

...

Analyse des pauses entre les chunks sur des données écrites en temps réel

...

Yingzi LIU,
Sarah ALMEIDA BARRETO,
Sandra JAGODZINSKA,
Diego ROSSINI.






LES PRINCIPES DU PROJET

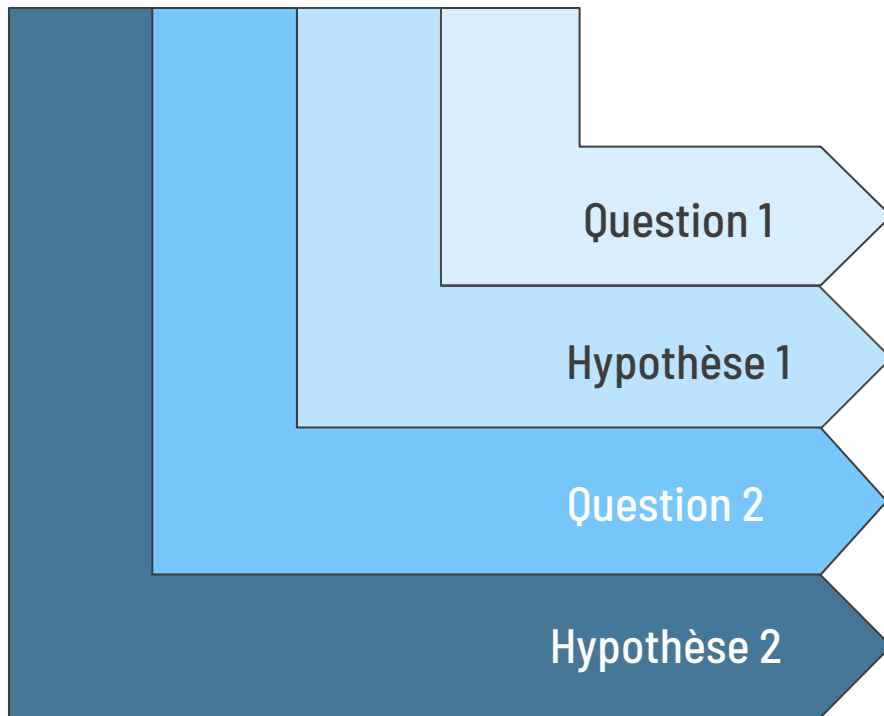
Corpus : il est constitué de données écrites enregistrées en temps réel. Il contient des informations comme la position initiale et finale du curseur dans un laps de temps donné, les caractères écrits et supprimés ainsi que les textes finaux. À chaque arrêt du participant, nous avons un retour à la ligne.

Objectif : le projet vise à explorer une tâche de chunking sur des données écrites en temps réel. La nature des données spécifiques rend cette tâche difficile et encore peu traitée en TAL.

Étapes :

1. *Hypothèses*
 2. *Présentation des données*
 3. *Recherche et choix entre les chunkers disponibles*
 4. *Collecte des données en format exploitable*
 5. *Statistiques*
 6. *Vérification des hypothèses et résultats*
- 

1. HYPOTHÈSES ET QUESTIONS DE RECHERCHE



Est-ce que les outils disponibles arrivent à bien segmenter des données ?

Les pauses arrivent aux frontières des chunks.

Si oui, est-ce que les pauses arrivent entre des chunks particuliers ?

Les différents processus de textualisation présentent les mêmes corrélation entre les pauses et les chunks.



2. PRÉSENTATION DES DONNÉES

Les données nous ont été fournies par Mme. Taravella et consistent en:

- 1)** Une documentation sur comment les données ont été enregistrées, traitées et formalisées ;
- 2)** Trois tableaux .csv où chaque ligne fournit des informations spatio-temporels sur ce qui a été écrit en temps réel entre deux pauses ;
- 3)** Un ensemble de documents .txt qui représentent les textes finaux et ce qui a été obtenu à la fin du processus d'écriture.

APERÇU DES DONNÉES .CSV

	J	K	L	M	N	O	P	Q
1	pct_pause	longueur_burst	burst	startPos	endPos	docLength	categ	charBurst
2	0	0		0	0	0	P	
3	0,627128663	21	L'arrêt du tabac est	0	21	21	P	IA_ La diminution du tabac est
4	0,173347779	42	un sujet qui est de plus en plus fréquent	21	63	63	P	un_sujet_ qui_ est_ de_ plus_ plus_
5	0,161602782	44	dans les conversations en France, où encore	63	107	107	P	dans_ no_ les_ conversations_ en_ France_ où_
6	0,432624113	0		107	63	63	RB	
7	0,815450644	0		63	56	56	RB	
8	0,374031008	7	équent	56	63	63	P	équent_ dans_
9	0,177136972	24	de plus en plus répandu	63	53	53	RB	
10	0,387387387	12	en France. L	53	65	65	P	en_ France_ L
11	0,74	0		65	65	65	P	a_
12	0,330729167	15	es différentes	65	80	80	P	es_ diff_ érenn_ tes_
13	0,773629933	10	e. L'Etat	80	71	71	RB	
14	0,907142857	2	a	71	73	73	P	a_
15	0,450581395	10	même crée	73	83	83	P	même_ crée_
16	0,087341893	142	différentes mesures pour faire que les personnes fumeuses diminues la cigarette. Il a par exem	83	217	226	P	différentes_ mesures_ pour_ faire_ que_ la_ les

Nombre de caractères présents dans le texte à la fin de l'intervalle de pause

Ce qui a été écrit à la fin de l'intervalle de pause

La position de départ et de fin du curseur au moment où l'écriture commence et se termine entre deux pauses ainsi que la longueur du document à la fin de l'intervalle de pause

Enregistrement de tout ce qui a été tapé sur le clavier y compris les effacements et les espaces

EXPLICATION DES DONNÉES .CSV

Le corpus de textes réalisé par Sirine découle de 3 études.

Dans ces études 3 processus mis en jeu dans la textualisation sont considérés : la planification, la formulation et la révision. La planification est la production des idées et leur organisation dans le texte. La formulation est l'étape suivante, à savoir la traduction de ces concepts en langage et sa mise en texte. Enfin, la révision est le fait de modifier des éléments du texte (révision directe) ou de modifier les concepts ou l'organisation des concepts avant la transcription (révision indirecte).

Dans les 3 expériences les étudiants rédigent sur ordinateur un texte argumentatif pendant 30min. Le texte argumentatif (produire des arguments pour et contre) est un type de document qui implique le participant à la tâche malgré des thèmes parfois non familiers.

La variable manipulée dans chaque étude est le **coût cognitif**. Comme chaque étude se focalise sur un processus différent, le coût cognitif sera opérationnalisé différemment. Un participant sera soit en condition contrôle "coût faible (-)" soit en condition "coût élevé (+)".

Dans l'étude 1 on contraint le processus de **planification** via le niveau de connaissance du thème.

- Condition contrôle : thèmes de rédaction évalués comme familiers (ex. installation de fumoirs dans l'université)
- condition coûteuse + : thèmes inconnus (ex. changement de règles de propulsion au décollage des avions).

Dans l'étude 2 on cherche à contraindre la **formulation**.

- Condition contrôle : pas de bruit diffusé
- Condition coûteuse + : diffusion d'un discours en fond sonore.

Dans l'étude 3 on va contraindre le processus de **révision**.

- Condition contrôle : feedback visuel (écran qui affiche le texte écrit)
- Condition coûteuse + : pas de feedback visuel (écran noir)

Comme expliqué par les chercheurs, les trois .csv sont issus d'une étude qui formalise les données récoltées lors de plusieurs expériences. Ces expériences diffèrent selon trois processus de mise en jeu dans la textualisation : formulation, planification et révision. Chacun des processus est ensuite testé dans deux conditions différentes : condition de contrôle et condition coûteuse au niveau cognitif.





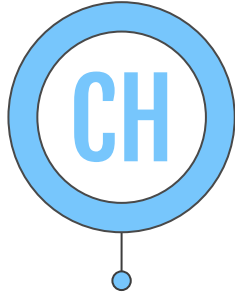
3. RECHERCHE ET CHOIX PARMIS LES CHUNKERS DISPONIBLES

Nous avons testé et évalué quatre chunkers différents. Le but était de déterminer lequel s'adapte le mieux aux données que nous avons à disposition. Afin d'effectuer ce test, nous avons procédé de la façon suivante :

- 1)** Nous avons choisi de lancer chaque chunker sur 30 phrases issues de la colonne "burst" du .csv "formulation". Les données de cette colonne représentent ce qui a été écrit dans un intervalle de temps, c'est à dire entre deux pauses ;
- 2)** Nous avons ensuite constitué un Gold Standard dans lequel nous avons corrigé les erreurs de chaque chunker tout en utilisant les mêmes étiquettes de POS et de chunks du chunker utilisé ;
- 3)** Nous avons comparé les résultats des chunkers avec leurs Golds et nous avons choisi celui qui avait les meilleurs scores d'évaluation.

