis de mieux conforter le compromis vitesse-qualité par rapport aux pipelines classiques [45].

- *Module de synthèse extrait simplifié* : Il s'agira d'un module qui permettra la réalisation de la synthèse mais en utilisant l'un des algorithmes de synthèse extraite (le plus rapide).
- *Module expérimental* : Il s'agira d'un module de synthèse abstraite qui sera essentiellement utilisé pour la synthèse des petits documents (quelques pages). Ce module sera entraîné à partir des synthèses collectées par le système, pour améliorer au fur et à mesure ses performances. Nous comptons réaliser l'entraînement par *transfer learning* avec le *transformer mBART* [52] comme base.

On peut aussi remarquer qu'il y a un module *pre-processing* dans la partie interfaces. C'est à la suite du fait que, pour des raisons de performance, on devra envoyer à l'API le fichier sous un format particulier. Il faudra réaliser l'acquisition des données dans divers formats (*pdf, docx, ...*) mais les données acquises seront envoyées dans un format plus léger à l'API (du JSON pour notre cas). **Ce *pre-processing* consistera donc à extraire les données des documents fournis en entrée. Cela devra être fait sans les corrompre pour s'assurer de la qualité des résumés.**

La base des données, que nous avons mentionné dans la figure II.4 a un double rôle :

- 1°) Le stockage des données de l'utilisateur (il s'agira en fait des identifiants des interfaces qui utiliseront l'API);
- 2°) Le stockage des paires document-synthèse, ainsi que l'appréciation de l'utilisateur (évaluation par les utilisateurs).

II.6.3 Architecture du module de synthèse extractive

Le module de synthèse extractive, que nous nommerons *merging*, se présente comme suit :

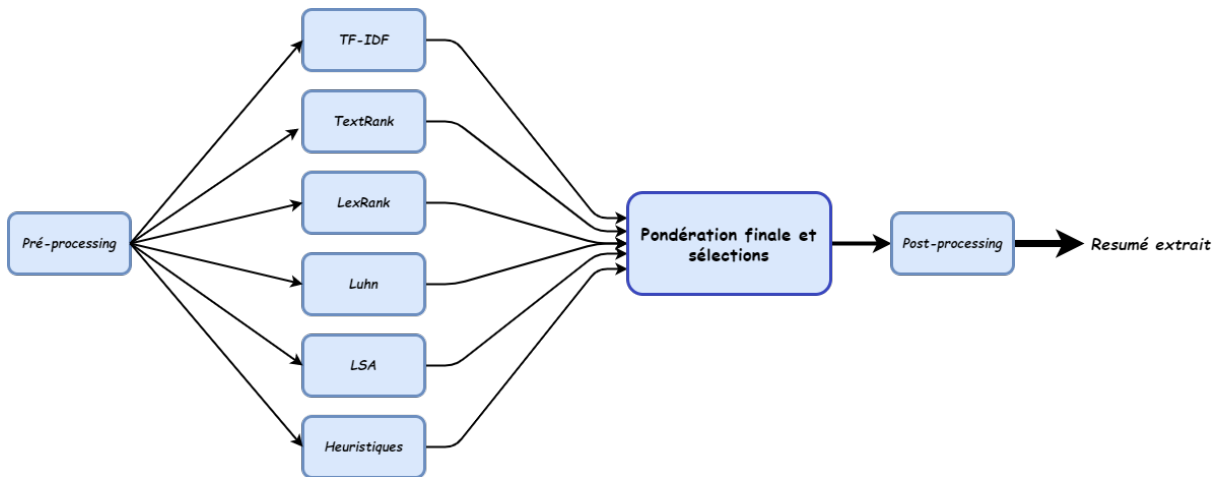


Figure II.6: Architecture globale du module de synthèse extractive

Comme nous pouvons le voir sur la figure II.6, un traitement sera fait pour adapter les données reçues à ce qui peut être traité par le système. Ce traitement consistera essentiellement à réaliser la *tokenisation* des textes (chaque *token* sera une phrase pour cette partie) et à affecter un identifiant unique à chaque phrase.

Après cela, les données seront invariablement passées aux divers algorithmes de synthèse extractive (*TFIDF*, *LexRank*, ...), qui généreront chacun un groupe de poids des phrases. Après cela, le module de pondération et sélection réalisera successivement ce qui suit :

- 1°) Acquisition des sorties de chaque algorithme de synthèse extractive (il s'agira des dictionnaires dont les clés seront les identifiants uniques des phrases et les valeurs seront les poids affectés par l'algorithme). A chaque algorithme, on donnera un poids qu'on nommera $W_{Nomdel'algo}$ compris entre 0 et 1, selon la confiance qu'on lui porte (la somme des poids sera égale à 1 et par défaut, tous les algorithmes pourront avoir le même poids) ;
- 2°) Élimination des phrases de poids faible (avec comme seuil, la taille maximale de résumé précisée par l'utilisateur) ;
- 3°) Réarrangement de chaque dictionnaire obtenu après expulsion des phrases non significatives (les éléments seront arrangés par ordre décroissant des poids pour chaque

sortie);

- 4°) Donner des probabilités aux espaces des poids de chaque dictionnaire par application d'un *softmax* sur chacun d'eux. Ce qui donnera, pour chaque phrase de chaque dictionnaire, un nouveau poids ω_{phr_i} , avec i le numéro du dictionnaire et phr le numéro de la phrase considérée dans ce dictionnaire ;
- 5°) Listage complet des éléments (leurs identifiants) de tous les dictionnaires.
- 6°) Pour chaque élément de la liste globale ainsi établie, appliquer la formule suivante pour obtenir un nouveau poids :

$$\mathcal{W}_j = \sum_{i \in \mathcal{D}} (W_i \cdot \omega_{phr_i}) \quad (\text{II.8})$$

Avec \mathcal{W}_j le nouveau poids affecté à la phrase ayant un identifiant global j (l'identifiant là d'origine) et \mathcal{D} la liste des dictionnaires (les sorties de chaque algorithme) ;

- 7°) Arranger toutes les phrases par ordre décroissant dans une unique liste et sélectionner les plus haut dans la liste jusqu'à atteindre le seuil fixé (nombre de mots fixé pour la synthèse).
- 8°) Constituer une liste avec les éléments sélectionnés.
- 9°) Réarranger les phrases de la liste selon leur ordre de succession dans le texte d'origine.
- 10°) Constituer la synthèse extraite.

Ce qui précède constitue en fait l'algorithme que nous allons implémenter pour le *module de pondération et sélection*. Précisons seulement que, les poids des algorithmes (voir le paramètre $W_{Nomdel'algo}$ du point 1 de la description ci-dessus) seront accordés de manière statique et non dynamique. Pour les choisir, nous allons nous servir des scores de chacun de ces algorithmes aux diverses métriques que nous présenterons au chapitre suivant. Et c'est à cette même occasion que nous préciserons les valeurs choisies pour ces paramètres (Valeurs à retrouver dans le tableau [III.2](#)).

II.6.4 Architecture du module de synthèse abstractive

Le module de synthèse abstraite n'est pas unique. Nous implémenterons plusieurs modèles (*BART*, *BART**hez*, *PEGASUS* et *mBART* entraîné avec nos données); Chaque module de synthèse se présentera néanmoins comme suit :

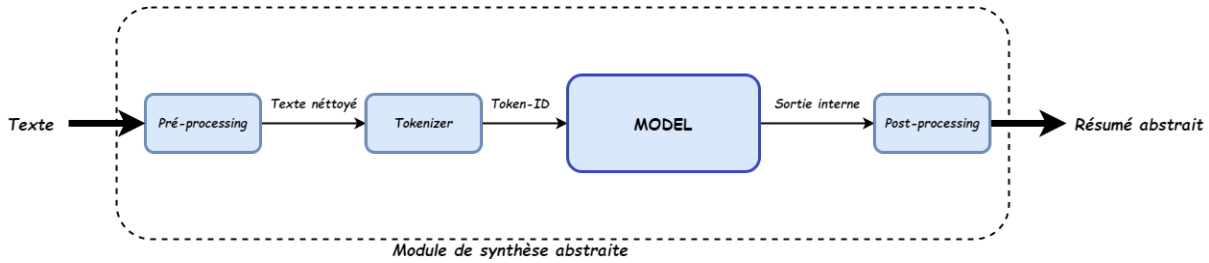


Figure II.7: Architecture globale du système de synthèse abstractive

Comme nous pouvons le remarquer, il y a toujours un module de mise en forme initial (*pre-processing*) qui nous permettra en gros de supprimer tous les caractères que nous ne pourrions pas gérer. Vient ensuite le module de tokenisation (le *tokenizer* ou tokeniseur) [45] qui consistera ici à diviser tout le texte en ses mots constitutifs et à leur affecter des identifiants numériques. Ce sont ces identifiants qui seront fournis au modèle et transformés en vecteurs par la couche d'*embedding* du modèle.

Le modèle quant à lui, aura toujours une architecture pareille :

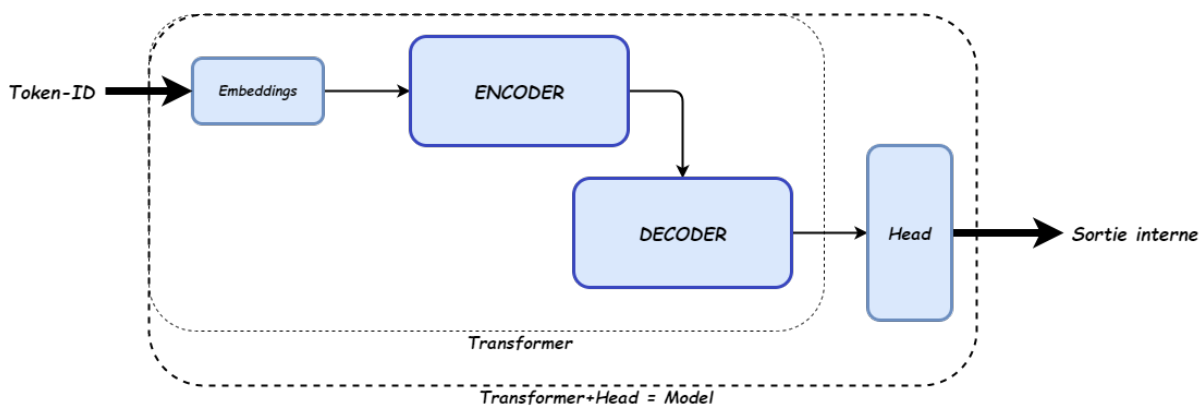


Figure II.8: Architecture interne du modèle mentionné sur la figure [II.7](#)

Il s'agit en effet de l'architecture classique d'un *transformer*, comme présenté sur la figure 1.7 à l'exception du fait qu'ici on fait explicitement apparaître l'existence de la sortie du modèle. Ça correspond au *réseau linéaire* suivi d'une couche de *softmax* tel que présenté sur la figure 1.7. Cette partie, que nous avons nommé *head* est différente selon les tâches [46], c'est pourquoi nous avons voulu la mentionner explicitement car, selon le besoin, on peut la modifier.

Nous devons finalement mentionner que les modules de *tokenisation* (nommés *tokenizer* en anglais) dépendront explicitement des modèles utilisés.

II.6.5 Présentation des interfaces

La partie interface nous permettra juste d'utiliser le service que nous aurons élaboré et d'évaluer par la même occasion ses performances. Voici donc une ébauche d'interface que nous comptons utiliser pour exploiter le service :

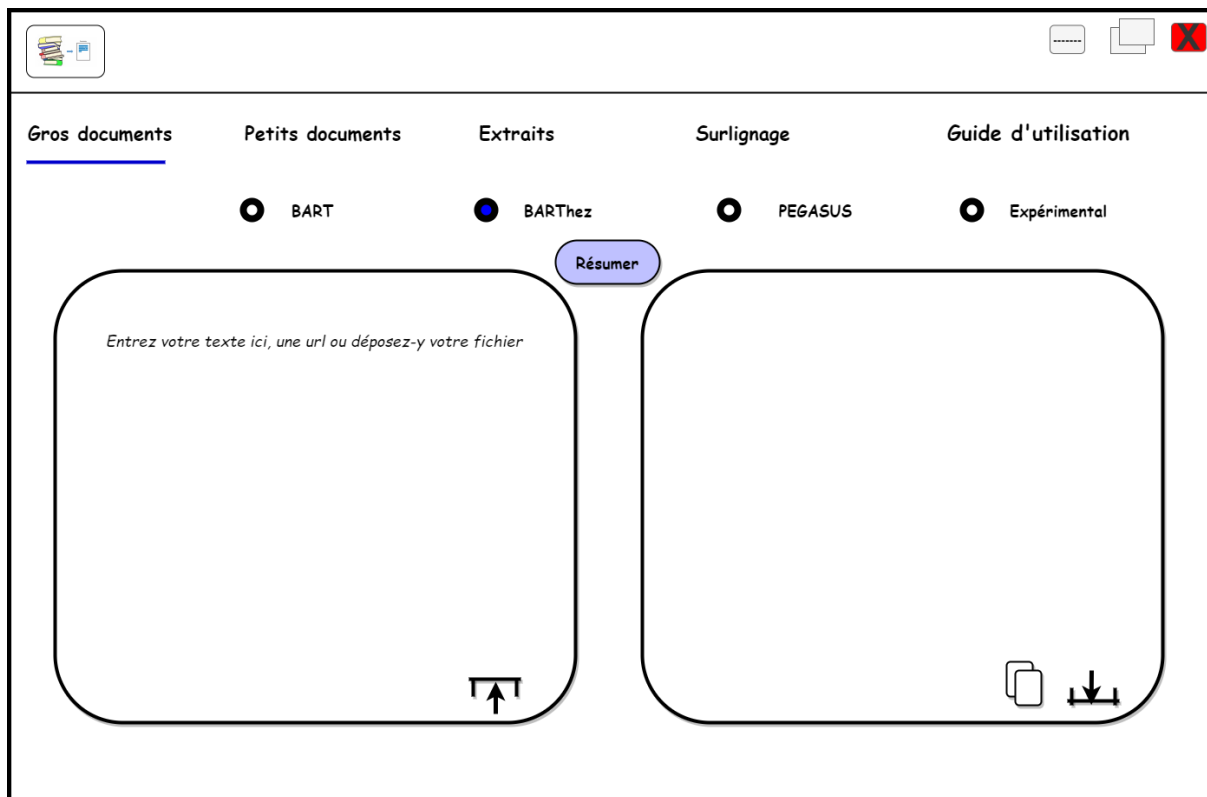


Figure II.9: Ébauche d'interface

Avec cette interface, on a une idée générale de la manière dont nous comptons servir le système aux utilisateurs.

II.7 Conclusion partielle

Dans cette partie, nous venons de présenter le résumé automatique des textes, tout en réalisant une vue d'ensemble des méthodes utilisées dans la littérature à cet effet. Nous avons mentionné que la classification des résumés que nous utiliserons sera celle les départageant en *abstractive summarization* et *extractive summarization* et que, pour notre cas, il s'agira de réaliser un système de résumé mono-document, avec une partie abstractive et une autre extractive, générant un résumé générique pour des documents de type narratif et argumentatif.

Nous avons également listé les divers modèles de *transformer* adaptés à la tâche de synthèse automatique abstraite, et nous avons mentionné devoir utiliser les modèles du type *BART* pour des raisons qui seront précisées dans le chapitre suivant.

Enfin, nous avons réalisé la conception préliminaire du système tout en précisant que, concernant l'*API*, la *BD* (Base des Données) et les interfaces, les détails d'implémentation utiles seront précisés dans la partie dédiée à la conception proprement dite et aux tests, c'est-à-dire au chapitre suivant. Le chapitre suivant nous permettra donc finalement de préciser, réaliser et tester les méthodes que nous avons jusque-là adoptées pour la mise au point de notre système de synthèse automatique des documents.

Chapitre III

Conception détaillée, réalisation et tests

III.1 Introduction partielle

Dans les chapitres précédents, nous avons esquissé tout l’environnement du système que nous avons prévu élaborer à travers ce mémoire. A présent, nous allons présenter avec précision les diverses parties de notre système.

Dans ce chapitre, nous comptons en premier lieu finaliser la conception entamée au chapitre précédent. En second lieu, nous allons décrire les diverses méthodes utilisées pour évaluer les résumés générés automatiquement, nous allons décrire l’implémentation de notre système, nous allons tester les algorithmes et modèles dont nous avons fait usage et nous allons justifier les choix d’implémentation que nous avons faits.

Les tests en particulier seront faits de manière à pouvoir donner une justification cohérente des choix d’implémentation que nous avons faits et à classer, par la même occasion, nos algorithmes les uns par rapport aux autres. Cela est donc l’objet de ce chapitre et nous estimons qu’il nous permettra de présenter les points techniques saillants de notre système.

III.2 Conception détaillée

Comme nous l’avons annoncé au précédent chapitre, dans cette partie nous allons finaliser la conception entamée. Cela consiste principalement en la conception de la base des données, de l’API et des interfaces. Plus précisément, il s’agira de conception et présentation.

III.2.1 Conception de la base des données

La conception d'une base des données passe en général par les étapes qui suivent [58] :

- L'analyse des besoins en stockage;
- Le regroupement des données repérées comme utiles en tables selon leur affinité;
- La spécification des clés primaires et l'analyse des relations entre tables;
- La normalisation de la base des données.

Étant donné ce que nous avons précisé au chapitre précédent comme spécifications du système, et ensuite ce que nous avons mentionné comme rôles principaux de la base des données dans notre système, il se dégage les besoins suivants en terme de stockage :

Il nous sera utile de stocker principalement les textes et leurs résumés, les évaluations des résumés par les utilisateurs, les dates et le modèle source. Il nous sera également utile de stocker les clés d'authentification des systèmes utilisant l'API, les noms et logins des développeurs se servant des services de l'API mais également leurs mots de passe.

Cela nous montre que nous n'aurons besoin que de 3 tables au maximum :

Une pour tout ce qui concerne les textes, une pour la gestion des clés des développeurs enregistrés sur la plateforme, et une autre pour la gestion de leur identité. Nous préférons séparer les tables sur l'identité des développeurs avec celle contenant leurs clés car on peut avoir égaré la clé d'authentification et vouloir en générer une autre, d'où c'est plus judicieux de considérer les informations sur les clés comme une entité à part.

