Gestion de corpus linguistiques : fusion de la Linguistique et du Traitement Automatique des Langues dans le projet PREFAB

Florine Hecquet

Gestion de Corpus Linguistiques : Fusion de la Linguistique et du Traitement Automatique des Langues dans le projet PREFAB

Florine
HECQUET

Sous la direction de Olivier KRAIF

Laboratoire : Laboratoire de Linguistique et didactique des langues étrangères et maternelles (LIDILEM)

UFR LLASIC
Département Sciences du Langage
Section Industries de la Langue

Mémoire de master 2 mention Industries de la Langue - 20 crédits
Parcours : Industries de la Langue, orientation professionnelle

Année universitaire 2022-2023

Gestion de Corpus Linguistiques : Fusion de la Linguistique et du Traitement Automatique des Langues dans le projet PREFAB

Florine
HECQUET

Sous la direction de Olivier KRAIF

Laboratoire : Laboratoire de Linguistique et didactique des langues étrangères et maternelles (LIDILEM)

UFR LLASIC
Département Sciences du Langage
Section Industries de la Langue

Mémoire de master 2 mention Industries de la Langue - 20 crédits
Parcours : Industries de la Langue, orientation professionnelle

Année universitaire 2022-2023

Remerciements

Tout d’abord, je souhaiterais remercier Olivier Kraif pour son offre de stage au sein du projet PREFAB. J’ai été touchée par la confiance qu’il a placée en moi en me proposant ce stage. Il a également été d’une aide précieuse tout au long de cette expérience, tant pour le guidage que pour le déblocage. Je n’aurais pas pu obtenir de tels résultats sans sa participation.

Ensuite, je pense à Agnès Tutin également pour la confiance qu’elle m’a accordée en me faisant participer à son projet, mais aussi pour sa disponibilité tout au long de mon stage. Grâce aux réunions régulières ainsi qu’aux informations qu’elle m’a transmises, j’ai pu mener à bien les différentes missions en étant accompagnée à chaque moment.

J’ai aussi une pensée pour Anja Smith et Yvon Keromnes avec qui j’ai eu l’occasion d’échanger lors de réunions à propos de l’avancée du projet, et qui m’ont permis de me sentir totalement intégrée dans ce dernier. Ils m’ont considérée comme un membre à part entière, se sont intéressés à mon travail, et cela m’a touchée.

Enfin, je voudrais remercier Xiao Ma. Mon associé, collègue et ami qui me supporte depuis 2 ans à l’université, que ce soit durant les cours, les projets ou les stages. Un partenaire aussi important qu’embêtant, mais présent à chaque étape de cette partie de ma vie, et un soutien moral indispensable.

J’en profite également pour faire un clin d'œil aux créateurs de ChatGPT (https://chat.openai.com/chat) dont l’IA conversationnelle a été sollicitée de temps en temps lors de mon stage pour permettre sa bonne réalisation.

DÉCLARATION ANTI-PLAGIAT

1. Ce travail est le fruit d'un travail personnel et constitue un document original.
2. Je sais que prétendre être l'auteur d'un travail écrit par une autre personne est une pratique sévèrement sanctionnée par la loi.
3. Personne d'autre que moi n'a le droit de faire valoir ce travail, en totalité ou en partie, comme le sien.
4. Les propos repris mot à mot à d'autres auteurs figurent entre guillemets (citations).
5. Les écrits sur lesquels je m'appuie dans ce mémoire sont systématiquement référencés selon un système de renvoi bibliographique clair et précis.

PRÉNOM : FLORINE

NOM : HECQUET

DATE : 15/04/2023

Sommaire

Introduction

Depuis plus de 35 ans, le laboratoire LIDILEM (Linguistique et Didactique des Langues Étrangères et Maternelles) innove grâce à son implication dans de nombreux projets et recherches dans le domaine des Sciences du Langage.

Du 06 mars 2023 au 07 juillet 2023 (4 mois), j'ai réalisé un stage au sein du laboratoire LIDILEM dans le cadre du projet PREFAB (Constructions des phrases préfabriquées dans les interactions langagières). Obtenu en acceptant la proposition faite par Olivier Kraif, l'un de mes tuteurs, je souhaitais participer à un projet de grande envergure autour de phénomènes linguistiques particuliers. Ce stage axé sur les phrases préfabriquées du français correspondait donc parfaitement à mes attentes.

Le laboratoire LIDILEM est rattaché à l'Université Grenoble Alpes. Il est né en 1987 de la fusion de six départements de recherche, à savoir :

- le Centre de Didactique des Langues (CDL)
- le Centre Écritures et Communications Écrites (ECOEC)
- le Centre de Linguistique Africaine
- le Centre de recherches sur le métalangage grammatical (Métagram)
- le Centre Interactions Verbales, Écritures, Lecture (IVEL)
- le Centre de Didactique du Français Langue Étrangère Grenoblois (CEDIFLEG)

Il est aujourd'hui spécialisé dans les Sciences du Langage, allant des descriptions linguistiques au traitement automatique des langues. Il possède 4 axes de recherches :

- Description et modélisation linguistiques, corpus, TAL
- Didactique des langues
- Acquisition du langage : multimodalité, variabilité et contexte
- Sociolinguistique : identités, cultures, interactions, usages

Mon stage a pris place dans le premier axe de recherche du laboratoire, encadré par Olivier Kraif et Agnès Tutin, deux enseignants chercheurs. Étudiante en 2ème année de Master Sciences du Langage parcours Industries de la Langue, mes missions au sein du projet concernaient uniquement l'aspect de traitement automatique des langues. Ce stage consistait en l'annotation et l'analyse de données linguistiques, la conversion de ces données dans différents formats, ainsi que la réalisation et l'analyse de l'alignement entre des romans en français et en allemand. Plus largement, lors de mon stage, j'ai eu l'occasion de me confronter aux enjeux d'un projet d'une grande ampleur, ainsi qu'aux différentes techniques actuelles. Au-delà

d’enrichir mes connaissances et mes compétences, ce stage m’a permis de mettre à contribution mes aptitudes et de découvrir le fonctionnement d’un projet de cette échelle.

Pour présenter mon expérience de stage au sein du laboratoire LIDILEM, je vais tout d’abord exposer le contexte dans lequel s’est déroulé mon stage, ainsi que les missions qui m’ont été confiées (Partie I). Puis, j’aborderai le travail concrètement effectué durant ces 4 mois, où seront détaillées chaque tâche, leur réalisation ainsi que les difficultés appréhendées (Partie II). Enfin, je conclurai par un bilan global sur la bonne ou mauvaise complétude des missions, les différentes perspectives proposées en conséquence, ainsi que les apports issus de cette expérience (Partie III).

Partie 1
-
Contexte et missions

Chapitre 1. Organisation

Tout d'abord, il est essentiel de faire un point sur l'organisation dans laquelle s'est déroulé mon stage, en abordant des thèmes comme l'organisation globale et la description des outils utilisés pour satisfaire les différentes demandes de mes tuteurs. Cette partie vise à présenter précisément les conditions d'exécution de ce stage.

1. Organisation générale

Concernant l'organisation générale de mon stage, je considère qu'il est important de s'attarder sur le projet auquel j'ai pris part, mais aussi sur le matériel utilisé lors des missions réalisées, sur les conditions de travail dont j'ai bénéficié, et enfin sur l'équipe participant au projet.

1.1. Description du projet

Le projet **PREFAB**[1] (Constructions des phrases préfabriquées dans les interactions langagières) est un projet piloté par le laboratoire **LIDILEM**[2] (Linguistique et Didactique des Langues Étrangères et Maternelles) associé à l'Université Grenoble Alpes, et plus précisément par Madame Agnès Tutin enseignant chercheur. Ce projet permet une collaboration entre les laboratoires suivants : l'UMR **ICAR**[3] (Interactions, Corpus, Apprentissages, Représentations) associée à l'Université Lumière Lyon 2, l'UMR **ATILF**[4] (Analyse et Traitement Informatique de la Langue Française) associée à l'Université de Lorraine, et l'UMR **BCL**[5] (Bases, Corpus, Langage) associée à l'Université Côte d'Azur, ayant chacun leur spécialité propre. Ainsi, ICAR est spécialisé dans l'étude des corpus oraux et des interactions, ATILF est le spécialiste des ressources linguistiques et des travaux de phraséologie pragmatique, et enfin BCL est expert dans l'exploration de corpus.

Ce projet a pour but de recenser et de modéliser les phrases préfabriquées des interactions les plus productives du français. Pour ce faire, une étude de corpus sera réalisée. L'objectif étant de décrire le fonctionnement syntaxique, sémantique, pragmatique,

[1] https://anr.fr/Projet-ANR-22-CE54-0013
[2] https://lidilem.univ-grenoble-alpes.fr/
[3] http://icar.cnrs.fr/
[4] https://www.atilf.fr/
[5] https://univ-cotedazur.fr/laboratoires/bases-corpus-langage-bcl

interactionnel, multimodal et prosodique de ces expressions préfabriquées, mais également de concevoir une modélisation de ces constructions.

La faisabilité de ce projet est liée à l'expérience et à la complémentarité des 4 équipes dans les domaines du traitement de la phraséologie pragmatique, des corpus oraux et écrits en linguistique outillée et des analyses syntaxiques et interactionnelles. Le projet s'appuie sur des corpus linguistiques disponibles, mais également sur les résultats des précédents projets réalisés, à savoir **ANR RHAPSODIE**[6], **ANR ORFEO**[7], **ANR SegCOR**[8], **ANR PHRASEOROM**[9], Wikiconflits et Wikidiscussions.

1.2. Le matériel

Dans le cadre de mon stage au sein du projet PREFAB, j'ai bénéficié de nombreux avantages. Tout d'abord, un ordinateur m'a été fourni par Olivier Kraif, l'un de mes tuteurs de stage. C'est avec ce dernier que j'ai réalisé la quasi-totalité des programmes, des exécutions de tests, ou encore des réunions régulières par **Zoom**[10]. Certaines réunions pouvaient se faire en présentiel en fonction des membres du projet concernés par ces dernières, mais dans la majorité des cas cela se faisait en visio-conférence avec l'équipe de l'ATILF de Nancy, composée de Anja Smith et de Yvon Keromnes.

Lors de ma formation, et également sur le plan personnel, je n'ai eu de contact qu'avec le système d'exploitation Windows. Or pour ce stage, j'ai travaillé exclusivement sur le système Linux. Il a donc fallu que je m'adapte rapidement aux différences de fonctionnement entre les 2 systèmes (téléchargement des outils, exécution des programmes etc). Je reviendrai plus en détails sur ce point dans le point « Système Linux ».

L'ensemble des outils ayant été utilisés dans le cadre de ce stage feront également l'objet d'une présentation détaillée dans le point suivant, à savoir la « Description des outils utilisés ». Ces derniers sont une dizaine, et possèdent chacun des fonctions et des objectifs différents.

[6] https://anr.fr/Projet-ANR-07-CORP-0030
[7] https://anr.fr/Projet-ANR-12-CORP-0005
[8] https://anr.fr/Projet-ANR-15-FRAL-0004
[9] https://phraseorom.univ-grenoble-alpes.fr/fr
[10] https://zoom.us/fr

1.3. L'autonomie

Au niveau de l'autonomie dont j'ai bénéficié lors de ce stage, j'ai eu le choix du télétravail ou du travail sur place dès le début. J'ai souhaité effectuer mes journées de travail sur place afin de conserver un cadre, et ainsi pouvoir maintenir une séparation entre le lieu de travail et le lieu d'habitation, synonyme de détente. Les premières semaines de stage, aucun emplacement spécifique n'était disponible pour me permettre de travailler. J'ai donc effectué ces premiers temps dans la salle informatique en libre-service du bâtiment K de Stendhal, à l'Université Grenoble Alpes. Par la suite, un accès à la salle I 110 du bâtiment Stendhal m'a été fourni.

Cependant, un mois après le début de mon stage, je me suis blessée, ce qui m'a contraint à passer entièrement en télétravail pendant les deux mois suivants. L'un des aspects que je considère comme étant important dans le télétravail est la confiance que mes tuteurs ont placée en moi. Grâce à cela, j'ai pu jouir d'une grande autonomie au niveau de mes horaires de travail, de mes conditions de travail, de mes apprentissages, tout en profitant de l'aide de mes tuteurs quand cela était nécessaire. Ces derniers ont toujours été joignables par mail et par téléphone lorsque l'occasion se présentait. Un point sur les bénéfices de cette autonomie sera accordé dans le point « Qualités personnelles ».

1.4. L'équipe

La plupart de mes tâches correspondent à des missions à effectuer seule, mais en récupérant des données précédentes. Ce projet est constitué de plusieurs tâches qui se suivent : mon intervention consiste à participer à quelques tâches précises au milieu du projet. Des tâches ont été effectuées avant mon arrivée, et d'autres seront réalisées après mon départ. Je n'aurais pas pu travailler au début de mon stage sans les données créées par Zhiyuan Qiu, la dernière stagiaire ayant travaillé sur ce projet environ un an avant mon arrivée. Le point « Avec une équipe » contient des précisions sur cette collaboration.

La récupération des fichiers de Zhiyuan Qiu s'est faite par le biais du serveur MIAI, d'où les données sont accessibles une fois que les identifiants permettant l'accès au serveur sécurisé sont créés. Une présentation du serveur est disponible dans le point « MIAI Serveur », et l'ensemble des traitements réalisés grâce à ces scripts est présenté dans le point « Annotation du discours direct ».

Ensuite, j'ai travaillé sur les romans français-allemand en collaboration avec l'équipe de l'Université de Lorraine. Je travaillais toujours seule sur des tâches précises, mais je devais partager mes données après traitement aux enseignants chercheurs de Nancy et également recevoir des données de leur part, afin de réaliser les programmes permettant d'analyser les romans français et allemands de la même manière. Mes connaissances en allemand n'étant pas suffisantes pour obtenir des corrections convenables au niveau de ces romans, la participation et la collaboration avec les équipes de l'ATILF étaient indispensables. De plus amples renseignements sont indiqués dans la partie « Chapitre 6. Corpus GLFA ».

2. Description des outils utilisés

Durant mon stage, j'ai eu besoin de multiples outils afin de mener à bien les missions qui m'ont été confiées. Parmi ces outils, certains sont des outils de traitement automatique des langues (TAL) aidant à l'analyse linguistique, d'autres sont simplement des assistants, ou des outils indispensables au bon fonctionnement de chacune des tâches à réaliser.

2.1. Outils de traitement automatique des langues

Dans la catégorie des outils de traitement automatique des langues dont j'ai eu besoin lors de mon stage, je pourrais citer 2 outils très performants et qui ont été bien utiles à ce projet. Ces outils sont Stanza et NooJ. Un point concernant le langage de programmation employé sera abordé, ainsi qu'une description d'un modèle de langage nécessaire à l'alignement.

2.1.1. Stanza

Stanza[11] est un outil de TAL permettant de faire une analyse linguistique à partir d'un texte brut. Il est décrit comme « *une collection d'outils précis et efficaces pour l'analyse linguistique de nombreuses langues humaines permettant le passage du texte brut à l'analyse syntaxique et à la reconnaissance d'entités* » par Qi et al. (2020). Il réalise plusieurs opérations indispensables à l'analyse linguistique, telles que la tokenisation, la lemmatisation et l'étiquetage morpho-syntaxique (cf. Figure 1).

La première étape est la tokenisation qui correspond au découpage du texte en entités plus petites, appelées tokens et se rapprochant des mots. L'étape suivante est celle de la lemmatisation, le fait d'associer une forme canonique à un token : par exemple, la forme conjuguée d'un verbe sera associée à l'infinitif de celui-ci, et dans le cas des noms le lemme

[11] https://stanfordnlp.github.io/stanza/

associé sera au singulier. L'étiquetage morpho-syntaxique occupe la troisième place et consiste à associer une catégorie syntaxique à un token, concrètement à décréter qu'une forme correspond soit à un verbe, soit à un nom etc. Cette étape est souvent liée à l'acronyme POS qui signifie Part of Speech en anglais, parties du discours en français. La quatrième étape est l'analyse en dépendances qui permet de définir le rôle de chaque token dans la phrase. Cette partie contribue à repérer l'élément tête de chaque phrase ainsi que les relations syntaxiques liant ce dernier aux autres tokens de la phrase. Elle est exprimée en nombres correspondant au nombre de l'élément auquel le token est directement relié. Enfin, la dernière fonctionnalité proposée par Stanza est la reconnaissance d'entités nommées. Cette étape consiste à repérer dans les textes des groupes spécifiques tels que des noms de personnes, d'entreprises, ou encore de lieux.

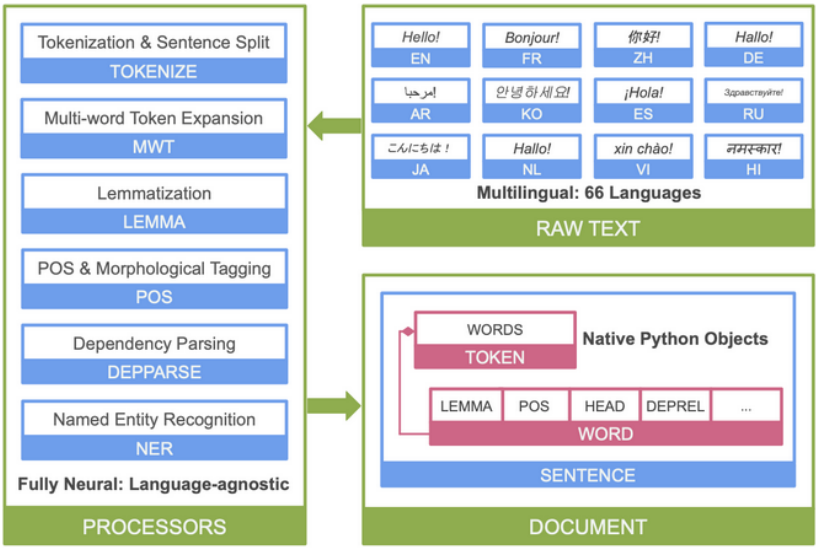


Figure 1. Schéma de fonctionnement de Stanza

Stanza est donc capable de gérer la tokenisation, la lemmatisation, l'étiquetage morpho-syntaxique en parties du discours (POS), l'analyse en dépendances et la reconnaissance d'entités nommées à partir d'un texte brut. Il fonctionne grâce à des modèles neuronaux pré-entraînés et est disponible en 66 langues humaines différentes (dont le français et l'allemand dont nous avons eu besoin ici). Il est étroitement lié au langage de programmation **Python**[12], que nous avons utilisé pour les scripts automatiques de ce projet et qui sera présenté ultérieurement, ainsi qu'au modèle UD (Universal Dependencies).

[12] https://www.python.org/

Le modèle de **dépendances universelles UD**[13] , dont le fonctionnement est développé dans Nivre et al. (2020), sert de base pour l'annotation des parties du discours, des dépendances syntaxiques, et des caractéristiques morphologiques. Il développe des annotations universelles dans le but de faciliter le développement d'analyseurs multilingues, l'apprentissage interlinguistique et la recherche sur l'analyse syntaxique du point de vue de la typologie des langues. Il souhaite fournir un inventaire universel de catégories et de lignes directrices pour faciliter l'annotation cohérente de constructions similaires à travers les langues, tout en permettant des extensions spécifiques à la langue si nécessaire. Il est actuellement disponible dans plus de 100 langues.

```
1   They    they    PRON    PRP    Case=Nom|Number=Plur              2   nsubj   2:nsubj|4:nsubj
2   buy     buy     VERB    VBP    Number=Plur|Person=3|Tense=Pres   0   root    0:root
3   and     and     CONJ    CC     _                                 4   cc      4:cc
4   sell    sell    VERB    VBP    Number=Plur|Person=3|Tense=Pres   2   conj    0:root|2:conj
5   books   book    NOUN    NNS    Number=Plur                       2   obj     2:obj|4:obj
6   .       .       PUNCT   .      _                                 2   punct   2:punct
```

Figure 2. Exemple d'analyse de Stanza utilisant le modèle UD

La figure 2 montre un exemple de sortie de Stanza où les colonnes contiennent chacune une information provenant des différentes analyses réalisées, à savoir (de gauche à droite) :

- la position du token dans la phrase (celle-ci contient 6 tokens)
- la forme du token telle qu'elle apparaît dans le texte
- le lemme du token correspondant à sa forme canonique
- la partie du discours ou catégorie syntaxique du token
- des informations supplémentaires et spécifique à la langue sur la partie du discours
- la liste des caractéristiques morphologiques du token
- le nombre correspondant au numéro de l'élément auquel se rattache le token (appelé la tête)
- la relation de dépendance du token avec l'élément racine de la phrase
- les dépendances supplémentaires spécifiques au token
- d'autres informations linguistiques ou supplémentaires sur le token (vide ici)

Ce format correspond au format Conll-U, format souhaité dans les fichiers après traitement. Le format Conll-U possède une syntaxe particulière équivalent à une ligne de mots contenant l'annotation d'un token dans 10 champs séparés par des caractères de tabulation simples, mais

[13] https://universaldependencies.org/

aussi à des lignes vides marquant les limites de la phrase, et enfin à des lignes de commentaires commençant par un dièse (#).

2.1.2. NooJ

NooJ[14] , dont le fonctionnement est développé dans Silberztein & Tutin (2005), est un outil de TAL similaire à Stanza, car il permet également de réaliser des analyses linguistiques à partir de texte brut, mais associées à des dictionnaires et des grammaires de son choix. En effet, lorsqu'on utilise Stanza, l'analyse est faite selon la langue sélectionnée sans possibilité de choisir un dictionnaire ou une grammaire sur lequel baser vos expériences. Or, dans le cas de NooJ, vous pouvez désigner un dictionnaire et une grammaire spécifique à partir desquels l'analyse sera effectuée. Cet outil est décrit par son créateur comme « *un environnement de développement linguistique open source permettant à la fois de formaliser des phénomènes linguistiques aux niveaux orthographique, lexical, morphologique, syntaxique et sémantique* » (Silberztein, M. (2002)), de faire des analyses syntaxiques de corpus, mais aussi de construire des applications de TAL.