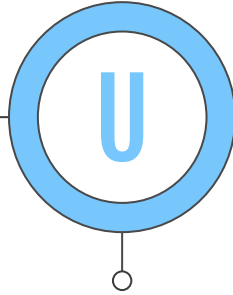


CHOIX DU CHUNKER



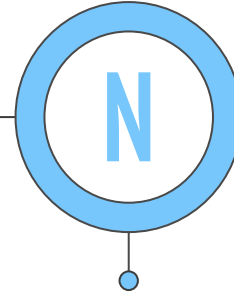
SEM

un logiciel permettant d'annotation du texte en POS, sur lequel il effectue un chunking. Il a été développé par Yoann Dupont et Isabelle Tellier.



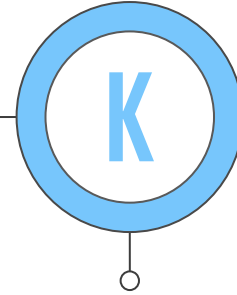
Treetagger

un outil pour annoter le texte en POS et lemmes. Il a été développé par Helmut Schmid à l'Université de Stuttgart.



SpaCy

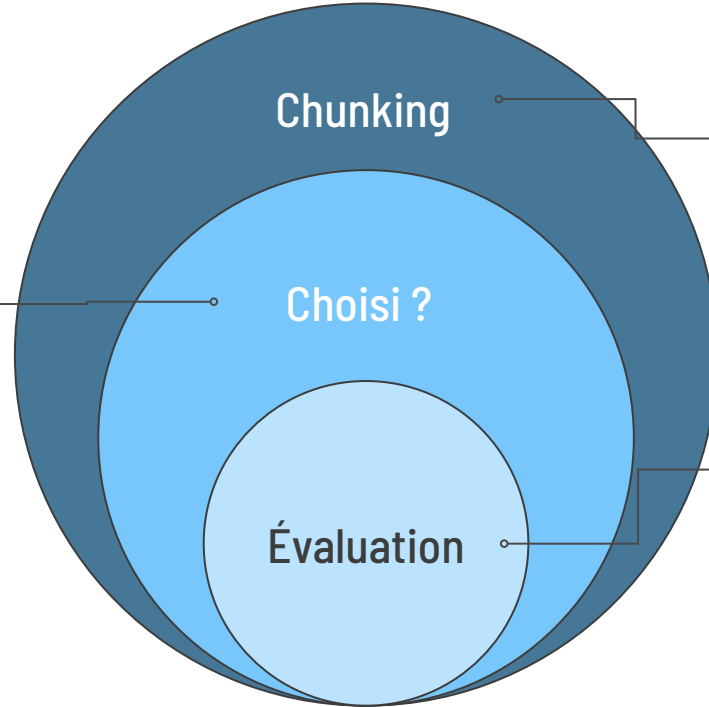
une bibliothèque en Python de traitement automatique des langues développée par Matt Honnibal et Ines Montani.



NLTK

une bibliothèque Python de traitement automatique des langues, développée par Steven Bird et Edward Loper du département d'informatique de l'université de Pennsylvanie.

NLTK



REJETÉ

NLTK était le premier outil rejeté. Après le POS tagging, l'outil exige de définir nos propres règles de chunking par regex, en se basant sur des étiquettes de POS.

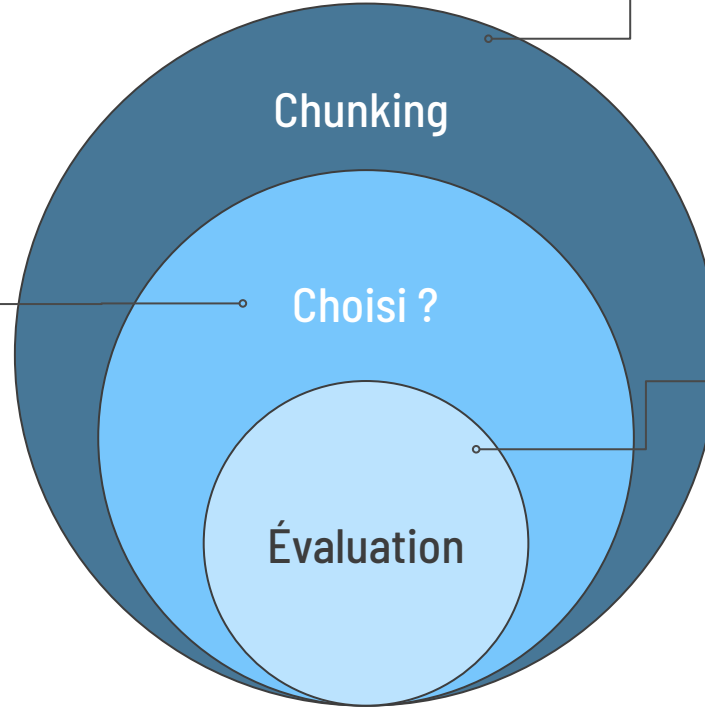
Comment ?

NLTK permet l'étiquetage du texte en POS. Le chunking se base sur ces étiquettes.

Résultats

Vue que l'outil ne permet pas de chunker de façon automatisé, nous avons pas recueilli des mesures d'évaluation

SPACY



REJETÉ

Spacy rencontre des difficultés pour identifier précisément les limites entre les différentes parties lors de la segmentation des phrases en chunks.

Comment ?

Dans SpaCy, le chunking est généralement réalisé en analysant les phrases en arbres de dépendance, il faut donc commencer par la tokenisation puis les annotations pour obtenir les dépendances et les POS.

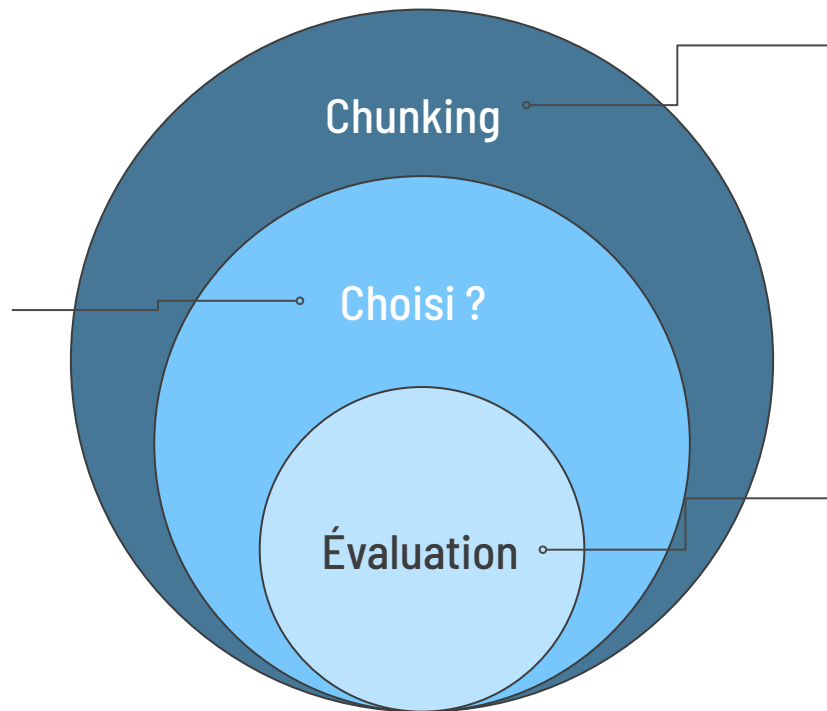
Résultats

Il y a plusieurs erreurs de syntaxe dans le résultat, telles que l'utilisation de chunks qui n'ont pas de sens, la combinaison incorrecte de mots dans un chunk, ainsi que la division incorrecte de phrases en chunks.