Des considérations qui précèdent on tire le diagramme des classes suivant :

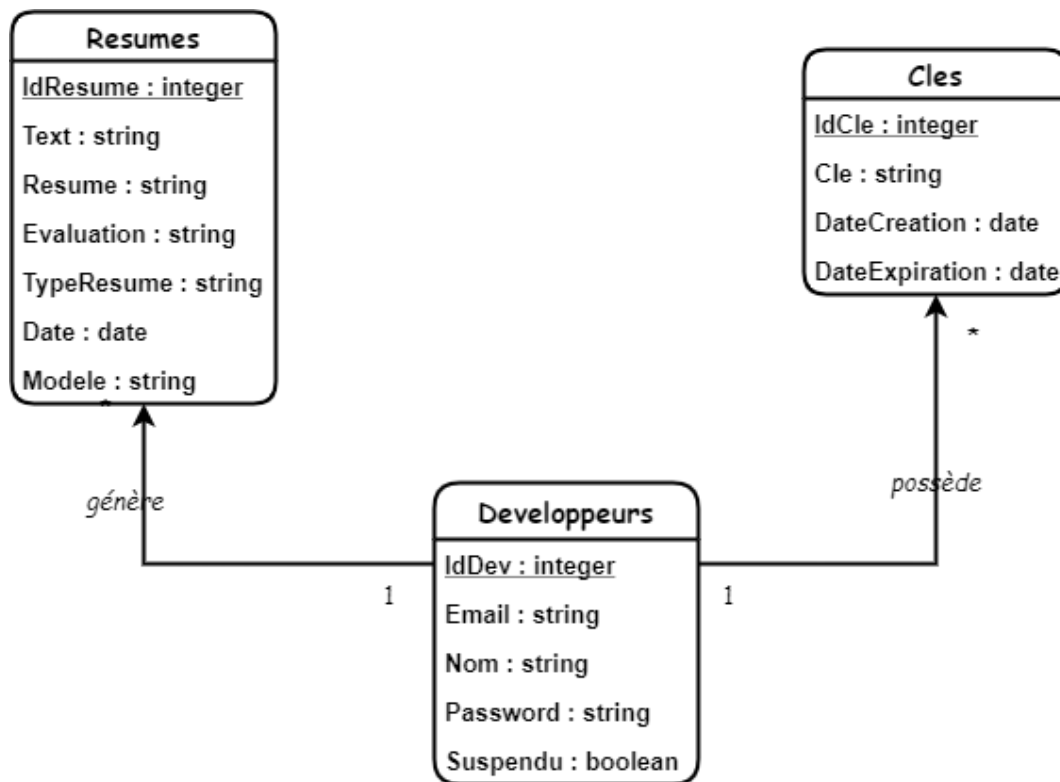


Figure III.1: Diagramme des classes représentant la base des données

On peut remarquer sur la figure III.1 l'ajout des attributs type de résumé et date dans la classe des résumés, ainsi que des attributs date de création et date d'expiration dans la classe des clés d'authentification, puis finalement l'attribut suspendu dans la classe des développeurs.

- L'attribut *type de résumé* permettra de sélectionner les résumés par type sans avoir à se fier des modèles mis en place dans le système;
- L'attribut *date* ajouté dans la classe des résumés pourra permettre de suivre l'évolution des performances car on aura également le modèle qui aura produit les résumés considérés.
- Les attributs *date de création* et *date d'expiration* ajoutés à la classe des clés permettront de faciliter plus tard le management des développeurs utilisateurs des services de

l'API dans leurs systèmes et cela va en connivence avec l'attribut *suspendu?* ajouté à la classe des développeurs.

Les *id* seront considérés comme clé primaire et l'id de développeur sera une clé étrangère dans les tables réservées aux résumés et aux clés. Ainsi, nous considérons terminée la phase de conception de la base des données.

III.2.2 Conception de l'API de synthèse

La création d'une interface de programmation des applications (*Application Programming Interface*) se fait en s'interrogeant sur les services qu'on veut pourvoir à travers cette API. Pour ce qui nous concerne, nous voulons mettre au point une plateforme capable de retourner les résumés aux textes qui lui seront présentés. Nous devons également pouvoir identifier les gens qui se serviront de l'API dans leurs systèmes (les développeurs dont on parlait dans la section précédente), d'où la nécessité de les enregistrer, de leur donner la possibilité de s'authentifier et de générer une clé d'authentification à utiliser dans leurs systèmes.

Nous allons également donner la possibilité de tester plusieurs modèles de résumé des textes et permettre ainsi aux utilisateurs de comparer diverses synthèses. Pour cela, nous avons prévu 4 familles d'*end-points* :

- 1°) *End-points* réservés à la synthèse extractive;
- 2°) *End-points* réservés à la synthèse abstractive;
- 3°) *End-points* réservés à tout ce qui concerne les vues de l'API (ses interfaces devant permettre l'authentification, la génération des clés et la documentation);
- 4°) *End-points* réservés à l'administration du système (que nous avons implémenté mais que nous passerons sous silence dans ce travail).

On peut aussi y joindre un *end-point* pour l'évaluation des résumés produits.

End-points pour la synthèse extractive

Pour la synthèse extractive, nous avons prévu essentiellement deux *end-points* pour comparer notre algorithme de synthèse extractive à l'un des systèmes de synthèse extractive le plus performant. Il s'est agit des *end-points* réservé au système basé sur le module python *gensim* et celui réservé à notre approche qui est un mélange de plusieurs algorithmes de synthèse extractive (on l'a nommé *merging*) comme on peut le voir sur la figure II.6 du chapitre précédent.

On a l'*end-point* de synthèse extractive basé sur le module *gensim* donné par :

POST /extractive-gensim

Puis celui de synthèse extractive par combinaison de divers algorithmes donné par :

POST /extractive-merge

End-points pour la synthèse abstractive

Pour la synthèse abstractive, nous avons prévu 5 *end-points* dont :

1. Un *end-point* utilisant le modèle *BART* dont nous avons optimisé l'utilisation en le servant à travers le processus présenté à la figure II.5 du chapitre précédent. L'*end-point* en question est :

POST /abstractive-large

2. Un autre *end-point* basé sur *BART* mais pouvant prendre en charge des documents plus gros :

POST /abstractive-very-large

3. Un *end-point* basé sur *BARThez*, le modèle de synthèse le plus utilisé pour la synthèse entièrement en français :

POST /abstractive-barthez

4. Un *end-point* basé sur *Pegasus*, l'un des modèles à grand pouvoir d'abstraction :

`POST /abstractive-pegasus`

5. Un *end-point* basé sur *BARTkrame*, notre modèle de synthèse élaboré pour fonctionner entièrement en français :

`POST /abstractive-experimental-bartkrame`

End-points pour les vues de l'API et l'authentification

Tout d'abord, le point d'entrée est donné naturellement par l'*end-point* :

`GET /`

Ensuite, l'*end-point* réservé à l'authentification :

`POST /login`

Vient alors l'*end-point* dédié à l'enregistrement :

`POST /sign-up`

Par après on a l'*end-point* réservé à la déconnexion :

`GET /logout`

Puis celui réservé à la génération des clés :

`GET /update_token`

Finalement, on a un *end-point* commun aux familles de synthèse extractive et abstractive pour l'évaluation des synthèses. C'est donné par :

`POST /evaluation`

Nous estimons que la description ici faite suffit comme conception des grandes lignes de notre *API*. Le reste de la documentation est dans l'annexe [.2](#).

III.2.3 Conception des interfaces

Étant donné l'API mise sur pieds, nous avons également implémenté un client web devant s'en servir pour en illustrer le fonctionnement. Pour cela, nous avons voulu exploiter toutes les possibilités de l'API en se basant sur les spécifications du système (nous les avons cités au chapitre précédent) et les objectifs de ce travail. C'est ainsi que les interfaces ont été conçues de manière à pouvoir :

- Permettre la synthèse abstractive en utilisant tous les modèles mis à notre disposition par l'API;
- Permettre la synthèse extractive en utilisant les deux approches pourvues par l'API;
- Permettre le téléchargement des synthèses obtenues;
- Permettre aux utilisateurs de coter les synthèses qui leur ont été fournies;
- Permettre le chargement des fichiers sous différents formats (**word**, **pdf** et **txt**).

Pour cela, et sur base de l'ébauche des interfaces qui a été présentée à la figure II.9 nous avons adopté ce qui suit comme interfaces du client de notre système.



Figure III.2: Interface générique du système

Sur la figure III.2 nous montrons la manière dont les interfaces se présentent en général. Durant le processus de synthèse, l'interface se présente comme suit :

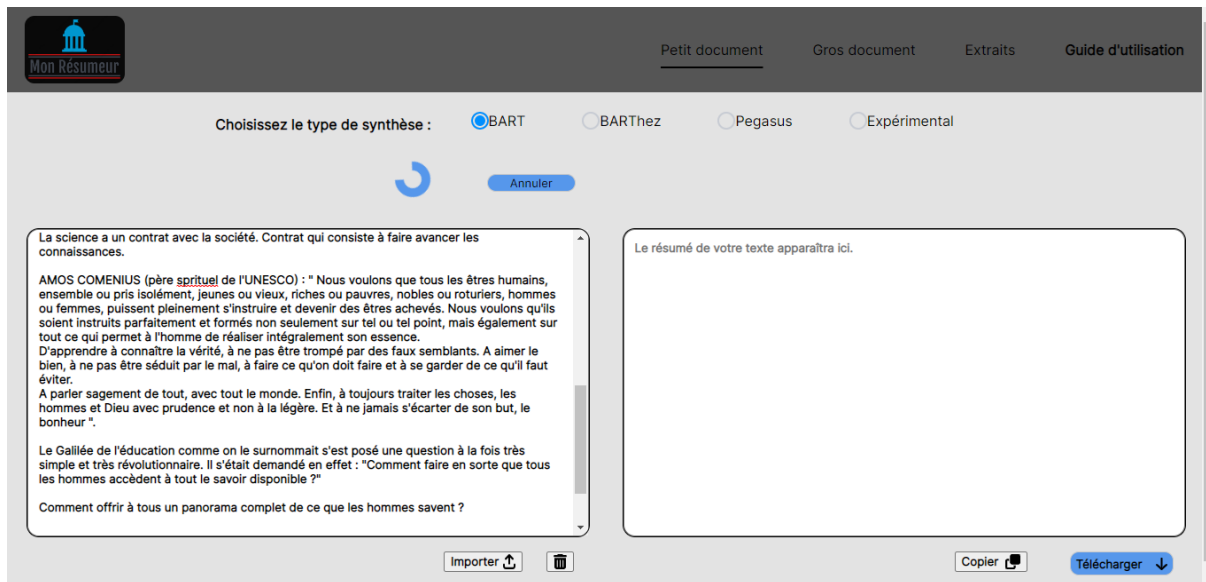


Figure III.3: Interface durant le processus de synthèse

On peut remarquer sur la figure III.3 que durant la synthèse, une possibilité d'annulation se présentera.

Après génération de la synthèse, l'interface se présentera comme suit :

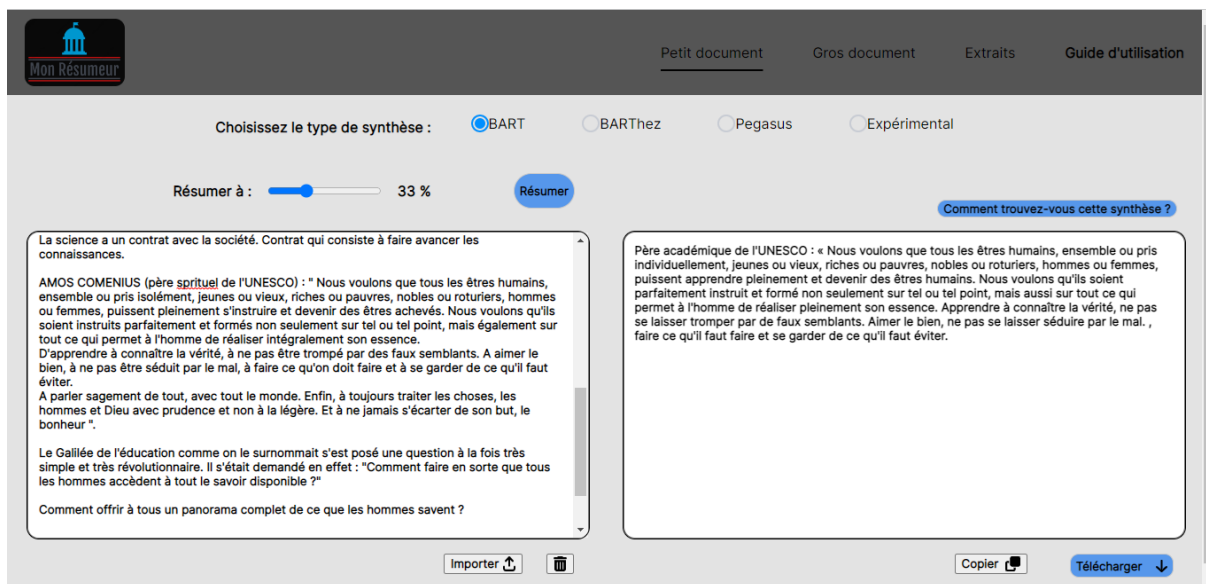


Figure III.4: Interface après génération de la synthèse

Il est clair également sur la figure III.4 qu'après synthèse, un bouton devant permettre d'évaluer la synthèse se présentera. En y cliquant, on pourra avoir la possibilité d'évaluer le résumé obtenu, comme cela se voit sur la figure suivante :

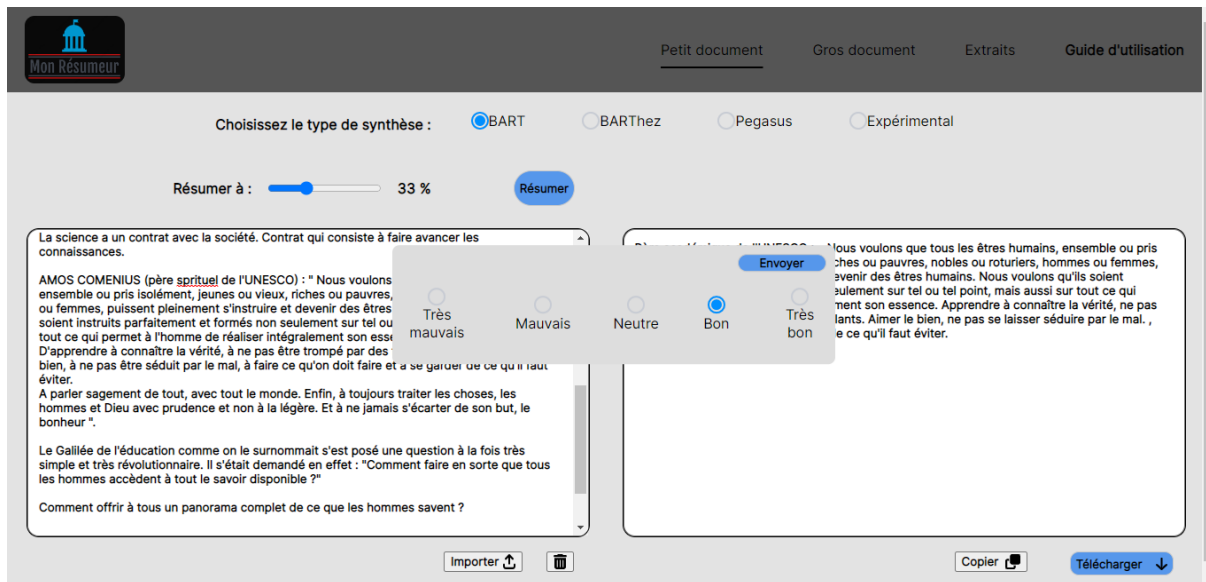


Figure III.5: Interface durant l'évaluation de la synthèse obtenue

Pour guider l'utilisateur, au survol du curseur de réglage de la taille du résumé, l'interface se présente comme suit :

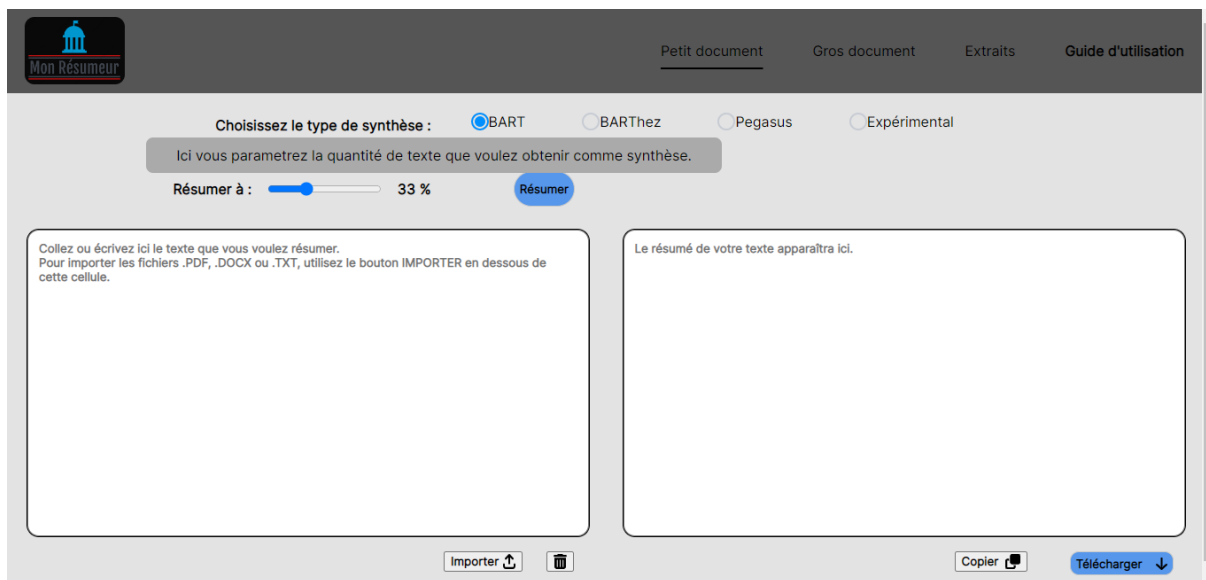


Figure III.6: Interface au survol du curseur de réglage de résumé

En cas de limitation des ressources serveur lors de l'hébergement de l'API, nous avons prévu disponibiliser uniquement le système réduit :