Dans le cadre de ce projet, NooJ a été employé pour analyser les textes en français tirés du corpus de l'ATILF. Ces textes étaient au format brut, et l'utilisation de NooJ a servi à repérer les mots inconnus grâce à un dictionnaire de référence. Une analyse lexicale de la totalité des textes français composant le corpus de romans français-allemand a donc été réalisée grâce à cet outil. Mais de nombreuses autres fonctionnalités telles que l'analyse de discours, la désambiguïsation, ou encore la création de champs lexicaux sont également possibles grâce à NooJ.

2.1.3. Python

Python est un langage de programmation, ce n'est pas un outil au même titre que Stanza et NooJ mais il offre de nombreux avantages permettant de le considérer à un niveau similaire. En effet, tous les programmes créés dans le cadre de ce stage ont été faits en langage Python. Cela s'explique par plusieurs raisons : sa puissance, sa lisibilité et son caractère complet.

Au niveau de la syntaxe, les fonctions ou blocs sont facilement identifiés grâce à la présence d'indentations, ce qui permet de faciliter la lecture et d'épurer le code. De plus, Python possède de nombreux mots-clés permettant de créer des conditions (if/else), des fonctions (def)

[14] https://nooj.univ-fcomte.fr/

et des boucles (for/while). Ces derniers sont capables de simplifier et d'apporter une puissance au code. Il faut également citer les différents types d'objets proposés par Python, offrant la possibilité de pratiquer différentes opérations lors de l'exécution. Par exemple, la création d'un dictionnaire, l'utilisation de tableaux et de listes, ou encore la gestion d'un booléen (une variable ayant une valeur vraie ou fausse).

Enfin, le langage Python possède de nombreuses bibliothèques (aussi appelées librairies), et méthodes pouvant favoriser son usage lors de la programmation. Parmi les plus connues se trouvent les méthodes *read()*, *readlines()*, *write()*, *writelines()*, ou encore *append()*. Respectivement, ces méthodes permettent de lire un fichier, d'écrire dans un fichier, et d'ajouter un élément à une liste. Les bibliothèques (ou librairies), quant à elles, apportent des fonctionnalités supplémentaires utilisables dans les scripts. Dans le cadre de ce projet, le traitement des fichiers XML a nécessité l'import de bibliothèques de Python ainsi que l'emploi de plusieurs librairies, dont les fonctions seront développées dans la partie suivante.

2.1.4. Bert

BERT[15] est un modèle de langage créé par Google en 2018 dont l'acronyme signifie Bidirectional Encoder Representations from Transformers. Le fonctionnement de BERT est développé dans Devlin et al. (2019), et consiste à masquer aléatoirement des mots dans des textes puis de faire des prédictions (cf. Figure 3). La lecture et le masquage ainsi qu'une phase de prédiction correspondent à l'action d'un encodeur, tandis que le décodeur sert à générer du langage : c'est ainsi qu'agit un modèle de type Transformer. À l'inverse des modèles précédents qui parcouraient les textes uniquement de gauche à droite ou de droite à gauche, BERT le parcourt de manière bi-directionnelle. Cela lui permet de prendre en compte le contexte dans sa globalité et ainsi d'avoir une meilleure compréhension du texte, donc de meilleurs résultats.

[15] https://github.com/google-research/bert

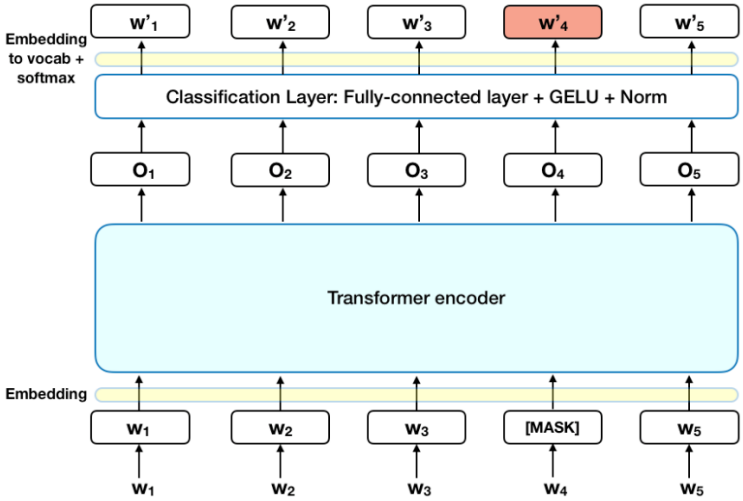


Figure 3. Schéma de fonctionnement de BERT

Ce modèle se sert d'un mécanisme d'attention pour apprendre les relations contextuelles entre les mots d'une phrase. Il fait partie des modèles d'apprentissage automatique basés sur des réseaux de neurones : en somme, c'est une intelligence artificielle qui s'appuie sur de l'apprentissage profond. Il fait des tâches répétitives pour apprendre de ses erreurs et est doté d'une mémoire, sous la forme de poids assignés aux connexions neuronales, pour retenir le contexte dans lequel il évolue. Dans le cadre de ce projet, il a été utile pour aligner les traductions du corpus français-allemand.

2.2. Assistants

Au niveau des outils correspondant à l'environnement de travail, que je qualifie ici d'assistants, je pourrais donner 3 dispositifs différents mais complémentaires dans le cadre de mon stage. Ces outils sont Geany, Regex101 et le tutoriel pour maîtriser les expressions régulières de Lucas Willems.

2.2.1. Geany

Geany[16] est une application téléchargeable gratuitement et accessible pour tous les systèmes d'exploitation. Elle est considérée comme « *un éditeur de texte de programmeur puissant, stable et léger qui fournit des tonnes de fonctionnalités utiles sans ralentir votre flux de travail. Il fonctionne sous Linux, Windows et macOS, est traduit dans plus de 40 langues et prend en charge plus de 50 langages de programmation* » par Wendling et al. (2005).

[16] https://www.geany.org/

Dans le cadre de ce stage, elle a été très utile pour la création des différents programmes développés en langage Python, mais aussi dans l'exécution de ces derniers. En effet, l'application possède une fonctionnalité d'exécution des scripts directement sur la machine. Cela permet de tester les programmes sur de petites données (une dizaine de textes par exemple) avant de les lancer sur la totalité des données. Je peux également citer la prise en charge de nombreux formats différents, allant du format de texte brut au format Conll-U en passant par le format XML, ce qui facilite grandement les comparaisons et le repérage des erreurs. De plus, Geany est dotée de fonctionnalités de base mais relativement importantes telles que la recherche et le remplacement, pouvant prendre en compte des expressions régulières (cet aspect est explicité dans le point suivant). Ces fonctions ont été utilisées à de multiples reprises lors de mon stage.

Dans l'annexe 1, on pourra observer que différents formats de fichiers sont présents dans l'éditeur, mais également que celui-ci possède une liste des variables et des imports initialisés dans le programme sur la gauche de l'écran. La partie inférieure de l'écran renvoie les messages de l'éditeur en fonction des requêtes exécutées, ce sera à cet endroit que les résultats des recherches effectuées avec la fonctionnalité de recherche simple ou bien celle de recherche et remplacement seront affichés (cf. Annexe 2). De plus, l'exécution du programme sur la machine peut se faire en cliquant sur le bouton représentant 2 écrous gris, ce qui aura pour effet d'ouvrir la console de l'ordinateur. Cette dernière transmettra alors les éventuels erreurs ou commentaires renvoyés par le script (cf. Annexe 3).

2.2.2. Regex101

Le second outil que je considère comme étant un assistant très précieux est **Regex101**[17], une console d'expressions régulières accessible en ligne qui permet de déboguer les expressions en temps réel. Une expression régulière, souvent appelée « regex », est une modélisation d'une chaîne de caractères pouvant être présentes dans un texte. Elle répond à une syntaxe précise, et permet de rechercher plusieurs chaînes de caractères dans un texte grâce à une seule expression. Cela permet de gagner en efficacité au niveau des programmes devant gérer des remplacements automatiques par exemple.

Une expression régulière a principalement une fonction de repérage, mais une fois la chaîne de caractères identifiée, plusieurs options s'offrent à l'utilisateur. Premièrement, celui-

[17] https://regex101.com/

ci peut choisir de supprimer le contenu sélectionné par son expression régulière. Pour ce faire, il doit initialiser un remplacement du contenu par une chaîne de caractères vide (cf. Annexe 4). Deuxième possibilité, celle du remplacement. Dans ce cas, l'utilisateur peut définir n'importe quel caractère ou chaîne de caractères à intégrer à la place du contenu de base.

Une option également puissante dans le cas des remplacements gérés par des regex est celle de la capture. Cette méthode consiste à conserver une partie des résultats de l'expression pour pouvoir la réinjecter lors du remplacement (cf. Annexe 5). Cela permet de gérer des cas particuliers de manière efficace.

Le mode de fonctionnement de Regex101 est simple : il suffit de construire une expression régulière et de constater son efficacité sur un texte entré par l'utilisateur. L'outil indique alors clairement les erreurs contenues dans l'expression régulière, mais aussi la pertinence de cette dernière. En effet, si l'expression ne correspond pas à la recherche initialement prévue ou bien au contenu du texte entré, elle ne réagira simplement pas. Dans le domaine des regex, la correspondance entre l'expression et le texte est appelée un *match*. Lorsque la regex ne donne aucun résultat (donc aucun *match*), une étape de débogage s'impose afin de déterminer si l'expression contient des erreurs ou bien si le texte associé ne possède simplement pas de chaîne de caractère liée à cette expression.

De plus, l'outil possède une partie explicative sur le côté droit de l'écran permettant à la fois de comprendre ce que recherche l'expression, mais aussi de déboguer cette dernière lorsque le *match* ne correspond pas à ce qui est souhaité ou lorsqu'aucun *match* n'est observé. L'encart proposant les différents *matchs* relevés par l'expression est également d'une grande aide lors des vérifications de bon fonctionnement des regex utilisées.

2.2.3. Tutoriel Regex

Pour finir avec les outils assistants de ce projet, je dois citer le **tutoriel pour maîtriser les expressions régulières**[18] de Lucas Willems. Ce site contient énormément d'informations sur la syntaxe si particulière des expressions régulières, mais également sur leur structure et les éléments de base les composant. Au départ, cette syntaxe n'est pas intuitive et ne paraît pas simple, mais elle n'évolue pas.

[18] https://www.lucaswillems.com/fr/articles/25/tutoriel-pour-maitriser-les-expressions-regulieres

Concrètement, il faut apprendre la manière de représenter les caractères, les nombres, ou encore les espaces pour pouvoir effectuer des recherches précises et efficaces. Mais une fois ces notions intégrées, elles ne se modifient pas et sont universelles. C'est-à-dire qu'une même expression régulière peut tout aussi bien fonctionner sur un texte en français et sur un texte en allemand (à condition de prendre en compte les caractères spéciaux et accentués présents dans chacune des langues). La pratique et les différentes situations nécessitant l'utilisation de regex lors de mon stage m'ont permis de me familiariser petit à petit avec cette syntaxe, et ainsi de me détacher de ce tutoriel.

2.3. Autres outils

Enfin, il me semble indispensable de présenter 3 outils essentiels ayant grandement facilité mes tâches lors de ce stage. Ces outils se nomment Cisco AnyConnect, FileZilla et MIAI Serveur.

2.3.1. Cisco AnyConnect

Tout d'abord il faut présenter **Cisco AnyConnect** [19], une application permettant une connexion sécurisée à un réseau défini. Certaines connexions sont considérées comme sécurisées car elles autorisent uniquement les appareils remplissant des conditions particulières à accéder au réseau. C'est le cas des 2 autres outils que je vais présenter ultérieurement. Afin de parvenir à utiliser ces outils, il faut impérativement se connecter avec un VPN reconnu par le réseau (cf. Annexe 6). Le rôle de Cisco AnyConnect est donc de transformer l'adresse IP de n'importe quel appareil en la redirigeant vers le VPN de l'université dans mon cas. Je pouvais ainsi accéder au serveur à distance et travailler sur n'importe quelle machine.

2.3.2. FileZilla

Une fois connecté au VPN de l'université, il est possible d'accéder à un outil obligatoire au bon fonctionnement de mon stage : il s'agit de **FileZilla**[20]. FileZilla est un logiciel libre capable de gérer les processus FTP (file transfer protocol) et SFTP (secure file transfer protocol). Il existe 2 versions actuellement : la version client (dont je me suis servi) permettant l'accès au serveur de votre choix et les échanges de données avec celui-ci, et la version serveur offrant un serveur open source et gratuit.

[19] https://www.cisco.com/c/fr_be/products/security/anyconnect-secure-mobility-client/index.html
[20] https://filezilla-project.org/index.php

Dans le cadre de mon stage, cet outil m'a permis de me connecter au serveur afin de récupérer les données ayant déjà fait l'objet de modification dans le cadre du projet, mais également d'échanger des données entre le serveur et la machine (cf. Annexe 7). Ces échanges servaient à tester les programmes, vérifier leur efficacité, mais aussi faire du débogage. Il était indispensable d'accéder au serveur, mais il était tout aussi important de travailler en local sur la machine selon les tâches. L'utilisation d'un outil tel que FileZilla était donc primordiale.

2.3.3. MIAI Serveur

Le dernier outil incontournable pour garantir le bon déroulement de mon stage est le serveur **MIAI Server**[21] de l'**institut MIAI Grenoble Alpes**[22] (Multidisciplinary Institute in Artificial Intelligence). Ce dernier « *vise à conduire des recherches au plus haut niveau en intelligence artificielle* », mais aussi à « *informer et interagir avec les citoyen.ne.s sur tous les aspects de l'IA* » selon ses concepteurs, en axant ses recherches sur 2 thèmes principaux qui sont l'IA du futur et l'IA pour l'humain et l'environnement. Le serveur, auquel j'ai eu accès grâce à Cisco AnyConnect et FileZilla, m'a permis de stocker les scripts et les fichiers en ligne plutôt que sur l'ordinateur. En effet, ces derniers étant nombreux et lourds, il était impossible de conserver toutes les données sur le disque dur de la machine. Il m'a également permis de récupérer les données précédemment créées afin de poursuivre le travail déjà commencé avant mon arrivée.

De plus, l'exécution des programmes sur le serveur était bien plus rapide que l'exécution sur le terminal de la machine. Les données étant souvent bien trop volumineuses pour la machine, seul le serveur était capable de toutes les traiter en même temps. Afin de procéder au lancement des scripts sur le terminal du serveur MIAI, il fallait utiliser un protocole SSH permettant la connexion sécurisée entre la machine locale et le serveur. Une fois cet accès créé, la navigation dans le serveur se fait de la même manière que sur la machine, c'est-à-dire en respectant l'arborescence et les chemins d'accès des fichiers.

Autre avantage du serveur, l'exécution des programmes se faisait à distance, ce qui signifie que ces derniers pouvaient tourner la nuit avec l'ordinateur déconnecté du serveur et éteint. Il n'était pas nécessaire de le laisser allumé et de contrôler que les scripts s'effectuaient bien, contrairement aux programmes qui tournent en local. Il était également possible d'utiliser

[21] miai-server.u-ga.fr

[22] https://miai.univ-grenoble-alpes.fr/institut-miai/

différentes « sessions » disponibles sur le serveur grâce à la commande « tmux attach -t » suivi du numéro de la salle désirée (cf. Annexe 8). De cette manière, plusieurs scripts pouvaient être exécutés en parallèle dans plusieurs sessions différentes, ce qui offre un gain de temps considérable.

L’organisation de mon stage et les descriptions des outils étant plus claires à présent, il est temps de passer au chapitre suivant présentant les missions qui m’ont été confiées pendant ces 4 mois.

Chapitre 2. Missions demandées

Les missions qui m'ont été confiées durant ce stage ont été multiples, mais toutes reliées à l'analyse et au traitement de corpus car je n'ai travaillé qu'avec des corpus de romans. Celles-ci peuvent se répartir selon moi en 4 points qui sont la prise en main des corpus, la prise en main des outils, les corrections diverses et l'implication dans le projet.

1. Prise en main des corpus et des outils

La première étape a donc été la familiarisation avec les corpus. En effet, avant toute création de programme modifiant les données, il faut d'une part s'intéresser aux métadonnées et d'autre part observer la construction des fichiers.

1.1. Description des corpus

Les corpus qui m'ont été fournis en début de stage faisaient partie du corpus **Phraseobase**[23]. Ce dernier est composé de romans appartenant à 6 genres différents répartis en 6 catégories. Ces catégories regroupent des romans de science-fiction, des romans policiers, des romans historiques, des romans de fantaisie, des romans sentimentaux et des romans de littérature générale. Une partie du corpus avait déjà été annotée par Zhiyuan Qiu lors d'un stage précédent, les parties de discours direct avaient donc été repérées auparavant. Mais une seconde partie du corpus n'avait pas été traitée par manque de temps, il a alors fallu réaliser l'annotation des parties de discours direct avant de pouvoir effectuer les modifications souhaitées (celles-ci sont explicitées dans la partie « Partie 2 - Réalisation concrète »). L'annexe 9 présente l'organisation du corpus Phraseobase sur le serveur.

Concernant les romans français-allemand de l'ATILF, ces derniers proviennent du **GLFA**[24] (Groupe de Lexicographie Franco-Allemande). Les métadonnées et la construction de fichiers étaient également à prendre en compte avant toute manipulation. Ce corpus est composé de 2 catégories différentes, elles-mêmes contenant 2 sous-catégories. Les catégories principales sont les romans en français et les romans en allemand. Les fichiers rassemblés dans chacun de ces groupes ont été classés en fonction de la langue source du roman. À l'intérieur de chaque classe, les romans en langue source ainsi que leur traduction en langue cible sont présents, ce qui correspond aux 2 sous-catégories. Cette organisation est représentée sur l'annexe 10.

[23] http://phraseotext.univ-grenoble-alpes.fr/phraseobase/index.html
[24] https://www.atilf.fr/recherche/equipes/lexique/lexicographie-franco-allemande/

Concrètement, la catégorie des romans en français contient à la fois les romans en français et leur traduction en allemand, de même pour la catégorie des romans en allemand. Il faut néanmoins souligner que les romans en français langue source sont bien plus nombreux que les romans en allemand langue source. En effet, les romans en français langue source sont au nombre de 205, tandis que les romans en allemand langue source ne sont que 35. Cela n'empêche en rien les comparaisons, ni la fidélité des résultats observés, mais peut expliquer certaines différences ou lacunes au niveau des conclusions tirées.

1.2. XML to XML ConLL

Mon intervention sur ces 2 corpus consistait à modifier le format XML (Extensible Markup Language) présent au départ en format XML ConLL, tout en annotant les parties de discours direct présentes dans les fichiers. Le format de départ était donc le format XML qui comportait de nombreuses annotations sous forme de balises (cf. Annexe 11).

L'acronyme XML signifie « langage de balisage extensible » en français. Comme son nom l'indique, il est formé de balises encadrées par des chevrons, et a pour but de faciliter l'échange de contenus entre différents systèmes. Il se veut interopérable et possède une structure stricte : dès lors qu'une erreur de syntaxe existe dans un fichier XML, celui-ci devient inutilisable. Bien souvent, les documents en XML suivent un schéma structurel défini appelé DTD (Document Type Definition) limitant l'apparition d'erreurs de syntaxe. Sophie Lavignasse (2005) explique le rôle de la DTD qui définit « *la structure d'un document, les éléments et attributs qui y sont autorisés, et le type de contenu ou d'attribut permis. La DTD va alors valider le document en vérifiant qu'il se conforme strictement aux règles établies par la DTD à laquelle il fait référence.* ». De plus, le format XML utilisé ici est étroitement lié à la **TEI**[25] (Text Encoding Initiative) qui décrit la structuration du document grâce à un langage de balisage. Cette dernière permet de rechercher des informations sur chaque balise existant dans un document XML, telles que les balises pouvant contenir la balise recherchée, ou bien les attributs pouvant être associés à cette balise, ainsi que des exemples d'utilisation.

Le roman (qu'il provienne du corpus Phraseobase ou du corpus français-allemand de l'ATILF) était composé de paragraphes notés <p>, eux-mêmes composés de phrases notées <s>. Chacune des phrases était composée de balises <tc> contenant des balises <t> comprenant toutes les analyses des tokens. Des balises <dc> contenant des balises <d> étaient ensuite

[25] https://tei-c.org/

présentes, contenant des informations supplémentaires sur chaque token ainsi que le nombre de la tête du token.

La version finale de présentation des romans doit être au format XML ConLL, qui est un format également structuré mais néanmoins différent du format XML précédemment décrit. En effet, celui-ci comporte beaucoup moins de balises qu'à l'origine : seules les balises des paragraphes <p> et celles des phrases <s> sont conservées (cf. Annexe 12). Chaque phrase accueille l'analyse faite par Stanza, qui elle est au format .conllu d'où le nom du format. Ce type de format est considéré comme plus lisible et plus pratique que le format XML précédent, c'est pourquoi le changement de format était souhaité.

Plusieurs étapes sont nécessaires pour arriver à ce résultat, notamment des étapes de conversion, de correction et d'analyse. Ces différentes phases sont développées dans la partie « Partie 2 - Réalisation concrète ».

1.3. Prise en main des outils

Le premier outil de TAL avec lequel je suis entrée en contact lors de ce stage a été Stanza. Pour pouvoir l'utiliser, il faut tout d'abord l'installer sur la machine grâce à la commande « pip install stanza » dans le terminal de l'ordinateur. Une fois Stanza installé, il faut également importer le module au début du programme qui va en avoir besoin, avec la ligne « import stanza ». Il faut ensuite définir la langue des textes à analyser avec Stanza, dans notre cas le français ou l'allemand, puis construire un pipeline (des règles à appliquer lors de l'exécution de l'analyse). L'étape suivante consiste à lancer l'analyse tout en contrôlant le procédé et la sortie : l'analyse est expliquée plus en détails dans la partie « Partie 2 - Réalisation concrète ».