TreeTagger

REJETÉ

Après une comparaison avec un Gold Standard, nous avons rejeté TreeTagger car la division en chunks n'est pas optimale et ne semble pas exploitable



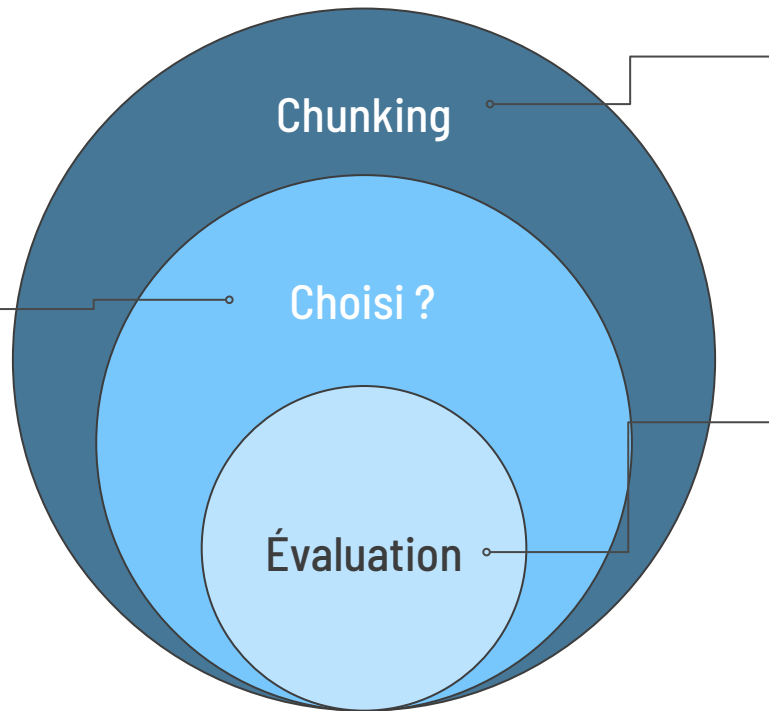
Comment ?

TreeTagger dispose d'un chunker qui doit être appliqué à un fichier .txt contenant tous les tokens des phrases que l'on veut parser en chunks

Résultats

TreeTagger semble avoir du mal avec certains signes de ponctuation et imbrique les chunks NP dans des chunks PP

SEM



ACCEPTÉ

SEM était depuis le début notre Saint-Graal. Il est très efficace et précis quand il s'agit de POS tagging. Alors, nous avions de grandes attentes pour le chunking et la curiosité de savoir comment il traiterait nos données spécifiques.

Comment ?

Au début, l'outil annote le texte avec des étiquettes de POS et ensuite en se basant sur ses résultats il regroupe des chunks

Résultats

Nous avons évalué la performance de POS tagging et chunking en mesures de : rappel, précision et f-mesure, calculés sur 30 premières lignes.

Évaluation SEM – Chunking

```
import os

with open('SEM30.txt') as fichier :
    textSEM = fichier.read()
with open('SEM_Gold.txt') as Gold :
    textGold = Gold.read()

#liste de chunkers

def chunker(text):
    nCh = ""
    listDeChunkers = []
    punct = ['.', ',', '!', '?']
    for i in range(len(text)):
        if text[i-1] == ")" and text[i] == " " or text[i-1] == ")" and text[i] == "\n" or text[i] == " " and text[i+1] == "(" or text[i] == "\n" and text[i+1] == "(" :
            listDeChunkers.append(nCh)
            nCh = ""
        else:
            nCh+=text[i]
    return listDeChunkers

chunkerSEM = chunker(textSEM)
chunkerGold = chunker(textGold)
```

Précision de chunking est
égale à 92%

Rappel de chunking est
égale à 98%

F-mesure de chunking est
égale à 95%

```
def prefix(ch):
    nch = ""
    for l in ch:
        if l == " ":
            break
        return nch
    nch+=l
    return nch

#Evaluation chunks
VP = 0
FP=0
FN=0
for i in range(len(chunkerSEM)) :
    prefixSEM = prefix(chunkerSEM[i])
    prefixGold = prefix(chunkerGold[i])
    if chunkerSEM[i] == chunkerGold[i] or prefixSEM == prefixGold :
        VP+=1
    else :
        if chunkerGold[i] == '(NULL )' or prefixSEM != prefixGold and prefixSEM != '(NULL) : #token w sem inny niz w gold ale token sem nie null
            FP+=1
        elif chunkerSEM[i] == '(NULL )' or prefixGold != prefixSEM and prefixGold != '(NULL) : #token w gold inny niz w sem ale token gold nie null
            FN+=1

precisionChunk, rappelChunk, FmeasureChunk = mesures(VP,FP,FN)
print(f'Précision de chunking est égale à {precisionChunk*100}% Rappel de chunking est égale à {rappelChunk*100}% Fmesure de chunking est égale à {FmeasureChunk*100}%')
```



Évaluation SEM – POS

e/NC e/NULL
L'/DET L'/DET
Etat/NPP Etat/NPP
a/V a/V

← Exemple des données
problématiques à cause
d'un mot coupé.

```
#prend liste de chunks en entrée
def POSlist(l):
    ele = l[0]
    g = ele.split()
    print(g)
    listDePos = g[1:-1]
    for i in range(1, len(l)) :
        ele = l[i]
        g = ele.split()
        #g index 0-->étiquette chunk 1 -->token/pos... -1 -->")"
        for ele in g[1:-1]:
            listDePos.append(ele)
    return listDePos
```

```
poslistSEM = POSlist(chunkerSEM)
poslistGold = POSlist(chunkerGold)
```

```
#traitement si il pas bien divisé les constituants marche seulement
#dans le cas quand sem a collé des choses ensembles
for i in range(len(poslistGold)):
    if (poslistSEM[i])[2] != (poslistGold[i])[2] :
        poslistSEM.insert(i, 'null/null')
```

```
for i in range(len(poslistSEM)) :
    print(poslistSEM[i], poslistGold[i])
```

```
def getThatPos(ele):
    b = ele.split('/')
    token = b[0]
    pos = b[-1]
    return token, pos
```

Précision de POStagging est
égale à 94%

Rappel de POStagging est
égale à 98%

F-mesure de POStagging est
égale à 96%



```
#evaluation pos
```

```
VP=0
FN=0
FP=0
```

```
for i in range(len(posGold)):
```

```
    token, pos = getThatPos(poslistSEM[i])
    tokenG, posG = getThatPos(poslistGold[i])
    if poslistSEM[i]==poslistGold[i]:
        VP+=1
```

```
    else :
```

```
        if poslistGold[i] == 'null/null' or pos != posG and pos != 'null': #tokensem diff. que tokengold mais tokensEM n'est pas null
            FP+=1
        elif posSEM[i] == 'null/null' or posG != pos and posG != 'null': #tokengold diff. que tokenSEM mais token gold n'est pas null
            FN+=1
```

```
precisionPOS, rappelPOS, FmesurePOS = mesures(VP,FP,FN)
```

```
print(f'Precision de POSTag est égale à {precisionPOS*100}% Rappel de POSTag est égale à {rappelPOS*100}% Fmesure de POSTag est égale à {FmesurePOS*100}%')
```



4. COLLECTE DES DONNÉES EN FORMAT EXPLOITABLE

Afin de vérifier notre question 1 (“Est-ce que les pauses arrivent aux frontières des chunks ?”), nous avons besoin de collecter les données présentes dans chaque .csv et de les transformer dans le but de les chunker et d’observer où se placent les pauses. Toutefois, les données à notre disposition présentaient trois inconvénients majeurs :

- 1) Les phrases de la colonne “burst” ne présentent pas toujours une mise en forme correcte ; des fautes de frappe, de grammaire/orthographe et plusieurs erreurs sont présents tout au long du .csv ce qui peut avoir une forte incidence sur le résultats du chunker SEM ;
- 2) Les phrases de la colonne “charBurst” contiennent tous les caractères qui on été tapé et ne semblent pas exploitable pour une analyse en chunks ;
- 3) Les pauses entre deux phrases ne sont signalées que par un retour à la ligne dans le .csv. L’absence d’un symbole de pause entre phrases nous a obligé à insérer un caractère spécial symbolisant la pause (@).