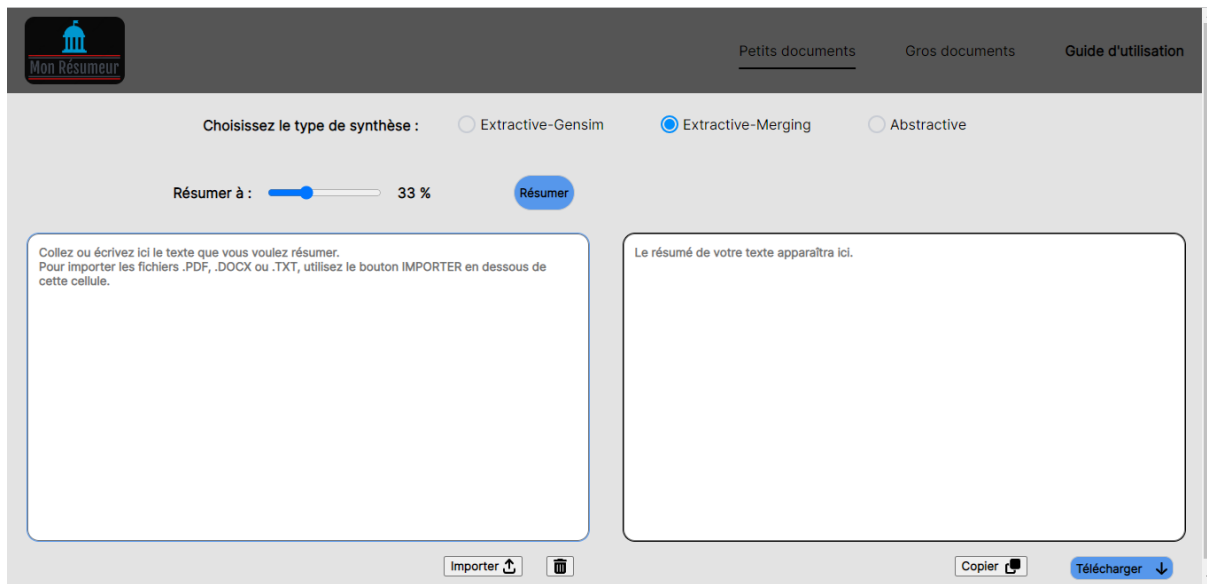


Figure III.7: Interface réduite côté petits documents

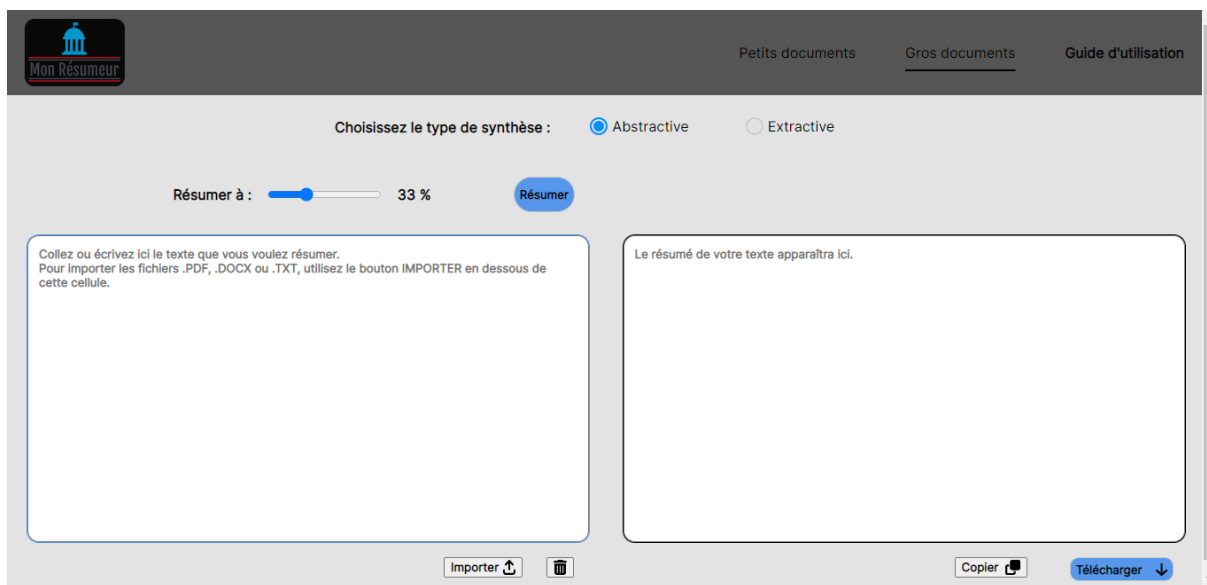


Figure III.8: Interface réduite côté gros documents

Les autres fonctionnalités peuvent être explorées en utilisant le système.

III.3 Évaluation des résumés

L'évaluation des résumés est une tâche assez complexe. Les méthodes classiquement utilisées pour évaluer les performances d'un système informatique, et en particulier celles utilisées pour évaluer les systèmes basés sur les réseaux de neurones, ne peuvent pas

s'appliquer de manière brute pour les systèmes de génération de texte comme ceux de résumé automatique. En effet, il n'existe pas de référence parfaite de résumé pour un document donné. Les professionnels humains d'ailleurs produisent des résumés différents pour un même texte, chacun selon ses penchants et ses goûts. Plus étonnant encore, un même humain ne produit en général pas le même résumé si la tâche lui est donnée à quelques temps d'intervalle [59]. C'est donc une tâche teintée d'une grande part de subjectivité. Cependant, la majorité s'accorde quand on est face à un bon résumé.

Comment alors faire pour parvenir à repérer, avec plus d'objectivité, qu'un résumé donné est de bonne qualité ? Comment même parvenir à comparer, sans équivoque, les qualités de deux résumés qui nous seraient présentés ? La solution à cette question est importante car elle permettra de pouvoir évaluer objectivement et rapidement les systèmes de résumé automatique mais, au-delà de cela, elle permettra d'en suivre l'évolution durant l'entraînement au cas où il s'agit des systèmes de résumé basés sur l'apprentissage automatique.

Généralement, on subdivise l'évaluation des résumés en deux principales catégories qui sont l'évaluation extrinsèque et l'évaluation intrinsèque[60]. L'évaluation intrinsèque est celle qui s'obtient par la lecture directe du résumé. Il peut s'agir d'un résumé manuel (évaluation directe par un humain selon un certain nombre de critères), un résumé semi-automatique ou bien un résumé automatique. L'évaluation extrinsèque quant à elle consiste à évaluer les résumés indirectement. Pour ce faire, on peut par exemple passer par l'évaluation de la possibilité de donner une réponse à un certain nombre de questions après lecture du résumé [61],... Beaucoup de recherches ont été menées sur l'évaluation des résumés mais les solutions ne sont que partielles le plus souvent [62, 63, 64, 65]. Dans ce travail, nous ne parlons que des méthodes d'évaluation intrinsèques qui nous intéressent.

III.3.1 Évaluation manuelle

L'évaluation manuelle consiste à se fier au jugement d'un certain nombre d'experts humains, selon un certain nombre de critères préétablis [12]. Pour plus d'information à ce sujet, se rapporter aux campagnes annuelles d'évaluation **DUC** ou *Document Understanding*

Conference (renommé **TAC** [66] ou *Text Analysis Conference* depuis 2008). Ces méthodes demeurant subjectives, les méthodes semi-automatiques ont été mises au point.

III.3.2 Évaluation semi-automatique

Nous allons présenter ici les méthodes les plus connues et les plus utilisées [67] pour évaluer les résumés. Il s'agit de **ROUGE** (*Recall Oriented Understudy for Gisting Evaluation*), que nous allons utiliser pour nos évaluations et de **PYRAMID** dont nous allons montrer la faiblesse par rapport à *ROUGE*, bien que celui-là soit plus corrélé aux jugements humains [68].

ROUGE [62]

Cette métrique est orientée rappel comme son nom l'indique. On cherche à évaluer jusqu'à quel point le résumé obtenu rappelle celui qui est pris comme référence. Pour réaliser ladite mesure, on se base essentiellement sur la correspondance mot à mot (on parle des *unigrammes*), de deux mots à deux mots (on parle alors de comparaison par *bigrammes*),... Pour être général, on parle des *n-grammes*. Considérons la phrase :

Il arrive demain avec deux objets rares.

Les unigrammes de cette phrase sont *il*, *arrive*, *demain*, *avec*, *deux*, *objets* et *rares*, soit les mots du texte. Les bigrammes sont *il arrive*, *arrive demain*, *demain avec*, *avec deux*, *deux objets*, *objets rares*. Dans le paquet des mesures proposées par *ROUGE*, il y a :

1°) ROUGE-N

Formellement, *ROUGE-N* est une mesure de rappel (*recall*) des *N-grammes* entre le résumé candidat et le ou les résumé(s) de référence. C'est régi par la relation qui suit :

$$ROUGE - N = \frac{\sum_{r \in R} Co_{c-r}}{\sum_{r \in R} (N - grammes)_r} \quad (III.1)$$

Avec *r* un résumé de référence parmi ceux pris pour référence (paquet *R*), $(N - grammes)_r$ le nombre total de *N-grammes* dans un résumé de référence *r* et Co_{c-r}

le nombre de *N-grammes* du résumé candidat qui se retrouvent dans le résumé de référence *r*.

C'est ainsi qu'on trouve *ROUGE-1* basé sur une comparaison des *unigrammes*, *ROUGE-2* basé sur celle des *bigrammes*,... *ROUGE-2* est particulièrement jugée très efficace pour l'évaluation des résumés[2].

2°) ROUGE-L

Il s'agit cette fois d'une mesure *ROUGE* basée non pas sur des *n-grammes* comme tel, mais plutôt sur l'évaluation des plus longues sous-séquences communes entre les phrases du résumé candidat avec celles du résumé de référence. Pour faire simple, considérons un résumé candidat *X* et un résumé de référence *Y*. Si *m* est le nombre d'unigrammes contenus dans *X* et *n* celui de *Y* on va définir deux rappels (sur *X* et sur *Y*) donnés par :

$$R_X = \frac{LCS(X, Y)}{m} \quad (\text{III.2})$$

$$R_Y = \frac{LCS(X, Y)}{n} \quad (\text{III.3})$$

Les expressions III.2 et III.3 nous permettent finalement de définir la mesure *ROUGE-L* par :

$$ROUGE - L = \frac{(1 + \beta^2)R_X R_Y}{R_X + \beta^2 R_Y} \quad (\text{III.4})$$

Dans les expressions III.2, III.3 et III.4, *LCS(X, Y)* désigne la plus longue séquence commune à *X* et *Y*. On la nomme habituellement *Long Common Subsequence (LCS)*. Pour illustration, considérons deux phrases :

A = Nous voulons savoir ce qui se passera.

B = Nous finirons par savoir ce qui se passe.

On remarque que la plus longue sous-séquence commune à *A* et *B* est *nous savoir ce qui se*, donc *LCS(A, B) = 5*. Il faut noter qu'ici nous n'avons décrit que la procédure de calcul de *ROUGE-L* pour des résumés à une phrase. Néanmoins, pour un résumé

à plusieurs phrases, l'idée est la même mais la comparaison est faite phrase après phrase [62].

3°) ROUGE-S

Cette méthode se base sur ce qu'on nomme les *skip-bigrammes* pour évaluer les résumés. Cela correspond à utiliser les bigrammes sans se limiter au fait que les termes soient consécutifs. Par exemple, dans la phrase :

La terre semble être plate

On a les *skip-bigrammes* suivants : *la terre, la semble, la être, la plate, terre semble, terre être, terre plate, semble être, semble plate* et finalement *être plate*, soit 10 *skip-bigrammes*. On a en fait toujours C_n^2 *skip-bigrammes* avec n le nombre d'unigrammes dans la phrase considérée. Pour notre cas, il s'agit de 5 unigrammes et on a bel et bien $C_5^2 = \frac{5!}{3!2!} = 10$. Après cela, on établit des relations du même type que celles établies pour ROUGE-L (voir la relation III.4).

Il faut noter que des améliorations peuvent être apportées à ces méthodes pour avoir d'autres variantes. Cela permet d'obtenir ROUGE-S avec un *gap maximal* (séparation maximale entre N -grammes formant les *skip-bigrammes*), ROUGE-SU_{ROUGE-L}Sum et ROUGE-W par exemple. Pour plus de détails à ce sujet, on peut regarder l'article fondateur du paquet d'évaluation ROUGE [62].

Il faudra noter également que l'une des sources du succès de ROUGE est sa facilité d'implémentation une fois qu'on a déjà des résumés de référence, ainsi que sa grande corrélation avec les jugements humains [62, 2, 67].

PYRAMID [68]

Cette méthode permet de comparer un résumé candidat à un ensemble de résumés de référence. L'inconvénient de cette méthode est que, par essence, elle exige l'existence de plusieurs résumés de référence, ce qui n'est pas exigé pour la méthode ROUGE.

Les méthodes semi-automatiques peuvent être vues comme automatique au cas où

trouver les résumés de référence ne pose aucun problème. Malheureusement, cela ne suffit pas car la subjectivité est considérée comme étant aussi un facteur de biais. D'où l'émergence des méthodes automatiques.

III.3.3 Évaluation automatique

Les méthodes d'évaluation automatiques réalisent l'évaluation en se basant uniquement sur le résumé obtenu et le texte résumé (texte d'origine). La non exigibilité des références rend ces méthodes particulièrement intéressantes.

Nous n'avons pas utilisé ces méthodes d'évaluation dans ce travail, en plus, elles ne sont pas encore mûres [2, 64, 69].

III.4 Description succincte de l'implémentation

Comme on peut le voir sur la figure II.4, notre système est conçu selon l'architecture trois tiers. Nous avons donc la partie cliente et la partie serveur. Néanmoins, s'agissant de notre système (que nous avons dénommé *Mon Résumeur*), cette distinction n'est pas trop claire. Nous allons donc préciser clairement les éléments qui le constituent. L'implémentation du système a été faite en considérant les parties suivantes :

- 1°) Une *API REST* que nous avons implémenté pour le traitement des données lui envoyés par les parties clientes. Le traitement en question consiste à synthétiser le texte qui lui est fourni, d'envoyer le résultat de ce traitement au système demandeur et de stocker les résultats du processus. Cette *API*, tel que nous l'avons implémenté est constituée d'une partie *frontend* et d'une partie *backend*. Le *frontend* c'est pour permettre aux programmeurs de se documenter sur les *end-points* disponibles par l'*API* et leur fonctionnement, tout en leur permettant de s'authentifier pour pouvoir utiliser les services de l'*API* dans leurs systèmes respectifs.
- 2°) Du point 1° on comprend directement qu'il y a également une base des données. Pour cette première version de notre système, nous avons d'abord opté pour une base de données du type *SQLite*.

3°) Une partie cliente qui a également un *frontend* et un *backend*. Cette partie a été implémentée pour montrer que l'*API* que nous avons mis au point est bel et bien utilisable. N'importe quel autre programmeur ayant les autorisations nécessaires peut donc se servir de l'*API* dans son système. Pour notre cas, le système que nous avons mis au point pour utiliser l'*API* a une partie *frontend* devant permettre aux utilisateurs de charger leurs documents et une partie *backend* devant permettre de réaliser quelques traitements avant d'envoyer les documents chargés par les utilisateurs. Ce petit traitement du début à pour objectif de corriger les erreurs de présentation de textes qui peuvent subvenir selon les outils et les langages utilisés pour cette fin.

III.4.1 API REST

Partie *frontend*

Quelques captures de la partie *frontend* de l'*API* sont mises à l'annexe [.3](#). Cette partie a été faite en utilisant *HTML*, *CSS* et très peu de *JavaScript* pour la génération des clés par les programmeurs voulant utiliser l'*API*. Ce *frontend* permet donc aux programmeurs de s'authentifier et de générer leurs clés mais également de consulter la documentation de l'*API*.

Partie *backend*

Cette partie est le coeur du système. C'est dans elle que nous implémentons les divers algorithmes de synthèse ainsi que la gestion des requêtes des utilisateurs. Elle a été réalisée en utilisant le *framework Python* dénommé **Flask** vu que nous avons implémenté nos algorithmes de synthèse en Python. Cette partie disponibilise plusieurs *end-points* comme on peut le voir dans la documentation mise à l'annexe [.2](#).

Les précisions sur l'implémentation des algorithmes de cette partie sont les suivantes :

- Pour la *synthèse extractive simplifiée* nous avons utilisé le module python dénommé **Gensim**.

- Pour la synthèse que nous avons réalisé en mélangeant divers algorithmes, nous avons modifié le module python *sumy* (version 0.0.1). Nous l'avons modifié de manière à pouvoir accéder aux poids et non aux synthèses, de manière à pouvoir mieux contrôler le ratio, de manière à y ajouter l'algorithme *Tf-Idf* et à le rendre compatible à la langue française. Cela avec aussi comme objectif d'avoir un contrôle total sur le pré-traitement, de l'uniformiser pour tous les algorithmes (même *tokenizer* pour tous) et de combiner les sorties selon l'algorithme conçu à la section II.6.3.
- Pour la synthèse des petits documents, nous avons utilisé le *transformer BART* que nous avons servi à travers le pipeline décrit à la figure II.5, en ajustant également les longueurs des entrées aux tailles que nous avons jugé optimales pour réduire la complexité de calcul, car les modèles du type *transformer* ont une *complexité algorithmique linéaire en la longueur de leurs têtes d'attention mais quadratique en la longueur des séquences d'entrée* [7]. Dans cette partie dédiée aux petits documents, nous avons également ajouté le *transformer BARThez* ainsi que notre *transformer* basé sur *mBART* et que nous serons entrain d'améliorer au fil de l'utilisation du système. Nous avons mis *BARThez* juste pour avoir un point de repère (de comparaison) en français.