Le second outil à apprivoiser était NooJ, permettant le repérage des mots inconnus. Cet outil a demandé une adaptation plus importante que Stanza, car il ne suffisait pas seulement de faire appel au module en décrétant la langue des textes avant de lancer l'analyse. En effet, la version de NooJ utilisée dans ce projet possède sa propre interface et ses propres fonctionnalités, et n'était donc pas compatible avec un programme. Afin de l'utiliser de manière efficace, plusieurs fichiers de grammaire sont nécessaires ainsi qu'un dictionnaire. Ces derniers régissent l'analyse qui est développée dans la partie « Partie 2 - Réalisation concrète ». Il existe

néanmoins une version compatible avec des lignes de commandes développée par **Thomas Baier**[26], mais nous ne l'avons pas expérimentée ici.

Concernant l'alignement français-allemand, l'outil le plus important a été BERT. En effet, l'alignement était basé en grande partie sur ce modèle de langage via un script d'alignement nommé *AIlign* et conçu par Olivier Kraif. Ce dernier procède en deux étapes, à savoir l'extraction de points d'ancrage et le calcul du meilleur chemin d'alignement avec un algorithme de type Dynamic Time Warping. Je n'ai pas eu à apprivoiser le modèle de façon similaire à NooJ, mais plutôt comme Stanza. En effet, pour pouvoir utiliser NooJ, il a tout d'abord fallu comprendre son fonctionnement afin d'en tirer le meilleur rendu possible. Pour Stanza et BERT, le procédé est différent : il suffit d'importer les modules pour pouvoir les utiliser. Aucun contrôle sur l'organisation des analyses n'est possible, les seules actions réalisables se font au niveau des sorties produites. Il est néanmoins possible d'ajuster les paramètres du script d'alignement pour influencer les résultats.

Associé à BERT, le module *Matplotlib* de Python a également son importance au niveau de la vérification de l'alignement. En effet, c'est sur cette fonctionnalité que se base le jugement de l'alignement, et que des améliorations ou ajustements des paramètres sont envisagés. Il agit de la même manière que Stanza, c'est-à-dire qu'il doit être importé en début de script pour pouvoir être utilisé. Le fonctionnement de ce module, ainsi que les résultats obtenus, sont également développés dans la partie « Partie 2 - Réalisation concrète ».

2. Corrections diverses

Ce point propose de présenter brièvement les corrections réalisées lors de ce stage, qui seront bien sûr développées plus en détails dans la partie « Partie 2 - Réalisation concrète ». Ces corrections concernent globalement les sorties des outils et les erreurs repérées dès le début dans les fichiers XML. Comme cela a été indiqué dans le point « XML to XML ConLL », une erreur de syntaxe dans un fichier XML perturbe totalement l'exécution du programme.

2.1. Corrections des sorties des outils

L'une des modifications importantes réalisée est celle concernant les sorties des analyses de Stanza. En effet, cette partie s'est révélée assez riche en erreurs de lemmatisation. Il a donc fallu gérer des comparaisons entre les fichiers de sortie de Stanza et un dictionnaire

[26] https://github.com/numblr/nooj-cmd

afin de remplacer les lemmes inconnus ou incorrects, mais également des erreurs que je qualifierais d'erreurs de mise en page. Ces corrections seront abordées plus en détails dans la partie « Partie 2 - Réalisation concrète ».

Les corrections suivantes ne concernent que le corpus des romans français-allemand de l'ATILF. Un passage par le nettoyage des fichiers a été incontournable car le procédé utilisé pour la récupération des romans est celui de la reconnaissance optique de caractères (OCR). Ce dernier consiste à récupérer le contenu écrit d'un texte imprimé afin de le sauvegarder dans un fichier lisible sur un support numérique. Le premier logiciel d'OCR appelé « omni-font OCR » a été créé par Ray Kurzweil, un ingénieur américain, dans les années 1970[27]. Concrètement, les textes au format papier ont été scannés puis enregistrés en format numérique grâce à l'OCR. C'est à partir du format numérique que les traitements informatiques peuvent avoir lieu.

Malgré l'efficacité de l'océrisation (fait d'utiliser le principe de l'OCR), de nombreuses erreurs subsistent. De ce fait, certains éléments présents sur les pages des romans en version papier ont été conservés sur les versions numériques, bien qu'ils ne soient pas nécessaires. Des numéros de pages, de paragraphes ou encore des titres apparaissaient alors au milieu des textes. Une suppression de ces éléments est apparue essentielle pour l'analyse et la comparaison des textes.

L'OCR a engendré d'autres types de problèmes que l'apparition d'éléments non désirés dans les textes. En effet, des erreurs dites systématiques ou récurrentes ont été découvertes. Ces erreurs concernent des mauvaises orthographes ou bien des reconnaissances de caractères faussées (les « I » sont souvent reconnus comme des « 1 »). Afin de limiter les conséquences de ce problème, il a fallu corriger ces erreurs grâce à un script de remplacement automatique et une sorte de dictionnaire. Une grande partie a pu être corrigée ainsi, mais la base de la correction était la sûreté : si la forme présente était trop ambiguë pour retrouver la forme d'origine, aucun changement n'était effectué. Un point sur l'OCR sera abordé dans la partie « Partie 2 - Réalisation concrète ».

2.2. Corrections des XML et des métadonnées

Afin de réaliser les modifications sur les différents fichiers, une première correction sur le format XML a été nécessaire. En effet, les premières tentatives de modification par le biais

[27] https://www.pitneybowes.com/content/dam/pitneybowes/uk/en/shipping-and-mailing/e-invoicing/Blog_E-invoicing-The_Brief_History_of_OCR.pdf?itid=lk_inline_enhanced-template

de scripts automatiques ont renvoyé énormément d'erreurs de syntaxe empêchant tout fonctionnement du programme. Une vérification des erreurs a donc été indispensable avant de réaliser une quelconque modification automatique. Des précisions sur cet aspect seront disponibles dans la partie « Partie 2 - Réalisation concrète ».

Enfin, une correction au niveau des métadonnées a été opérée afin d'harmoniser ces dernières. En effet, les noms des auteurs n'apparaissaient pas de la même manière dans le fichier : parfois le nom était devant le prénom, parfois l'inverse. La forme choisie fut « Nom, Prénom » pour tous les fichiers.

3. Implication dans le projet

Certaines de mes missions ne sont pas directement reliées à l'analyse de corpus, mais possèdent une place toute aussi importante dans la réalisation de mon stage. Ce sont des tâches d'investissement au sein du projet. Elles sont regroupées ici en tâches concernant l'avancement du projet, et en vulgarisation des actions effectuées.

3.1. Avancement

La première de ces tâches consiste à faire des comptes-rendus réguliers aux membres de l'équipe, et à mes tuteurs. Cette étape est essentielle pour plusieurs raisons. Premièrement, les participants au projet peuvent ainsi savoir comment avancent les phases, et envisager la suite des évènements. Deuxièmement, cela me permettait de synthétiser mon travail, mes difficultés et de dater plus ou moins précisément la durée nécessaire à chaque phase. Cette partie m'a également permis de formuler mes activités de manière à ce que des personnes non expertes dans ce domaine puissent comprendre ce qui se faisait.

Une autre tâche qui m'était demandée était la participation aux réunions d'équipe, en visio-conférence ou en présentiel. De cette façon, je me tenais informée des autres étapes du projet, de l'avancée globale, et des missions à venir. Je pouvais aussi expliquer mon travail lors de ces réunions, je me sentais ainsi impliquée dans le projet et j'y avais une place importante.

3.2. Vulgarisation

La partie qui m'a semblé la plus difficile mais aussi la plus importante, puisque ce projet est un travail d'équipe, est celle de l'explication des termes et des notions évidentes pour moi, autrement dit la vulgarisation des termes spécifiques. En effet, la plupart des membres de

l’équipe ne sont pas des experts du traitement automatique des langues, ni des outils utilisés, encore moins du vocabulaire.

Lors des réunions, mes tuteurs étant présents, j’avais l’habitude d’utiliser des termes spécifiques à ma formation tout d’abord, car je faisais un compte-rendu de mon avancée. Mais les membres de l’équipe de l’ATILF posaient énormément de questions après mon intervention pour obtenir des précisions et comprendre ce que je venais d’annoncer. Je devais alors reformuler mes propos pour essayer de clarifier mes tâches. Petit à petit, j’ai intériorisé le fait de devoir réfléchir en amont à la manière de faire le point sur la situation actuelle pour éviter de perdre du temps en explications peu claires. Malgré cela, des questions pouvaient être posées, et je m’efforçais d’y répondre en évitant les notions trop caractéristiques du TAL.

Les missions ainsi annoncées, un point sur le sujet global du projet va être abordé dans le chapitre suivant.

Chapitre 3. Pourquoi ce sujet de recherche ?

Le laboratoire LIDILEM est spécialisé dans la linguistique et la didactique des langues, il est donc logique que celui-ci s'intéresse de près à des phénomènes tels que le fonctionnement et le rôle des phrases préfabriquées dans la langue. Cette partie du mémoire est inspirée de la description du projet qui m'a été présentée en amont de mon stage afin de me transmettre les informations nécessaires à ma participation.

1. Les phrases préfabriquées

Les phrases préfabriquées sont le centre du projet PREFAB, il est donc essentiel de s'attarder sur ce que sont ces phrases préfabriquées, quel est leur rôle dans l'interaction et quelles études ont déjà été réalisées sur ce thème-là.

1.1. Qu'est-ce qu'une phrase préfabriquée ?

Les interactions orales et écrites ont fait l'objet d'études linguistiques approfondies ces dernières années, mais les aspects lexicaux liés à la préfabrication n'ont pas été autant observés que la syntaxe ou la linguistique interactionnelle (Traverso (2018), cité par Tutin 2021). Pourtant, les interactions présentent des phénomènes phraséologiques spécifiques tels que les phrases préfabriquées. Ces dernières sont définies par les auteurs du projet comme des énoncés « prêts à l'emploi » dans des contextes spécifiques avec une lexicalisation contrainte.

Quelques exemples de ces phrases préfabriquées :

- *ça marche*
- *tu plaisantes*
- *tu peux le dire…*

Ces phrases sont essentielles à maîtriser lors d'une interaction, ce qui renforce l'importance de la compréhension de leur fonctionnement et de leur construction selon les chercheurs engagés dans ce projet. Pourtant, malgré l'intérêt de plusieurs chercheurs dans ce domaine, il existe peu d'études faisant l'objet d'un inventaire ou d'une modélisation de ce phénomène. Cependant, des études dans le domaine des phrases préfabriquées ont bel et bien eu lieu.

1.2. Différentes études sur le sujet

Des modélisations prenant en compte les aspects sémantiques, syntaxiques et pragmatiques pour un sous-ensemble de ces expressions ont déjà été proposées par Blanco

Escoda & Mejri (2018) cité par Tutin 2021. Ces derniers ont fourni une description détaillée des « pragmatèmes », qui sont définis comme des énoncés restreints dans leur signifié par la situation de communication dans lesquels ils sont produits (par exemple, « après vous »).

De plus, les chercheurs de l'ATILF ont fait une étude sur des énoncés appelés « actes de langage stéréotypés » tels que « la belle affaire » ou encore « tu parles », présentant un degré de figement plus important que celui des « pragmatèmes » et étant associés à des fonctions pragmatiques telles que la menace, le refus ou encore l'étonnement (Kauffer (2019), cité par Tutin 2021).

Il existe aussi une étude sur les « formules expressives de la conversation » en italien, français et polonais, réalisée dans le cadre du projet **PRAGMALEX**[28] par le laboratoire LIDILEM et l'Université de Cracovie (Krzyżanowska et al. (2021), cité par Tutin 2021).

Des travaux ont également été réalisés au niveau des grammaires de constructions en ayant pour but de modéliser l'ensemble des paramètres qui entrent en jeu, c'est-à-dire le phénomène multidimensionnel, ainsi que le type de construction propre aux interactions orales (Imo (2015), cité par Tutin 2021). Les études basées sur cette approche, et plus particulièrement sur les « constructicons », offrent une modélisation globale des phénomènes phraséologiques (Lyngfelt et al. (2018), cité par Tutin 2021), ce qui, selon les chercheurs en charge de ce projet, peut permettre de décrire deux caractéristiques essentielles des CPP (construction de phrases préfabriquées) :

- les régularités observées au sein d'une construction de phrases préfabriquées, comme dans l'exemple « c'est pas vrai/possible/croyable »
- l'aspect multidimensionnel de ces constructions, qui nécessitent de multiples paramètres linguistiques

À ce jour, la phraséologie traditionnelle traite peu les paramètres à analyser dans le cadre de la construction des phrases préfabriquées, car ces derniers relèvent plutôt de la phraséologie pragmatique. Les dimensions interactionnelle ainsi que multimodale (gestes) sont donc déterminantes dans ce domaine.

[28] https://www.umcs.pl/fr/descriptif-du-projet,15298.htm

2. *Objectifs du projet*

L'objectif global de ce projet est d'expliquer le fonctionnement des phrases préfabriquées les plus productives du français. Les participants au projet ont tout d'abord formulé une hypothèse de départ, puis ont mis en place un plan en 3 étapes pour affirmer leur raisonnement.

2.1. Hypothèse de départ

L'hypothèse formulée par les chercheurs est que les phrases préfabriquées fonctionnent selon un paradigme limité de constructions lié à des fonctions paradigmatiques et interactionnelles. Ces dernières sont modélisables grâce à des approches comme les « constructicons » (Lynfgelt et al. (2018), cité par Tutin 2021).

De plus, il semble que les PPI (phrases préfabriquées des interactions) répondent à des schémas d'enchaînement réponses-redondances pouvant expliquer leur fonctionnement interactionnel et discursif, et que leur organisation forme un paramètre essentiel pour catégoriser les sous-genres discursifs et interactionnels.

Pour parvenir à modéliser le fonctionnement de ces phrases préfabriquées, ce projet se compose de 3 objectifs principaux présentés dans le point suivant.

2.2. Différentes étapes

Premièrement, il faut identifier et caractériser les phrases préfabriquées et les constructions les plus fréquentes à l'aide d'une large étude de corpus. Parmi ces différents corpus, il y a :

- des corpus oraux d'interactions tirés des corpus **ORFEO-CEFC**[29] et **ESLO2**[30]
- des corpus écrits d'interactions tirés de Wikiconflits et de Wikidiscussions
- des corpus romanesques contemporains issus du projet ANR PHRASEOROM et du corpus GLFA de l'ATILF qui ont été annotés syntaxiquement.

L'étape suivante consiste à analyser les paramètres linguistiques en considérant les aspects syntaxiques, sémantiques et pragmatiques. Pour un sous-ensemble, les paramètres interactionnels et multimodaux seront également analysés. À la suite de cette étude, une

[29] https://repository.ortolang.fr/api/content/cefc-orfeo/11/documentation/site-orfeo/home/index.html
[30] http://eslo.huma-num.fr/index.php/pagecorpus/pagepresentationcorpus

modélisation des constructions inspirée du modèle des « constructicons » sera réalisée avec une perspective contrastive français-allemand pour un sous-ensemble d'éléments.

La dernière étape est de déterminer l'influence du genre discursif et de la fonction pragmatique dans les patrons constructionnels. L'hypothèse d'un noyau commun pour un large sous-ensemble d'expressions préfabriquées, et de divergences formelles et fonctionnelles entre les types de corpus ou les activités interactionnelles est envisagée. La variation sera observée :

- entre les interactions écrites et les interactions orales
- entre l'oral romanesque représenté et l'oral authentique
- entre le français et l'allemand pour un sous-ensemble de constructions.

3. Apports du projet

Grâce aux compétences des différents laboratoires participant à cette étude, les instigateurs du projet PREFAB visent à proposer une approche novatrice basée sur plusieurs points développés ci-après.

3.1. Globalité, inventaire et fonctionnement

Premièrement, à l'inverse des projets précédents, l'équipe traitera un large ensemble de phrases préfabriquées des interactions afin de considérer ce phénomène dans sa globalité, tout en accordant de l'importance aux fonctions linguistiques de phrases considérées comme banales mais centrales dans la langue.

Différents types de phrases seront incluses dans cet ensemble :

- des phrases à fonction expressive comme « il manquerait plus que ça »
- des phrases à fonction phatique comme « tu vois »
- des phrases à fonction métalinguistique comme « comment dirais-je »
- des « pragmatèmes » comme « je vous en prie »

Deuxièmement, la création d'un inventaire des phrases préfabriquées n'est pas le seul but scientifique de ce projet, la compréhension du fonctionnement de celles-ci fait également partie de la mission. Pour cela, les participants au projet proposent de mettre en œuvre une approche utilisant les dimensions syntaxique, sémantique, pragmatique et interactionnelle. De plus, les approches des 4 équipes doivent permettre de créer un modèle original et innovant, adapté aux 3 types d'interactions, à savoir les interactions orales, écrites et représentées.

Selon les chercheurs de ce projet, le modèle est intéressant lors d'une analyse linguistique pour plusieurs raisons. Tout d'abord, il permet d'expliquer le fonctionnement des phrases préfabriquées, qu'elles soient lexicalisées comme dans la phrase « ça m'en bouche un coin », ou bien générales. Même si ce dernier a surtout été exploité en syntaxe, l'intégration de la dimension pragmatique est pertinente selon l'équipe.

L'équipe considère que le niveau de traitement de base choisi pour le projet, celui des CPP semi-abstraites, correspondant à des constructions de type <tu Verbe ?>, est préférable à celui des phrases préfabriquées entièrement lexicalisées. Selon elle, ce niveau de traitement permet de mettre à jour les régularités du fonctionnement linguistique dans les différents genres abordés. L'aboutissement du projet offrira une ressource pour le français composée d'une centaine de CPP correspondant aux 1000 PPI les plus productives ayant été analysées et traitées, ce qui en fera la première ressource de ce type pour le français.

3.2. Nouvelles approches et variation linguistique

Ensuite, sur le plan méthodologique, le projet propose une expérimentation des approches nouvelles de linguistique outillée utilisées dans le traitement des patrons constructionnels. Cette expérimentation se fera en testant l'adaptation de la méthode ALR (« arbres lexico-syntaxiques récurrents », qui sont des sortes de segments répétés utilisant la syntaxe de dépendance et des mesures d'association (Kraif (2019), cité par Tutin 2021)) à des corpus non standards, tels que les corpus oraux et les interactions dans Wikipédia. Jusqu'à présent, la méthode des ARL n'a été appliquée qu'à l'écrit à partir de corpus arborés. Le traitement réalisé sera unifié pour les 3 types de corpus.

Enfin, les chercheurs de ce projet souhaitent accorder une place essentielle à la question de la variation linguistique des phrases préfabriquées dans ce projet. En effet, plusieurs comparaisons sont envisagées, à savoir la comparaison des types d'interactions orales basées sur le nombre de locuteurs, sur le présentiel vs distanciel, sur la situation professionnelle vs privée, sur le contexte et sur les sous-genres. Mais également la comparaison de l'oral représenté dans les sous-genres romanesques, ou encore la mise en contraste des interactions orales et écrites atteste de l'importance de l'observation de la variation. De plus, la comparaison français-allemand réalisée en fin de projet n'a pas encore été étudiée sous cet angle avec ce couple de langues.

Nous allons maintenant décrire avec plus de détails les corpus et outils disponibles au démarrage du projet.

Chapitre 4. L'existant

Comme avancé précédemment, le projet se base sur l'analyse de la phraséologie des interactions du français, car peu d'études existent sur ce sujet. Il existe cependant plusieurs travaux sur le sujet en linguistique allemande (Meliss et al. (2019), cité par Tutin 2021) ainsi qu'en linguistique anglaise (Aston (2015), cité par Tutin 2021). En ce qui concerne le français, il existe le projet ANR PHRASEOROM (Novakova & Siepmann (2020), cité par Tutin 2021), mais ce dernier n'aborde pas les expressions spécifiques aux interactions. D'autres travaux basés sur la lexicographie serviront de base pour la modélisation et l'analyse des phrases préfabriquées de ce projet : ceux des laboratoires LIDILEM (Krzyżanowska et al. (2021), cité par Tutin 2021) et ATILF (Kauffer (2019), cité par Tutin 2021).

1. Les corpus écrits

Dans le cadre de ce stage, j'ai été amenée à travailler principalement sur des corpus écrits. Ces derniers proviennent d'anciens projets réalisés précédemment que nous avons récupérés, dans le but de mener une nouvelle étude sur les phrases préfabriquées. Différents corpus ont donc été réutilisés afin de disposer de genres et de types d'interactions multiples, pour pouvoir effectuer des comparaisons par la suite.

1.1. Les corpus Wiki

Les interactions écrites ont très peu été analysées par rapport aux interactions orales, ce projet propose donc de se pencher sur ce type d'interactions dans le but de proposer une vue d'ensemble sur les CPP. Selon Herring (2010) cité par Tutin 2021, le développement du web a généré de nouvelles formes de communications et d'interactions écrites qui oscillent entre l'aspect conversationnel (oral) et l'aspect textuel (écrit).

L'équipe de ce projet considère qu'il est pertinent d'observer les préfabrications dans les interactions écrites et d'effectuer une comparaison avec les interactions orales. Les corpus textuels de cette étude sont tirés de Wikidiscussions et de Wikiconflits, qui sont des interactions entre les rédacteurs de Wikipédia. C'est un corpus libre de droits, riche en échanges, qui ne nécessite pas d'analyse ou d'annotation au préalable, et qui a déjà fait l'objet d'une étude en 2019 par Poudat et Ho-Dac (cité par Tutin 2021). Ces derniers ont analysé le discours polémique sur le plan pragmatique.

1.2. Les corpus romanesques

Ce projet intègre, en plus des interactions orales et écrites, un troisième type de genre discursif composé de dialogues issus de corpus romanesques contemporains. Ces romans sont tirés du corpus Phraseobase issu du projet ANR PHRASEOROM (Novakova & Siepmann (2020), cité par Tutin 2021) et permettent de disposer d'un volume de données important.

De plus, selon les protagonistes du projet, il est intéressant de se pencher sur le cas de l'oral représenté dans la fiction car il constitue un objet d'étude pour la linguistique de l'oral et les études stylistiques (Rouayrenc (2010), cité par Tutin 2021). Enfin, l'étude des dialogues dans les romans donnera lieu à une comparaison avec un corpus bilingue français-allemand, le GLFA de l'ATILF. Cette comparaison permettra d'élargir la réflexion sur l'étude contrastive des CPP d'une part, et d'approfondir les études existantes dans le cadre de la phraséologie et des grammaires de constructions d'autre part.