ESSAI N°1 : RECONSTRUCTION DU TEXTE

Nous avons essayé de prendre en considération la colonne "charBurst" et de reconstruire le texte de chaque intervalle de pause. Le but était de suivre pas à pas l'écriture en temps réel en obtenant finalement un texte propre et prêt pour la division en chunks de SEM. Un script Python a été produit dans ce but :

```
#!/usr/bin/env python3
#---TREATMENT---

outputLines=[]
outputLine=[]
outputIndex=0
correctionIndex=0
for i,j in enumerate(fileInput):
    if j[charBurstIndex][0]=="\n" and j[charBurstIndex][len(j[charBurstIndex])-1]=="\n":
        fileInput[i][charBurstIndex]=j[charBurstIndex][1:len(j[charBurstIndex])-1].replace("\n","")
for i in fileInput:
    toAddIndex=0
    for j in i[charBurstIndex]:
        if j!="\n":
            while True:
                if int(i[startPosIndex])+toAddIndex-correctionIndex>len(outputLine):
                    correctionIndex+=1
                else:
                    break
            outputLine.insert(int(i[startPosIndex])+toAddIndex-correctionIndex,j)
            toAddIndex+=1
        else:
            while True:
                try:
                    outputLine.pop(int(i[startPosIndex])-1+toAddIndex-correctionIndex)
                    break
                except:
                    correctionIndex+=1
            toAddIndex-=1
    outputLines.append(outputLine)
    outputLine=outputLines[outputIndex][:]
    outputIndex+=1

fileOutput=open(f"output_{fileToOpen}.csv","w",encoding="utf-8")
fileOutput.write("charBurst\twholeTextTyped\n")
for i,j in zip(fileInput,outputLines):
    fileOutput.write(f"{i[charBurstIndex]}\t{''.join(j).replace(' ',' ')}\n")
fileOutput.close()
```


ESSAI N°1 : RECONSTRUCTION DU TEXTE

Ce script Python essaie de reconstituer le texte en s'appuyant aussi sur la position du curseur au début d'un intervalle (colonne "startPos"). Toutefois il nous a été impossible de produire un résultat satisfaisant car plusieurs intervalles présentent des déplacements en leur sein sans qu'aucune indication de l'emplacement du curseur ne soit fournie.

```
#!/usr/bin/env python3
#---TREATMENT---

outputLines=[]
outputLine=[]
outputIndex=0
correctionIndex=0
for i,j in enumerate(fileInput):
    if j[charBurstIndex][0]=="\n" and j[charBurstIndex][len(j[charBurstIndex])-1]=="\n":
        fileInput[i][charBurstIndex]=j[charBurstIndex][1:len(j[charBurstIndex])-1].replace("\n\n","\n")
for i in fileInput:
    toAddIndex=0
    for j in i[charBurstIndex]:
        if j!="\n":
            while True:
                if int(i[startPosIndex])+toAddIndex-correctionIndex>len(outputLine):
                    correctionIndex+=1
                else:
                    break
            outputLine.insert(int(i[startPosIndex])+toAddIndex-correctionIndex,j)
            toAddIndex+=1
        else:
            while True:
                try:
                    outputLine.pop(int(i[startPosIndex])-i+toAddIndex-correctionIndex)
                    break
                except:
                    correctionIndex+=1
            toAddIndex-=1
    outputLines.append(outputLine)
    outputLine=outputLines[outputIndex][:]
    outputIndex+=1

fileOutput=open(f"output_{fileToOpen}.csv","w",encoding="utf-8")
fileOutput.write("charBurst\twholeTextTyped\n")
for i,j in zip(fileInput,outputLines):
    fileOutput.write(f"{i[charBurstIndex]}\t{' '.join(j).replace(' ',' ')}\n")
fileOutput.close()
```



ESSAI N°2 : DES BIGRAMMES À CHUNKER

Une fois compris que la colonne "charBurst" ne pouvait pas représenter pour nous des données exploitables, nous nous sommes tournés vers la colonne "burst" qui contient le texte final de chaque intervalle. Afin de pouvoir chunker chaque ligne et vérifier si les pauses se trouvent ou pas à la frontière entre deux chunks, nous avons pensé que la constitution de bigrammes aurait pu être une solution. L'idée était de prendre deux phrases successives, les coller ensemble, les diviser en chunk et ensuite insérer le symbole de pause entre eux et vérifier si cette dernière se trouve ou non à la frontière entre deux chunks. Un traitement d'une telle sorte nous permet aussi d'observer si des patterns sont identifiables, c'est-à-dire si des types de chunks particuliers semblent provoquer la pause. Toutefois deux grandes difficultés se sont présentées face à nous :

- 1) Comment chunker les bigrammes de phrases sur SEM étant données que nous ne disposons que de la version en ligne de SEM ?
- 2) Comment insérer les pauses une fois obtenus les bigrammes de phrases ?



ESSAI N°2 : DES BIGRAMMES À CHUNKER

Un long script python a été écrit à ce propos. Dans un premier temps, le script récupère les textes des phrases "burst" et crée des bigrammes puis les stocke dans une variable sur la base du modèle suivant :

- Phrase 1 + symbole de pause "@" + Phrase 2
- Phrase 2 + symbole de pause "@" + Phrase 3
- Phrase 3 + symbole de pause "@" + Phrase 4
- Etc...

Le but est d'enregistrer la position de la pause entre deux phrases.

En même temps une autre liste de bigrammes est créée mais cette fois-ci sans les pauses et encore sur le modèle précédent :

- Phrase 1 + Phrase 2
- Phrase 2 + Phrase 3
- Phrase 3 + Phrase 4
- Etc...

C'est cette deuxième liste qui sera passée, bigramme par bigramme, aux chunker SEM en ligne.

ESSAI N°2 : DES BIGRAMMES À CHUNKER

Pour passer à SEM le contenu de notre liste de bigrammes sans les pauses, nous avons utilisé la librairie Selenium qui permet d'effectuer du scraping sur des sites web. Grâce à une fonction, nous passons en boucle tous les bigrammes de notre liste en obtenant des bigrammes divisé en chunks en retour. Les résultats sont ensuite écrits dans un fichier texte pour chacun des .csv à notre disposition (formulation, planification et révision).

```
def write_chunks_from_bigrams_burst(burst_clean_no_pause):
    options = Options()
    options.headless = True
    for idx, bigram in enumerate(burst_clean_no_pause):
        bigram = str(bigram)
        if bigram == " ":
            with open("../Selenium_SEM_output//bigram_chunks_complete_formulation", "a+", encoding="utf-8") as file:
                file.write("\n")
                file.close()
            continue
        else:
            driver = webdriver.Firefox(options=options)
            url = "https://apps.lattice.cnrs.fr/sem/index"
            driver.get(url)
            driver.implicitly_wait(5)
            driver.switch_to.alert.dismiss()
            time.sleep(2)
            time.sleep(1)
            driver.find_element(By.XPATH, "//textarea").send_keys(bigram)
            time.sleep(1)
            driver.find_element(By.ID, "inlineCheckbox2").click()
            time.sleep(1)
            driver.find_element(By.XPATH, "//button").click()
            time.sleep(1)
            driver.switch_to.alert.dismiss()
            time.sleep(1)
            driver.find_element(By.XPATH, "//label[contains(., 'Chunking')]").click()
            time.sleep(1)
            driver.find_element(By.XPATH, "//a[contains(., ' text')]").click()

            # Changer le path d'entrée du fichier txt de SEM et la destination du nouveau
            with open("../Selenium_SEM_output//bigram_chunks_complete_formulation", "a+", encoding="utf-8") as file:
                file_path = glob.glob("/home/diego/Downloads/sem_*.text")
                file_path = str(file_path)[2:len(str(file_path))-2]
                with open(file_path, "r", encoding="utf-8") as f:
                    unique_line = ""
                    for line in f.readlines():
                        line = line.strip('\n')
                        unique_line = unique_line + " "+line
                    file.write(unique_line)
                    file.write("\n")
                    os.remove(file_path)
            file.close()
            driver.close()
```