Le modèle *BART* étant en anglais, nous devons préciser que notre *API* passe par l'*API* de *google translate* pour pouvoir tout présenter en français. Notre modèle dit *expérimental* ainsi que *BARThez* n'ont pas eu besoin de traduction car ils sont complètement en français.

- Pour la partie *gros documents*, pour la synthèse abstractive, nous avons juste appliqué le modèle *BART optimisé* tel que nous venons de le décrire de manière à pouvoir prendre en compte des documents plus longs sans perte de qualité. Pour cela, la partie a juste consisté à appliquer notre pipeline d'optimisation (figure II.5) au complet. Pour la synthèse extractive des longs documents, nous avons utilisé *gensim* à cause de sa rapidité.

Concernant des précisions supplémentaires sur nos modèles et les résultats, il faudra se rapporter à la section III.5.

III.4.2 Système client

Le système client que nous avons implémenté est également constitué de deux parties principales dont le *backend* et le *frontend*.

Le *frontend* correspond à ce que nous avons présenté quand nous avons traité de la conception des interfaces dans ce travail (il s'agit du point III.2.3). La partie visible (*frontend*) a été réalisée en utilisant le *framework JavaScript* dénommé *ReactJS*.

Le *backend* pour cette partie, nous a juste permis de réaliser l'extraction des textes et le formatage avant envoi à l'*API*. Il a été réalisé en se servant du module *NodeJS* dénommé *ExpressJS*. Pour cette partie, nous n'avons prévu ni authentification, ni stockage mais cela peut évidemment être fait selon le choix du programmeur se servant de l'*API* de synthèse dans son système.

III.5 Tests et interprétation des résultats

Dans cette partie, chaque fois que nous parlerons de *checkpoint* ou de *point d'entrée*, il faudra comprendre qu'il s'agit du nom du modèle ou bien du *dataset* sur la plateforme *Hugging Face*¹. Nous devons également souligner que les résultats *ROUGE* sont ici présentés en pourcentage et plus c'est élevé, plus le modèle considéré peut être quantitativement jugé performant sur le *dataset* pris comme référence (pour plus de précisions sur le paquet *ROUGE*, voir la section III.3.2).

III.5.1 Datasets utilisés pour les tests

En parlant des tests, nous voulons dire ceux réalisés pour évaluer les performances des modèles les uns par rapport aux autres. Pour cela, il nous est nécessaire d'utiliser les mêmes *datasets* ainsi que les mêmes métriques de mesure des performances. Les *datasets* que nous avons utilisés font partie de ceux qui sont généralement utilisés dans divers *benchmarks* dédiés à la synthèse automatique.

Ainsi, nous avons utilisé les *datasets* *CNN / DailyMail* (version 3.0.0), *XSum*, *OrangeSum*

¹<https://huggingface.co>

et *for-ULPGL-Dissertation* (Version 0.0.1, la première version). Tous ces *datasets* sont accessibles sur *Hugging Face*² et sont constitués de trois sous-catégories dont les données de *test*, les données d'*entraînement*, ainsi que les données de *validation*.

Dataset CCN / DailyMail

Le jeu de données *CNN / DailyMail* est un jeu de données en langue anglaise contenant un peu plus de 300 000 articles d'actualité uniques, rédigés par des journalistes des chaînes *CNN* et du *Daily Mail*. La version actuelle prend en charge le résumé extractif et abstraktif, bien que la version originale ait été créée pour la lecture et la compréhension automatiques et la réponse aux questions abstractives³.

Pour nos tests, nous avons utilisé la **version** 3.0.0 retrouvable au point d'entrée `ccdv/cnn_dailymail`. Ce qui veut dire qu'on peut le télécharger en utilisant les lignes de code python suivantes :

```
1 from datasets import load_dataset
2
3 dataset = load_dataset("ccdv/cnn_dailymail", '3.0.0')
```

Code à utiliser en ayant installé au préalable le module *datasets* grâce à `pip install datasets`.

Dataset XSUM

Il s'agit d'un jeu de données de résumés de nouvelles avec des résumés généralement qualifiés très abstraits. Ces données ont été constituées en collectant des articles en ligne de la *BBC (British Broadcasting Corporation)* qui sont accompagnés des petits résumés. Il s'agit d'un jeu de données en anglais donc. Plus précisément, chaque article est précédé d'une phrase d'introduction (appelée résumé) rédigée par un professionnel, généralement l'auteur de l'article. Le résumé porte la classe HTML "storybody introduction" et peut être facilement identifié et extrait du corps du texte principal. Ce *dataset* a été introduit dans l'article [70] et tous les détails sur sa formation et ses spécificités peuvent y être trouvés. En ce qui nous concerne, nous avons utilisé la version retrouvable sur *hugging face* au

²<https://huggingface.co/datasets/>

³https://huggingface.co/datasets/ccdv/cnn_dailymail

checkpoint éponyme **xsum** ⁴. Ce qui veut dire qu'il peut être chargé avec *python* grâce aux lignes de code :

```
1 from datasets import load_dataset
2
3 datasetXSUM = load_dataset("xsum")
```

Dataset OrangeSum

Ces données sont introduites à travers l'article [53] et ont été constituées en récupérant les contenus du site Web **Orange Actu**⁵. Les pages scrappées couvrent presque une décennie, de février 2011 à septembre 2020. Elles appartiennent à cinq catégories principales : France, monde, politique, automobile, et société . La catégorie société est elle-même divisée en 8 sous-catégories : santé, environnement, people, culture, médias, high-tech, insolite, et divers. Ce jeu de données est considéré comme l'équivalent en langue française du jeu de données *XSum* en ce qui concerne le degré d'abstraction des résumés. Pour ce mémoire, nous avons utilisé les données *OrangeSum* tel que retrouvées sur le point d'entrée **GEM/OrangeSum** ⁶. Il peut donc être chargé identiquement que les 2 jeux de données précédents, *XSum* et *CNN/DailyMail*, en utilisant le nom de checkpoint *GEM/OrangeSum*.

Dataset for-ULPGL-Dissertation

Le jeu des données que nous avons dénommé **for-ULPGL-Dissertation** a été constitué par nous, à partir du sous-ensemble dénommé *abstract* du *dataset OrangeSum* mentionné dans la sous section précédente. Les données d'*OrangeSum-Abstract* peuvent être chargées avec *python* en utilisant :

```
1 from datasets import load_dataset
2
3 orange_datas = load_dataset("GEM/OrangeSum", "abstract")
```

⁴<https://huggingface.co/datasets/xsum>

⁵<https://actu.orange.fr/>

⁶<https://huggingface.co/datasets/GEM/OrangeSum>

A ces données, nous avons ajouté les quelques résumés jusque là générés grâce à notre système (dénommé **Mon Résumeur**). Plus exactement, la version actuelle (version 0.0.1 du 15/10/2022) de ce *dataset* est constituée de 24401 données en provenance d'*OrangeSum-Abstract* et 450 données en provenance de **Mon Résumeur**. Soit environs 2% du total en provenance de **Mon Résumeur**.

Nous ferons constamment évoluer ce *dataset*, avec l'augmentation des résumés générés par notre système, de manière à ce qu'il soit majoritairement constitué des données jugées *bonnes* par les utilisateurs de notre système.

Nous avons *uploadé* le *dataset* ainsi constitué dans notre compte *hugging face*. Il y est publiquement accessible et on peut y voir tous les détails supplémentaires ⁷. Comme il est au point d'entrée **krm/for-ULPGL-Dissertation**, on peut le charger avec *Python* grâce aux bouts de code :

```
1 from datasets import load_dataset
2
3 my_dataset = load_dataset("krm/for-ULPGL-Dissertation")
```

III.5.2 Modèles utilisés pour la synthèse abstractive

Modèle BART optimisé

Le modèle *BART* [10] est celui que nous avons choisi comme modèle principal pour la synthèse abstractive dans notre système. Son objectif de pré-entraînement est assez général pour former un excellent modèle de langue comme nous l'avons décrit au chapitre précédent (voir le point II.5.3). Vu que notre système est destiné à fonctionner en production et non pas uniquement dans le monde de la recherche, il nous a été important de choisir une déclinaison de ce modèle adaptée à nos fins. C'est pour cela que nous avons choisi, pour notre système, d'utiliser la déclinaison accessible au *checkpoint* **sshleifer/distilbart-cnn-12-6** finetuné pour la synthèse des textes. Ce *transformer* peut être chargé très simplement en utilisant le *pipeline* traditionnel disponibilisé sur *Hugging Face* ⁸ par son auteur mais ce n'est pas très optimal de s'y prendre comme cela. C'est la raison pour laquelle,

⁷<https://huggingface.co/datasets/krm/for-ULPGL-Dissertation>

⁸<https://huggingface.co/sshleifer/distilbart-cnn-12-6>

nous avons implémenté un *pipeline* (comme on peut le voir sur la figure II.5) nous permettant d'utiliser ce modèle de façon optimale, sans augmenter la complexité de calcul associée aux têtes d'attention de tous les modèles du type *transformer* [7]. Les justifications des choix d'implémentation sont données au dernier point de ce chapitre.

Il faut noter que ce modèle est prévu exclusivement pour la langue anglaise.

Modèle BARThez

Le modèle *BARThez* [53] est un modèle pré-entraîné sur du texte en français. Pour la synthèse abstractive, nous avons utilisé sa déclinaison finetunée sur le dataset *OrangeSum* et disponible au *checkpoint* **moussaKam/barthez-orangesum-abstract**. Néanmoins, on ne peut pas se contenter de ce modèle pour la synthèse en langue française car, étant donné le *dataset* sur lequel il a été entraîné, la qualité de ses résumés n'est pas très satisfaisante quand on veut l'utiliser pour la production des résumés génériques sur des documents du type narratif et littéraire comme on le voudrait pour un système à usage général. En plus l'une des ses limitations est qu'il a une grande tendance à attribuer les propos des textes du résumé aux personnalités publiques actuelles quand bien même elles n'apparaissent pas dans le texte source (biais introduit par son *dataset*). Les objectifs de pré-entraînement sont similaires à ceux du modèle *BART*, donc il s'agit bien d'un modèle du type *BART*.

Modèle mBARThez

Le modèle *mBARThez* est un modèle introduit dans l'article [53] (le même que celui de *BARThez*) et basé sur **mBART**. Il s'agit d'un modèle pré-entraîné sur plusieurs langues et entraîné juste à prédire les mots masqués dans un texte. Nous l'avons finetuné pour pouvoir l'utiliser pour la synthèse automatique et espérer corriger avec le temps les faiblesses du modèle *BARThez*. Pour le *finetuning*, nous sommes parti du modèle *mBARThez* retronvable au *checkpoint* **moussaKam/mbarthez** ⁹.

Après entraînement de *mBARThez*, nous avons produit un modèle pouvant être utilisé pour la synthèse (nous l'avons dénommé **BARTkrame-abstract**).

⁹<https://huggingface.co/moussaKam/mbarthez>

Modèle BARTkrame-abstract

Il s'agit d'un modèle obtenu par *finetuning* de *mBART* sur notre *dataset* dénommé *for-ULPGL-Dissertation*. Cela nous a permis d'obtenir le *checkpoint* **krm/BARTkrame-abstract** dont tous les détails sont accessibles sur la plateforme *Hugging Face* où nous l'avons *uploadé* ¹⁰.

En terme d'architecture, il s'agit d'un *transformer* du type *BART*, de même architecture que *mBART*[52] car c'est ce qui a été utilisé comme modèle de base pour produire *mBART* que nous avons finetuné enfin pour obtenir *BARTkrame-abstract*.

C'est-à-dire qu'il s'agit d'un *transformer* d'architecture exactement similaire à celle proposée à l'initial par *Vaswani et al.* [7] constitué de 12 couches dans l'encodeur et 12 dans le décodeur avec une dimension de 1024 *tokens* pour le vecteur caché, sur 16 têtes d'attention.

Modèle Pegasus

Il s'agit de l'un des modèles jugés les plus performants pour les tâches de synthèse abstractive [44]. En effet, son objectif de pré-entraînement est très proche d'une tâche de synthèse. Néanmoins, nous avons fini par le supprimer des modèles que nous rendons accessibles à travers notre *API* à cause de sa lenteur et de sa taille. Cela est dû au fait que, du point de vu qualitatif, l'abstraction est parfois poussée trop loin de telle sorte que ce qui est dans la synthèse ne correspond pas vraiment à l'idée véhiculée par le texte source, le rapprochant beaucoup plus des systèmes de génération automatique des textes. Pour tirer ces conclusions, nous nous sommes servi du *checkpoint* dénommé **google/pegasus-xsum** ¹¹.

III.5.3 Tests

Nous avons mené plusieurs tests, à la fois sur les modèles abstraits et sur les algorithmes de synthèse extractive de notre système. Les tests avaient pour objectif de démontrer le bien-fondé des techniques que nous avons choisi pour la synthèse automatique.

¹⁰<https://huggingface.co/krm/BARTkrame-abstract>

¹¹<https://huggingface.co/google/pegasus-xsum>

Nous avons utilisé comme métriques d'évaluation les mesures *ROUGE-1*, *ROUGE-2*, *ROUGE-L* et une variante de cette dernière dénommée *ROUGE-LSum* car il s'agit de loin des mesures les plus utilisées en synthèse automatique [2].

Tests sur les algorithmes de synthèse extractive

Ce test a deux principaux objectifs :

- choisir les poids à affecter aux divers algorithmes qui forment le système *merging* (II.6).
- établir une mesure des performances de la partie qui fonctionne par combinaison des divers algorithmes de synthèse extractive (voir la figure II.6 qui représente le système en question que nous avons nommé *merging*) par rapport au module python dédié à la synthèse extractive, module dénommé *gensim* (choisi principalement pour sa rapidité);

Le test a été effectué en se servant de 2 *datasets* bien connus en synthèse automatique, à savoir *CCN / DailyMail* et *XSum*. Ce test a été mené en anglais pour ne pas avoir à traduire les *datasets* et ainsi avoir des résultats les plus authentiques possible. Pour chacun de ces *datasets*, nous avons réalisé nos expérimentations sur les données de test. Nous avons choisi ces deux *datasets* car ils sont en quelques sortes complémentaires. En effet, *CCN / DailyMail* est essentiellement extractif tandis que *XSum* est beaucoup plus abstraktif [53].

Tests sur BART et BART optimisé

Ce test avait pour objectif d'évaluer les performances du modèle *BART* quand on l'utilise à travers le pipeline implémenté par nous selon la figure II.5 par rapport à une utilisation classique (selon le pipeline classique [45]). Nous avons pour cela utilisé les *datasets* *CCN / DailyMail* et *XSum* sur le *checkpoint* dénommé *sshleifer/distilbart-cnn-12-6* que nous avons utilisé comme modèle *BART* dans l'ensemble du système.

Précisions sur l'entraînement et les tests de BARTKrame-abstract

La version actuelle de ce modèle (le 16/10/2022, version 0.0.1) a été obtenue après entraînement de *mBART* avec 5000 **données d'entraînement** et 100 **données de validation** provenant du *dataset for-ULPGL-Dissertation* (le nombre de données a été réduit suite aux limitations en terme de ressources matérielles dont nous disposions). Nous avons utilisé, pour cet entraînement, un environnement sur *google colab*¹² équipé d'un *GPU* et à présent l'entraînement a été fait sur 12 époques et a duré 8 heures. Les hyperparamètres utilisés peuvent être ainsi présentés :

- *Learning rate initial* : $5.6 * 10^{-6}$;
- Taille des batchs (*batch size*) : 4;
- Optimiseur : *Adam* (avec $\beta = 0.999$ et $\epsilon = 1 * 10^{-8}$)
- Le *Learning rate* était à décroissance linéaire;
- Le grain de sélection aléatoire des données était de 42;
- Métrique : *ROUGE*.

Pour plus d'informations sur notre modèle et le processus d'entraînement, se rapporter à son dépôt sur *Hugging Face*¹³.

Le test réalisés sur ce modèle ont été faits sur le dataset *for-ULPGL-Dissertation* car notre objectif est d'obtenir un modèle principalement en français. Une partie des résultats de test a été obtenue immédiatement après entraînement (c'est-à-dire sur les données de validation) et nous avons réalisé après cela un autre test sur les données de test de notre *dataset*.

III.5.4 Résultats des tests

Pour ces résultats, il faut noter que plus une mesure est supérieure, plus le modèle a de chance de donner des résultats de bonne qualité. Ce n'est pas toujours le cas mais la

¹²<https://colab.research.google.com/>

¹³<https://huggingface.co/krm/BARTkrame-abstract>

mesure *ROUGE* ici utilisée est un excellent indicateur des performances d'un système de résumé automatique (voir la section [III.3](#)).