1.3. Analyse des corpus

Les travaux précédemment effectués dans le domaine de la phraséologie ont été basés en grande partie sur des corpus en raison de l'usage des phénomènes à observer (Novakova & Siepmann (2020), cité par Tutin 2021). Selon les chercheurs engagés dans ce projet, cette utilisation permet entre autres d'affiner les paramètres d'étude comme la topologie ou encore la structure syntaxique. Ainsi, la méthode d'extraction des phrases préfabriquées utilisera l'approche des « arbres lexico-syntaxiques récurrents » (ALR). Cette méthode a déjà été exploitée dans le projet ANR PHRASEOROM.

De plus, la méthode des ALR, technique à base de corpus arborés (Kraif & Tutin (2016), cité par Tutin 2021), sera utilisée pour repérer les expressions polylexicales. Couplée à cette technique, l'exploitation de ressources existantes de listes de phrases préfabriquées structurées syntaxiquement permettra une identification optimale des phrases préfabriquées. En effet, l'équipe du projet pense que l'utilisation de ressources lexicales phraséologiques est indispensable pour ce type d'étude. Les extractions réalisées serviront à repérer les CPP, puis seront analysées pour effectuer des comparaisons.

2. Les outils

Comme développé dans le point 2 de la partie « Organisation », à savoir « Description des outils utilisés », les outils Stanza et Nooj existaient déjà et se sont révélés très utiles dans le cadre de mes missions. De même pour le langage de programmation Python.

2.1. Stanza

L'outil Stanza a été créé par le **Stanford NLP Group**[31], un groupe d'étudiants, de professeurs, d'ingénieurs de recherche et de post-doctorants. Leur domaine de prédilection est plutôt vaste, mais lié au traitement automatique des langues en général. En effet, ces derniers correspondent à la recherche scientifique fondamentale sur la linguistique informatique, l'apprentissage automatique, les applications pratiques de la technologie du langage humain et les travaux interdisciplinaires en sciences sociales informatiques et en sciences cognitives, et leur travail est centré sur des algorithmes permettant aux ordinateurs de traiter, de générer et de comprendre les langues humaines. Ils développent de nombreux outils associés au TAL, dont Stanza. L'acronyme NLP, signifiant Natural Language Processing, est l'équivalent du Traitement Automatique des Langues (TAL) en français.

Des travaux récents ont été réalisés avec Stanza, notamment ceux de Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton et Christopher D. Manning. (2020) qui présentent le fonctionnement et appliquent des comparaisons entre Stanza et d'autres modèles existants, en l'occurrence **FLAIR**[32] et **spaCy**[33]. Le modèle FLAIR a été présenté dans les travaux de Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter et Rolan Vollgraf (2019) qui présentent le fonctionnement du modèle. Concernant le modèle spaCy, les travaux de Mark Neumann, Daniel King, Iz Beltagy et Waleed Ammar (2019) visent à expliquer le fonctionnement de spaCy, mais également à l'améliorer en créant une version plus performante. Il existe également le modèle **NLTK**[34] ayant des fonctions similaires à Stanza et spaCy, présenté par Steven Bird, Ewan Klein, Edward Loper et Jason Baldridge (2008).

2.2. NooJ

L'outil NooJ a été conçu par Max Silberztein, un professeur de linguistique, de linguistique computationnelle et d'informatique, exerçant à l'université de Franche-Comté. Il est également l'auteur d'un second outil de traitement automatique des langues appelé **INTEX**[35], considéré comme la version obsolète de NooJ. Cette version comprenait des dictionnaires et des grammaires, pouvait analyser des textes de plusieurs millions de mots en temps réel, et était capable de créer des ressources lexicales, ainsi que des grammaires

[31] https://nlp.stanford.edu/
[32] https://github.com/flairNLP/flair
[33] https://spacy.io/
[34] https://www.nltk.org/
[35] http://intex.univ-fcomte.fr/

morphologiques et syntaxiques. Les dictionnaires et les grammaires étaient appliqués aux textes afin de localiser les modèles morphologiques, lexicaux et syntaxiques, de lever les ambiguïtés et d'étiqueter les mots simples et composés. Selon Max Silberztein, NooJ est plus performant que INTEX car il possède plus de 20 fonctionnalités linguistiques, informatiques et statistiques qui lui donnent une valeur technique et scientifique objectivement supérieure à celle de son prédécesseur.

Max Silberztein a également publié un document en 2005 pour présenter son nouvel outil, ainsi qu'un article sur la manière de formaliser les langues grâce à NooJ en 2015. Puis d'autres études se basant sur l'emploi de l'outil sont venues renforcer ses différentes utilisations possibles. Parmi ces études, il y a celle de Andrea Rodrigo, Mario Monteleone et Silvia Reyes (2018) qui se servent de NooJ pour l'apprentissage des langues.

2.3. Python

Le langage de programmation Python a été créé par Guido van Rossum, un développeur informatique néerlandais, en 1991. Plusieurs versions ont été développées depuis, la plus récente est la version 3.11.4 datant du 7 juin 2023. La version utilisée pour ce projet est la version 3.10.6. L'avantage de ce langage est qu'il fonctionne sur tous les systèmes informatiques, de Windows à macOS en passant par Linux. Il se veut interopérable et a pour but d'optimiser la productivité des programmeurs. De plus, il s'est adapté à d'autres langages comme Java, C, ou C# : ce qui donne respectivement Jython, CPython et IronPython. Ces implémentations visent à permettre une utilisation du langage Python sur des plateformes ou des machines virtuelles non compatibles.

De nombreuses études basées sur les fonctionnalités offertes par Python ont déjà eu lieu, notamment celle de Ossama Obeid, Nasser Zalmout, Salam Khalifa, Dima Taji, Mai Oudah, Bashar Alhafni, Go Inoue, Fadhl Eryani, Alexander Erdmann et Nizar Habash (2020) basée sur la langue arabe, ou encore celle de Jonathan Verner et Anna Vernerová (2020) basée sur la reconnaissance automatique de lexique.

Maintenant que le contexte et les missions de ce stage sont globalement plus explicites, le moment est venu de rentrer dans la partie concrète de mon stage.

Partie 2
-
Réalisation concrète

Chapitre 5. Corpus Phraseobase

En premier lieu, les tâches qui m'ont été demandées concernaient le corpus Phraseobase appartenant au projet ANR PHRASEOROM (Novakova & Siepmann (2020), cité par Tutin 2021). Ma mission consistait à annoter les romans composant le corpus, à analyser ces derniers grâce à l'outil Stanza présenté précédemment, et également à convertir les fichiers du format XML au format XML ConLL. Le corpus Phraseobase était disponible dans les répertoires de Zhiyuan Qiu et était scindé en 2 parties : les romans déjà annotés et les romans non annotés. Ces types de romans ont donc nécessité un traitement différent. La totalité du corpus traité équivaut à 648 fichiers.

1. Romans annotés

Les romans dits annotés sont des romans qui ont déjà été traités par Zhiyuan Qiu lors d'un précédent stage. Le corpus a été annoté avec l'outil XIP (Aït et al. (2000) cité par Roux 2004), dont l'acronyme signifie « Xerox Incremental Parser », et basé sur le XML. Il faut le réannoter avec Stanza tout en conservant le découpage en phrases. Les fichiers auxquels j'ai eu accès pour commencer étaient au format XML et possédaient déjà les annotations des passages de discours direct au niveau des tokens. Les passages considérés comme du discours direct correspondent à des discussions par exemple, ou à des citations entre guillemets. Cette catégorie de romans concerne 411 fichiers dont 47 romans de science-fiction, 81 romans policiers, 37 romans historiques, 42 romans de fantaisie, 38 romans sentimentaux et 166 romans de littérature générale.

1.1. Première gestion des fichiers XML

Avant toute chose, en sachant que l'analyse avec Stanza se fait à partir d'un texte brut, il est clair que la récupération des fichiers XML seule ne suffira pas. Une gestion des fichiers de départ est envisagée dès le début, notamment une étape de conversion du format .xml au format .txt.

1.1.1. Conversion

Les passages de discours direct ayant déjà été traités auparavant, ma première tâche a été de convertir le format .xml en format .txt. Pour ce faire, un programme de conversion a été créé, permettant de lire le contenu des fichiers XML et de récupérer uniquement le contenu des

balises. Ce script se nomme *conversion_A.py*. En fonction des balises rencontrées, l'action à effectuer est différente :

- si la balise rencontrée correspond à <t>, le contenu de la balise est récupéré et ajouté au texte de sortie
- si la balise rencontrée correspond à <dc>, un retour à la ligne est ajouté au texte de sortie
- les espaces sont gérés en fonction de l'attribut « e » de la balise <t>

La lecture des fichiers XML a été faisable grâce à la bibliothèque *ElementTree* proposée par Python. En effet, bien qu'il soit possible de lire un fichier XML avec la fonction *read()*, il est nécessaire d'utiliser un parser capable de lire à la fois les balises et leur contenu. Ainsi, une recherche des balises présentes dans le fichier est envisageable, et des actions à effectuer en fonction du résultat de la recherche sont abordables. Ce qui a été nécessaire pour le premier programme de ce stage, et faisable grâce à la commande « import xml.etree.ElementTree ».

De plus, afin de gagner du temps lors de l'exécution des scripts, étant donné que chaque dossier contenait quelques quarantaines de fichiers, la librairie *os* de Python a été utilisée avec la commande « import os ». Celle-ci permet de traiter des fichiers de plusieurs dossiers en exécutant le programme une seule fois, il suffit de fournir les différents chemins d'accès dans une variable. Ainsi, le programme cherchera chaque fichier dans chaque dossier renseigné et ne s'arrêtera qu'une fois le dernier fichier du dernier dossier traité. Cela se fait grâce à la commande *os.walk()*.

Cette librairie a également servi à créer les nouveaux dossiers où ont été enregistrés les fichiers sortants de chacun des programmes, de même il a suffi de renseigner le chemin d'accès et le nom du nouveau dossier à créer avant d'y enregistrer le nouveau fichier. La commande correspondant à la création du nouveau dossier est *os.makedirs()*, et celle créant le nom du nouveau fichier est *os.path.join()*. De cette façon, aucun fichier n'a été supprimé ou écrasé à la suite d'une exécution de script, ce qui évite la perte des fichiers dans le cas où le programme comportait des erreurs trop importantes. Ainsi, lorsqu'un script devait être modifié puis relancé, il était toujours possible d'utiliser la version précédente des textes.

Cependant, l'exécution du script de conversion n'a pas fonctionné au départ en raison d'erreurs dans les fichiers XML. En effet, comme précisé dans le point « XML to XML ConLL », lorsqu'une erreur est détectée par un programme dans un fichier XML celui-ci devient inutilisable. Il a donc fallu reprendre tous les fichiers XML du corpus des romans annotés et vérifier leur syntaxe. La plupart du temps, le problème provenait de balises ouvertes

mais pas fermées. L'étape de correction des fichiers a été longue et fastidieuse, mais nécessaire pour parvenir à apporter des modifications au corpus. Un moyen plus rapide et efficace de repérer les erreurs de syntaxe présentes dans les fichiers XML a été trouvé plus tard : il sera présenté dans la seconde partie concernant les romans non annotés.

1.1.2. Nettoyage

Une fois que les fichiers XML ont été corrigés, le programme de conversion a créé les fichiers au format .txt indispensables pour l'analyse avec Stanza. Néanmoins, une modification des textes obtenus après la conversion s'est révélée essentielle. En effet, de nombreuses erreurs présentes dans les fichiers XML ont été retranscrites dans les textes. Il s'agissait par exemple de « \n » symbolisant un retour à la ligne et apparaissant tel quel dans le texte, ou encore de mots coupés en deux, de *underscores* indésirables, d'espaces multiples, et pour finir de *underscores* à la place des tirets présents dans des dialogues.

Afin de nettoyer ces textes avant de les analyser, un deuxième programme, se nommant *text_suppr_n_A.py*, a été créé. Ce dernier a également utilisé la librairie *os* de Python pour apporter des modifications à tous les fichiers en un seul lancement du script et créer les emplacements des nouveaux fichiers, mais il a aussi utilisé la librairie *re* de Python. Cette dernière a été importée grâce à la commande « import re » et permet de recourir à des expressions régulières qui, comme avancé dans le point « Assistants », sont puissantes lors des recherches et des remplacements souhaités. Ainsi, le programme est capable de rechercher les formes erronées présentes dans les fichiers textes et de les remplacer par la forme correcte. Cela se fait avec la commande *re.sub()*. Le format .txt est conservé pour cette phase.

Lors de cette étape, une modification manuelle a été nécessaire pour deux fichiers afin d'éviter un décalage du nombre de lignes entre les deux fichiers textes. En effet, la vérification du nombre de lignes dans le fichier à modifier et dans le fichier modifié était indispensable pour ne pas se retrouver avec des décalages au niveau des paragraphes. Si cela se produisait, un écart apparaissait et engendrait une erreur sur le plan des annotations de discours direct.

Concrètement, les annotations de discours direct étant indiquées par rapport au fichier XML de départ, les modifications faites par les scripts suivants ne devaient pas altérer le nombre de paragraphes présents dans ce fichier, ni le nombre de lignes du fichier texte obtenu grâce au script de conversion, sous peine de se retrouver avec des annotations liées aux mauvais paragraphes ou des paragraphes vides en fin de fichier. Le programme de nettoyage a permis

de corriger de nombreuses erreurs automatiquement, mais les deux modifications manuelles ont dû venir contrer l'action de correction du précédent programme pour permettre de conserver le bon nombre de lignes dans le fichier texte.

Le choix d'annuler la rectification réalisée a été fait pour se prévenir d'erreurs plus importantes et plus difficiles à supprimer par la suite, telles que des décalages au niveau des paragraphes lors de l'injection finale engendrant des erreurs d'annotations. La vérification du nombre de lignes s'est faite grâce à un programme dont le fonctionnement sera développé dans la partie sur le corpus GLFA.

1.2. Utilisation de Stanza

Dès lors que les fichiers XML ont été corrigés syntaxiquement et que la conversion vers le format .txt a été opérée, l'analyse via Stanza a pu se mettre en place. Celle-ci a également dû se faire en plusieurs étapes.

1.2.1. Lancement de l'analyse

Le programme suivant est celui qui gère le lancement de l'analyse en faisant appel à Stanza. Il se nomme *analyse_A.py*. Comme précisé dans le point « Prise en main des outils », Stanza nécessite un import via la commande « import stanza ». La librairie *os* de Python a également été utilisée dans ce programme pour analyser tous les fichiers des dossiers concernés en une seule exécution, et pour obtenir les nouveaux dossiers contenant les résultats des analyses. Stanza étant disponible en 66 langues différentes, il est indispensable de préciser quelle langue doit être utilisée lors de l'analyse des textes nettoyés qui vont être transmis : dans ce cas-ci « Partut » et « GSD », les modèles par défaut du français. La langue sélectionnée doit alors être téléchargée grâce à la commande *stanza.download()*. Une condition a ensuite été mise en place afin de contraindre l'analyse : si les phrases étaient séparées par 2 retours à la ligne consécutifs, la tokenisation était la seule à être effectuée, la segmentation ne l'était pas.

Une fois toutes les exigences satisfaites, l'analyse peut avoir lieu. La suite du script concerne la manière dont l'analyse est réalisée. Tout d'abord, le texte fourni est lu ligne par ligne et découpé en phrases. Puis, chaque phrase est analysée par Stanza qui renvoie le résultat sous forme de ligne contenant les caractéristiques de chaque token composant la phrase (cf. Figure 2). Le détail de l'analyse est présenté dans le point « Stanza ». Les phrases analysées sont ensuite concaténées dans un nouveau fichier. Ce fichier est alors au format .conllu et plus

au format .txt. Cette étape est extrêmement longue car elle prend plus de 24h en se faisant sur le serveur MIAI.

1.2.2. Modification des sorties obtenues

Le contrôle des sorties obtenues à la suite de l'analyse de Stanza a révélé des erreurs de taille. En effet, la plupart des fichiers possédaient une première partie du texte analysé en double. Concrètement, le texte commençait par être analysé correctement, puis une phrase était incomplète et le début du texte apparaissait à nouveau pour être analysé jusqu'à la fin cette fois. Il a donc fallu supprimer la partie incomplète au début de chaque fichier pour obtenir une analyse valide et correspondant bien au texte fourni. Pour ce faire, un nouveau programme de modification a été créé, appelé *stanza_modif_A.py*.

La librairie *os* de Python a de nouveau été utilisée pour analyser tous les fichiers des dossiers concernés d'une seule traite, et pour obtenir les nouveaux dossiers accueillant les fichiers modifiés. En premier lieu, le fichier .conllu est lu par le script, puis il recherche une suite de caractères correspondant au recommencement de l'analyse. Cette suite de caractères peut être un *underscore* suivi du chiffre 1 et d'une tabulation, ou bien un *underscore* suivi d'une parenthèse contenant les chiffres 1 et 2 lorsque le fichier commence par une forme amalgamée, dont la notion sera expliquée dans le prochain point. Dès lors que l'une ou l'autre de ces suites est trouvée dans le fichier, la ligne contenant la suite de caractères ainsi que toutes les lignes suivantes sont ajoutées au nouveau fichier. Ensuite, une étape de nettoyage est également effectuée pour supprimer les *underscores* présents devant le chiffre 1 ou la parenthèse, et se retrouvant alors en tête de ligne. Cette modification n'entraîne pas de changement de format, les fichiers restent au format .conllu.

Néanmoins, comme précisé au début de ce point, la plupart des fichiers sont concernés par la présence de cette répétition, mais tous ne le sont pas. Après avoir exécuté ce script de modification, quelques fichiers se sont retrouvés vides car ils ne contenaient aucune des suites de caractères recherchées. Il a alors fallu récupérer les fichiers obtenus après l'analyse de Stanza pour ces quelques textes, d'où la création de nouveaux dossiers pour chaque phase.

1.2.3. Gestion des formes amalgamées

La langue française comporte de nombreuses formes amalgamées, ou contractées. Ces dernières correspondent à des articles tels que « au », « du » ou encore « des ». Ces formes sont analysées par Stanza de la manière suivante (cf. Figure 4) :

- une ligne commençant par des parenthèses et contenant la forme amalgamée apparaît
- la forme amalgamée est décomposée en 2 lignes contenant chacune les caractéristiques des tokens la constituant

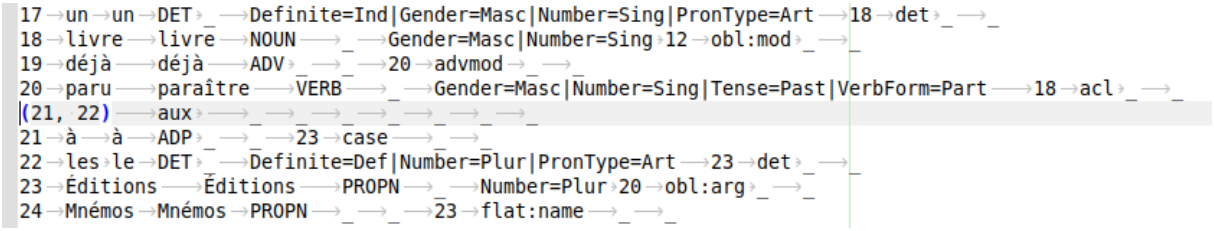


Figure 4. Forme amalgamée sortie par Stanza

De cette manière, une ligne est ajoutée à chaque nouvelle forme amalgamée, ce qui engendre un décalage dans les identifiants des tokens. Afin d'éviter cela, un nouveau programme automatique visant à conserver la forme amalgamée et ses caractéristiques tout en supprimant la ligne commençant par des parenthèses, nommé *mep_A.py*, a été créé.

La librairie *os* de Python est toujours présente pour les mêmes raisons que dans les scripts précédents. Chaque fichier est d'abord lu et décomposé en éléments correspondant à chaque caractéristique du token analysé. Le premier élément du token équivaut à son identifiant, qui doit obligatoirement commencer par un chiffre. Tant que le programme ne rencontre pas de parenthèse en premier élément, la ligne est ajoutée dans le nouveau fichier. Dans le cas où le premier élément commence par une parenthèse, cela signifie qu'il s'agit d'une forme amalgamée qui doit être traitée par le programme. La première action est alors de récupérer la forme amalgamée contenue dans la ligne et de l'injecter dans la ligne suivante, à la place de la forme qui s'y trouve. La ligne modifiée est ensuite ajoutée au nouveau fichier. L'action suivante consiste à remplacer la forme de la ligne suivant celle qui vient d'être modifiée par un *underscore*, puis à l'ajouter au nouveau fichier. Ce script ne nécessite pas de changement de format, les nouveaux fichiers obtenus sont toujours au format .conllu.

Les formes amalgamées ainsi obtenues se présentent comme suit (cf. Figure 5) :

- une première ligne contenant la forme amalgamée ainsi que les caractéristiques du premier token la composant
- une seconde ligne avec une forme vide et les caractéristiques du second token la composant

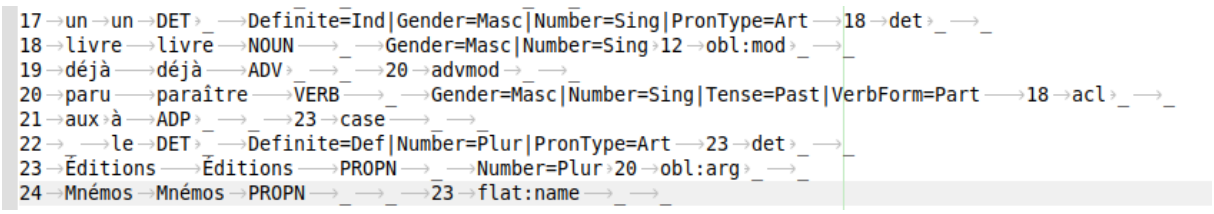


Figure 5. Forme amalgamée modifiée

De cette manière, la continuité des identifiants des tokens composant chaque phrase est respectée. Les fichiers sont plus simples à gérer. Ce script s'occupe également d'ajouter des retours à la ligne à chaque fois qu'une ligne commence par le chiffre 1. Ainsi, le fichier paraît plus propre, plus lisible et il est plus simple d'y repérer les phrases. Mais c'est également une modification qui permettra de faciliter l'insertion du format .conllu dans le fichier XML.