ESSAI N°2 : DES BIGRAMMES À CHUNKER

Voici un aperçu du résultat obtenu:

```
14 (NP 'la'/DET création/NC ) (PP 'de'/P paquet/NC neutre/ADJ ) ./PONCT
15
16 ./PONCT (NP On/CLS ) (VN peut/V ) (VN voir/VINF ) (CONJ que/CS ) (NP certaines/PRO ) (PP de/P ces/DET )
17 ./PONCT (NP On/CLS ) (VN peut/V ) (VN voir/VINF ) (CONJ que/CS ) (NP certaines/PRO ) (PP de/P ces/DET mesures/NC ) (VN augmentent/V )
18 (NP mesures/NC ) (VN augmentent/V s/VPP )
19 (NP s/NC ) (VN permettent/V ) (PP aux/P+D personnes/NC ) (PP de/P diminuer/VINF )
20 (VN permettent/V ) (PP aux/P+D personnes/NC ) (PP de/P diminuer/VINF ) (AdP vraiment/ADV ) (NP la/DET cigarette/NC ) ./PONCT (PP Comme/P par_exemple/ADV ) ./PONCT (NP l'/DET augmentation/NC )
  (PP du/P+D prix/NC ) (PP du/P+D tabac/NC ) ./PONCT (CONJ car/CC ) (NP les/DET personnes/NC ) (VN a/V ) (AdP plus/ADV ) (NP faibles/ADJ revenu/NC ) (CONJ ou/CC ) (VN ne/ADV voulant/VPR pas/ADV )
21 (AdP vraiment/ADV ) (NP la/DET cigarette/NC ) ./PONCT (PP Comme/P par_exemple/ADV ) ./PONCT (NP l'/DET augmentation/NC ) (PP du/P+D prix/NC ) (PP du/P+D tabac/NC ) ./PONCT (CONJ car/CC ) (NP
  les/DET personnes/NC ) (VN a/V ) (AdP plus/ADV ) (NP faibles/ADJ revenu/NC ) (CONJ ou/CC ) (VN ne/ADV voulant/VPR pas/ADV mettre/VINF ) (AdP trop/ADV ) (PP d'/P argent/NC ) (PP dans/P les/DET
  cigarettes/NC ) (VN vont/V ) (VN diminuer/VINF ) (CONJ ou/CC ) (AdP alors/ADV ) (VN passer/VINF ) (NP des/DET cigarettes/NC )
22 (VN mettre/VINF ) (AdP trop/ADV ) (PP d'/P argent/NC ) (PP dans/P les/DET cigarettes/NC ) (PP dans/P les/DET cigarettes/NC ) (VN vont/V ) (VN diminuer/VINF ) (CONJ ou/CC ) (AdP alors/ADV ) (VN passer/VINF ) (NP des/DET
  cigarettes/NC pré-faites/ADJ ) (PP aux/P+D cigarettes/NC ) (PP à/P roulées/NC ) ./PONCT
```

À ce stade, il est nécessaire d'insérer les pauses au bon index dans chacun des bigrammes divisés en chunks. Pour cela faire nous nous sommes basés sur la liste créée auparavant qui contient les mêmes bigrammes mais avec le symbole de pause "@" en plus (voir slide suivante).

ESSAI N°2 : DES BIGRAMMES A CHUNKER

Pour insérer la pause "@" à la bonne position, nous avons utilisé des expressions régulières qui vont déterminer où placer ce symbole et si le symbole "@" se place effectivement à la frontière entre deux chunks. Si le symbole se trouve entre deux parenthèses dont la première fermante (") et la deuxième ouvrante ("("), alors cela signifie que la pause est à la frontière entre deux chunks. Si ce n'est pas le cas alors le symbole de pause "@" se trouve à l'intérieur d'un chunk. Voici un aperçu de ce deuxième traitement du script et du résultat obtenu :

```
with open("../Selenium_SEM_output/bigram_chunks_complete_formulation", "r", encoding="utf-8") as f:
    with open("../Selenium_SEM_output/bigram_chunks_complete_formulation_with_pauses", "a+", encoding="utf-8") as final:
        for idx, line in enumerate(f.readlines()):
            mot_1 = pauses_entre_mot[idx][0]
            mot_2 = pauses_entre_mot[idx][2]
            match_1 = re.search(mot_1, line)
            match_2 = re.search(mot_2, line)
            set_pause = line[match_1.span()[1]:match_2.span()[0]]
            between_chunks = re.search("\)\s\(", set_pause)
```

(AdP même/ADV) (VN crée/))@(NP différentes/DET mesures/NC) (PP pour/P faire/VINF)



5. STATISTIQUES

Grâce à l'insertion de "@", nous pouvons accéder facilement aux informations concernant les pauses, surtout ses emplacements et ses corrélations avec des chunks.

Pour repérer les motifs et les statistiques qui peuvent répondre à notre hypothèses, nous avons écrit un script python qui parcourt le texte, crée une liste de chunks, et ensuite grâce aux @ permet de voir entre (ou à l'intérieur de) quels chunks le locuteur produit les pauses ainsi que les étiquettes de ces chunks.

Ce script a été appliqué aux trois corpus (formulation, planification, révision) et construit à partir de chaque .csv un tableau de résultat.



ETAPE 1 : CONSTITUTION D'UNE LISTE DE CHUNKS

```
In [117]: import os
with open('bigram_chunks_complete_formulation_with pauses_2.txt') as f:
    t = f.read()
```

```
In [121]: #une fonction qui crée une liste de chunks
def chunker(text):
    #une chaine représentant un chunk, liste de chunks
    nCh = ""
    listDeChunkers = []
    for i in range(len(text)):
        #des conditions pour fermer le chunk, pour diviser les chunks d'une chaine de car /
        if text[i-1] == ")" and text[i] == " " or text[i-1] == ")" and text[i] == "\n" or te
            listDeChunkers.append(nCh)
            nCh = ""
        #condition nécessaire pour fermer le chunk precedent et ajouter un @ qui se trouve e
        elif text[i-1] == ")" and text[i] == "@" and text[i+1] == "(":
            if nCh!=")":
                listDeChunkers.append(nCh)
                listDeChunkers.append("@")
                nCh=""
            #condition nécessaire pour un @ qui se trouve entre les deux balises ouvrantes --> r
            elif text[i] == "@" and text[i-1] == "(" and text[i+1] == "(":
                listDeChunkers.append("@")
                nCh=""
        else:
            nCh+=text[i]
    return listDeChunkers

chunks = chunker(t)
print(chunks)
```


ETAPE 2 : NETTOYAGE D'UNE LISTE DE CHUNKS

```
In [122]: #fonction qui nous donne accès à l'étiquette du chunk, à la nature du chunk
#il faut lui donner en entre un chunk duquel nous voulons savoir l'étiquette
def prefix(ch):
    nch = ""
    for l in ch:
        if l == " ":
            break
        return nch
        if l == "(":
            pass
        else:
            nch+=l
    return nch

#l'enlèvement des PONCT qui empêchent l'analyse et des element vides comme chunk vide ()
chunksNettoyes=[]
for ele in chunks :
    if ele != "" and ele != "()" and "PONCT" not in ele :
        chunksNettoyes.append(ele)
print(chunksNettoyes)
```

```
[('(NP L'/DET arrê't/NC )', '(PP du/P+D tabac/NC )', '(VN est/)', '@', '(NP un/DET sujet/NC )', '(NP qui/PROREL )', '(VN est/V )', '(AdP de_plus_en_plus/ADV )', '(AdP fréquent/ADV )', '(NP un/DET sujet/NC )', '(NP qui/PROREL )', '(VN est/V )', '(AdP de_plus_en_plus/ADV )', '(AdP fréquent/)', '@', '(PP dans/P les/DET conversations/NC )', '(PP en/P France/NPP )', '(PP où/PROREL )', '(NP encore/ADV )', '(PP dans/P les/DET conversations/NC )', '(PP en/P France/NPP )', '(PP où/PROREL )', '(AdP encore/)', '@', '(AdP équent/ADV )', '(NP équent /@/NC de_plus_en_plus/ADV )', '(NP répandu/VPP )', '(AdP de_plus_en_plus/ADV )', '(VN répandu/)', '@', '(us_en_plus/ADV )', '(VN répandu/VPP )', '(PP en/P France/NPP )', '(NP L/NC )', '(PP en/P France/NPP )', '(NP L/@/DET es/NC différentes/ADJ )', '(NP es/DET différentes /@/ADJ e/NC )', '(NP L'/DET Etat/NPP )', '(NP e/NC )', '(NP L'/DET Etat/)', '@', '( Etat/NP P )', '(VN a/V )', '(VN a/)', '@', '(AdP même/ADV )', '(VN crée/VPP )', '(AdP même/ADV )', '(VN crée/)', '@', '(NP différentes/DET mesures/NC )', '(PP pour/P faire/VINF )', '(CONJ qu e/CS )', '(NP les/DET personnes/NC fumeuses/ADJ diminuees/NC )', '(NP la/DET cigarette/NC )', '(NP Il/CLS )', '(VN a/V )', '(AdP par_exemple/ADV )', '(VN interdi/VINF )', '(NP la/D
```