Algorithmes de synthèse extractive

1°) Recherche des poids des modèles constituant *merging*

Pour cet essai, il nous a d'abord fallu évaluer la confiance que nous pouvons accorder à chaque modèle constituant *merging*. Ce que nous appelons *merging* est formé, comme on peut le voir sur la figure [II.6](#) par les algorithmes de synthèse extractive suivants : *TF-IDF*, *LSA*, *Heuristiques*, *TextRank*, *LexRank* et *Luhn*. Pour la recherche des poids, nous avons considéré uniquement les résultats des tests sur le *dataset CNN/DailyMail* car il est plus adapté à la synthèse extractive. Les résultats ont été les suivants :

Table III.1: Résultats ROUGE des algorithmes de *merging* sur *CNN/DailyMail*

Algorithmes	R-1	R-2	R-L	R-LSum
Heuristique	24.72	7.95	16.44	19.64
LexRank	22.88	5.88	15.02	18
TextRank	19.91	5.06	13.52	16.13
LSA	20.43	3.68	12.84	15.70
Luhn	19.66	4.79	12.66	15.39
Tf-Idf	10.98	1.99	7.77	8.95

Pour la recherche des poids, nous avons appliqué une méthode heuristique simple. En vertu des résultats que nous avons dans le tableau [III.1](#), nous avons accordé les scores 1, 2, 3, 4, 5 et 6 respectivement aux algorithmes *Tf-Idf*, *Luhn*, *LSA*, *TextRank*, *LexRank* et *Heuristique* comme ils se suivent dans le tableau. Ensuite, nous avons normalisé ces poids en appliquant une fonction *softmax* à cet ensemble de rangs. Donc, en appliquant $e^i / (\sum_{n=1}^6 e^n)$ pour tout $i \in \{1, 2, 3, 4, 5, 6\}$ nous avons trouvé les poids suivants :

Table III.2: Poids des algorithmes constituant *merging*

Heuristique	LexRank	TextRank	LSA	Luhn	Tf-Idf
0.633	0.233	0.0857	0.0315	0.0116	0.0042

2°) Comparaison quantitative de Merging avec Gensim

L'évaluation de ces deux modèles sur *CNN/DailyMail*, en se servant des poids listés dans le tableau III.2, a donné les résultats suivants :

Table III.3: Comparaison des algorithmes Gensim et *merging* sur *CNN/DailyMail*

Algorithmes	R-1	R-2	R-L	R-LSum
Gensim	31.74	12.28	20.69	27.70
Merging	23.19	6.70	15.05	18.17

L'évaluation de ces deux modèles sur *XSum*, en se servant des poids listés dans le tableau III.2, a quant à elle donné les résultats suivants :

Table III.4: Comparaison des algorithmes Gensim et *merging* sur *XSum*

Algorithmes	R-1	R-2	R-L	R-LSum
Gensim	15.10	2.50	10.13	11.29
Merging	14.90	1.88	10.09	10.12

Résultats de BART en usage normal puis en usage optimisé

Les tests sur le jeu des données *CNN/DailyMail* ont donné les résultats suivants :

Table III.5: Comparaison de BART avec BART optimisé sur *CNN/DailyMail*

Algorithmes	R-1	R-2	R-L	R-LSum
BART optim.	43.72	20.22	30.06	36.62
BART normal	36.66	14.31	25.88	31.038

Et les tests sur le jeu des données *XSum* ont donné :

Table III.6: Comparaison de BART avec BART optimisé sur *XSum*

Algorithmes	R-1	R-2	R-L	R-LSum
BART optim.	20.57	3.50	13.42	13.41
BART normal	17.67	2.66	12.08	12.091

Résultats des tests de BARTkrame-abstract

Les tests sur le jeu des données *for-ULPGL-Dissertation* ont donné les résultats suivants:

Table III.7: Résultats de test de *BARTkrame-abstract* sur *for-ULPGL-Dissertation*

Données	R-1	R-2	R-L	R-LSum
Validation	27.03	13.34	23.92	24.19
Test	24.08	7.28	19.30	19.29

Il faudra noter que le dataset utilisé pour entraîner ce modèle, ainsi que le modèle lui-même, seront entrain d'être améliorés. Pour avoir les résultats actualisés au-delà de la date de soumission de ce document, il faut se rapporter à leurs liens (dépôts HuggingFace sus-mentionnés pour *BARTkrame-abstract* et pour *for-ULPGL-Dissertation*).

III.5.5 Interprétation des résultats

Système de synthèse extractive

Comme on peut le voir dans les tableaux [III.3](#) et [III.4](#), les performances du module python *gensim* demeurent supérieures à celles de *merging*. Cela est un fait, du point de vu quantitatif. Néanmoins, comme pourra en juger le lecteur au vu des exemples mis en annexe, cela n'est pas complètement vrai du point de vu qualitatif.

En effet, le modèle *merging* tel que nous l'avons implémenté gère mieux le ratio entré par l'utilisateur et est moins sensible aux problèmes introduits par les sauts de ligne arbitraires dans le texte. Également, *merging* repère très bien les éléments saillants du

texte (voir l'annexe .1) .

Il faut finalement constater que les mesures sur *XSum* sont toujours inférieures à celles faites sur *CNN/DailyMail*. Cela est très normal et justifié par le fait que le *dataset CNN/DailyMail* est essentiellement du type extractif alors que *XSum* est du type *abstraktif*.

BART et BART optimisé

Comme on peut le voir au travers des tableaux III.5 et III.6, les résultats obtenus avec le modèle *BART* en usage optimal sont largement supérieurs à ceux obtenus en usage classique (normale). Le test sur *CNN/DailyMail* donne des résultats sans équivoque sur la supériorité de *BART optimisé* par rapport au *BART classique*. Néanmoins, les résultats sur *XSum* sont proches (comme on le voit dans le tableau III.6) et cela est bien évident car, comme *XSum* est formé des résumés à tendance plus abstraite et que le modèle *BART* est la base de ces deux implémentations (classique et optimisée), aucune ne devait dépasser un certain seuil en ce qui concerne la ressemblance à *XSum*. En plus, la mesure *ROUGE* est incapable de repérer les similarités sémantiques quand des tournures différentes sont utilisées. Néanmoins, l'utilisation de la mesure *ROUGE* sur un *dataset* à tendance abstractive est un bon indicateur de la fidélité du modèle au texte d'origine quand il s'agit d'un modèle abstraktif.

Commentaires sur les résultats de BARTkrame-abstract

Les résultats obtenus sur *BARTkrame-abstract* ne devaient pas différer assez sur les données de validation par rapport aux données de test car les données proviennent d'un même *dataset* et, comme le *dataset* a été constitué par un choix aléatoire des données à affecter aux différentes catégories (*train*, *test* et *validation*), les caractéristiques statistiques des données de validation sont approximativement les mêmes que celles des données de test. Cela devait suffire pour assurer que les résultats obtenus sur les données de test et ceux obtenus sur les données de validation soient assez proches.

La signification des résultats du tableau III.7 est simplement que le modèle est encore très sensible à des petites variations. Donc il n'est pas encore stable et l'entraînement doit être poursuivi. Cela est évidemment clair quand on regarde les résultats qualitatifs (les

synthèses) obtenus en utilisant ce modèle dans notre système. On peut voir en effet que le modèle parvient à capturer les points importants mais ne parvient pas encore à repérer clairement où s'arrêter, voir l'annexe [.1](#).

III.6 Justification des choix d'implémentation

Voici donc quelques justifications très sommaires des choix d'implémentation que nous avons faits. Les points saillants sont principalement ici présentés :

- L'un des modules utilisés pour la synthèse extractive est *gensim*. Nous l'avons principalement choisi comme algorithme de base de la synthèse extractive suite à sa rapidité. Il implémente néanmoins l'algorithme *Tf-Idf* avec quelques optimisations [\[71\]](#).
- Pour le module mélangeant un grand nombre d'algorithmes utilisés pour la synthèse extractive, nous avons opté pour cette approche car nous sommes partis de l'idée selon laquelle, du point de vue qualitatif, une synthèse qui serait générée en prenant en compte divers aspects serait plus susceptible d'être riche que les autres. Bien que les résultats quantitatifs (mesure *ROUGE*) n'ont pas été très satisfaisants, nous estimons avoir eu raison d'implémenter ces algorithmes au vu de la valeur ajoutée en terme qualitatif comme on peut en juger grâce aux synthèses mises à l'annexe [.1](#).
- Pour la synthèse abstractive, le fait de tourner notre choix vers les modèles du type *BART* a été inspiré par la nécessité de réaliser des synthèses abstraites mais pas trop pour ne pas s'éloigner du contenu du texte source. Cela nous a amené à exclure *PEGASUS* bien que ce soit l'un des modèles les mieux cotés en synthèse abstractive. En fait, *PEGASUS* s'éloigne souvent des vraies idées véhiculées par le texte source. En plus, étant donné que le modèle *BART* est moins pesant et plus manipulable que le modèle *T5*, il a été l'unique à considérer (comme *PEGASUS* et *T5* étaient *ipso facto* exclus).
- Comme point de repère avec les modèles complètement conçus en français, nous avons utilisé *BARThez* car c'est la référence française en ce qui concerne les *transformers*

de type encodeur-décodeur.

- Pour l'entraînement de notre modèle, nous sommes parti de *mBART* qui est à la base un modèle *mBART* dont le pré-entraînement a été poursuivi en insérant une grande masse des données en français [53]. Comme nous devons réaliser le *finetuning* sur du texte en français pour obtenir un modèle complètement en français, nous estimons que c'est l'approche la plus appropriée et devant nous permettre de partir d'un modèle de langue déjà très bon en français. Ce dernier fait a pour avantage de nous permettre de diminuer les données d'entraînement.
- Pour l'implémentation de l'API, nous avons choisi d'utiliser *Flask* étant donné qu'il donne une grande liberté aux développeurs. Il n'a pas une structure préconçue et imposante.
- L'utilisation de *ReactJS* pour la partie client a été motivée par sa popularité, mais également nous nous sentons plus à l'aise avec la syntaxe et l'architecture de ce *framework*.
- Concernant le fait de réaliser une API parfaitement distinguée, l'objectif est de permettre l'implémentation, par divers développeurs, de toute forme de systèmes pouvant en utiliser les services, participant par la même occasion à l'amélioration de notre modèle de synthèse. C'est d'ailleurs la raison pour laquelle nous avons également donné aux utilisateurs la possibilité d'évaluer les résumés qu'ils obtiennent. Cela constituera en fait un écosystème d'annotation des résumés par des humains.

III.7 Conclusion partielle

Dans ce chapitre, qui est le dernier de notre travail, nous venons de finaliser la conception de notre système. Cela nous a permis de montrer que le système a une architecture *3-tiers* classique, avec une petite base des données qui nous permet également de récolter les données de synthèse pouvant par la suite s'utiliser pour améliorer les modèles de *machine learning* dédiés à la synthèse automatique des documents.

Nous y avons également présenté les diverses métriques couramment utilisées pour l'évaluation des systèmes de synthèse automatique, tout en utilisant le paquet *ROUGE* qui est la métrique la plus utilisée pour cette fin.

Ce chapitre nous a non seulement permis de décrire complètement l'implémentation de notre système mais également, il nous a permis de montrer que le modèle *BART* tel que nous l'avons utilisé pour la synthèse abstractive est plus performant que si on l'utilisait dans une pipeline classique. Nous avons également montré que les résultats du module *gensim* pour la synthèse automatique sont meilleurs que ceux de l'algorithme *merging*, mais que ce dernier est qualitativement très satisfaisant et plus robuste aux variations mineures dans les textes. Nous avons également montré que le modèle *BARTkrame-abstract*, qui a été obtenu après *finetuning* de *mBARThez* (un modèle basé sur *mBART*) sera entrain d'être amélioré au fil de l'utilisation du système, étant donné que tout l'ensemble a été conçu en tenant compte de cet aspect.

Conclusion générale

Ce travail est un rapport accompagnant le système que nous avons mis au point en guise de mémoire. Le système en question a pour rôle de générer automatiquement les résumés des documents qui lui sont fournis.

Au début de ce travail, nous avons fait quelques hypothèses. Dans la première hypothèse, nous affirmons qu'un traitement purement linguistique ne suffirait pas pour pouvoir produire des résumés performants en utilisant un système informatique. A l'issu de nos recherches, et des expérimentations, nous avons pu vérifier cette hypothèse car, nous avons démontré que le traitement purement linguistique du langage naturel est très gourmand en ressources et très fastidieux. En plus, le langage naturel n'est pas aussi précis que peut l'être un langage formel quelconque.

A la seconde hypothèse, nous avons avancé qu'on ne peut se passer de l'intelligence artificielle si on veut obtenir des systèmes de résumé de performance presque humaine. Cela se vérifie à travers la littérature que nous avons présentée, ainsi que la complexité inhérente au traitement du langage naturel tel que nous l'avons montré dans ce travail. L'usage de l'intelligence artificielle permet de réaliser des tâches à grande complexité à un coût faible par rapport à une approche algorithmique détaillée.

La troisième et dernière hypothèse que nous avons émise était qu'une architecture du type *transformer*, à laquelle on joint des traitements linguistiques sommaires comme la segmentation du texte par exemple, permettrait d'obtenir des bons systèmes de résumé automatique. Cette hypothèse doit être précisée au vu des études que nous avons menées à travers ce travail. En effet, les recherches que nous avons menées nous ont permis de nous rendre compte que, pour des systèmes destinés à un usage en production, il est mieux de se limiter aux algorithmes classiques pour la synthèse extractive au lieu d'utiliser des systèmes d'intelligence artificielle très lourds. Cela permet de gagner en performance et en expérience utilisateur. Néanmoins, grâce aux modèles de *deep learning* basés sur les mécanismes d'attention (les *transformers* en l'occurrence), on peut se passer de la nécessité d'implémenter explicitement les règles linguistiques car le pré-entraînement

des *transformers* capture déjà les règles de la langue considérée. Ainsi, les règles linguistiques à implémenter peuvent être réduites au strict minimum. C'est pour cela que, bien que les modèles soient assez lourds en général, ils sont incontournables si on veut obtenir des résumés de qualité presque humaine (abstractifs), et corriger les erreurs de coupure de sens inhérentes à la synthèse par sélection brute des phrases saillantes (extraction). Ainsi donc, nous avons montré qu'on ne peut se passer des modèles d'intelligence artificielle si on veut obtenir des systèmes de résumé automatique performants, surtout que tous les aspects du langage naturel ne peuvent pas être formalisés.

L'objectif de ce travail était, comme énoncé dans l'introduction générale, de concevoir et réaliser un système, en l'occurrence une application web, devant permettre la génération automatique des résumés des textes. Pour atteindre ledit objectif, il nous a fallu réaliser 6 objectifs spécifiques.

Le premier objectif, concernant l'évaluation des failles et limites des techniques de synthèse existantes a été mené à terme. Il a débouché sur le constat selon lequel, les systèmes de résumé extractif par usage d'algorithmes classiques (sans traitement intelligent) sont couramment utilisés à cause de la facilité d'implémentation mais sont sujets aux erreurs de coupure de sens, étant donné que les phrases saillantes sont sélectionnées en les enlevant de leur contexte initial. Ce problème peut être réduit en essayant d'exclure au maximum les éléments hors contexte. Malheureusement, cette dernière manipulation a ses limites car elle ne peut pas supprimer complètement les coupures de sens inhérents au processus même de synthèse extractif. C'est pourquoi, les modèles abstractifs se présentent comme modèle de choix pour résoudre ce problème. Nous avons également montré que, comme le module *gensim* de python a moins de chance de capturer un large éventail des points saillants dans les textes, l'utilisation d'un algorithme qui combinerait plusieurs approches de synthèse extractive a plus de chances de capturer une multitude d'aspects. C'est pourquoi nous avons implémenté un algorithme de synthèse extractive fonctionnant par combinaison de divers algorithmes qui ont chacun sa spécificité. Cela a contribué à une nette amélioration qualitative sur les résultats, ce qui fait partie du second objectif de ce travail qui consistait à corriger le cas échéant, les failles observées.

Et, s'agissant justement de ce second objectif, qui consiste à apporter les améliorations

nécessaires, nous devons préciser que, pour la synthèse abstractive, que nous avons réalisé grâce aux modèles du type *transformer*, nous avons implémenté un pipeline qui permet de réaliser la synthèse des très longs documents (allant au-delà de 100 pages) alors qu’au préalable, les *transformers* utilisés se limitent à environ 1 page.

Le troisième objectif consistant à établir une architecture logique fonctionnelle et optimale pour obtenir des synthèses de qualité a été atteint, d’abord à travers l’optimisation que nous avons faite durant l’usage du modèle *BART* pour la synthèse abstractive (et cela contribue également au second objectif du travail), puis à travers l’architecture globale que nous avons utilisé pour notre système. Il s’agit en fait d’une architecture *3-tiers* classique mais qui est intrinsèquement évolutive étant donné la séparation nette faite entre *API* et *client web*, auquel on joint un stockage continu des résumés obtenus.

On conclut, de ce qui vient d’être mentionné, que le quatrième objectif, consistant à élaborer une *API* devant servir de moteur de résumé a également été atteint, ainsi que le très important cinquième objectif consistant à mettre au point une base des données devant nous permettre de constituer à la longue un *dataset* riche, susceptible d’être utilisé pour élaborer des modèles de résumé de plus en plus performants.

Le sixième et dernier objectif de ce travail, consistant à mettre au point une interface web devant consommer les services de l’*API* a également été atteint au vu du système que nous avons réalisé.

Les 6 objectifs spécifiques ayant été atteints, nous estimons avoir mis au point un écosystème complet de synthèse automatique des documents.

Nous devons néanmoins préciser que ce système est mieux adapté à la synthèse générique des documents de type narratif, argumentatif et littéraire en général (romans, nouvelles, discours, articles de presse, documents à faible densité mathématique et figures,...). Pour la synthèse extractive, nous avons utilisé deux approches comme nous l’avons déjà mentionné (le module *gensim* de python et *merging* que nous avons mis au point en combinant plusieurs méthodes de synthèse extractive).

Pour la synthèse abstractive, celle qui se rapproche le plus de la synthèse humaine, nous avons utilisé des modèles du type *BART*, pour éviter de pousser trop loin l’abstraction comme le ferait *PEGASUS*, et en même temps éviter d’utiliser un modèle inutilement

trop général pour notre contexte, comme c'est le cas du modèle *T5*.

Nous avons également mis en place un *dataset* de résumé automatique que nous avons nommé *for-ULPGL-Dissertation* qui accueillera au fil du temps les données de synthèse recueillies à travers la base des données de notre système (que nous avons nommé *Mon Résumeur*) et deviendra de plus en plus riche et performant. Ce *dataset*, dans sa version actuelle (pas encore très riche) a été utilisé pour mettre au point le modèle *BATkrame-abstract* basé sur le modèle *mBARThez*, lui-même basé sur *mBART* que nous continuerons également à améliorer en même temps que le *dataset*. Finalement, nous avons utilisé le paquet *ROUGE* comme métrique d'évaluation de notre système.