1.3. Création du format XML ConLL

Une fois l'analyse réalisée et les fichiers obtenus nettoyés de manière à récupérer des fichiers propres, l'initialisation du format XML ConLL peut avoir lieu. Différentes étapes sont encore nécessaires avant de parvenir au résultat souhaité.

1.3.1. Ajout des annotations de discours direct

Comme avancé en introduction de cette partie sur les romans annotés, Zhiyuan Qiu avait déjà réalisé le repérage des parties de discours direct dans les fichiers XML. Ces derniers contenaient des annotations notées « type=”dd” » à chaque token considéré comme relevant du discours direct (cf. Annexe 13). Cependant, pour obtenir le format XML ConLL, ces annotations allaient être supprimées avec le contenu des fichiers. Il fallait donc un moyen de conserver ces annotations. Pour ce faire, un programme appelé *ajout_type_dd_A.py*, gérant l'annotation automatiquement, a été créé.

Les librairies *os* et *re* de Python ont été nécessaires à ce script. Le choix fait pour gérer les annotations de discours direct a été d'annoter les phrases et non les tokens. Une phrase était considérée comme du discours direct si les 3 premiers tokens la composant possédaient l'annotation « type=”dd” ». La méthode adoptée pour repérer les passages correspondant aux conditions citées précédemment fut celle de l'expression régulière. Pour ce faire, la syntaxe des fichiers XML de départ a été observée puis retranscrite dans la regex, en utilisant des groupes de capture pour permettre une réinjection des contenus souhaités par la suite. Concrètement, 2 groupes de capture ont été utilisés et aucun contenu n'a été supprimé. Seule l'annotation

« type=”dd” » a été rajoutée à la balise <s> avant de réinjecter le contenu dans le nouveau fichier (cf. Annexe 14). Cela a permis de conserver l’annotation du discours direct même après la suppression des contenus spécifiques au format .xml. Aucun changement de format n’a été opéré par ce programme : les fichiers d’entrée étaient les fichiers XML dont la syntaxe avait été corrigée, et les fichiers de sortie sont restés au format .xml.

1.3.2. Suppression des contenus inadaptés

Afin de se rapprocher du format souhaité, à savoir le format XML ConLL, certaines suppressions sont nécessaires. Premièrement, lors des vérifications réalisées à la suite de chaque transformation de fichier, certains paragraphes vides ont été mis à jour. Pour une raison inconnue, des paragraphes respectant parfaitement la syntaxe XML mais ne présentant aucun contenu apparaissaient. Ces paragraphes étaient susceptibles d’engendrer des décalages à la fois au niveau des paragraphes, et de laisser des paragraphes vides en fin de fichier, mais aussi au niveau des annotations de discours direct. Pour éviter cela, un programme automatique repérant et supprimant les paragraphes vides a été conçu. Il se nomme *vides_suppr_A.py*.

Pour ce script, les librairies *os* et *re* de Python ont été utilisées, mais également la bibliothèque *lxml* qui possède à peu près les mêmes fonctions que la bibliothèque *ElementTree* de Python en étant plus puissante. Cette dernière est importée grâce à la commande « from lxml import etree » et nécessite l’utilisation de la librairie *io* de Python, importée avec « import io », permettant de transformer les données en objet compatible avec l’action du module *lxml*. La première action a été de lire le fichier, puis de rechercher des paragraphes ou des phrases vides à l’aide d’expressions régulières. Si l’une des deux regex utilisées renvoie un *match* (cf. point « Regex101 »), le paragraphe ou la phrase vide est supprimé. La seconde partie du programme s’occupe de la modification des identifiants des phrases dans le cas où des phrases ont été supprimées dans le fichier, pour éviter un décalage. Cette phase nécessite d’avoir accès à l’attribut d’une balise, ce que le module *lxml* est capable de faire, mais pas sans le module *io*. Afin de trouver les identifiants des phrases, il a fallu parser le fichier avec la commande *etree.parse()*. Cette phase a pris des fichiers au format .xml en entrée et n’a pas modifié le format des fichiers.

L’étape suivante consiste à supprimer le contenu des fichiers XML qui ne correspond pas au format XML ConLL. Concrètement, les seules balises qui doivent être conservées sont les balises de paragraphes et de phrases, ainsi que les métadonnées présentes en tête et en fin de fichier. Il faut donc supprimer le contenu des balises <s> pour pouvoir le remplacer par le

contenu du dernier fichier au format .conllu obtenu suite à toutes les modifications effectuées. Encore une fois, un programme a été créé pour gérer la suppression automatiquement : *suppr_xml_A.py*.

Le script de suppression finale a nécessité la librairie *os* de Python pour accéder à plusieurs fichiers en une fois, et la bibliothèque *lxml* pour accéder aux balises des fichiers et à leur contenu. Le fichier a d'abord été parsé avec l'aide de *lxml*, puis les balises inutiles ainsi que leur contenu ont été supprimés. Le format des fichiers a été conservé.

1.3.3. Insertion du ConLL dans le XML

Cette ultime phase a nécessité l'utilisation de la librairie *os* de Python et de la bibliothèque *lxml* comme lors de l'étape précédente. Cependant, contrairement aux programmes précédents qui récupéraient la sortie produite par leur prédécesseur pour y apporter leur modification, celui-ci a dû récupérer la sortie produite par 2 programmes différents. Ainsi, les fichiers en entrée provenaient à la fois de la sortie de la gestion des formes amalgamées au format .conllu, et de la sortie de la suppression finale au format .xml. L'insertion est expliquée ci-dessous, et schématisée dans la figure 6.

La première étape a été de lire le fichier .conllu d'une part et de parser le fichier XML d'autre part. Ensuite, les phrases ont été récupérées dans le fichier .conllu en séparant le contenu lorsque 2 retours à la ligne consécutifs étaient rencontrés. En parallèle, le nombre de balises <s> dans le fichier XML, correspondant au nombre de phrases, sont comptées. Deux variables i et j sont alors initialisées à 0.

L'insertion se fait alors grâce à une boucle. Une boucle sert à exécuter plusieurs fois une même action. Elle fonctionne grâce une condition d'entrée, une condition qui doit être satisfaite pour pouvoir entrer dans la boucle et effectuer toutes les actions souhaitées, et une condition de sortie, dès que la condition initiale n'est plus satisfaite. Les conditions et actions de la boucle utilisée ici sont les suivantes :

TANT QUE i < nb_phrases_conllu ET j < nb_phrases_XML FAIRE :
    Associer indice de phrase XML à j
    Récupérer identifiant
    SI identifiant commence par « s » ALORS
        Récupérer identifiant numéral
        SI identifiant numéral est un nombre entier ET identifiant numéral == i + 1 ALORS
            Insérer contenu de phrase .conllu dans la phrase XML correspondante

i = i + 1
Fin SI
Fin SI
j = j + 1
Fin TANT QUE

Le nouveau fichier contenant à la fois les balises XML et les phrases du fichier .conllu est alors créé. Il est enregistré au format .xml mais contient le format XML ConLL (cf. Annexe 12). Ce dernier programme se nomme *conllu_to_xml_A.py*.

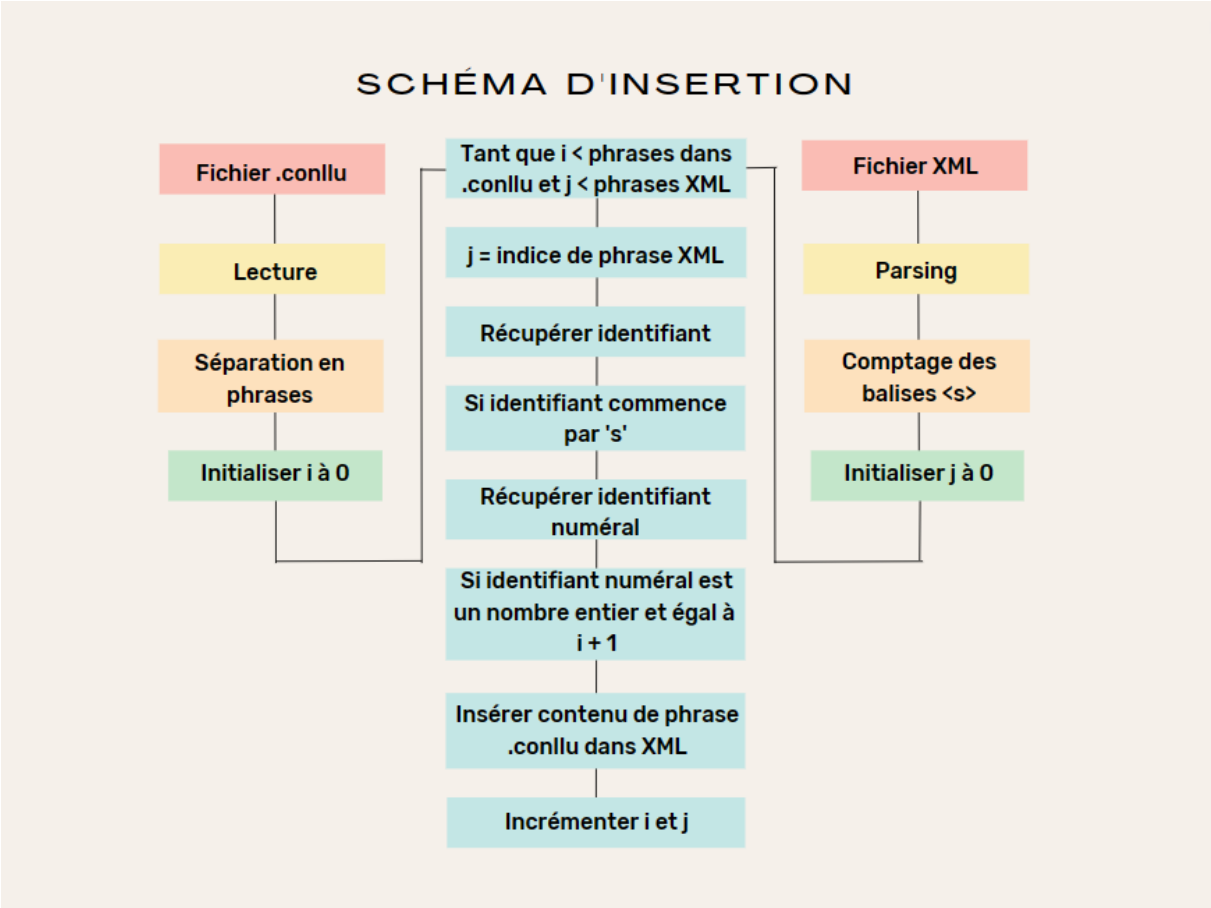


Figure 6 : Schéma d'insertion du format ConLL dans le format XML

2. Romans non annotés

Contrairement aux romans annotés, Zhiyuan Qiu n'a pas traité les romans de cette catégorie lors de son stage. Ils ne possèdent donc pas d'annotations sur les passages de discours direct. Or, ces passages sont importants dans les corpus écrits car ils rapportent une utilisation de l'oral dans un document écrit, ce qui équivaut à de l'oral représenté et se diffère de l'oral authentique. Les phrases préfabriquées les plus productives dans les corpus oraux et écrits seront comparées à la fin de cette étude, mais également selon le type d'oral concerné. Il est

donc primordial de repérer les passages de discours direct dans les corpus écrits. Cette catégorie de romans concerne 237 fichiers dont 37 romans de science-fiction, 49 romans policiers, 32 romans historiques, 48 romans de fantaisie, 14 romans sentimentaux et 57 romans de littérature générale.

2.1. Gestion des fichiers XML

De même que pour les romans déjà annotés, une conversion du format .xml vers le format .txt est nécessaire avant de pouvoir analyser les fichiers avec Stanza. Cependant, les annotations de discours direct n'apparaissant pas dans les fichiers XML de départ, la première étape est de les ajouter.

2.1.1. Correction

Avant de pouvoir réaliser l'annotation des fichiers XML, il a de nouveau fallu les corriger syntaxiquement. En effet, les scripts d'annotation ne fonctionnaient pas car ils détectaient des erreurs rendant les fichiers inutilisables. Ces erreurs correspondaient à nouveau à des balises ouvertes mais pas fermées dans les fichiers. Dans le point « Conversion » de la partie « Romans annotés », cette correction a été très chronophage. Pour éviter de perdre à nouveau beaucoup de temps à rechercher les erreurs, un programme automatique capable de repérer et d'indiquer l'emplacement des erreurs de syntaxe a été créé, appelé *erreur_xml.py*.

Ce dernier était basé sur la bibliothèque *lxml* de Python et fonctionnait grâce au parsing. Si le programme réussissait à parser le fichier, cela signifiait que le fichier ne contenait pas d'erreurs de syntaxe ou de balisage : le script renvoyait alors un message indiquant la validité du fichier. Dans le cas contraire, le script indiquait que le fichier était invalide et renvoyait l'erreur ainsi que son emplacement dans le fichier. De cette manière, la correction des erreurs a été grandement simplifiée et le temps nécessaire a bien diminué.

2.1.2. Annotation du discours direct

Les fichiers XML corrigés, l'annotation pouvait avoir lieu. Les passages de discours direct ayant été traités par Zhiyuan Qiu dans le corpus de romans annotés, les scripts permettant de réaliser les annotations existaient. Olivier Kraif me les a transmis afin d'exécuter les traitements sur la totalité du corpus Phraseobase. Les programmes gérant les annotations étaient au nombre de 5, et la seule modification apportée a été l'utilisation de la librairie *os* de Python pour pouvoir annoter tous les fichiers en un seul lancement, Zhiyuan s'occupant des fichiers un

par un. Tous les programmes prennent en entrée un fichier au format .xml et ne modifient pas le format de sortie.

Le premier programme, appelé *Regle_1.py*, utilise les librairies *re* et *xml.dom.minidom*, qui est une alternative à la bibliothèque *ElementTree* de Python, importée grâce à la commande « import xml.dom.minidom ». Zhiyuan a d'abord initialisé une expression régulière afin de rechercher les tirets de discussion dans le fichier XML, puis elle a parsé le fichier pour obtenir les différentes balises présentes à l'aide de la commande *xml.dom.minidom.parse()*. Elle a ensuite recherché les balises <t> contenues dans des balises de paragraphes <p>, et a appliqué son expression régulière sur le contenu des balises <t> repérées. Dans les cas où un tiret était présent, elle a ajouté l'attribut « type="dd" » à la balise <t> correspondante.

Le deuxième programme, appelé *Regle_2.1.py*, utilise également les librairies *re* et *xml.dom.minidom*, mais s'intéresse cette fois aux guillemets français présents dans le contenu des balises <t>. À nouveau le fichier est parsé, les balises sont repérées, et la regex est appliquée au contenu des balises <t>. Afin de différencier les guillemets ouvrants, signifiant le début d'un passage de discours direct, des guillemets fermants, signifiant la fin de ce passage, Zhiyuan a utilisé un booléen. Un booléen est une variable ne pouvant avoir que 2 valeurs : vrai ou faux. Des actions peuvent alors être effectuées dans une boucle en fonction de la valeur renvoyée par le booléen. La boucle utilisée ici fonctionne de la manière suivante :

```
Initialiser le booléen à faux au début de la boucle
TANT QUE la regex renvoie un match FAIRE :
    SI le booléen est faux ALORS
        SI le match est un guillemet ouvrant ALORS
            Changer la valeur du booléen à vrai
        SINON
            Ne rien faire
        Fin SI
    SINON
        SI le match est un guillemet fermant ALORS
            Ne rien faire
        SINON
            Ajouter l'attribut « type ="dd" » à la balise <t> correspondante jusqu'à
            ce que la regex trouve le guillemet fermant
        Fin SI
    Fin SI
    SI le booléen est vrai ET la regex ne renvoie pas de match ALORS
        Ajouter l'attribut « type ="dd" » à la balise <t> correspondante
    Fin SI
Fin TANT QUE
```

Le programme suivant, appelé *Regle_2.2.py*, utilise toujours les librairies *re* et *xml.dom.minidom*, mais son but est de supprimer les attributs « type=”dd” » abusifs entre des parenthèses ou des crochets. De la même façon que lors des scripts précédents, le fichier est parsé, les balises sont repérées, et la regex est appliquée au contenu des balises <t>. À nouveau, un booléen est mis en place pour faire la différence entre les parenthèses ou crochets ouvrants et les fermants. Lorsque la regex renvoie un *match*, l’attribut « type=”dd” » est remplacé par « type=”/” », et lorsque la regex ne renvoie pas de *match* mais que le booléen renvoie vrai l’opération est également appliquée.

Les quatrième et cinquième programmes, appelés respectivement *Regle_3.1.py* et *Regle_3.2.py,* visent à supprimer les attributs « type=”dd” » abusifs en fonction des pronoms et des verbes apparaissant autour de ces attributs. Ils fonctionnent aussi avec les librairies *re* et *xml.dom.minidom*, le parsage des fichiers, le repérage des balises, l’application de regex, et utilisent des booléens et des listes de pronoms et de verbes. À partir de ces listes, Zhiyuan détermine les situations où l’attribut « type=”dd” » n’a pas lieu d’être, car il ne correspond pas à du discours direct mais à du discours rapporté. Le principe est toujours le même : la regex renvoie ou non un *match*, le booléen renvoie une valeur et une comparaison avec les listes est effectuée. Dans la situation où le discours est rapporté et pas direct, l’attribut « type=”dd” » est remplacé par « type=”/” ».

2.1.3. Lancement des scripts

Une fois les annotations effectuées grâce aux scripts de Zhiyuan Qiu, les étapes de conversion du format .xml vers le format .txt similaires aux romans annotés ont été possibles. La conversion des fichiers XML au format texte brut des romans non annotés est identique à celle des romans annotés, de même pour les étapes de nettoyage réalisées après la conversion et toutes les étapes suivantes. Les erreurs retrouvées dans les fichiers en sortie d’analyse avec Stanza, les modifications à opérer pour rendre les fichiers plus propres, les suppressions, ou encore l’insertion du format .conllu dans les fichiers .xml sont globalement équivalents pour les romans non annotés et pour les romans annotés.

Cependant, une copie des scripts gérant les fichiers des romans annotés a été créée pour la gestion des romans non annotés. Cela se justifie notamment par le fait que certaines modifications étaient spécifiques à l’une ou l’autre des catégories, mais également parce que le répertoire, et donc le chemin d’accès, des fichiers non annotés était différent de celui des fichiers annotés. Ainsi, les noms des programmes utilisés pour les romans annotés ont été

réutilisés en changeant simplement le « A » à la fin du nom en « NA » (*conversion_A.py* est devenu *conversion_NA.py*). De plus, le temps de traitement des fichiers aurait été bien plus conséquent si les dossiers annotés et non annotés avaient été modifiés à la suite. En faisant le choix de séparer les programmes, cela laissait la possibilité de les faire tourner en parallèle sur le serveur via les sessions disponibles (cf. point « MIAI Serveur »).

2.2. Vérification des lemmes de Stanza

Bien que l'analyse de Stanza soit précise, il subsiste quelques erreurs au niveau de la lemmatisation à la sortie du traitement. Ces erreurs étant assez problématiques, il a fallu trouver un moyen de procéder à leur vérification et à leur correction lorsque cela était possible.

2.2.1. Utilisation d'un dictionnaire de formes fléchies

Pour cette étape, Olivier Kraif possédait un dictionnaire de formes fléchies du français extrait du **GLAFF**[36]. Le GLAFF (Gros Lexique À tout Faire du Français) a été construit grâce au dictionnaire de Wikipédia, le Wictionnaire. Chaque entrée de ce lexique est composée de sa forme, de sa catégorie syntaxique, de son lemme, de sa transcription phonétique en API (Alphabet Phonétique International), de sa ou ses transcriptions(s) équivalentes en SAMPA (Speech Assessment Methods Phonetic Alphabet), et enfin de ses fréquences absolues et relatives (par million de mots) pour sa forme et son lemme dans différents corpus. Il contient essentiellement des formes nominales, verbales, adjectivales et adverbiales. N'ayant pas besoin de toutes les caractéristiques proposées par le GLAFF, le dictionnaire créé par Olivier Kraif ne contenait que la forme, la catégorie syntaxique et le lemme pour chacune des entrées (cf. Annexe 15).

L'utilité d'un dictionnaire de ce type est de réaliser une comparaison entre la sortie produite par Stanza et les caractéristiques présentes dans le dictionnaire. Ainsi, ce dernier servira de référence : si un lemme apparaît différemment dans la sortie de l'outil et dans le dictionnaire, il faut se référer au dictionnaire et remplacer la forme initiale par la forme correcte.

Ce dictionnaire au format shelve a été récupéré sur le serveur MIAI d'Olivier Kraif sous forme de dossier contenant deux formats différents : le format .tsv et le format .dat. Le format .tsv a été utilisé en premier lieu pour effectuer des tests sur un seul fichier et contrôler la qualité des modifications réalisées, car c'est le seul format de fichier lisible avec des applications sur

[36] http://redac.univ-tlse2.fr/lexiques/glaff.html

la machine. En second lieu, le format .dat a été privilégié pour gérer les changements sur tous les fichiers directement sur le serveur, car il est plus rapide et bien plus efficace.

2.2.2. Correction automatique

Comme précisé précédemment, le programme de vérification et de correction automatique des lemmes erronés dans les sorties de Stanza se base sur un dictionnaire de référence tiré du GLAFF. Il se nomme *conllu_verif.py*, et ne possède pas de version spécifique aux romans annotés et non annotés : un seul script gère les 2 types de romans. Pour pouvoir réaliser ces changements, la librairie *os* a été importée ainsi que la librairie *shelve* de Python afin d'utiliser le dictionnaire au format .dat. Les commandes nécessaires à ces importations sont « import os » et « import shelve ».