ETAPE 3.a : ANALYSE DES PAUSES ENTRE LES CHUNKS

```
In [125]: #ce bout de code parcourt les chunks et une fois il rencontre @ tout seule
#il repère l'étiquette de chunk avant et après sous la condition que l'étiquette
#est conforme au standard d'étiquetage, sinon c'est compté comme les données bruitées.
#les étiquettes possibles sont pris de siteweb de SEM.

étiquettes = ["NP", "PP", "AdP", "AP", "CONJ", "VN", "__UNKNOWN__"]
bruit=0
listeEntreChunks = []
#parcours d'une liste de chunks
for i in range (len(chunksNettoyes)-1) :
    #if élément d'une liste est un arobase et
    if chunksNettoyes[i] == "@" :
        #extractions d'étiquettes
        prefix1 = prefix(chunksNettoyes[i-1])
        prefix2 = prefix(chunksNettoyes[i+1])
        #si les deux étiquettes sont conforme au standard on les compte --> les données sûre
        if prefix1 in étiquettes and prefix2 in étiquettes :
            key = (prefix1,prefix2)
            listeEntreChunks.append(key)
            print('bon :', chunksNettoyes[i-1], chunksNettoyes[i], chunksNettoyes[i+1] )
        else:
            print('bruuuuuit :', chunksNettoyes[i-1], chunksNettoyes[i], chunksNettoyes[i+1])
            bruit+=1
```

```
bon : (NP brasserie/NC ) @ (NP brasserie/NC )
bon : (CONJ que/) @ (NP les/DET différentes/ADJ mesures/NC )
bruuuuuit : (AP toute/) @ (NCT
,/)
bruuuuuit : (NCT
,/) @ (NP des/DET fumeurs/NC )
bon : (NP des/DET fumeurs ) @ (AdP trop/ADV )
bon : (PP de/P fumeur/NC ) @ (PP en/P )
bon : (NP d/) @ (PP De/P nos/DET jours/NC )
bon : (AP présente/) @ (PP dans/P notre/DET société/NC )
```

ETAPE 3.b : ANALYSE DES PAUSES ENTRE LES CHUNKS

```
In [124]: #ce bout de code génère le dico avec des résultats d'analyse pour le cas où les pauses arrivent
dicoEntre = {}
for ele in listeEntreChunks :
    n=0
    for i in range(len(listeEntreChunks)):
        if ele == listeEntreChunks[i] :
            n+=1
    dicoEntre[ele] = n
dicoEntre['données bruitées'] = bruit
print(dicoEntre)
```

#pour la formulation :

*#1237 données de dico concaténées. Données bruitées c-à-d l'étiquette donnée par SEM n'est pas
#ou chunker a fait n'importe quoi, mais cela ne veut pas dire que les pauses ne sont pas ent
#dans la fenêtre avant on peut voir des cas des bruits*

```
{('VN', 'NP'): 89, ('AdP', 'PP'): 23, ('AdP', 'AdP'): 19, ('VN', 'AdP'): 30, ('AdP', 'NP'): 47, ('NP', 'VN'): 114, ('PP', 'AdP'): 24, ('CONJ', 'NP'): 44, ('CONJ', 'CONJ'): 2, ('NP', 'PP'): 95, ('PP', 'PP'): 92, ('PP', 'VN'): 66, ('NP', 'NP'): 130, ('NP', 'AdP'): 22, ('AP', 'PP'): 16, ('PP', 'CONJ'): 44, ('AdP', 'VN'): 13, ('PP', 'NP'): 136, ('NP', 'AP'): 3, ('AdP', 'AP'): 6, ('VN', 'VN'): 26, ('VN', 'PP'): 55, ('CONJ', 'AdP'): 8, ('CONJ', 'PP'): 6, ('NP', 'CONJ'): 35, ('AdP', 'CONJ'): 10, ('VN', 'AP'): 18, ('PP', 'AP'): 1, ('AP', 'NP'): 16, ('VN', 'CONJ'): 19, ('AP', 'CONJ'): 13, ('CONJ', 'VN'): 9, ('AP', 'AP'): 1, ('AP', 'VN'): 4, ('AP', 'AdP'): 3, 'données bruitées': 811}
```

ETAPE 4.a : ANALYSE DES PAUSES DANS LES CHUNKS

```
In [127]: #nettoyage de données brutes (les lettres séparés, chunks vides) et des espaces entre les c
#nettoyage des cas où la pause arrive avec la ponctuation ou entre les chunks et est étiquet
chunksComplex = []
for ele in chunks :
    if len(ele) > 7 :
        if "PONCT" not in ele :
            chunksComplex.append(ele)
```

```
#ce bout de code repère l'info dedans quel chunks il y a des pauses
#aussi le nombre des données difficiles à analyser
#il retourne la liste de chunks qui contient la pause dedans
étiquettes = ["NP", "PP", "AdP", "AP", "CONJ", "VN", "__UNKNOWN__"]
listeInsideChunk = []
bruit = 0
for ele in chunksComplex :
    if "@" in ele :
        nature = prefix(ele)
        if nature in étiquettes :
            print(nature, 'in chunk :', ele)
            listeInsideChunk.append(nature)
        else :
            print("données brutes :", ele)
            bruit+=1
```

```
PP in chunk : (PP pour/P diminuer/@/VINP )
PP in chunk : (PP en/@/P d/NC )
AdP in chunk : (AdP de_plus_@_en_plus/ADV )
PP in chunk : (PP de/P douze/DET ans/@/NC )
PP in chunk : (PP du/P+D prix/@/NC d/ADJ )
```

ETAPE 4.b : ANALYSE DES PAUSES DANS LES CHUNKS

```
dicoInside = {}  
for ele in listeInsideChunk :  
    n=0  
    for i in range(len(listeInsideChunk)):  
        if ele == listeInsideChunk[i] :  
            n+=1  
    dicoInside[ele] = n  
dicoInside['donné bruitées'] = bruit  
print(dicoInside)
```

```
#NP in chunk (NP les/@/DET t/NC )
```

```
#PP in chunk (PP de/P cigarette/NC neutre/@/ADJ s/NC )
```

```
{'NP': 369, 'VN': 64, 'PP': 329, 'AdP': 57, 'AP': 25, 'CONJ': 17, 'donné bruitées': 7}
```

TABLEAUX : FORMULATION

PAUSES AUX FRONTIÈRES

Nature	Nombre	Exemple
PP, NP	136	(PP depuis/P plusieurs/DET années/NC) @ (NP le/DET prix/NC)
NP, NP	130	(NP une/DET voiture/NC @ NP qui/PROREL)
NP, VN	114	(NP curité/NC routière/ADJ @ VN est/V)
NP, PP	95	(NP une/DET diminution/NC @ PP dans/P ; (NP terrasse/NC @ PP pour/P) fumeur/NC)
PP, PP	92	(PP le/P diminution/NC @ PP dans/P l'/DET achat/NC)
VN, NP	89	(VN crée/VPP @ NP différentes/DET mesures/NC)
PP, VN	66	(PP les/P-D paquets/NC @ VN profitent/V)
VN, PP	55	(VN est/V @ PP /P cause/NC)
AdP, NP	47	(AdP encore/ADV @ NP la/DET création/NC)
CNOJ, NP	44	(CONJ Mais/CC @ NP la/DET création/NC)
PP, CONJ	44	(PP le/P cigarettes/NC @ CONJ ou/CC)
NP, CONJ	35	(NP 150/DET ans/NC @ CONJ que/CS)
VN, AdP	30	(VN a/V @ AdP même/ADV)
VN, VN	26	(VN déplacer/VINF @ VN est/V)
PP, AdP	24	(PP jusqu' à/P devenir/VINF @ AdP quasiment/ADV)
AdP, PP	23	(AdP fréquent/ADV @ PP dans/P les/DET conversations/NC)
NP, AdP	22	(NP des/DET maladies/NC pulmonaires/ADJ) @ AdP uns/ADV)
VN, CONJ	19	(VN abstient/V @ CONJ quand/CS)
AdP, AdP	19	(AdP Seulement/ADV @ AdP dépendant/ADV)
VN, AP	18	(VN verait/V @ AP nng/ADJ)
AP, NP	16	(AP intéressant/ADJ @ NP la/DET sécurité/NC routière/ADJ)
AP, PP	16	(AP gratuit/ADJ @ PP aux/P-D étudiants/NC)
AdP, VN	13	(AdP en _effet/ADV @ VN diminuer/VINF)
AP, CONJ	13	(AP nolen/ADJ @ CONJ e/CC)
AdP, CONJ	10	(AdP vite/ADJ @ CONJ et/CC)
CONJ, VN	9	(CONJ que/CS @ VN éradiquer/VINF)
CONJ, AdP	8	(CONJ et/CS @ AdP dépendant/ADV)
AdP, AP	6	(AdP aussi/ADV @ AP indisponible/ADJ)
CONJ, PP	6	(CONJ car/CC @ PP le/P bus/NC)
AP, VN	4	(AP moindres/ADJ @ VN sont/V)
NP, AP	3	(NP a/DET population/NC @ AP es/DET plus/ADV pauvre/ADJ)
AP, AdP	3	(AP Hevé/ADJ @ AdP dépendant/ADV)
CONJ, CONJ	2	(CONJ ou/CC @ CONJ ou/CC)
PP, AP	1	(PP d'/P être/VINF @ AP respectueux/ADJ)
AP, AP	1	(AP oust/ADJ @ AP oust/ADJ)
Bruit	811	*(PP de/P tabac/NC @ P)