Ce domaine est très intéressant et surtout très vaste, c'est pourquoi l'ouverture de ce sujet est très large. Ainsi, aux futurs étudiants, qui comptent améliorer ce travail ou tout simplement œuvrer dans le même domaine, nous pouvons suggérer :

- d'implémenter, selon leurs besoins, d'autres clients web pouvant consommer les services de cette *API* mise au point;
- d'étudier minutieusement les limites de la métrique *ROUGE* ainsi que d'autres métriques d'évaluation des systèmes génératifs pour mettre au point, le cas échéant, des nouvelles métriques, aussi moins gourmandes en ressources que *ROUGE* mais plus performantes en matière d'évaluation de qualité;
- d'étendre notre système de résumé automatique au *résumé guidé* et multi-documents;
- d'étendre le processus au résumé des vidéos ou même d'audios tout simplement;
- d'étudier la possibilité de mise au point des systèmes devant permettre la synthèse des documents à forte teneur mathématique;
- d'étudier la possibilité de réaliser un résumeur universel, c'est-à-dire qui serait adapté à la majorité de types de documents connus (documents spécialisés, littéraires,...);
- d'optimiser le temps de calcul des modèles de synthèse automatique;
- etc

Bibliographie

- [1] D. Sarkar. *Text Analytics with Python: A Practitioner's Guide to Natural Language Processing*. Apress, 2019.
- [2] Maali Mnasri. *Résumé automatique multi-document dynamique*. Theses, Université Paris-Saclay, September 2018.
- [3] SINAT Blog. Présentation des réseaux de neurones artificiels. <https://rb.gy/ddborn>, 06 2021.
- [4] *Introduction to Deep Learning*. Massachussets Institute of Technology, 2020.
- [5] Patrick Hairy. Les réseaux de neurones récurrents pour les séries temporelles. <https://rb.gy/ndwuks>, 09 2021.
- [6] A. Géron. *Deep Learning avec Keras et TensorFlow: Mise en oeuvre et cas concrets*. Dunod, 2020.
- [7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [8] Lilian Weng. The transformer family. <https://rb.gy/i00on2>, 04 2020.
- [9] H. P. Luhn. The automatic creation of literature abstracts. *IBM Journal of Research and Development*, 2(2):159–165, 1958.
- [10] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019.
- [11] S. Lamsiyah, S. Ouatik El Alaoui, and B Espinasse. Résumé automatique guidé de textes: Etat de l’art et perspectives (guided summarization : State-of-the-art and

- perspectives). In *Actes de la Conférence TALN. Volume 2 - Démonstrations, articles des Rencontres Jeunes Chercheurs, ateliers DeFT*, pages 55–72, Rennes, France, 5 2018. ATALA.
- [12] Juan-Manuel Torres-Moreno. *Automatic text summarization*. John Wiley & Sons, 2014.
- [13] D. Adams. *Combining State-of-the-art Models for Multi-document Summarization Using Maximal Marginal Relevance*. University of Lethbridge, 2021.
- [14] Min-Yen Kan, Kathleen R. McKeown, and Judith L. Klavans. Applying natural language generation to indicative summarization. *CoRR*, cs.CL/0107019, 2001.
- [15] Stergos D. Afantenos, Vangelis Karkaletsis, and Panagiotis Stamatopoulos. Summarization from medical documents: A survey. *CoRR*, abs/cs/0504061, 2005.
- [16] O. Carton and D. Perrin. *Langages formels : calculabilité et complexité : cours et exercices corrigés*. Vuibert, 2014.
- [17] D. Jurafsky and J.H. Martin. *Speech and Language Processing*. Always learning. Pearson, 2014.
- [18] M. Hagiwara. *Real-World Natural Language Processing: Practical Applications with Deep Learning*. Manning, 2021.
- [19] S. Srivastava. *Natural Language Processing with Python: Hands-On Labs to Apply Deep Learning Architectures to NLP Applications*. Independently Published, 2021.
- [20] Akshay Kulkarni and Adarsha Shivananda. *Natural language processing recipes*. Springer, 2019.
- [21] Mohamed Hedi Maaloul. *Approche hybride pour le résumé automatique de textes. Application à la langue arabe*. Theses, Université de Provence - Aix-Marseille I, December 2012. Date Soutenance 18-12-2012.
- [22] Youcef Djeriri. *Les Réseaux de Neurones Artificiels*. 09 2017.

- [23] Thushan Ganegedara. *Natural Language Processing with TensorFlow: Teach language to machines using Python's deep learning library*. Packt Publishing Ltd, 2018.
- [24] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- [25] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.
- [26] Ministère de l'éducation. Expression écrite : Le résumé de texte en classe de terminale-généralités. <https://fasoeducation.bf/>, 10 2022.
- [27] Chin-Yew Lin. Training a selection function for extraction. In *CIKM '99*, 1999.
- [28] Inderjeet Mani. *Automatic summarization*, volume 3. John Benjamins Publishing, 2001.
- [29] Ani Nenkova and Rebecca Passonneau. Evaluating content selection in summarization: The pyramid method. In *HLT-NAACL 2004 - Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, Proceedings of the Main Conference*, HLT-NAACL 2004 - Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics, Proceedings of the Main Conference, pages 145–152. Association for Computational Linguistics (ACL), 2004.
- [30] Marie-Francine Moens. Automatic indexing and abstracting of document texts. *Computational Linguistics*, 27(1):149–149, 2001.
- [31] Barry Schiffman, Ani Nenkova, and Kathleen McKeown. Experiments in multidocument summarization. In *Proceedings of the Second International Conference on Human Language Technology Research*, HLT '02, page 52–58, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc.

- [32] Jan Žižka, František Dařena, and Arnošt Svoboda. *Text mining with machine learning: principles and techniques*. CRC Press, 2019.
- [33] Michael W Berry and Jacob Kogan. *Text mining: applications and theory*. John Wiley & Sons, 2010.
- [34] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.
- [35] Catherine Benincasa, Adena Calden, Emily Hanlon, Matthew Kindzerske, Kody Law, Eddery Lam, John Rhoades, Ishani Roy, Michael Satz, Eric Valentine, et al. Page rank algorithm. *Department of Mathematics and Statics, University of Massachusetts, Amherst, Research*, 2006.
- [36] Nitin Indurkha and Fred J Damerau. *Handbook of natural language processing*. Chapman and Hall/CRC, 2010.
- [37] Rada Mihalcea and Paul Tarau. Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, pages 404–411, 2004.
- [38] Günes Erkan and Dragomir R Radev. Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal of artificial intelligence research*, 22:457–479, 2004.
- [39] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
- [40] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.
- [41] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

- [42] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146, 2017.
- [43] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- [44] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In *International Conference on Machine Learning*, pages 11328–11339. PMLR, 2020.
- [45] Lewis Tunstall, Leandro von Werra, and Thomas Wolf. *Natural language processing with transformers*. " O'Reilly Media, Inc.", 2022.
- [46] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.
- [47] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.
- [48] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.
- [49] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

- [50] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- [51] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.
- [52] Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. Multilingual denoising pre-training for neural machine translation. *Transactions of the Association for Computational Linguistics*, 8:726–742, 2020.
- [53] Moussa Kamal Eddine, Antoine J-P Tixier, and Michalis Vazirgiannis. Barthez: a skilled pretrained french sequence-to-sequence model. *arXiv preprint arXiv:2010.12321*, 2020.
- [54] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67, 2020.
- [55] Linting Xue, Noah Constant, Adam Roberts, Mihir Kale, Rami Al-Rfou, Aditya Siddhant, Aditya Barua, and Colin Raffel. mt5: A massively multilingual pre-trained text-to-text transformer. *arXiv preprint arXiv:2010.11934*, 2020.
- [56] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.
- [57] Boudjemaa Boudaa. Cours de génie logiciel (avec exercices résolus). Université Ibn-Khaldoun-Tiaret, Algérie, 02 2020.
- [58] Philippe Rigaux. *Cours de bases de données..* CNAM/Médias, 2001.

- [59] GJ Rath, A Resnick, and Terry R Savage. The formation of abstracts by the selection of sentences. part i. sentence selection by men and machines. *American Documentation*, 12(2):139–141, 1961.
- [60] Karen Sparck Jones and Julia R Galliers. Evaluating natural language processing systems: An analysis and review. 1995.
- [61] Mani Maybury. *Advances in automatic text summarization*. MIT press, 1999.
- [62] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.
- [63] Julia Bell Hirschberg, Kathleen McKeown, Rebecca Passonneau, David K Elson, and Ani Nenkova. Do summaries help? a task-based evaluation of multi-document summarization. 2005.
- [64] Annie Louis and Ani Nenkova. Automatic summary evaluation without human models. In *TAC*, 2008.
- [65] Horacio Saggion, Juan-Manuel Torres-Moreno, Iria da Cunha, Eric SanJuan, and Patricia Velázquez-Morales. Multilingual summarization evaluation without human models. In *Coling 2010: Posters*, pages 1059–1067, 2010.
- [66] TAC. Text analysis conference. <https://tac.nist.gov/>, 09 2020.
- [67] Kai Hong, John Conroy, Benoit Favre, Alex Kulesza, Hui Lin, and Ani Nenkova. A repository of state of the art and competitive baseline summaries for generic news summarization. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC’14)*, pages 1608–1616, 2014.
- [68] Ani Nenkova and Rebecca J Passonneau. Evaluating content selection in summarization: The pyramid method. In *Proceedings of the human language technology conference of the north american chapter of the association for computational linguistics: Hlt-naacl 2004*, pages 145–152, 2004.

- [69] Dragomir Radev, Simone Teufel, Horacio Saggion, Wai Lam, John Blitzer, Hong Qi, Arda Celebi, Danyu Liu, and Elliott Franco Drabek. Evaluation challenges in large-scale document summarization. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 375–382, 2003.
- [70] Shashi Narayan, Shay B Cohen, and Mirella Lapata. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *arXiv preprint arXiv:1808.08745*, 2018.
- [71] Bhargav Srinivasa-Desikan. *Natural Language Processing and Computational Linguistics: A practical guide to text analysis with Python, Gensim, spaCy, and Keras*. Packt Publishing Ltd, 2018.
- [72] Hervé Broquet, Catherine Lanneau, and Simon Petermann. *Les 100 discours qui ont marqué le XXe siècle*. A. Versailles, 2008.
- [73] Wikipédia. Albert einstein. https://fr.wikipedia.org/wiki/Albert_Einstein, 10 2022.
- [74] David Krame. Tp d'éducation à la paix (ulppl/goma). <https://rb.gy/aln4om>, 08 2022.
- [75] David Krame. Tous mes résumés sur google drive. <https://rb.gy/uuplm4>, 10 2022.

Annexes

.1 Exemples des synthèses pour une évaluation qualitative du système

Table 8: Tableau d'exemples de résumés

Modèles	Résumés
Adresse à la nation (P.E. LUMUMBA, le 30 Juin 1960)[72]	
<i>BART</i>	La République du Congo a été proclamée et notre cher pays est désormais entre les mains de ses propres enfants. Nous sommes fiers de cette lutte, qui a été de larmes, de feu et de sang, jusqu'au plus profond de nous-mêmes. Le nouveau Congo, notre chère République que créera mon gouvernement, sera un pays riche, libre et prospère. Nous allons faire régner non pas la paix des fusils et des baïonnettes, mais la paix des cœurs et la bonne volonté
... ↓ ...	

Table 8 – suite à partir de la page précédente

Modèles	Résumés
<i>BARThez</i>	<p>Au nom du gouvernement congolais, je vous demande de faire de ce 30 juin 1960 une date illustre que vous garderez ineffaçablement gravée dans vos cœurs, une date dont vous enseignerez avec fierté la signification à vos enfants. Nous avons connu le travail harassant, exigé en échange de salaires qui ne nous permettaient ni de manger à notre faim, ni de nous vêtir ou nous loger décentement, ni d'élever nos enfants comme des êtres chers. Nous avons connu les souffrances atroces des relégués pour opinions politiques ou croyances religieuses ; exilés dans leur propre patrie, leur sort était vraiment pire que la mort elle-même. « Nous avons souffert dans notre corps et dans notre cœur de l'oppression colonialiste, nous vous le disons tout haut, tout cela est désormais fini. Le président de la République démocratique du Congo (RDC) Joe Biden a annoncé mercredi qu'il allait faire du Congo le centre de rayonnement de l'Afrique tout entière. Il a également annoncé que le pays allait revoir toutes les lois qui étaient en vigueur. Le président de la République française, François Hollande, prononce un discours liminaire à l'Assemblée nationale. « Mon pays, la France, sera prête à vous accorder son aide et son amitié, dit le président français. Dans un entretien fleuve, le président de la République démocratique du Congo a demandé à ses compatriotes de ne pas baisser les bras et de continuer à travailler pour que le Congo soit un pays riche, libre et prospère. Au nom du gouvernement, je vous demande de respecter la vie et les biens de vos concitoyens et des étrangers établis dans notre pays. Au nom de la République démocratique du Congo, l'Assemblée générale de l'Organisation des nations unies (ONU) a adopté un vigoureux texte appelant les Congolais à travailler pour créer une économie nationale.</p>
... ↓ ...	

Table 8 – suite à partir de la page précédente

Modèles	Résumés
<i>Gensim</i>	<p>Congolais et Congolaises, Combattants de l'indépendance aujourd'hui victorieux, Je vous salue au nom du gouvernement congolais, À vous tous, mes amis, qui avez lutté sans relâche à nos côtés, je vous demande de faire de ce 30 juin 1960 une date illustre que vous garderez ineffaçablement gravée dans vos cœurs, une date dont vous enseignerez avec fierté la signification à vos enfants, pour que ceux-ci à leur tour fassent connaître à leur fils et à leurs petits-fils l'histoire glorieuse de notre lutte pour la liberté. Car cette indépendance du Congo, si elle est proclamée aujourd'hui dans l'entente avec la Belgique, pays ami avec qui nous traitons d'égal à égal, nul Congolais digne de ce nom ne pourra jamais oublier cependant que c'est par la lutte qu'elle a été conquise (applaudissements), une lutte de tous les jours, une lutte ardente et idéaliste, une lutte dans laquelle nous n'avons ménagé ni nos forces, ni nos privations, ni nos souffrances, ni notre sang. Cette lutte, qui fut de larmes, de feu et de sang, nous en sommes fiers jusqu'au plus profond de nous-mêmes, car ce fut une lutte noble et juste, une lutte indispensable pour mettre fin à l'humiliant esclavage qui nous était imposé par la force. Ce que fut notre sort en quatre-vingts ans de régime colonialiste, nos blessures sont trop fraîches et trop douloureuses encore pour que nous puissions les chasser de notre mémoire.</p>
... ↓ ...	

Table 8 – suite à partir de la page précédente

Modèles	Résumés
<i>Merging</i>	<p>Congolais et Congolaises, Combattants de l'indépendance aujourd'hui victorieux, Je vous salue au nom du gouvernement congolais, À vous tous, mes amis, qui avez lutté sans relâche à nos côtés, je vous demande de faire de ce 30 juin 1960 une date illustre que vous garderez ineffaçablement gravée dans vos cœurs, une date dont vous enseignerez avec fierté la signification à vos enfants, pour que ceux-ci à leur tour fassent connaître à leur fils et à leurs petits-fils l'histoire glorieuse de notre lutte pour la liberté. Car cette indépendance du Congo, si elle est proclamée aujourd'hui dans l'entente avec la Belgique, pays ami avec qui nous traitons d'égal à égal, nul Congolais digne de ce nom ne pourra jamais oublier cependant que c'est par la lutte qu'elle a été conquise(applaudissements), une lutte de tous les jours, une lutte ardente et idéaliste, une lutte dans laquelle nous n'avons ménagé ni nos forces, ni nos privations, ni nos souffrances, ni notre sang. Mais pour que nous arrivions sans retard à ce but, vous tous, législateurs et citoyens congolais, je vous demande de m'aider de toutes vos forces. J'invite tous les citoyens congolais, hommes, femmes et enfants, de se mettre résolument au travail en vue de créer une économie nationale prospère qui consacrera notre indépendance économique.</p>
... ↓ ...	

Table 8 – suite à partir de la page précédente

Modèles	Résumés
<i>BARTkrame-abs</i>	Nous avons connu le travail harassant, exigé en échange de salaires Nous avons connu les souffrances atroces des relégués pour opinions Ensemble, mes frères, mes sœurs, nous allons commencer une nouvelle lutte, Nous allons montrer au monde ce que peut faire l'homme noir quand il travaille dans Nous allons régner non pas la paix des fusils et des baïonnettes Je vous demande à tous de ne reculer devant aucun sacrifice pour assurer la réussite Je vous demande de respecter inconditionnellement la vie et les biens de vos J'invite tous les citoyens congolais, hommes, femmes et enfants
I have a dream (M.LUTHER KING, le 28 août 1963)[72]	
<i>BART</i>	Cent ans se sont écoulés et l'existence du nègre est toujours entravée par les liens de la ségrégation. L'Amérique a émis aux Noirs un chèque sans valeur; un chèque qui est revenu avec la mention « fonds insuffisants » Ils ont compris que leur liberté est inextricablement liée à notre liberté. Et si l'Amérique veut être une grande nation, cela doit se réaliser.
... ↓ ...	

Table 8 – suite à partir de la page précédente

Modèles	Résumés
<i>BARThez</i>	<p>Il y a un siècle, un grand Américain signait notre acte d'émancipation. Aujourd'hui, cent ans ont passé et le Noir n'est pas encore libre. L'Amérique a failli à sa promesse en ce qui concerne ses citoyens de couleur. C'est pourquoi nous sommes montés à la capitale de notre pays pour toucher un chèque. Nous sommes venus en ce lieu sanctifié pour rappeler à l'Amérique les exigeantes urgences de l'heure présente. Nous sommes venus encaisser un chèque qui nous fournira les richesses de la liberté et la sécurité de la justice. Dans un discours prononcé en 1963 à l'Assemblée nationale, George Floyd déclare qu'il n'y aura « plus ni repos ni tranquillité en Amérique tant que le Noir n'aura pas obtenu ses droits de citoyen ». Comme le relaye le magazine américain <i>The Conversation</i>, « Black Lives Matter », le mouvement de libération de la communauté noire appelle à la vigilance face à la violence physique. « Nous ne pourrions jamais être satisfaits tant que nos corps recrus de la fatigue du voyage ne trouveront pas un abri dans les motels des grand-routes ou les hôtels des villes. » C'est la devise de la lutte pour les droits civiques aux Etats-Unis. « Nous ne pourrions être satisfaits tant qu'un Noir du Mississippi ne pourra pas voter. Nous ne serons pas satisfaits tant que le droit ne jaillira pas comme les eaux et la justice comme un torrent intarissable. » Ne nous vautrions pas dans les vallées du désespoir ». Le jour où les enfants de Dieu pourront chanter ensemble l'hymne américain, ils donneront une signification nouvelle : « Mon pays c'est toi. » Libres enfin. Merci Dieu tout-puissant, nous voilà libres enfin. » C'est la devise du mouvement de libération des Noirs américains.</p>
... ↓ ...	