La première étape a été l'ouverture et l'enregistrement du dictionnaire dans une variable pour pouvoir y accéder par la suite. Puis, le fichier au format .conllu obtenu après les différents traitements effectués sur les sorties de Stanza (cf. point « Utilisation de Stanza ») est ouvert et lu ligne par ligne pour faire les comparaisons. Pour chaque ligne rencontrée dans le fichier .conllu, chaque élément composant la ligne et séparé par une tabulation est récupéré. Les éléments utiles à la comparaison sont ensuite enregistrés dans des variables, à savoir la forme (correspondant au 2ème élément de la ligne), le lemme (correspondant au 3ème élément) et la catégorie syntaxique (correspondant au 4ème élément). Une clé composée de la forme transcrite en minuscules, suivie d'une tabulation, suivie de la catégorie syntaxique est alors formée : cette clé sera recherchée dans le dictionnaire de formes fléchies lors des vérifications. Le fait de transformer la forme présente dans le fichier en minuscules permet de prendre en compte n'importe quelle forme, qu'elle se trouve en début de phrase (avec une majuscule) ou dans une phrase. En effet, la position de la forme dans la phrase ne modifie pas son lemme.

Un exemple de clé formée peut être : mangera + \t + VERB

Les étapes suivantes se font grâce à des conditions dans une boucle :

SI la catégorie syntaxique est un nom, un verbe, un adjectif ou un adverbe ET la clé existe dans le dictionnaire ALORS
    SI le lemme du fichier .conllu n'existe pas dans le dictionnaire ALORS
        Remplacer le contenu du lemme par la forme correspondante dans la clé du dictionnaire (en minuscules)
        Remplacer le 3ème élément de la ligne du fichier .conllu par le nouveau contenu de la variable lemme
    Fin SI

Fin SI

De cette manière, lorsqu'un lemme est inventé dans le fichier .conllu, le fait de le remplacer par la forme présente dans le dictionnaire permet de limiter les erreurs. En effet, la forme et le lemme sont souvent similaires dans les cas d'invention de lemme. De plus, si le lemme est vide, c'est-à-dire correspondant à un *underscore,* le remplacement de celui-ci par la forme permet également de corriger l'erreur de lemmatisation.

Ce programme a aussi permis de remplacer les *underscores* abusifs par des tirets cadratins, plus appropriés aux discours directs. La vérification s'effectue également avec une boucle :

- si les 2ème et 3ème éléments de la ligne dans le fichier .conllu sont des *underscores*, et que le 4ème élément correspond à la catégorie syntaxique « PUNCT » (condition d'entrée)
- les 2ème et 3ème éléments sont remplacés par des tirets cadratins
- sinon la ligne n'est pas modifiée

Une condition indispensable au bon fonctionnement de ce script est la fermeture du dictionnaire à la fin des modifications. En effet, si le dictionnaire au format .dat est ouvert mais pas fermé, cela bloque l'exécution du programme et engendre une erreur alors que le contenu est bien valide.

2.3. Ordre des programmes

La création du script de vérification des lemmes de Stanza bouleverse l'ordre des modifications effectuées à la fois sur les romans annotés et non annotés. En effet, les remplacements de lemme sont à effectuer après le programme *mep_A.py* pour les romans annotés et *mep_NA.py* pour les romans non annotés. Étant donné que les scripts *ajout_type_dd_A.py*, *vides_suppr_A.py* et *suppr_xml_A.py* (et leurs homologues non annotés) se basent sur les fichiers XML et non les fichiers .conllu, il n'est pas nécessaire de les relancer après avoir fait tourner *conllu_verif.py*. Par contre, il est indispensable d'effectuer à nouveau *conllu_to_xml_A.py* et *conllu_to_xml_NA.py*, car les fichiers .conllu sont affectés par le nouveau script.

Les traitements du corpus Phraseobase terminés, le chapitre suivant présente ceux concernant le corpus GLFA.

Chapitre 6. Corpus GLFA

Dans un second temps, les romans du corpus GLFA de l'ATILF ont dû être traités de la même manière que les romans du corpus Phraseobase. Il a fallu annoter les romans composant le corpus, les analyser grâce à Stanza, et les convertir du format XML au format XML ConLL. Ce corpus était disponible sur un sharedocs appartenant à la plateforme Huma-Num, permettant à tous les membres du projet de pouvoir s'échanger des dossiers rapidement et simplement. Cependant, étant stagiaire, je n'ai pas eu de droit d'accès à cette plateforme. Olivier Kraif a donc récupéré ce corpus pour le publier sur le serveur MIAI afin que je puisse y accéder et travailler dessus.

Une étape supplémentaire par rapport au traitement du précédent corpus a été nécessaire : celle de l'alignement des romans en français avec leurs traductions en allemand, et inversement. Étant donné la variation de langue entre les fichiers, les traitements à effectuer ont été différents.

1. Romans français

La première partie de la gestion du corpus GLFA était axée sur les romans en français langue source et en français langue cible. Les romans considérés comme langue source sont des romans écrits en français, tandis que les romans en français langue cible sont des romans allemands traduits en français. De même, les romans en allemand langue source sont des romans écrits en allemand, et les romans en allemand langue cible sont des traductions de romans français vers l'allemand.

1.1. Gestion initiale des fichiers XML

La gestion des fichiers XML de départ n'a pas été exactement la même pour le corpus GLFA que pour le corpus Phraseobase. En effet, il a fallu également corriger les erreurs de syntaxe présentes dans les fichiers et annoter le discours direct qui n'était pas repéré, mais de nouvelles modifications ont dû être réalisées en supplément.

1.1.1. Correction et annotation

La première étape a été de récupérer les scripts de Zhiyuan Qiu pour gérer l'annotation des parties de discours direct n'étant pas encore repérées dans ce corpus. De la même manière que pour les précédents corpus, des erreurs sont apparues lors du lancement du premier programme, empêchant ainsi l'exécution du script. Une réutilisation du programme

erreur_xml.py a donc été faite (cf. point « Gestion des fichiers XML ») afin de repérer rapidement le type et l'emplacement des erreurs contenues dans ce corpus. Certaines erreurs étaient identiques à celles trouvées dans le précédent corpus, c'est-à-dire des erreurs de balisage ou de syntaxe, d'autres étaient différentes et correspondaient à des erreurs d'entités non définies. Ce type d'erreur est particulier et assez long à corriger, même si cela peut également se faire via un programme automatique.

Dans le cas du corpus GLFA, les entités non définies correspondaient à des caractères spéciaux comme des chevrons ouvrants et fermants ou des esperluettes. L'erreur provenait du fait que ces caractères spéciaux devaient apparaître comme des entités HTML, c'est-à-dire codées, ce qui n'était pas le cas. Un mélange d'entités était également présent, ainsi que des entités composées de 3 lettres. Un programme de remplacement automatique appelé *glt_replace.py* a été créé pour gérer ces changements directement sur les fichiers XML d'origine grâce aux librairies *os* et *re* de Python. Celui-ci agissait à la fois sur les romans en français et en allemand.

Une fois les erreurs d'entités non définies corrigées, les programmes suivants ont pu être lancés sans blocage. Les programmes gérant l'annotation des parties de discours direct des romans non annotés du corpus Phraseobase ont été adaptés au corpus actuel, et appliqués aux romans en français langue source et en français langue cible. Ces étapes n'ont pas nécessité de changement de format, les fichiers en entrée et en sortie étaient au format .xml.

1.1.2. Conversion et nettoyage

L'étape suivant l'annotation du discours direct est celle de la conversion du format .xml vers le format .txt. La structure des fichiers XML étant similaire dans les 2 corpus au niveau des balises et de leur contenu visés par le programme de conversion (cf. point « Conversion »), l'adaptation du script *conversion.py* au nouveau corpus a donc été rapide. S'en est suivi l'étape de nettoyage des fichiers textes obtenus car, comme lors de la conversion des romans annotés et non annotés, des erreurs subsistent. Certaines erreurs sont identiques à celles retrouvées dans le corpus Phraseobase, à savoir des mots coupés en deux, des *underscores* abusifs et des retours à la ligne apparaissant avec « \n », et d'autres erreurs spécifiques au corpus GLFA sont apparues.

Le programme *text_suppr_n.py* a donc été créé pour corriger les erreurs similaires au précédent corpus, et de nouvelles erreurs comme l'apparition de guillemets seuls autour de

ponctuation, des erreurs d'OCR (cf. point « Corrections des sorties des outils »), ou encore le passage dans les fichiers textes des entités HTML codées aux caractères spéciaux. Ce script ne provoquait pas de changement de format, les fichiers en entrée étaient au format .txt. Les erreurs spécifiques aux romans de ce corpus sont développées dans la partie suivante.

1.2. Erreurs spécifiques dans les textes

Les erreurs considérées comme spécifiques le sont car elles n'apparaissent que dans le corpus GLFA, elles n'ont pas été identifiées dans le corpus Phraseobase. Ces erreurs sont globalement de deux types différents : des erreurs directement liées à l'OCR et des erreurs provoquées par le scan des romans faisant apparaître des éléments indésirables dans la structure des textes.

1.2.1. OCR

Des erreurs spécifiques à l'OCR ont été repérées dans les romans au format .txt, correspondant à des mauvaises associations à la suite du scan des romans (comme précisé dans le point « Corrections des sorties des outils »). Afin de repérer ces mots inconnus résultant d'une transcription erronée, l'outil NooJ a été utilisé pour effectuer une analyse de ces textes.

La première étape de cette analyse a été de combiner tous les textes français du corpus en un seul texte pour n'obtenir qu'un seul fichier en sortie. La taille de ce fichier étant assez conséquente, l'analyse a mis un certain temps à s'effectuer. La sortie obtenue était une liste des mots inconnus repérés dans les textes, correspondant à un dictionnaire au format .dic (cf. Annexe 16). L'étape suivante a été la conversion de ce dictionnaire de mots inconnus au format .xlsx permettant la création de 2 colonnes distinctes : l'une présentant les formes inconnues présentes dans les textes, et l'autre la forme correcte devant remplacer la forme erronée. Ce format de données est utilisable par un script de remplacement automatique gérant les substitutions. La recherche des mots inconnus dans les textes en français a été réalisée par ma tutrice Agnès Tutin, car elle maîtrise mieux l'outil que moi.

Une fois le fichier Excel créé, il a fallu compléter la 2ème colonne avec les nouvelles formes devant corriger les mauvaises. Cette tâche fut assez chronophage car le fichier de départ contenait plus de 20 000 lignes, donc potentiellement 20 000 formes inconnues à rectifier. La ligne directrice de cette phase était liée à la certitude absolue, c'est-à-dire que lorsqu'une forme inconnue apparaissait dans la 1ère ligne du fichier Excel, il fallait la corriger uniquement si la forme correcte était déterminée de manière évidente et qu'aucune autre forme ne pouvait être

possible. Par exemple, dans le cas de mots collés il était certain que le décollement des mots était la seule possibilité, le changement pouvait donc avoir lieu. En revanche, lorsqu'un mot inconnu ne pouvait pas être rattaché de manière certaine à une forme existante, aucun changement n'était opéré. Le but étant de limiter au maximum les erreurs et pas d'en créer de nouvelles en modifiant abusivement les données. De plus, les mots en ancien français et en langue étrangère apparaissaient également dans le fichier Excel car ils sont considérés comme inconnus selon le dictionnaire de NooJ, mais ils ne devaient pas être modifiés.

Après avoir complété le fichier Excel avec les formes à remplacer, et supprimé les lignes dont les formes inconnues ne seraient pas traitées, celui-ci contenait encore plus de 4000 lignes (cf. Annexe 17). Plusieurs conversions de format ont alors été opérées pour passer du format .xlsx au format .csv, puis au format .dat permettant des transformations plus rapides et plus efficaces (cf. point « Utilisation d'un dictionnaire de formes fléchies »). Le parcours et les remplacements automatiques ont ensuite été gérés par un programme appelé *correction_OCR_fr.py*. Ce dernier utilise également les librairies *os* et *shelve* de Python, comme le script faisant appel au dictionnaire de formes fléchies pour le corpus Phraseobase, car il va réaliser des actions similaires.

Dans un premier temps, le dictionnaire et le fichier texte sont ouverts, puis le fichier texte est lu ligne par ligne pour effectuer la comparaison. Chaque mot de la ligne est récupéré grâce à la commande *line.split()* permettant le découpage d'un texte en mots, puis une boucle est initialisée :

```
POUR CHAQUE mot DANS les mots de la ligne FAIRE :
        SI le mot est dans le dictionnaire ALORS
                Enregistrer la forme correcte du dictionnaire dans une variable
                Ajouter le contenu de la variable au nouveau fichier
        SINON
                Ajouter le mot directement au nouveau fichier
        Fin SI
Fin POUR
```

À nouveau, il est important de penser à fermer le dictionnaire .dat pour éviter les erreurs d'exécution. Les fichiers en entrée étaient des fichiers texte, les fichiers en sortie l'étaient également.

1.2.2. Mise en page

Des erreurs pouvant s'apparenter à de la mise en page ont été découvertes après la conversion du format .xml au format .txt. En effet, des numéros de pages, des balises héritées du XML, des titres ou encore des numéros de paragraphes pouvaient apparaître à n'importe quel endroit du texte. Cela est dû au scan des versions papier des romans, où le titre du roman ou bien celui du chapitre peut apparaître en tête de chaque page par exemple. Idem pour les numéros de pages. Cependant, ces éléments perturbent la lecture, car les pages n'apparaissent pas sur les versions numériques : ce sont donc des éléments abusifs à retirer des fichiers textes.

Pour ce faire, un second programme de nettoyage, nommé *titre_suppr_fr.py*, a été créé. Comme son nom l'indique, il a pour but de supprimer les titres apparaissant n'importe où dans les fichiers textes, mais pas seulement. En effet, les apparitions de paragraphes abusifs, de numéros de pages et de balises indésirables sont également gérées par ce script. Une version locale a d'abord été réalisée afin de s'assurer que les changements provoqués par les expressions régulières correspondaient à ce qui était attendu, et que le nombre de lignes des fichiers modifiés était identique à celui des fichiers non modifiés (cf. point « Nettoyage »). Les titres apparaissant dans les romans français sont toujours en majuscules et ont été repérés grâce à un programme automatique appelé *specific_search.py*. Celui-ci fonctionne avec les librairies *os* et *re* de Python, recherche une expression régulière et affiche chaque *match* trouvé.

Le script *titre_suppr_fr.py*, quant à lui, n'utilise que la librairie *re* de Python et effectue, selon le cas, des suppressions ou des remplacements. Il était impossible d'utiliser la librairie *os* ici, car tous les textes des dossiers n'étaient pas concernés par les modifications. De plus, le risque de perte de données était grand si les expressions régulières utilisées modifiaient des contenus qui ne devaient pas l'être. Par exemple, un titre apparaissant dans un texte et devant être supprimé à l'aide d'une regex compatible ne créera pas de problème, mais la même regex agissant sur un texte ne devant pas être modifié engendrera des changements indésirables. Il faut donc créer une regex pour chaque texte séparément afin d'éviter toute altération involontaire. Ce script a comme objectif de rendre les textes plus propres, prend des textes en entrée et renvoie des textes en sortie.

Une vérification manuelle des titres a été nécessaire pour repérer les cas où ces derniers coupent des mots en deux, ainsi que les différentes orthographes possibles (notamment la présence ou non d'accents). En effet, les expressions régulières sont très puissantes mais ne sont pas efficaces lors de cas particuliers : si la recherche ne prend pas en compte les accents et

que les titres en contiennent, ils ne seront pas supprimés. De même pour les titres coupant des mots en deux, ils seront supprimés mais le mot ne sera pas reformé et contiendra un tiret signalant la séparation. Tandis que la prise en compte du tiret dans la regex peut reformer le mot initial, il est donc important de vérifier les cas particuliers dans chaque texte pour gérer au mieux la suppression.

1.3. Suite des opérations

Une fois les erreurs spécifiques aux fichiers XML puis aux fichiers textes corrigées, les romans sont prêts à être analysés par Stanza et à subir les modifications suivantes, de manière similaire au corpus Phraseobase.

1.3.1. Vérification du nombre de lignes

Comme précisé précédemment, la vérification du nombre de lignes dans les fichiers est une étape importante pour éviter de nombreux décalages et erreurs créés par les programmes agissant sur les romans (cf. point « Nettoyage »). Lorsque l'on travaille sur un seul fichier, il est aisé de vérifier le nombre de lignes manuellement après chaque opération réalisée en ouvrant chaque version. Mais dans le cas de centaines de fichiers modifiés en parallèle, une vérification manuelle n'est pas envisageable. Il faut alors trouver un moyen de contrôler ces lignes automatiquement.

Le programme créé pour gérer les lignes des différentes versions des fichiers au format .txt se nomme *text_comp.py*. Il utilise la librairie *os* de Python pour parcourir plusieurs fichiers et les comparer. Il est composé d'une fonction permettant une modification rapide des dossiers concernés. En effet, la fonction est identique quels que soient les fichiers et leurs emplacements. Le fait de ne modifier que le chemin d'accès pour réaliser une nouvelle comparaison offre un gain de temps et d'efficacité sans précédent (cf. Annexe 3).

Le script compare deux fichiers dans deux répertoires différents : dans le cas servant d'exemple, le chemin d'accès du répertoire du texte de référence (obtenu après le script de conversion) est indiqué dans une variable, puis le chemin d'accès du répertoire du texte à comparer est enregistré dans une seconde variable. Une liste des fichiers présents dans les deux répertoires est ensuite récupérée grâce à la commande *os.listdir()*, puis une boucle est initialisée :

POUR CHAQUE texte DANS la liste obtenue FAIRE :

Former le chemin d'accès du texte de référence en utilisant le répertoire et le texte lui-même, et l'enregistrer dans une nouvelle variable
Former le chemin d'accès du texte à comparer en utilisant le répertoire et le texte lui-même, et l'enregistrer dans une nouvelle variable
Ouvrir et lire chaque texte de référence
Ouvrir et lire chaque texte à comparer
SI le nombre de lignes lues est différent dans le fichier de référence et dans le fichier à comparer ALORS
Afficher un message indiquant les noms des fichiers concernés ainsi que le répertoire
Fin SI
Fin POUR

Pour pouvoir utiliser la fonction, il faut simplement renseigner le répertoire souhaité dans la variable correspondante : le répertoire est utilisé automatiquement car la 1ère ligne de la fonction est « def comparer_fichiers_repertoire(repertoire) ».

Afin de s'assurer qu'aucun décalage n'apparaîtra dans le fichier XML ConLL final, il suffit de contrôler le nombre de lignes présentes dans le fichier texte obtenu après le script de conversion (correspondant au fichier de référence) et de vérifier que ce nombre ne diffère pas dans les fichiers textes suivants, c'est-à-dire ceux obtenus après l'exécution des scripts *text_suppr_n.py*, *correction_OCR_fr.py* et *titre_suppr_fr.py*. Si un changement au niveau des lignes est observé dans l'un des fichiers sortant de l'un de ces programmes, il sera conservé jusqu'à l'insertion du format .conllu dans le fichier XML. Le maintien d'un nombre de lignes identique dans toutes les versions des fichiers textes garantit donc l'absence de décalage au niveau des paragraphes et des annotations de discours direct.

1.3.2. Analyse et corrections

La phase réalisée après les vérifications du nombre de lignes des fichiers textes et les corrections des erreurs repérées auparavant fut celle de l'analyse avec Stanza. L'adaptation du programme utilisé pour le corpus Phraseobase fut très rapide puisque seuls les chemins d'accès des fichiers différaient pour les romans en français. Un doublon a néanmoins été créé pour gérer les romans en allemand avec le modèle germanique de Stanza en parallèle, grâce aux différentes sessions disponibles sur le serveur (cf. Annexe 18). Ce point sera développé dans la partie suivante.

Les programmes *stanza_modif.py* et *mep.py* hérités du traitement du corpus Phraseobase ont ensuite été lancés sur les romans en français. Les erreurs de Stanza étaient similaires à celles rencontrées précédemment, ce qui permettait de réutiliser les mêmes scripts pour éliminer ces

erreurs récurrentes (cf. points « Modification des sorties obtenues » et « Gestion des formes amalgamées »).

Le script *conllu_verif.py* a également été récupéré pour gérer les erreurs de lemmatisation de Stanza sur les romans en français. Le dictionnaire des formes fléchies du français a été réutilisé, et les remplacements effectués ont eu lieu dans les mêmes conditions que précédemment (cf. point « Correction automatique »). Sachant que cette version n'était compatible qu'avec les romans en français, un doublon a été créé pour s'adapter aux romans en allemand. Ce point sera aussi développé dans la partie suivante.

Puis, le programme *ajout_type_dd.py* a repris du service en indiquant les phrases concernées par du discours direct, cette fois-ci pour le corpus GLFA. Les annotations de discours direct ayant été réalisées avec les scripts de Zhiyuan Qiu à la fois sur le corpus Phraseobase et le corpus GLFA, le programme a pu être réutilisé en ne modifiant que les chemins d'accès des fichiers.

1.3.3. Suppression des contenues inadaptés et insertion

Concernant les opérations de suppression des contenus indésirables, plusieurs modifications ont été nécessaires contrairement aux programmes précédents. En effet, la structure des fichiers XML de départ était visiblement similaire pour conserver le script de conversion, mais pas suffisamment pour réutiliser ceux de suppression. Dans le corpus Phraseobase, les paragraphes ne possédaient pas d'identifiants tandis que les phrases en possédaient. Dans le corpus GLFA, non seulement les phrases possèdent un identifiant, mais les paragraphes aussi. De ce fait, il a fallu adapter les expressions régulières initialisées dans le programme *vides_suppr.py* afin de prendre en compte la nouvelle structure.

De plus, les identifiants des phrases dans le corpus Phraseobase étaient composés de la lettre « s » suivie d'un nombre (cf. point « Insertion du ConLL dans le XML »), tandis que ceux des phrases du corpus GLFA étaient formés d'une suite de nombres, d'un tiret, de la langue du roman (« fr » ou « de »), d'un tiret et d'un autre nombre. Afin de procéder à l'évitement des décalages dans le cas de suppression d'un paragraphe vide, les identifiants des phrases ont été simplifiés et transformés en lettre « s » suivie du nombre de la phrase. De cette façon, les identifiants des romans en allemand et en français pouvaient être traités par le même programme sans que cela ne prive le fichier d'informations essentielles.