Total des bigrammes : 2050

PAUSES À L'INTÉRIEUR DES CHUNKS

Nature	Nombre	Exemple
NP	369	(NPL/@/DET es/NC différentes/ADJ)
PP	329	(PP à/@/P Dans/NPP un/DET)
VN	64	(VN il _@_ y _a/V)
AdP	57	(AdP à _@_ moins/ADV)
CONJ	17	(CONJ ou _même/@/CC qu'/_/CS)
AP	25	(AP cour@rante/ADJ)
Bruit	7	

Total des bigrammes : 868

TABLEAUX : PLANIFICATION

PAUSES AUX FRONTIÈRES

Nature	Nombre	Exemple
PP, NP	164	PPde/P médecine/NC traditionnelle/ADJ @ NP ce/DET centre/NC)
NP, NP	164	NPun/DET avantage/NC) @ NP sette/DET médecine/NC)
NP, VN	118	NPsa/DET disposition/NC) @ VN place/V)
NP, PP	118	NP l'/DET efficacité/NC) @ PP de/p cette/DET démarche/NC)
VN, NP	109	VN voient/V @ NP un/DET renouveau/NC)
PP, PP	103	PP dans/p le/DET reacteur/NC) @ PP des/p-D avions/NC)
PP, VN	47	PP en/p meilleure/ADJ santé/VPP) @ VN Réduire/VINF)
VN, PP	65	VN permettrait/V @ PP des/p-D gaz/NC)
AdP, NP	57	AdP de _plus/ADV @ NP elle/CS)
CONJ, NP	50	CONJ que/CS @ NP toutes/ADJ les/DET personnes/NC)
VN, VN	39	VN devra/V @ VN être/VINF mis/VPP)
AdP, VN	27	AdP également/ADV @ VN être/VINF)
PP, PP	22	AdP régulièrement/ADV @ PP de/p problème/NC)
AP, NP	21	AP fatale/ADJ @ NP la/DET médecine/NC alternative/ADJ)
AP, PP	14	AP proches/ADJ @ PP des/p-D villes/NC)
CONJ, VN	10	CONJ Mais/CC @ VN ne/ADV pensons/V pas/ADV)
AdP, AP	8	AdP plus/ADV @ AP rapide/ADJ)
VN, AP	5	VN sont/V @ AP nombreux/ADJ)
CONJ, PP	4	CONJ mais/CC @ PP dans/p l'/DET ensemble/NC tout/ADJ le/DET monde/NC)
PP, AdP	4	PP dans/p le/DET froid/NC) @ AdP de _plus/ADV)
NP, CONJ	3	NP sa/DET prescription/NC) @ CONJ et/CC)
NP, AdP	3	NP leur/DET réussite/NC scolaire/ADJ) @ AdP réellement/ADV)
AP, VN	2	AP fatale/ADJ @ NP la/DET médecine/NC alternative/ADJ)
VN, CONJ	2	VN prospérer/VINF @ CONJ et/CC)
PP, AP	2	PP sur/p le/DET long/ADJ terme/NC) @ AP négatifs/ADJ)
NP, AP	2	NP des/DET remèdes/@/NC naturels/ADJ) @ AP naturels/ADJ)
PP, CONJ	2	PP de/p fumer/VINF @ CONJ et/CC)
AP, CONJ	1	AP nécessités/ADJ @ CONJ et/CC)
CONJ, AP	1	CONJ que/CS @ AP certain/ADJ)
AdP, AdP	1	AdP etc/ADV @ AdP etc/ADV)
VN, AdP	1	VN ne/ADV fumant/VPR pas/@/ADV pas/ADV) @ AdP pas/ADV)
Bruit	1135	

Total des bigrammes : 2304

PAUSES À L'INTÉRIEUR DES CHUNKS

Nature	Nombre	Exemple
NP	434	NP la/DET mont/@/NC é/ADJ plus/ADV rapide/ADJ)
PP	372	PP en/@/P sa/DET disposition/NC)
VN	89	VN sont/@/V reconnu/VPP)
AdP	47	AdP Prem _@ _ièrement/ADV)
CONJ	25	CONJ Mais/CC que/@/CS qu' _est-ce _que/CS)
AP	24	AP peu@uplé/ADJ)
Bruit	13	

Total des bigrammes : 1004

TABLEAUX : RÉVISION

PAUSES AUX FRONTIÈRES

Nature	Nombre	Exemple
PP, NP	147	PP de/P titres/NC @ NP Cette/DET idée/NC
NP, NP	143	NP quelques/DET années/NC @ NP Cette/DET hausse/NC
NP, VN	134	NP es/DET fabriquants/NC @ VN mt/V décidé/VPP
VN, NP	119	VN lument/V @ NP des/DET cigarettes/NC
NP, PP	109	NP a/DET fumée/NC @ PP pour/P réduire/VINF
PP, PP	107	PP de/P morts/NC @ PP sur/P les/DET routes/NC
PP, VN	63	PP comme/P les/DET étudiants/NC @ VN ont/V
CONJ, NP	58	CONJ et/CC @ NP les/DET transports/NC
AdP, NP	55	AdP Encore/ADV @ NP la/DET mise/NC
VN, PP	54	VN permet/V @ PP aux/P-B non-fumeurs/NC
VN, VN	28	VN pourraient/V @ VN continuer/VINF
AdP, PP	24	AdP Ici/ADV @ PP de/P discuter/VINF
AdP, VN	23	AdP régulièrement/ADV @ VN diffuses/VPP
AP, PP	10	AP libre/ADJ @ PP de/P toutes/DET restrictions/NC
AP, NP	10	AP traité/ADJ @ NP les/DET bus/NC
CONJ, VN	9	CONJ et/CC @ VN ne/ADV mettrait/V pas/ADV
CONJ, PP	9	CONJ car/CC @ PP en/P les/DET exposant/NC
VN, AP	8	VN varaitre/VINF @ AP itopiste/ADJ
PP, AP	5	PP de/P problèmes/NC @ AP cher/ADJ
AP, VN	3	AP judicieux/ADJ @ VN demander/VINF
CONJ, AP	3	CONJ et/CC @ AP condamnable/ADJ
PP, CONJ	2	PP au_courant_des/P risques/NC encourus/ADJ @ CONJ pourquoi/CS
AdP, AdP	2	AdP notamment/ADV @ AdP notamment/ADV
AdP, AP	2	AdP exclus/ADV @ AP Nombreux/ADJ
AP, AP	2	AP social/ADJ @ AP social/ADJ
PP, AdP	2	PP en/P place/NC @ AdP en_place/ADV
VN, CONJ	2	VN recherché/VPP @ CONJ et/CC
AP, AdP	1	AP dangereux/ADJ @ AdP peut-être/ADV
NP, CONJ	1	NP a/DET sécurité/NC intérieure/ADJ @ CONJ et/CC
NP, AP	1	NP a/DET loi/NC @ AP la/DET plus/ADV récente/ADJ
VN, AdP	1	VN apparaît/VINF @ AdP Aujourd'hui/ADV
AP, CONJ	1	AP évastateur/ADJ @ CONJ mais/CC
NP, AdP	1	NP es/DET @ AdP C'est_pourquoi/ADV
Bruit	1114	

Total des bigrammes : 2253

PAUSES À L'INTÉRIEUR DES CHUNKS

Nature	Nombre	Exemple
NP	389	(NP leur/@/DET habitude/NC)
PP	363	(PP Malgré/@/P les/DET bonnes/ADJ volontés/NC)
VN	96	(VN pourr/@/V aient/VS être/VINF)
AdP	51	(AdP à_première_@_vue/ADV)
AP	25	(AP r/@/DET plus/ADV élevé/ADJ)
CONJ	17	(CONJ Mais/@/CC si/CS)
Bruit	22	

Total des bigrammes : 963

6. CONCLUSION

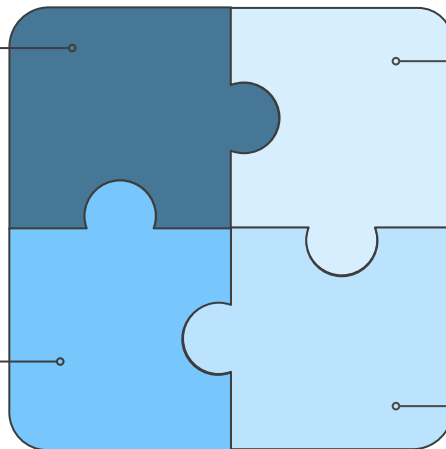
Question 1

Est-ce que les outils disponibles arrivent à bien segmenter nos données ?