Table 8 – suite à partir de la page précédente

Modèles	Résumés
<i>Gensim</i>	Je suis heureux de participer avec vous aujourd’hui à ce rassemblement qui restera dans l’Histoire comme la plus grande manifestation que notre pays ait connue en faveur de la liberté. Il y a un siècle de cela, un grand Américain qui nous couvre aujourd’hui de son ombre symbolique signait notre acte d’émancipation. En ce sens, nous sommes montés à la capitale de notre pays pour toucher un chèque. En traçant les mots magnifiques qui forment notre constitution et notre Déclaration d’indépendance, les architectes de notre république signaient une promesse dont héritait chaque Américain. Aux termes de cet engagement, tous les hommes, les Noirs, oui, aussi bien que les Blancs, se verraient garantir leurs droits inaliénables à la vie, à la liberté et à la recherche du bonheur. Il est aujourd’hui évident que l’Amérique a failli à sa promesse en ce qui concerne ses citoyens de couleur.
... ↓ ...	

Table 8 – suite à partir de la page précédente

Modèles	Résumés
<i>Merging</i>	<p>Je suis heureux de participer avec vous aujourd’hui à ce rassemblement qui restera dans l’Histoire comme la plus grande manifestation que notre pays ait connue en faveur de la liberté. Il y a un siècle de cela, un grand Américain qui nous couvre aujourd’hui de son ombre symbolique signait notre acte d’émancipation. »</p> <p>Et si l’Amérique doit être une grande nation, cela doit devenir vrai. Aussi, que résonne la liberté depuis les prodigieux sommets du New Hampshire. Qu’elle résonne depuis les puissantes montagnes de l’État de New York. Qu’elle résonne depuis les hautes Alleghany de Pennsylvanie. Qu’elle résonne depuis les sommets neigeux des Rocheuses du Colorado. Qu’elle résonne depuis les pentes capricieuses de la Californie. Mais cela ne suffit pas. Qu’elle résonne depuis la Stone Mountain de Georgie. Qu’elle résonne depuis la Lookout Mountain du Tennessee. Qu’elle résonne depuis chaque colline et chaque butte du Mississippi, qu’elle résonne du flanc de chaque montagne. Quand nous ferons en sorte que la liberté puisse sonner, quand nous la laisserons carillonner depuis chaque village et chaque hameau, depuis chaque État et chaque cité, nous pourrons hâter la venue du jour où tous les enfants de Dieu, les Noirs et les Blancs, les Juifs et les Gentils, les catholiques et les protestants, pourront se tenir par la main et chanter les paroles du vieux negro spiritual : « Libres enfin.</p>
... ↓ ...	

Table 8 – suite à partir de la page précédente

Modèles	Résumés
<i>BARTkrame-abs</i>	Cent ans ont passé et le Noir n'est pas encore libre. Cent ans Nous sommes montés à la capitale de notre pays pour toucher un chèque. En Nous refusons de croire que la banque de la justice ait fait faillite. Nous Il n'y aura plus ni repos ni tranquillité en Amérique tant que Livrons toujours notre bataille sur les hauts plateaux de la dignité et Nous ne pourrons jamais être satisfaits tant que nos enfants seront dépouillés Nous ne pourrons être satisfaits tant qu'un Noir du Mississippi ne pourra Je suis désolé, mais même si nous devons affronter des difficultés aujourd » Je rêve que mes quatre petits enfants vivront un jour dans un pays où on Avec une telle foi nous serons capables de transformer la cacophonie Ce sera le jour où les enfants de Dieu pourront chanter ensemble cet hymne Mais cela ne suffit pas. Qu'elle résonne depuis la Stone Mountain de
Discours <i>Ich bin ein Berliner</i> (J.F. KENNEDY, le 26 juin 1963)[72]	
<i>BARThez</i>	ICH BIN EIN BERLINER. J'ai été invité par votre distinguéBourgmestre W. Brandt à venir visiter la République fédérale. « Lass'sie nach Berlinkommen ». C'est ce qu'a dit Donald Trump sur Twitter. Lors d'une visite à Berlin, le Bourgmestre de la ville a dit ne ressentir aucune satisfaction en voyant le mur qui divise Berlin entre ses habitants. Le pape François appelle l'humanité à se tourner vers la seule ville de Berlin ou de votre pays, l'Allemagne, vers le progrès de la liberté partout dans le monde. Tous les habitants de Berlin-Ouest pourront tirer une sobre satisfaction du faitqu'ils ont été sur la ligne de front pendant deux décennies.
... ↓ ...	

Table 8 – suite à partir de la page précédente

Modèles	Résumés
<i>BART</i>	Berlin-Ouest a conservé autant de vitalité, de force, d'espoir et de détermination que Berlin-Ouest. Le mur fournit la démonstration la plus éclatante de la faillite du système communiste. Une paix réelle et durable en Europe ne pourra jamais être assurée tant qu'un Allemand sur quatre sera privé du droit fondamental des hommes libres à l'autodétermination.
<i>Gensim</i>	Je ne connais aucune ville assiégée durant dix-huit ans qqui ait conservé autant de vitalité, de force, d'espoir et de détermination que Berlin-Ouest. Le mur fournit la démonstration la plus éclatante de la faillite du système communiste et cette faillite est visible aux yeux du monde entier. Mais nous n'éprouvons aucune satisfaction en voyant ce mur,car il constitue, comme l'a dit votre Bourgmestre, une offense non seulement à l'histoire mais aussi à l'humanité, séparant les familles,les maris de leurs femmes et les frères de leurs soeurs, et divisant un peuple qui voudrait vivre uni.Ce qui est vrai pour cette ville l'est pour toute l'Allemagne.
... ↓ ...	

Table 8 – suite à partir de la page précédente

Modèles	Résumés
<i>Merging</i>	Je suis fier d'être venu dans votre ville, invité par votre distingué Bourgmestre [W. Brandt], qui a symbolisé aux yeux du monde entier l'esprit combatif de Berlin-Ouest. En conclusion, je vous demande donc de regarder au-delà des dangers d'aujourd'hui vers les espoirs de demain, au-delà de la seule liberté de cette ville de Berlin ou de votre pays, l'Allemagne, vers le progrès de la liberté partout dans le monde, au-delà de vous-mêmes et de nous-mêmes vers l'humanité tout entière. La liberté est indivisible et tant qu'un seul homme se trouve en esclavage, tous les autres ne peuvent se considérer comme libres. Mais quand tous les hommes seront libres, nous pourrons attendre avec impatience le jour où cette ville sera réunifiée, de même que ce pays et ce grand continent qu'est l'Europe dans un monde de paix et d'espoir.
Biographie d'Albert Einstein (Copie brute de la page [73])	
... ↓ ...	

Table 8 – suite à partir de la page précédente

Modèles	Résumés
<i>BART</i>	L'intérêt d'Albert Einstein pour la science a été éveillé par une boussole lorsqu'il avait cinq ans. En 1901, il publie son premier article scientifique dans les <i>Annalen der Physik</i> , article consacré à ses recherches sur la capillarité. En 1922, il reçoit le prix Nobel de physique pour ses contributions à la physique théorique. Albert Einstein se lie d'amitié avec le chanteur noir Paul Robeson et devient, à ses côtés, un militant des droits civiques et de la lutte contre le racisme. Ben Gourion lui propose la présidence de l'Etat d'Israël en 1952, ce qu'il refuse. Pendant la guerre froide, il s'est prononcé contre la course aux armements et a appelé les scientifiques à renoncer ensemble à la prolifération des armes atomiques. La citation apocryphe qui lui est attribuée est : « L'astrologie est une science en soi, éclairante. Les connaissances géophysiques mettent en évidence le pouvoir des étoiles et des planètes sur le destin de la Terre. A son tour, dans un certain sens, l'astrologie la renforce. C'est pourquoi c'est une sorte d'élixir de vie pour l'humanité. L'expression a été inventée environ cinq ans après la mort d'Einstein.
Biographie de 11 pages sur Gandhi accessible au lien [74]	
<i>BART</i>	L'idéologie non violente de Gandhi revient à rejeter tout ce qui détruit la personne humaine. Martin Luther King, Nelson Mandela et Ernst Friedrich Schumacher sont tous de grandes figures de la non-violence. La pensée de Gandhi n'est pas explicitement ou directement appliquée par ceux qui prétendent le suivre, mais elle est toujours présente aujourd'hui.

De ces quelques exemples on peut remarquer que *BAThez* est trop biaisé et insère souvent

des bouts de textes et des auteurs n'ayant aucun rapport avec son contexte. Mais ces bouts insérés ont vraisemblablement un grand rapport avec le *dataset OrangeSum* utilisé pour entraîner le modèle *BARThez*.

Résumé de ce mémoire par BART optimisé (ratio=0.3) sans retouche :

A l'ère du numérique, le texte est l'un des principaux moyens de communication et surtout de transmission des savoirs. En 2018, c'était environ 80% des informations circulant sur le web. L'évolution de l'informatique ne cesse de démontrer la possibilité de simplifier toujours grandement la vie humaine en automatisant de plus en plus de tâches. Les réseaux de neurones artificiels (ANN) sont un ensemble de neurones (artificiels) assemblés pour résoudre des tâches considérées comme nécessitant une certaine intelligence. Le neurone artificiel est un algorithme développé en s'inspirant du modèle théorique simplifié d'un neurone naturel. Les cellules LSTM sont utilisées à la place des cellules RNN conventionnelles (appelées vanille) pour permettre au réseau de traiter des séquences de plus en plus longues sans perte rapide d'informations. TextRank est un algorithme de résumé extractif, basé sur la théorie des graphes et inspiré de l'algorithme PageRank de Google. La formation des transformateurs est semi-supervisée. Il existe des résumés monodocument et multidocument selon le nombre de documents sources. Un résumé peut également être classé selon le public auquel il est destiné. Le système doit permettre de : - Synthétiser les textes qui lui sont fournis en entrée (saisis directement ou importés dans des fichiers .pdf, .docx et .txt non scannés ; - Obtenir des synthèses produites par plusieurs algorithmes et les évaluer ; - Stocker les couples document-synthèse ; - Faciliter la navigation des documents en mettant en évidence les parties saillantes ; - Permettre l'affinement d'un modèle de synthèse automatique. Nous voulons développer une plateforme capable de restituer les résumés des textes qui lui seront présentés. Nous devons également être en mesure d'identifier les personnes qui utiliseront l'API dans leurs systèmes (les développeurs dont nous avons parlé dans la section précédente), d'où la nécessité de les enregistrer, de leur donner la possibilité de s'authentifier et de générer une clé d'authentification. . Nous avons prévu 4 familles d'end-points réservés à tout ce qui concerne les vues de l'API (ses interfaces pour permettre l'authentification, la génération de clés et la documentation)

.1. EXEMPLES DES SYNTHÈSES POUR UNE ÉVALUATION QUALITATIVE DU SYSTÈME

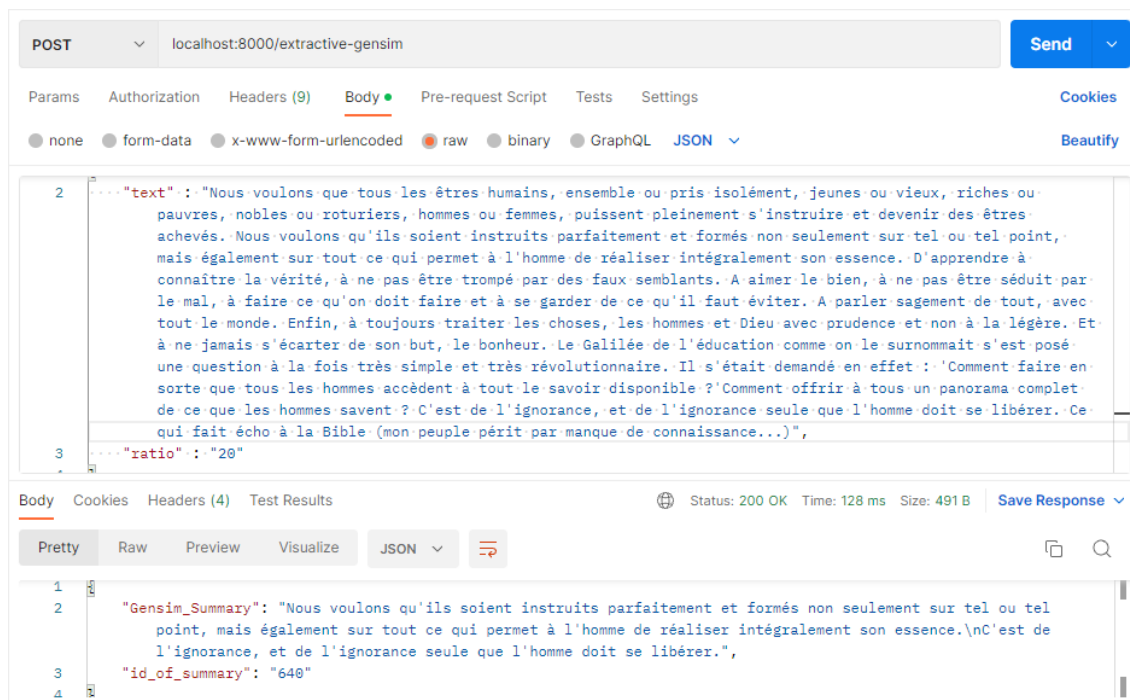
Pour la synthèse abstraite, le choix vers des modèles de type BART a été inspiré par la nécessité de réaliser des synthèses abstraites mais pas trop pour ne pas s'éloigner du contenu du texte source. BART est moins lourd et plus maniable que le modèle T5, c'était le seul à considérer (car PEGASUS et T5 étaient ipso facto exclus). Pour la synthèse abstraite, celle qui se rapproche le plus de la synthèse humaine, nous avons utilisé des modèles de type BART, pour éviter de pousser l'abstraction trop loin comme le ferait PEGASUS, et en même temps éviter d'utiliser inutilement un modèle. Un ensemble de données de synthèse automatique que nous avons nommé for-ULPGL-Dissertation hébergera au fil du temps les données de synthèse collectées via la base de données de notre système.

Pour d'autres exemples

Pour voir d'autres exemples réalisés grâce à ce système, visiter le lien [\[75\]](#).

.2 Documentation sur les end-points utilisables de l'API

Cette annexe a pour raison d'être, de donner une précision sur les end-points de l'implémentation finale de notre API de synthèse. Pour l'end-point **POST** */extractive-gensim*, voici un exemple de requête réussie :



Mais pour la réussite de toute requête, il faudra préciser le paramètre **api-key** dans les **headers**. Ce paramètre aura pour valeur la clé générée à travers l'API pour avoir les autorisations. Voici comment c'est précisé dans la requête précédente :

2. DOCUMENTATION SUR LES END-POINTS UTILISABLES DE L'API

POST localhost:8000/extractive-gensim

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

Key	Value	Description
Content-Type	application/json	
Content-Length	<calculated when request is sent>	
Host	<calculated when request is sent>	
User-Agent	PostmanRuntime/7.29.2	
Accept	*/*	
Accept-Encoding	gzip, deflate, br	
Connection	keep-alive	
api-key	eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJlb...	

Body Cookies Headers (4) Test Results Status: 200 OK Time: 128 ms Size: 491 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "Gensim_Summary": "Nous voulons qu'ils soient instruits parfaitement et formés non seulement sur tel ou tel
3   point, mais également sur tout ce qui permet à l'homme de réaliser intégralement son essence.\nC'est de
4   l'ignorance, et de l'ignorance seule que l'homme doit se libérer.",
  "id_of_summary": "640"
```

En cas d'erreur de clé, un exemple de message est le suivant :

POST localhost:8000/extractive-gensim

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies

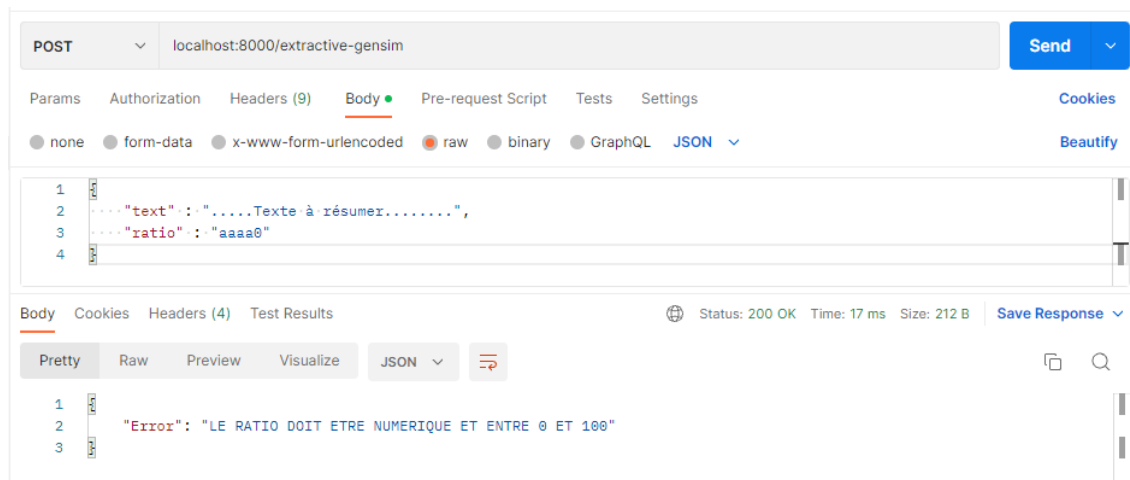
none form-data x-www-form-urlencoded raw binary GraphQL JSON

Body Cookies Headers (4) Test Results Status: 401 UNAUTHORIZED Time: 33 ms Size: 195 B Save Response

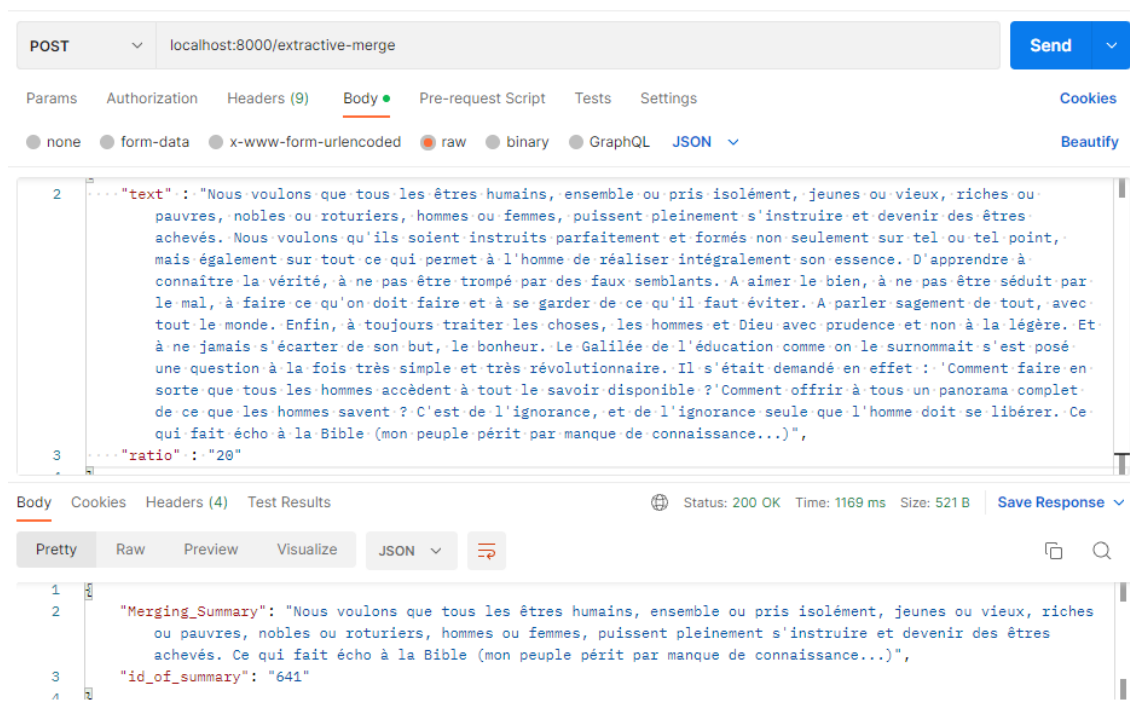
Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Token is invalid!"
3 }
```