Puis, le programme de suppression du contenu des fichiers XML *suppr_xml.py* a simplement été réutilisé car la structure des romans le permettait. Et enfin, le programme d'insertion du format .conllu dans le fichier au format .xml étant similaire à celui utilisé pour le corpus Phraseobase, avec seulement les chemins d'accès des fichiers qui diffèrent, il a également pu être réutilisé. Le fait d'avoir transformé le format des identifiants des phrases dans l'étape précédente a permis de ne pas avoir à modifier les conditions de la boucle d'insertion, cette dernière étant basée sur la présence de la lettre « s » au début des identifiants. Cela a simplifié et accéléré le traitement des romans de ce corpus.

2. Romans allemands

Le traitement des romans en allemand est globalement similaire à celui des romans en français, bien que quelques différences existent. En effet, la structure des romans étant identique, certains programmes ont pu être utilisés à la fois pour les romans en français et ceux en allemand. Mais des traitements spécifiques ont dû se faire en supplément pour ce type de romans.

2.1. Gestion des erreurs spécifiques

Une partie importante des traitements concernant les romans en allemand est la gestion des erreurs dites spécifiques, c'est-à-dire les erreurs un tant soit peu semblables à celles trouvées dans les romans en français mais dont le traitement a dû être différent par moments.

2.1.1. Erreurs résultant de la conversion

De façon identique aux romans français, et même au corpus Phraseobase, des erreurs ont subsisté après la conversion du format .xml vers le format .txt. Ces erreurs sont essentiellement des mots coupés en deux qui doivent être réunis, et des *underscores* apparaissant à la place des tirets de dialogue. Un programme jumeau de *text_suppr_n.py* a donc été créé, appelé *text_modif_de.py*, afin de corriger ces erreurs. Son fonctionnement est sensiblement le même que celui de son homologue, développé dans le point « Nettoyage », il se base simplement sur les erreurs concernant les mots en allemand. Les données en entrée sont au format .txt et le restent après le traitement.

2.1.2. Erreurs d'OCR

En ce qui concerne les erreurs d'OCR retrouvées dans les romans en allemand, ces dernières n'étaient pas évidentes à repérer en raison du niveau en allemand des membres du

projet basés à Grenoble. En se basant sur les cas retrouvés dans les romans en français, il était logique de penser que des erreurs similaires devaient apparaître dans les romans en allemand. L'outil NooJ a alors été à nouveau utilisé pour rechercher les mots inconnus en allemand présents dans les textes.

Les données utilisées lors de l'analyse des romans en français (les différentes grammaires et le dictionnaire), ainsi que celles concernant les romans en allemand, m'ont été transmises afin que je puisse reproduire le protocole. L'utilisation de NooJ faite par Agnès Tutin devait me servir d'exemple pour réaliser ensuite la même analyse au niveau des textes en allemand. Cependant, je me suis aperçue que la version classique de l'outil n'était pas disponible pour le système d'exploitation Linux, avec lequel j'effectuais mon stage. J'ai donc téléchargé la version Java de NooJ, qui elle est compatible avec le système Linux. Toutefois, les données fonctionnant avec la version classique sont soit au format .nod pour les dictionnaires, soit au format .nom pour les grammaires. Or, la version Java de NooJ exige un dictionnaire au format .jnod et des grammaires au format .jnom. La réalisation de l'analyse des textes en allemand a donc été impossible à effectuer avec le système Linux, mais Agnès Tutin travaillant avec le système Windows a pu achever une partie de leur analyse de son côté. En effet, la totalité du corpus en allemand était d'une taille trop importante pour l'outil. Il a donc fallu procéder à la correction des erreurs systématiques dans les textes en allemand de façon manuelle : cette vérification a été confiée à l'équipe de l'ATILF car nos connaissances en allemand sont loin d'être suffisantes pour gérer ces changements.

Cependant, l'équipe de l'ATILF n'ayant pas suffisamment de temps, ni pour corriger manuellement les erreurs repérées, ni pour créer un fichier Excel semblable à celui utilisé pour corriger les romans français (cf. point « OCR »), cette étape n'a pas été réalisée avant la gestion de l'alignement. Les romans en allemand possèdent donc un traitement de moins que les romans en français.

2.1.3. Erreurs de mise en page

Concernant les erreurs de mise en page, celles-ci sont assez similaires à celles repérées dans les romans en français. En effet, des balises héritées des fichiers XML étaient présentes dans le corps des textes, mais aussi des numéros de paragraphes indésirables et des titres (cf. point « Mise en page »). Le script *specific_search.py* a à nouveau servi à repérer ces éléments. Cependant, des caractères non connus ont également été repérés dans certains textes, ainsi que des entités composées de 3 lettres n'ayant pas perturbé la conversion mais ne pouvant pas

permettre à Stanza d'effectuer une analyse correcte. Le pendant du programme *titre_suppr_fr.py* a donc été créé, appelé *titre_suppr_de.py*, prenant en entrée des fichiers au format .txt et ne subissant pas de changement de format. De la même façon que pour son homologue agissant sur les romans en français, la première version de ce script a été testée en local avant de modifier tous les textes sur le serveur pour s'assurer de son bon fonctionnement. Une vérification manuelle a également été mise en place afin de repérer les cas où un titre coupe un mot en deux, et où la regex de suppression doit donc être adaptée à la situation.

Le caractère inconnu le plus présent dans les textes en allemand est « ﬠ », il a été supprimé grâce à des expressions régulières. Quant aux entités de 3 lettres, ces dernières apparaissaient à la place de caractères accentués allemands. Il y avait par exemple la suite « HOP » correspondant à « ü », la suite « IND » correspondant à « ä », ou encore la suite « CCH » correspondant à « ö » (cf. Annexe 19). Afin d'identifier quel caractère accentué devait être présent au niveau de l'entité, l'aide du traducteur en ligne **Google Traduction**[37] a été très précieuse. En effet, il a suffi de copier le texte contenant les entités dans le traducteur et de lui demander de traduire les phrases en allemand. Ainsi, les caractères accentués apparaissaient dans la traduction, et pouvaient être remplacés à l'aide d'une regex (cf. Annexe 20).

2.2. Autres traitements spécifiques

Les erreurs spécifiques développées dans la partie précédente ne sont pas les seules à nécessiter un traitement particulier au niveau des romans en allemand. En effet, d'autres étapes réalisées ont également eu besoin d'une adaptation différente des romans en français.

2.2.1. Annotation du discours direct

La correction des erreurs présentes dans le format .xml ayant été faite en même temps que pour les romans en français, la première étape de traitement des romans en allemand a été celle de l'annotation des parties de discours direct avec les scripts de Zhiyuan Qiu. Les premier, troisième, quatrième et cinquième programmes ont été adaptés en modifiant les chemins d'accès des fichiers, mais le deuxième a nécessité quelques changements. En effet, les guillemets allemands diffèrent des guillemets français, il a donc fallu les inclure dans l'expression régulière cherchant les guillemets ouvrants et fermants pour annoter les passages de discours direct (cf. point « Annotation du discours direct »).

[37] https://translate.google.fr/?hl=fr

Afin de prendre en compte un maximum de guillemets possibles, cette version du script recherche à la fois les guillemets français, allemands et anglais, car une vérification rapide a permis de se rendre compte que des guillemets anglais sont également présents dans ces romans. Ce doublon a été identifié par son nom : *Regle_2.1_de.py*.

2.2.2. Analyse et mise en page

Concernant l'analyse, le programme utilisé pour analyser les romans en français du corpus pouvait être réutilisé sur les romans en allemand, à condition de télécharger le bon modèle de langue. Une fois ce paramètre renseigné, l'analyse s'effectuait de la même façon que pour les romans en français et les romans du corpus Phraseobase (cf. point « Lancement de l'analyse »). Le doublon créé se nomme *analyse_de.py*. Les erreurs repérées dans les romans en allemand sont similaires à celles apparaissant dans les romans en français, les programmes *stanza_modif.py* et *mep.py* ont donc été réutilisés sur ces fichiers.

Le plus gros changement observé entre les romans en français et ceux en allemand au sujet de l'analyse se situe au niveau de la mise en page des sorties produites par Stanza. En effet, le modèle de langue utilisé étant différent, les sorties obtenues le sont également. Ce qui ressort de la comparaison des fichiers au format .conllu des romans en français, et de ceux des romans en allemand, est la présence d'informations supplémentaires dans les fichiers des romans en allemand. Effectivement, comme le montrent les annexe 21 et annexe 22, la ligne contenant des informations supplémentaires et spécifique à la langue sur la partie du discours (cf. point « Stanza ») est vide dans les fichiers au format .conllu des romans en français (représentée par un *underscore*), tandis qu'elle contient des éléments dans les fichiers au format .conllu des romans en allemand. Cette différence paraît insignifiante, mais elle a son importance pour les étapes de correction suivantes, développées dans la partie ci-après.

2.3. Dictionnaire de formes fléchies

De la même manière que pour le corpus Phraseobase et les romans en français, Stanza a produit des erreurs de lemmatisation nécessitant l'utilisation d'un dictionnaire de référence afin de procéder à leur correction. Cependant, pour des raisons évidentes, il est impossible de se servir du dictionnaire des formes fléchies du français dans le cas des romans en allemand. Il a donc fallu trouver un équivalent au GLAFF pour l'allemand.

2.3.1. Création du dictionnaire

Afin de créer un dictionnaire de référence similaire à celui qu'Olivier Kraif a construit pour le français, il a tout d'abord fallu chercher un équivalent du GLAFF pour l'allemand. À force de recherches, le **Dereko**[38] est apparu comme un bon candidat pour ce rôle. En effet, ce dernier est disponible et téléchargeable gratuitement au format .freq, et présente une liste de formes, lemmes, catégories syntaxiques et fréquence. Il date de 2014 et est la version la plus récente et complète créée par l'Institut de la langue allemande. L'annexe 23 présente la version de base du Dereko, c'est-à-dire sans les modifications réalisées par la suite dans l'intérêt des fichiers.

Ce dictionnaire de formes fléchies est certes très complet et pratique pour effectuer les corrections des lemmes erronés, mais en l'état actuel il n'est pas utilisable sans réaliser des modifications soit dans le programme de correction des lemmes, soit sur lui-même. Le choix adopté a été la modification du dictionnaire, car il devait déjà subir des transformations visant à le rendre plus propre : ainsi seul un fichier subirait des changements au lieu de deux.

Le premier changement opéré sur le dictionnaire a été la conversion du format .freq au format .csv afin de pouvoir ensuite le convertir du format .csv au format .dat, et obtenir une version plus efficace lors des remplacements. Ensuite, la colonne contenant les fréquences des mots a été supprimée car elle est inutile dans notre cas, ce qui rend le dictionnaire plus propre. Puis, afin de se rapprocher au maximum de la structure du dictionnaire de formes fléchies du français dans le but de ne pas avoir à modifier le script de correction des lemmes, les colonnes des lemmes et des catégories syntaxiques ont été inversées. En effet, l'annexe 15, présentant un extrait du dictionnaire de formes fléchies du français, montre un dictionnaire organisé en forme-catégorie-lemme. Or, la version de base du Dereko est organisée en forme-lemme-catégorie-fréquence. L'annexe 24 présente le dictionnaire de formes fléchies de l'allemand tel qu'il a été utilisé dans le cadre de ce projet.

2.3.2. Correction de la lemmatisation

Grâce à la création et à l'adaptation du dictionnaire de formes fléchies de l'allemand, il est désormais possible de corriger les lemmes erronés dans les sorties de Stanza. Pour cette étape, le programme *conllu_verif.py* a été récupéré et adapté, mais son fonctionnement reste le même : lorsque le lemme est inconnu, il est remplacé par la forme correspondante. Cependant,

[38] https://www.ids-mannheim.de/digspra/kl/projekte/methoden/derewo/

comme précisé dans le point « Analyse et mise en page », le fait que la ligne contenant des informations supplémentaires et spécifiques à la langue sur la partie du discours ne soit pas vide dans le format .conllu des fichiers en allemand a son importance.

En effet, en comparant les catégories syntaxiques renseignées dans le dictionnaire de formes fléchies du français (cf. Annexe 15) et celles dans le dictionnaire de formes fléchies de l'allemand (cf. Annexe 24), une différence non négligeable apparaît. De fait, les catégories syntaxiques renseignées dans le dictionnaire extrait du GLAFF correspondent à la 3ème colonne présente dans le format .conllu, tandis que celles présentes dans le dictionnaire du Dereko correspondent à la 4ème colonne du format .conllu. Cette différence nécessite une adaptation du programme gérant la correction des lemmes : les 4 catégories contrôlées dans les fichiers français deviennent 25 catégories à contrôler dans les fichiers allemands (cf. point « Correction automatique »). En effet, au lieu de contrôler uniquement les verbes, adverbes, noms et adjectifs, il faut vérifier ici les différentes catégories de verbes (à l'infinitif, fini, à l'impératif, à l'infinitif complet…), d'adjectifs (attributif et adverbial) etc. Cela augmente considérablement le temps nécessaire au traitement des lemmes.

De plus, la clé utilisée pour les fichiers français était composée de la forme en minuscules, suivie d'une tabulation, suivie de la catégorie syntaxique. Celle utilisée dans les fichiers allemands est composée de la forme telle qu'elle est présente dans le fichier, suivie d'une tabulation, suivie de la catégorie syntaxique. Ce changement permet de prendre en compte les noms communs qui commencent par une majuscule en allemand, et qui apparaissent donc avec une majuscule dans le dictionnaire. Le fait de transcrire toutes les formes présentes dans les fichiers en minuscules engendre des absences de remplacements car les formes ne sont simplement pas trouvées dans le dictionnaire, donc pas corrigées dans les fichiers .conllu.

3. Alignement

Les traitements sur les romans en français et en allemand réalisés, l'étape suivante fut celle de l'alignement. Celui-ci s'est fait grâce à une liste des romans, mais aussi de nouvelles bibliothèques de Python. Ce point est important pour la mise en contraste et la comparaison des phrases préfabriquées en allemand et en français, comme avancé dans la partie « Chapitre 3. Pourquoi ce sujet de recherche ? » point « Objectifs du projet ». En effet, le corpus romanesque émanant du GLFA, cet axe d'analyse est privilégié pour les chercheurs de l'ATILF.

3.1. Appariement des fichiers

La première étape de cet alignement a été l'appariement des fichiers. En effet, les romans et leur traduction ne comportaient pas forcément le même titre, et plusieurs romans pouvaient appartenir au même auteur. Afin de s'assurer du bon fonctionnement des traitements futurs, une vérification de la compatibilité des romans a été indispensable.

3.1.1. Création d'une liste des romans

À ce stade, les romans du corpus GLFA sont enregistrés dans différents répertoires sur le serveur selon leur provenance. En effet, comme stipulé dans le point « Description des corpus », les romans sont classés en fonction de leur langue source et de leur statut de traduction. Afin de conserver ce classement de fichiers, tout en permettant un appariement des romans en langue source avec leur traduction en langue cible, le choix de créer une liste des romans et de leur traduction est apparu comme évident.

Avant toute chose, il a fallu dresser une liste des romans en français langue source ainsi que de leur traduction en allemand. Ces fichiers étaient au nombre de 205 pour chaque catégorie, soit un total de 410 romans. Les romans en allemand langue source ainsi que leur traduction en français ont également fait l'objet d'un listage, mais ces derniers n'étaient que 35 dans chaque catégorie, soit un total de 70 fichiers. L'étape suivante a été la comparaison des noms d'auteurs et des titres des romans dans chaque catégorie afin d'identifier les paires de romans. Dans les cas où l'appariement n'était pas évident, une recherche d'entités uniques telles que des noms de personne ou de lieu a été opérée à l'intérieur des romans. De cette façon, les ressemblances ont pu être mises à jour, et les fichiers ont pu être reliés.

Une fois les romans et leur traduction repérés, il y avait 480 romans à aligner automatiquement, donc 240 paires. Ces romans ne pouvaient évidemment pas être alignés un par un, il fallait trouver un moyen de transmettre au programme la liste des romans appairés, mais également leur emplacement, car ces derniers sont dans 4 répertoires différents. Pour ce faire, le format .tsv a été utilisé pour accueillir les listes obtenues précédemment. La première colonne du tableau affichait les fichiers en français (qu'ils soient en langue source ou en langue cible) et la seconde colonne affichait les fichiers en allemand (cf. Annexe 25). De cette manière, le programme pouvait s'appuyer sur le tableau pour récupérer les 2 fichiers à analyser conjointement.

Cependant, renseigner le nom des romans et de leur traduction n'était pas suffisant pour que le programme puisse les trouver. En effet, cela aurait marché dans le cas où les romans et la liste des romans s'étaient trouvés dans un même répertoire. Or, le fait que les romans soient classés dans plusieurs répertoires oblige la liste des romans à contenir également le chemin d'accès des fichiers. La version finale de cette liste est donc différente de celle présentée sur l'annexe 25, elle est plus complète mais moins lisible, d'où le choix de partager la première version.

3.1.2. Création de nuages de points

Grâce à la liste gérant l'appariement des romans français et allemands, l'étape suivante a pu être entamée. Celle-ci consiste à vérifier la correspondance entre les fichiers français et allemands appairés. Pour ce faire, il a fallu obtenir un bon nuage de points grâce à la bibliothèque *Matplotlib* de Python. Cette fonctionnalité permet de créer des graphiques au format .png basés sur les données fournies, et s'active avec « import matplotlib.pyplot ».

L'analyse se fait grâce à un **préprocesseur**[39] et un **encodeur**[40] basés sur BERT, un modèle de langage pré-entraîné utilisé dans le traitement automatique des langues dont le fonctionnement est expliqué dans la partie « Outils de traitement automatique des langues » point « Bert ». Ces derniers proviennent de *Labse* (Language-specific BERT Sentence Embedding), une variante de BERT entraînée spécifiquement pour des langues autres que l'anglais, et sont implémentés dans le script d'alignement *AIlign*.

Les 2 fichiers composant la paire à analyser sont d'abord lus phrase par phrase, puis chaque phrase du premier texte est comparée à chaque phrase du second texte pour obtenir des points de similarité. Les points candidats sont créés lorsqu'on obtient, entre une phrase x et une phrase y, un seuil de similarité supérieur à la valeur définie (paramètre labseThreshold). Ensuite, on ne retient pour chaque colonne ou chaque ligne que les k points obtenant les meilleurs scores (paramètre kBest). Un filtrage est effectué dès cette étape avec le paramètre margin, qui permet de ne retenir que les points ayant un score supérieur à la valeur choisie par rapport à celui de leur meilleur concurrent.

On applique ensuite un filtre passe-haut en deux étapes. Le premier filtrage s'appuie sur un calcul de la densité de candidats autour de chaque point candidat. On ne calcule pas cette

[39] https://tfhub.dev/google/universal-sentence-encoder-cmlm/multilingual-preprocess/2
[40] https://tfhub.dev/google/LaBSE/2

densité dans un carré centré autour du point mais plutôt dans un couloir centré sur la diagonale qui passe par le point (le chemin d'alignement suivant en général cette diagonale). La largeur de ce couloir correspond au paramètre deltaY. La longueur de ce couloir correspond au paramètre deltaX. Le nombre de points candidats divisé par la taille de cet espace donne une valeur de densité. Si cette densité, divisée par la densité moyenne de tous les points candidats, est supérieure à un certain ratio (paramètre minDensityRatio) alors le point est conservé. Ce premier filtrage correspond à la fonction *filterPoints()*.

Le deuxième filtrage, correspondant à la fonction *resolvingConflicts()*, s'attache à résoudre les conflits respectivement sur les axes verticaux et horizontaux. Cela se produit quand pour une même coordonnée x, on a plusieurs points avec des y différents, et réciproquement, pour une même coordonnée y on a plusieurs points avec des x différents. La suppression des concurrents s'effectue sur la base de la densité : on ne garde que le point qui obtient une meilleure densité le long de sa diagonale. Ce filtrage par densité peut être réitéré une fois si le paramètre reiterateFiltering est donné.

Ces points forment un nuage affichant le taux de correspondance entre le fichier 1 et le fichier 2. Grâce à ces filtrages, les points significatifs sont repérés et affichés en noir sur le graphique. Les autres points apparaissent en rouge. Pour qu'une paire de romans soit considérée comme correctement appairée, il faut que le graphique obtenu représente une diagonale de points noirs allant du bord inférieur gauche au bord supérieur droit de l'image. La position et le nombre de points rouges n'a pas d'importance sur le graphique, dès lors qu'une ligne droite de points noirs est présente (cf. Annexe 26).

La création de nuages de points a d'abord été testée en local sur la machine paire par paire, ce qui a permis de repérer une paire dont la traduction ne correspondait pas, et qui a donc été écartée. Mais le temps d'exécution était bien trop long pour effectuer l'analyse sur toutes les paires ainsi. C'est pour cela que la liste des romans a son intérêt : grâce à elle, les paires roman-traduction ont été repérées automatiquement et analysées les unes après les autres en n'exécutant le programme qu'une seule fois. Les images obtenues ont ensuite pu être vérifiées dans l'éventualité d'autres mauvais appariements, d'autres textes à éliminer du corpus, ou de textes nécessitant des traitements spécifiques. Les annexe 27 et annexe 28 montrent respectivement un nuage de points de textes non appairés, et un nuage de points de textes bien appairés mais dont la correspondance n'est pas totalement identique.

3.2. Utilisation du Dynamic Time Warping

À partir du moment où les fichiers étaient bien appariés, c'est-à-dire qu'ils avaient un nuage de points satisfaisant, la prochaine phase a pu être abordée. Il s'agit de l'utilisation du Dynamic Time Warping (DTW).

3.2.1. Définition et application

Le Dynamic Time Warping, ou déformation temporelle dynamique en français, est un procédé permettant de mesurer la similarité entre deux séquences pouvant varier au cours du temps. Le principe est de déterminer le chemin optimal entre les points des phrases comparées. Une fois ce chemin trouvé, il est possible de calculer un score de similarité entre les textes.