Question 2

Hypthèse 1

Hypothèse 2





CONCLUSION QUESTION 1

Selon les résultats de notre évaluation, tous les outils disponibles n'ont pas réussi à segmenter nos données de manière optimale. SEM s'est avéré être l'outil le plus performant en termes de segmentation des données, avec des résultats prometteurs pour le POS tagging et le chunking.

Cependant, TreeTagger, SpaCy et NLTK ont présenté des limitations et des difficultés dans l'identification précise des limites des segments. Ces outils ont montré des résultats moins satisfaisants en termes de précision et de cohérence dans la segmentation des données.

6. CONCLUSION

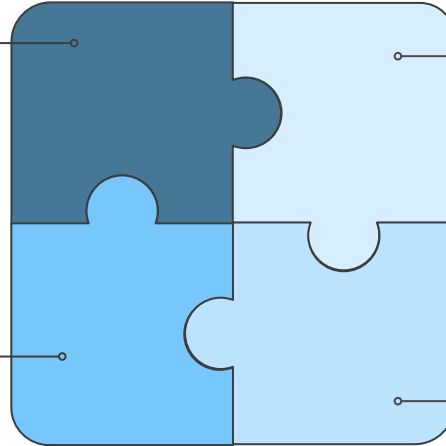
Question 1

Question 2

Hypthèse 1

Les pauses arrivent aux
frontières des chunks ?

Hypothèse 2





CONCLUSION HYPOTHÈSE 1



Selon le chunking effectué avec SEM sur les données des trois tableaux, nous avons constaté que sur un total de 6607 exemples, la pause se situait aux frontières des chunks, tandis que sur 2835 exemples, la pause était située à l'intérieur des chunks. Par conséquent, nous pouvons conclure que la plupart des pauses se trouvent effectivement aux frontières des chunks.

Statistiques	Tableau “formulation”	Tableau “planification”	Tableau “révision”
Pause entre chunks	2050	2304	2253
Pause à l'intérieur des chunks	868	1004	963

6. CONCLUSION

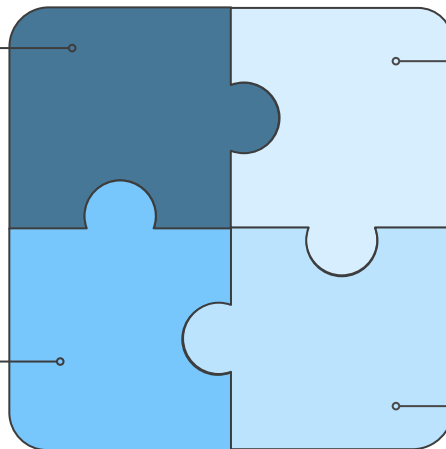
Question 1

Question 2

Si oui, est-ce que les pauses arrivent entre des chunkers particuliers ?

Hypthèse 1

Hypothèse 2





CONCLUSION QUESTION 2

Selon nos résultats expérimentaux, il est clair que la plupart des pauses se situent principalement entre les chunks PP (Chunk prépositionnel) et NP (Chunk nominal), NP et NP, NP et VN (Chunk verbal).

Nous pouvons également affirmer qu'elles sont souvent positionnées avant ou après les chunks NP.

Ces observations suggèrent que les pauses sont fréquemment associées à des points de transition entre différentes structures grammaticales. Elles marquent des pauses naturelles dans le flux de la phrase et peuvent indiquer des changements de sujet, des compléments d'information ou des séparations entre différentes entités nominales.

6. CONCLUSION

Question 1

Question 2

Hypthèse 1

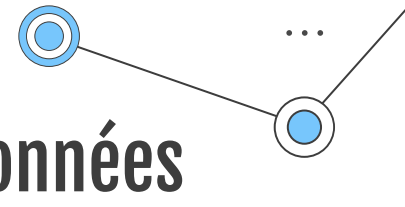
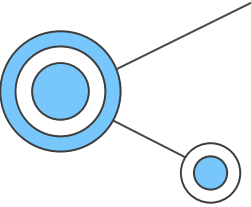
Hypothèse 2

Les différents processus de textualisation présentent les mêmes corrélation entre les pauses et les chunks.



CONCLUSION HYPOTHÈSE 2

Selon nos observations, les différents processus de textualisation présentent des corrélations similaires entre les pauses et les chunks. Qu'il s'agisse du processus de formulation, de planification ou de révision, nous avons constaté que les pauses se manifestent généralement aux frontières des chunks et que les pauses se situent principalement entre les chunks PP (Chunk prépositionnel) et NP (Chunk nominal), NP et NP, NP et VN (Chunk verbal).



Limites de la recherche – Exploiter des données

Traiter des données écrites en temps réel s'est révélé être une tâche plus compliquée que prévu. Nous avons dû faire face à un dilemme empirique qui nous a mis face à des choix importants. D'abord, la qualité de la division en chunks de la part de SEM dépendait aussi de la bonne formation des phrases qu'on lui passe en entrée ; nous ne disposons pas de phrases bien formées et son output a représenté aussi un challenge qui nous a obligé à post-traiter une grande partie des données en output. Cependant, reconstituer un texte bien formé dans le but d'obtenir une division en chunks plus correcte aurait impliqué la perte de certaines informations liée au processus d'écriture en temps réel, notamment la perte de l'indication temporelle d'intervention de la pause.

Dès lors, nous avons décidé de donner une importance majeure aux pauses et, par conséquent, nous n'avons pas voulu trop modifier les données pré-chunking dans le but de ne pas perdre l'indication temporelle concernant les pauses entre les phrases. De toute manière, ce qui nous intéressait était surtout de vérifier le statut syntaxique des chunks qui gravitent autour d'une pause, ce qui a comme contrepartie le risque de se retrouver avec des fautes de frappe segmentées en chunks de la part de SEM.

Finalement, bien que incomplets et demandant un traitement plus approfondi, les résultats indiquent une certaine tendance et répondent partiellement aux questions que nous nous sommes posées au début du projet.



Limites de la recherche – Extraire des statistiques

Nous avons travaillé sur les textes produits en temps réel. Vu la nature des productions, il était difficile de traiter les données et de reconstruire le corpus sans bruit.

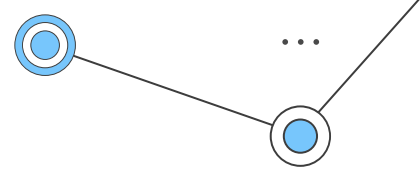
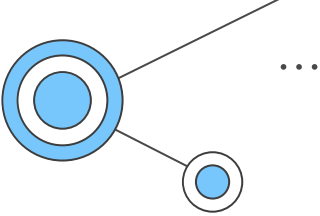
Parfois nous avons rencontré des lettres simples et des mots ou des phrases coupés. Cela empêchait le programme de diviser les chunks correctement mais également a rendu très difficile l'insertion des arobases.

Nous étions un peu limités et ne pouvions pas répondre aux hypothèses sans être équivoque. Les statistiques ont relevé de grandes masses des données bruitées desquelles, nous n'étions pas capables d'extraire des informations de façon claire ou automatique.

Nous avons mis l'accent sur les données plus ou moins "propres" plutôt que de bricoler dans le bruit de façon non contrôlée.

Exemples pour illustrer :

(PP d'/P autrui/NC) @ (P aujourd'_hui/ADV)
(CONJ que/) @ (ailleurs/NC)
(PP de/P l'/DET argent/NC) @ (C)
(NP s/) @ (H)



**Nous vous remercions pour l'attention que
vous nous avez porté.**