Si on précise un *ratio* inapproprié, le message, pour tous les cas, est le suivant :

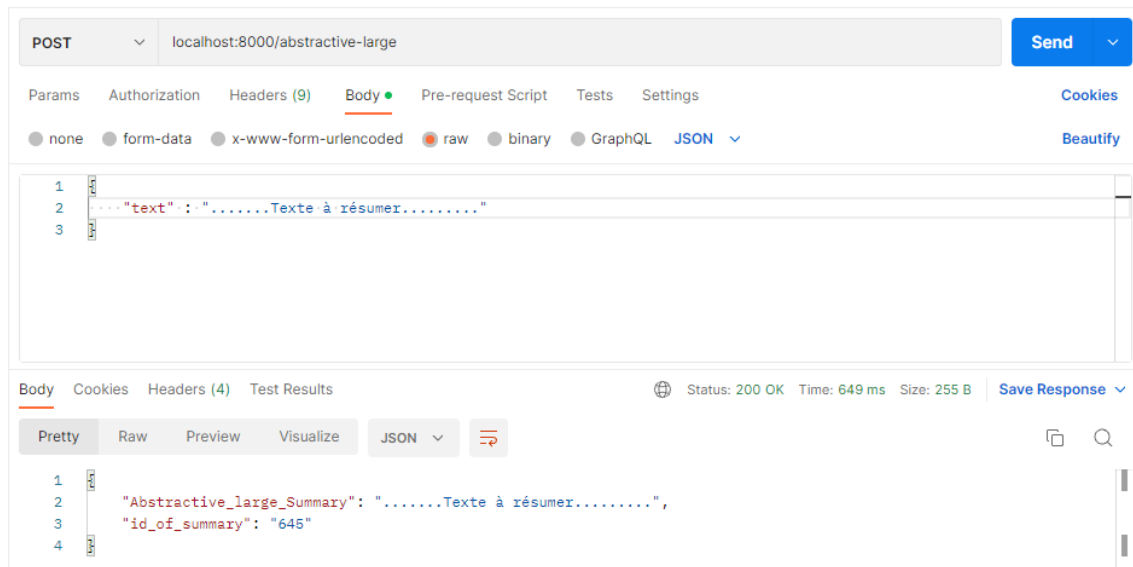


Pour l'end-point *POST /extractive-merge*, voici un exemple de requête :



Pour la partie de synthèse abstractive en utilisant *BART*, pour les petits documents, c'est-à-dire l'end-point *POST /abstractive-large*, voici un exemple de requête réussie :

2. DOCUMENTATION SUR LES END-POINTS UTILISABLES DE L'API



Au cas où la taille est dépassée pour la partie petits documents, voici un exemple de requête :



Pour l'end-point *POST /abstractive-very-large*, voici un exemple :

The screenshot shows a REST client interface with a POST request to `localhost:8000/abstractive-very-large`. The request body is a JSON object: `{ "text": ".....Texte à résumer....." }`. The response status is 200 OK, with a time of 863 ms and a size of 260 B. The response body is a JSON object: `{ "Abstractive_very_large_Summary": ".....Texte à résumer.....", "id_of_summary": "647" }`.

```
POST localhost:8000/abstractive-very-large

{
  "text": ".....Texte à résumer....."
}
```

Status: 200 OK Time: 863 ms Size: 260 B

```
{
  "Abstractive_very_large_Summary": ".....Texte à résumer.....",
  "id_of_summary": "647"
}
```

Pour l'end-point **POST /abstractive-barthez**, voici également un exemple :

The screenshot shows a REST client interface with a POST request to `localhost:8000/abstractive-barthez`. The request body is a JSON object: `{ "text": ".....Texte à résumer....." }`. The response status is 200 OK, with a time of 1 m 0.49 s and a size of 223 B. The response body is a JSON object: `{ "Summary": ".....", "id_of_summary": "648" }`.

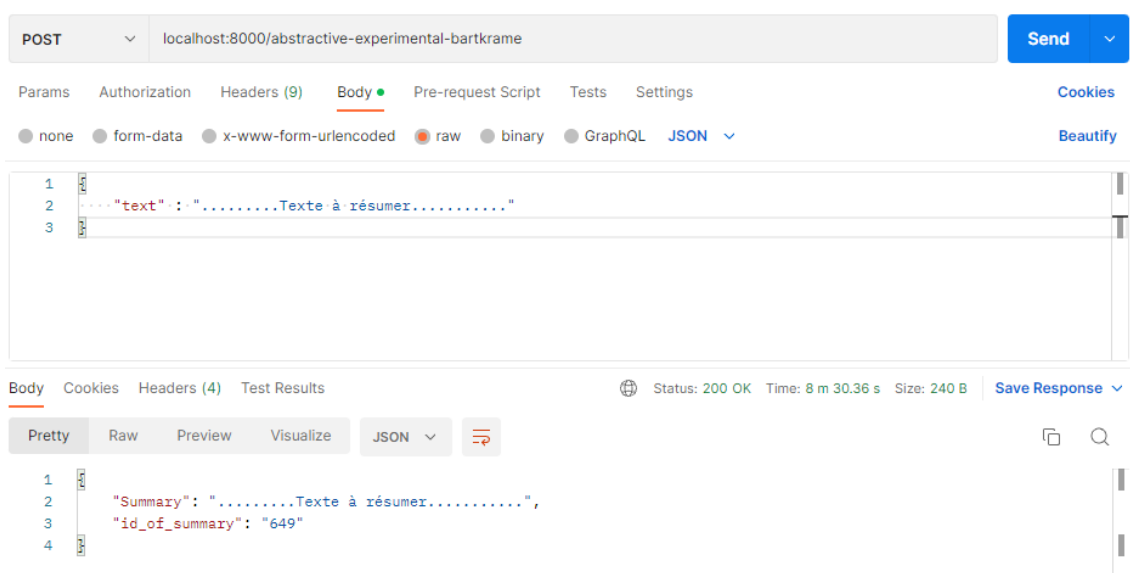
```
POST localhost:8000/abstractive-barthez

{
  "text": ".....Texte à résumer....."
}
```

Status: 200 OK Time: 1 m 0.49 s Size: 223 B

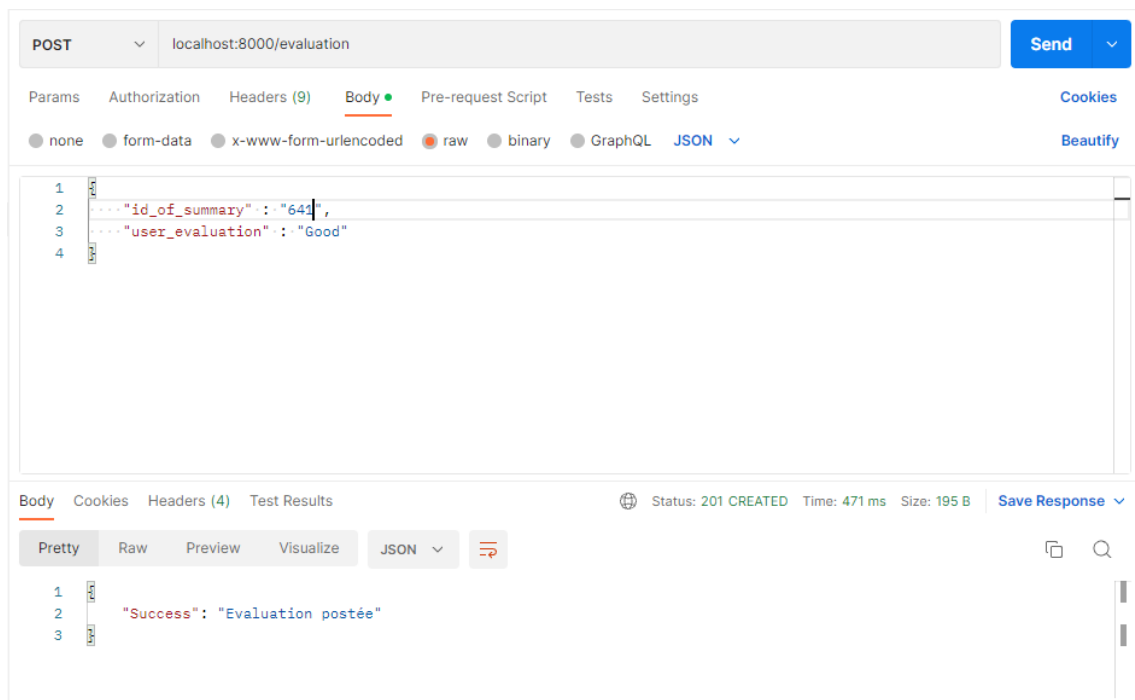
```
{
  "Summary": ".....",
  "id_of_summary": "648"
}
```

Pour l'end-point **POST /abstractive-experimental-bartkrame**, voici aussi un exemple :



Finalement, pour l'évaluation des résumés, c'est-à-dire l'end-point *POST /evaluation*, voici un exemple complet de requête :

- En cas de réussite :



- En cas d'échec :

The screenshot shows a REST client interface with the following details:

- Method:** POST
- URL:** localhost:8000/evaluation
- Body (Request):**

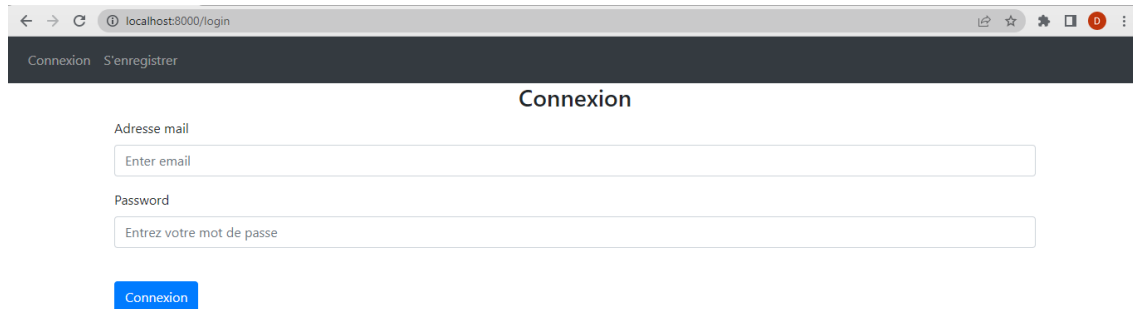
```
1 {
2   ... "id_of_summary": "624",
3   ... "user_evaluation": "Neutral"
4 }
```
- Status:** 400 BAD REQUEST
- Time:** 44 ms
- Size:** 235 B
- Body (Response):**

```
1 {
2   "Failed": "Vous ne pouvez modifier que ce que vous avez créé"
3 }
```

.3 Captures des interfaces d'authentification de l'API

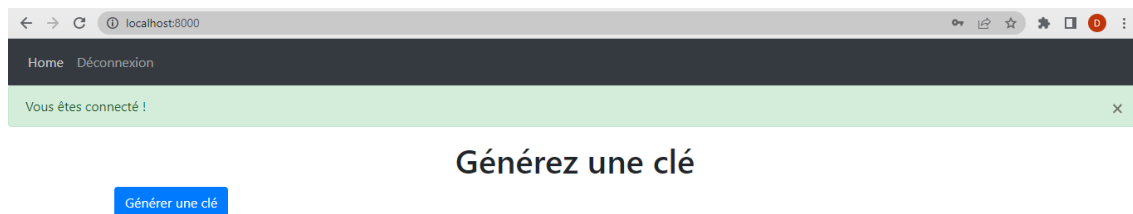
Mis à part dans la partie consacrée à la documentation, nous présentons ici le reste du *frontend* de l'API.

Avant connexion, la page suivante se présente :



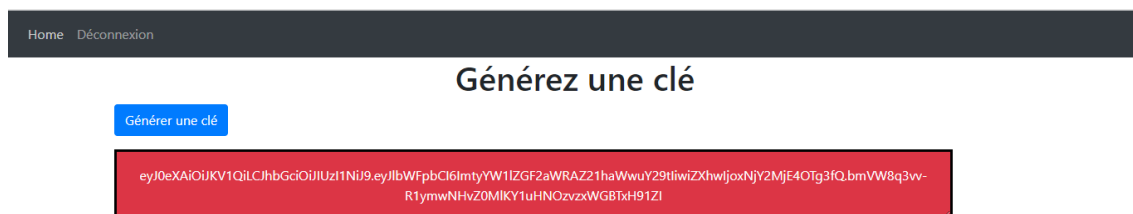
The screenshot shows a web browser at localhost:8000/login. The page has a dark header with 'Connexion' and 'S'enregistrer' links. The main content area is titled 'Connexion' and contains a form with two input fields: 'Adresse mail' (with placeholder 'Enter email') and 'Password' (with placeholder 'Entrez votre mot de passe'). Below the fields is a blue 'Connexion' button.

Après connexion, voici l'interface très succincte qui se présente :



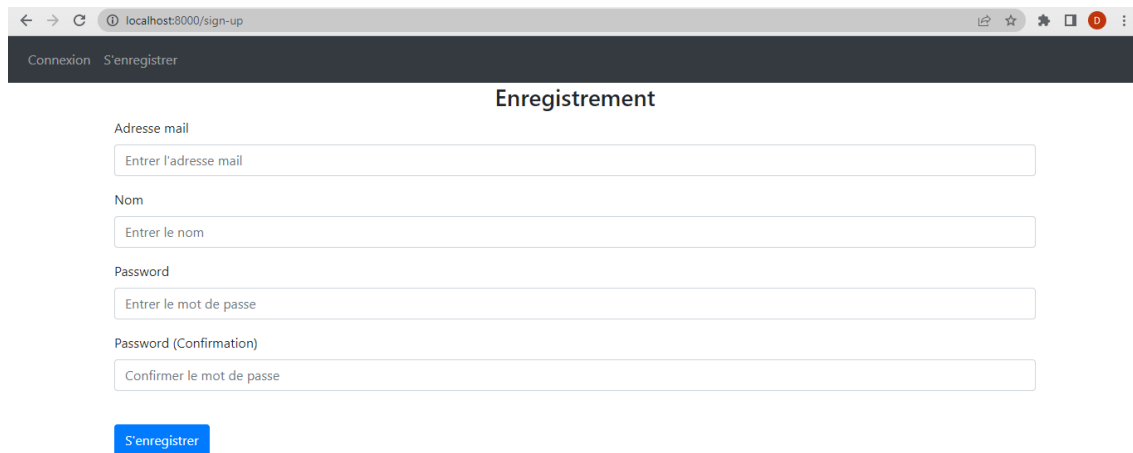
The screenshot shows the home page at localhost:8000. The header has 'Home' and 'Déconnexion' links. A green banner at the top says 'Vous êtes connecté !'. The main content area is titled 'Générez une clé' and contains a blue 'Générer une clé' button.

Après génération de la clé, voici ce qui se présente :



The screenshot shows the home page at localhost:8000. The header has 'Home' and 'Déconnexion' links. The main content area is titled 'Générez une clé' and contains a blue 'Générer une clé' button. Below the button is a red box containing a long alphanumeric string representing the generated API key: 'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJlbWFnbnCI6ImtyYW1IZGF2aWRhZ21haWwY29tIiwiaXNjaXNjY2MjE4OTg3fQ.bmVW8q3v-R1ymwNHvZ0MIKY1uHNOvzxWGBTxH91ZI'.

Finalement, pour l'enregistrement, voici ce qui se présente :



The screenshot shows a web browser window with the address bar displaying 'localhost:8000/sign-up'. The browser's navigation bar includes 'Connexion' and 'S'enregistrer' links. The main heading of the page is 'Enregistrement'. Below this, there are four input fields with labels: 'Adresse mail' (with placeholder 'Entrez l'adresse mail'), 'Nom' (with placeholder 'Entrez le nom'), 'Password' (with placeholder 'Entrez le mot de passe'), and 'Password (Confirmation)' (with placeholder 'Confirmer le mot de passe'). At the bottom of the form is a blue button labeled 'S'enregistrer'.

Ceci est un ensemble d'interfaces très minimaliste mais on peut écrire quelques textes d'explication si nécessaire.