Le DTW a été appliqué aux romans français et allemands en considérant chaque mot des textes comme un point dans l'espace-temps. Il a ainsi permis de comparer des séquences de longueurs différentes, et de détecter des variations de vitesse ou des déformations temporelles entre les mots. Il suffisait pour cela d'ajouter la commande *runDTW* dans la ligne de commande, qui faisait appel à une fonction créée par Olivier Kraif dans le script d'alignement.

Chaque mot était représenté par un point dans l'espace-temps, permettant à l'algorithme de calculer le meilleur alignement entre ces points, tout en minimisant la distance cumulative entre les séquences comparées. Le chemin obtenu représente la correspondance entre des groupes de phrases sources et cibles. Une fois le chemin optimal trouvé, le calcul du score de similarité entre les textes se fait en utilisant la distance totale du chemin. Un score faible indique une forte similarité entre les deux textes, tandis qu'un score élevé indique une faible similarité. Ce score peut ensuite servir à analyser la précision de la traduction.

3.2.2. Problème rencontré

Le Dynamic Time Warping est une étape très coûteuse en termes de mémoire. En effet, la première utilisation sur tous les romans et leur traduction via la liste créée précédemment a engendré une saturation de la mémoire du serveur. Plusieurs solutions ont été envisagées à ce stade, dont le découpage de la liste en portions plus petites et l'analyse des textes par bloc de 5 ou 10 paires, mais cela n'a pas fonctionné. Une tentative paire par paire a alors été tentée, sans plus de succès.

Cela était dû à une fonction permettant de conserver les paramètres utilisés lors des lancements afin de pouvoir les modifier si nécessaire. En effet, les paramètres étaient

enregistrés dans un fichier grâce à la commande *useShelve,* et ce dit fichier avait atteint une taille de plus de 10 Go. En supprimant ce fichier du serveur et en n'utilisant plus la commande de sauvegarde des paramètres, le DTW a pu être exécuté sans engendrer une saturation de mémoire.

3.2.3. Résultats obtenus

Lors du lancement de l'alignement en prenant en compte le Dynamic Time Warping, une nouvelle version du script avait été créé par Olivier Kraif. Celle-ci avait la même fonction que la version précédente, mais quelques paramètres différaient. Ainsi, les nuages de points ne renvoyaient plus une ligne de points noirs entourée de points rouges, mais simplement une ligne de points noirs. Les autres fichiers étaient assez similaires, ils seront développés dans une prochaine partie.

Au niveau des résultats renvoyés par le DTW, concrètement les scores de chaque paire de textes analysés, ces derniers étaient assez différents selon les paires. En effet, comme précisé dans le point « Définition et application », plus le score est faible plus la similarité est forte, et plus le score est fort plus la similarité est faible. Pour pouvoir comparer les scores, il faut diviser celui-ci par le nombre de phrases alignées. Cependant, les scores obtenus ne correspondaient pas toujours à des scores chiffrés permettant une comparaison.

Factuellement, 3 types de retours du DTW ont été observés. Il s'agit soit de l'apparition du score chiffré signifiant que le processus a fonctionné, soit d'un score considéré comme infini, soit d'une erreur ayant empêché le processus de se poursuivre. Comme annoncé dans le point « Création de nuages de points », une paire de textes a déjà été écartée car la traduction ne correspondait pas au texte source. Une autre paire a également été supprimée du corpus car elle ne correspondait ni à du français langue source ni à de l'allemand langue source, c'était un texte en anglais traduit en français et en allemand.

Au début du lancement du processus de DTW, il restait donc 238 paires de textes à analyser. Sur ces 238 paires, 25 d'entre elles ont abouti à un échec en raison d'un mauvais nuage de points, d'un nuage comportant un trou, voire d'un nuage de points inexistant. Cela peut s'expliquer soit par un mauvais appariement des fichiers, donc une traduction trop éloignée de la version d'origine, soit par les différents traitements effectués sur les romans en français et pas en allemand (cf. points « Erreurs spécifiques dans les textes » et « Gestion des erreurs spécifiques »). En effet, le fait d'avoir corrigé certaines erreurs de mise en page dans des textes

d'origine, mais pas dans leur traduction peut engendrer des différences et perturber l'alignement et le DTW.

Il faut également citer les 106 paires ayant obtenu un score infini. Ces dernières ne possèdent pas de nuage de points inexistant, mais peuvent avoir un nuage imparfait. C'est-à-dire que la diagonale de points noirs ne représente pas une ligne totalement droite reliant le bord inférieur gauche au coin supérieur droit. Cependant, plusieurs nuages de points observés dans ce cadre-là ne justifient pas la présence du score infini. En effet, un score infini signifie que l'algorithme n'a pas été capable de trouver un alignement satisfaisant. Une amélioration du script d'alignement est nécessaire pour prendre en compte ces cas particuliers et comprendre pourquoi aucune correspondance n'est trouvée alors que le nuage de points est correct.

Actuellement, sur les 238 paires de textes analysés :

- 25 ont été victimes d'un échec de traitement
- 106 ont renvoyé un score infini
- 107 ont obtenu un score
- environ 45% des paires ont obtenu un score convenable

Au vu de ces chiffres, il est clair que des améliorations peuvent être apportées afin d'augmenter la précision du DTW et d'obtenir de meilleurs résultats par la suite.

3.3. Le script d'alignement

Le programme permettant l'alignement complet m'a été fourni par Olivier Kraif, mais quelques modifications ont dû y être apportées. En effet, la version d'origine du script prenait en entrée seulement 2 fichiers : il fallait l'adapter pour prendre en compte la liste des romans et de leur traduction. De plus, la gestion des fichiers de sortie du script devait également être revue en raison de la prise en compte de la liste des paires, et la récupération des scores du Dynamic Time Warping dans un fichier texte était de rigueur.

3.3.1. Prise en compte des paires repérées via la liste

La partie de code gérant la prise en compte de la liste des paires de romans s'est faite par le biais d'une fonction. Celle-ci initialise 2 tableaux correspondant aux fichiers français et allemands, puis lit la liste ligne par ligne en séparant chaque ligne lorsqu'il rencontre une tabulation.

La première valeur présente dans la ligne est alors ajoutée au tableau des fichiers français et la seconde est ajoutée au tableau des fichiers allemands. Les paires sont ensuite construites grâce aux tableaux maintenant remplis de fichiers : le premier fichier du premier tableau est comparé avec le premier fichier du second tableau, et ainsi de suite jusqu'à ce qu'il n'y ait plus aucun fichier dans les 2 tableaux.

Au lieu de renseigner les noms des 2 fichiers d'entrée dans la ligne de commande, ce qui signifie devoir exécuter le programme autant de fois qu'il y a de paires de romans à analyser, il a suffi de renseigner le nom de la liste utilisée et d'exécuter le programme une seule fois. Encore une fois, le but était de gagner en temps et en énergie.

Exemple de ligne de commande sans la liste de paires :

> *python3 ailign.py --inputFile1 "/home/hecquetf/phraseorom_annotation/ROMANS/romans.de/romans.de.xml/fr/Ultime/bettina violet_le sauvage enfant-lion.fr.xml" --inputFile2 "/home/hecquetf/phraseorom_annotation/ROMANS/romans.de/romans.de.xml/Ultime/violet bettina_das wilde löwenkind.de.xml" --inputFormat xml-conll --outputFormats txt tmx ces --outputFile bettina_violet_aligned_fr.de --savePlot --verbose --margin 0.05 --labseThreshold 0.4 --k 3 --deltaX 60 --minDensityRatio 0.5 --save2shelve*

Exemple de ligne de commande avec la liste de paires :

> *python3 ailign2.py --inputFileList "appariement_fr-de.tsv" --inputFormat xml-conll --outputFormats txt tmx ces --savePlot --verbose --margin 0.05 --labseThreshold 0.4 --k 3 --deltaX 60 --minDensityRatio 0.5 --save2shelve*

Les deux premiers arguments des lignes de commande correspondent à l'appel et au lancement du script d'alignement avec la version de Python appropriée (ici, la troisième). Puis, chaque argument suivant correspond soit à une fonction développée dans le programme, soit à une valeur limite. Dans la ligne de commande prenant en compte la liste des paires, le nom du fichier de sortie n'est pas précisé, contrairement à sa prédécesseuse. Cela s'explique dans le point suivant.

3.3.2. Gestion et description des sorties

Il a également été nécessaire d'ajouter une partie concernant le nom du fichier de sortie. En effet, dans la version de base du script d'alignement, le fichier en langue source et sa

traduction en langue cible étaient précisés dans la ligne de commande, ce qui permettait de renseigner également le nom du fichier de sortie.

Dans le cas où le code ne faisait pas l'objet d'une modification, le nom du fichier de sortie souhaité serait indiqué au lancement du script de la même manière. Mais toutes les paires analysées auraient alors le même nom, ce qui engendrerait un écrasement du fichier précédent à chaque nouvelle paire.

Pour remédier à ce problème, le nom du premier fichier de chaque paire a été récupéré, enregistré dans une variable et associé au nom du fichier de sortie directement. Ainsi, chaque paire analysée était enregistrée différemment et aucun fichier n'a été perdu. Le nom du fichier de sortie correspondait au contenu de la variable, étant modifié à chaque nouvelle paire analysée. Les fichiers en entrée étaient au format ConLL, et les fichiers en sorties étaient aux formats .tmx, .ces, .txt et .png.

Le fichier .tmx est un fichier signifiant Translation Memory Exchange, permettant de stocker et d'échanger des données de mémoire de traduction. La mémoire de traduction est une base de données qui enregistre des segments de texte alignés, généralement des phrases, dans une langue source ainsi que leur traduction correspondante dans une langue cible. Ce format permet de stocker ces phrases, mais aussi des métadonnées associées telle que le couple de langue comparé. Il ressemble au format .xml car il possède une syntaxe similaire composée de balises. Son utilité réside dans la possibilité de réutiliser les traductions existantes pour accélérer le processus de traduction. En effet, il est plus rapide de rechercher les phrases déjà traduites présentes dans ce fichier, plutôt que de recommencer la traduction de zéro (cf. Annexe 29).

Le fichier .ces représente l'alignement des phrases, ou segments de phrases, dans les deux langues sous forme de numéros. C'est un fichier indiquant le numéro de la ou des phrase(s) en langue source, suivi du numéro de la ou des phase(s) associée(s) en langue cible (cf. Annexe 30). Il est également utilisé dans le but de conserver les données afin de pouvoir les réutiliser par la suite.

Le fichier .txt présente les phrases des 2 romans comparés considérées comme appariées, séparées par des sauts de ligne (cf. Annexe 31). Il est alors possible de contrôler la fiabilité de l'alignement réalisé, mais également de conserver les données pour un projet ultérieur. Enfin, le fichier .png contient le nuage de points obtenu après l'analyse de similarité,

dont le développement et la fonction ont été présentés dans le point « Création de nuages de points ».

3.3.3. Récupération des scores du DTW

Dans la version du script créé par Olivier Kraif, l'utilisation de l'algorithme de Dynamic Time Warping permettait d'afficher le score obtenu pour chaque paire dans le terminal. Dans ces conditions, il était impossible de le récupérer pour pouvoir faire des comparaisons par la suite, ou bien simplement vérifier les cas problématiques. Il a donc fallu modifier une partie de la fonction afin d'enregistrer les scores de chaque paire dans un fichier texte à part.

Pour cela, la méthode *append()* de Python a été requise afin de sauvegarder un à un les scores sortis par l'algorithme dans un fichier texte. L'utilisation de la méthode *append()* dans ce cas, et non de la méthode *write(),* se justifie par le fait que celle-ci ajoute une ligne à la fin d'un fichier texte sans modifier les lignes précédentes. Contrairement à la méthode *write()* qui va d'abord supprimer le contenu du fichier texte existant avant d'écrire la nouvelle ligne. Le but étant de pouvoir créer une liste des scores obtenus pour chaque paire, il fallait que chaque ligne soit conservée et qu'aucune donnée ne soit écrasée au profit d'une autre.

Ce procédé a fonctionné comme prévu, cependant de petits ajustements manuels ont été indispensables pour profiter de l'utilité de cette liste de scores. En effet, le fichier texte créé ne contenait que les scores ligne par ligne, aucune information sur la paire de textes analysés n'était présente. Il a donc fallu tenir compte du fichier .tsv contenant la liste des paires, ainsi qu'une liste conservant les paires ayant rencontré des erreurs et pour lesquelles le procédé n'a pas abouti. De cette façon, il a été plus facile de retrouver à quelle paire appartenait chaque score, et de les ajouter manuellement sur la liste (cf. Annexe 32). Lorsqu'un auteur ne possédait qu'un texte dans le corpus, seul son nom a été ajouté avant le score. Dans le cas où un même auteur possédait plusieurs ouvrages, le titre du roman a été écrit en plus du nom afin d'éviter toute confusion.

Le travail concret réalisé lors de ce stage a été développé dans cette partie, la partie suivante concerne les retours, perspectives envisagées et conclusions.

Partie 3
-
Conclusion, perspectives et bilans

Chapitre 7. Retour sur les missions

Plusieurs missions m'ont été confiées lors de ce stage. Celles-ci étaient toutes liées au traitement automatique de corpus et à la programmation avec le langage Python. Cependant, elles n'ont pas nécessité le même temps de traitement, n'ont pas fait appel aux mêmes compétences, et n'ont pas obtenu les mêmes résultats.

1. Analyse et conversion des corpus

La première mission m'ayant été demandée était la réanalyse des corpus du projet Phraseobase provenant du projet ANR PHRASEOROM (Novakova & Siepmann (2020), cité par Tutin 2021), tout en annotant le discours direct, puis la conversion au format XML ConLL. Après avoir finalisé ces étapes, le schéma a dû être répété sur le corpus GLFA de l'ATILF.

1.1. Analyse

Au niveau de l'analyse des corpus, celle-ci a été réalisée grâce à l'outil Stanza pour le corpus Phraseobase ainsi que pour les romans en français et en allemand du corpus GLFA, dont le fonctionnement est expliqué dans le point « Utilisation de Stanza ». Comme stipulé précédemment, la manière dont l'analyse est faite n'est pas contrôlable. Les seuls paramètres pris en compte à l'entrée de l'analyse sont la langue dans laquelle l'analyse doit être exécutée, et quelques conditions telle que l'annulation de la segmentation dans le cas où les phrases étaient séparées par 2 retours à la ligne consécutifs.

Le lancement de l'analyse, ainsi que l'analyse en elle-même ont été une réussite au premier abord. En effet, les phrases contenues dans les textes ont toutes été analysées par l'outil, bien que certaines erreurs de traitement aient été repérées par la suite ainsi que des problèmes de double analyse. Ces aspects inattendus ont retardé les opérations futures en nécessitant des corrections post-analyses. Ces traitements supplémentaires ne sont pas synonymes d'échec de l'analyse, mais apportent des éléments d'améliorations potentielles qui seront développés dans la partie « Chapitre 8. Perspectives ». La lenteur d'exécution de l'analyse peut également être soulignée, et peut être améliorée en contraignant l'analyse à se faire sur un nombre restreint de phrases appelé *batches*. Ce procédé peut permettre de gagner du temps.

De plus, le fait d'ajouter des rectifications à la suite des analyses, afin de rendre ces dernières plus propres et de limiter les erreurs, offre la possibilité d'accroître ses connaissances et ses capacités en matière de programmation, en les mettant en pratique dans de nouvelles

conditions. Ainsi, des erreurs engendrant une perte de temps au niveau global du projet peuvent également apporter des compétences supplémentaires. Ce n'est donc pas une perte.

1.2. Conversion du XML au XML ConLL

Concernant la conversion du format .xml au format XML ConLL, celle-ci ne s'est pas réalisée exactement de la même manière pour le corpus Phraseobase et le corpus GLFA. En effet, les fichiers d'origine possédaient une syntaxe similaire mais pas totalement identique. Cela a permis de conserver des scripts (la conversion du format .xml au format .txt) pour l'un et l'autre des corpus, mais a nécessité quelques ajustements pour d'autres scripts (les suppressions).

La première conversion était celle du format .xml au format .txt. Quel que soit le corpus, cette étape laissait transparaître de nombreuses erreurs qui ont dû être corrigées grâce à d'autres scripts. Ces erreurs étaient, pour la plupart, communes aux romans des 2 corpus, qu'ils soient en français ou en allemand. Elles ont cependant nécessité un temps assez important de traitement afin de les corriger, ce qui ouvre la porte à des hypothèses de perfectionnement disponibles dans la partie « Chapitre 8. Perspectives ». À la suite de l'analyse, le format .txt avait été automatiquement converti au format .conllu, qui lui aussi a subi des modifications comme précisé dans le point précédent.

La partie relative à l'insertion du format .conllu dans le format .xml, quant à elle, n'a pas engendré de problèmes exigeant de multiples corrections, mais a nécessité des adaptations en fonction du corpus traité. Cela est dû aux différences de syntaxe précédemment citées. Ces corrections ne peuvent malheureusement pas amener de possibilité d'améliorations, car il est normal que chaque corpus possède sa propre syntaxe et que les programmes devant effectuer des traitements dessus aient à s'adapter à chacune de ces syntaxes. Ces adaptations offrent également des mises en pratique de compétences, assurant ainsi une progression. La conversion du format .xml au format XML ConLL, ainsi que l'annotation du discours direct, peuvent néanmoins être considérées comme une réussite.

2. Vérifications et corrections

La partie de vérification et de correction n'était pas vraiment définie au début de ce stage. Il était évident que des opérations de ce genre risquaient d'être nécessaires, mais il était impossible de décrire précisément sur quoi celles-ci allaient porter.

2.1. Syntaxe XML

Les premières corrections apportées aux corpus concernaient la syntaxe des fichiers XML recueillis. En effet, comme explicité dans le point « Conversion », les fichiers récupérés contenaient beaucoup d'erreurs empêchant tout traitement. À ce stade, les corpus étaient inutilisables.

Grâce au module *lxml* proposé par Python, dont le fonctionnement est expliqué dans le point « Correction », le repérage et l'analyse des erreurs a été très rapide. Les fichiers erronés étant automatiquement repérés par le script prévu à cet effet, et les erreurs étant indiquées précisément, la correction n'en a été que plus simple et n'a posé aucun problème.

La vérification et la correction de cette syntaxe ont été un succès, car sans cela aucun traitement n'aurait pu être effectué. De plus, la nécessité de rectifier la syntaxe a permis à la fois de découvrir et de se familiariser avec le module *lxml*. Des compétences ont alors été développées grâce à cette étape.

2.2. Sorties de Stanza

Concernant les sorties obtenues après le lancement de l'analyse de Stanza, celles-ci ont demandé plusieurs opérations. En effet, quelques problèmes ont été remarqués dans les fichiers produits, au niveau de la lecture des fichiers mais aussi de l'analyse en elle-même.

2.2.1. Lecture des fichiers

Premièrement, la lecture des fichiers a engendré des problèmes au niveau de l'analyse de Stanza, car certains textes possédaient leur première partie analysée en double comme expliqué dans le point « Modification des sorties obtenues ». Pour remédier à ce problème, il a fallu créer un nouveau script permettant de repérer ces parties analysées en double et de les supprimer.

L'utilisation de ce script était pratique pour gérer tous les fichiers en un seul lancement. Étant donné le nombre de fichiers dans chaque corpus traité, il était évident qu'une vérification manuelle était inenvisageable. Néanmoins, cette manière de faire a engendré une obligation de contrôle humain des fichiers corrigés par ce script, car tous les textes ne possédaient pas ce type d'erreur. Dès lors qu'un fichier qui n'avait pas de partie analysée en double était traité par ce script automatique, celui-ci perdait la totalité de son contenu. Pour ne pas se retrouver avec un

fichier vide, il fallait procéder à une vérification de la taille des fichiers et à la récupération du fichier de l'étape précédente lorsqu'un fichier vide était rencontré.

Cette étape peut être considérée comme une réussite car, même si une attention manuelle est requise pour éviter de perdre du contenu entre les opérations effectuées, plus aucun fichier ne contient de partie analysée en double. De plus, le corpus est considéré comme plus propre après ce traitement qu'avant. Enfin, la création du script a permis de développer des compétences de programmation en se confrontant à des erreurs de traitement et en les rectifiant étape par étape jusqu'à obtenir la version optimale.

2.2.2. Lemmatisation

Deuxièmement, les sorties de Stanza contenaient des erreurs de lemmatisation qui ont été repérées grâce à une vérification manuelle des résultats. La source de ces erreurs est inconnue, l'analyse ne peut pas être contrôlée, mais une correction était nécessaire. Pour ce faire, une comparaison avec un dictionnaire de formes fléchies a été effectuée (cf. point « Vérification des lemmes de Stanza »).

La création du script gérant cet aspect a posé de nombreux problèmes, même si son fonctionnement semble simple. En effet, la formation de la clé servant de repère ainsi que la recherche dans le dictionnaire n'ont pas été évidentes à mettre en place. De même pour effectuer le remplacement du lemme par la forme correspondante. Plusieurs essais ont été nécessaires avant d'obtenir des remplacements satisfaisants.

Cette phase peut être définie comme réussie, car le script créé est capable à la fois de corriger de nombreuses erreurs en remplaçant le lemme par la forme, mais aussi de prendre en compte les cas où le lemme est vide. Ce changement n'est pas parfait car le lemme d'un token et sa forme sont différents, mais cela limite les erreurs. Il vaut mieux avoir la forme d'un token à la place de son lemme plutôt qu'un lemme totalement inventé ou vide. C'est le choix qui a été adopté ici.

2.3. OCR

Au sujet de l'OCR, ce procédé est très pratique mais contient encore de nombreuses lacunes se retrouvant dans les corpus traités dans le cadre de ce projet. Plusieurs modifications ont été essentielles pour tenter de limiter ces erreurs.

2.3.1. Mots inconnus

Le premier aspect nécessitant des vérifications et des corrections en matière d'OCR est l'apparition de mots inconnus dans les fichiers. En effet, le procédé d'OCR étant basé sur un scan des romans (cf. point « Corrections des sorties des outils »), de nombreuses erreurs ont été repérées notamment grâce à l'emploi de l'outil NooJ (cf. point « OCR »).

Ces erreurs ont fait l'objet d'une correction sélective basée sur la certitude du remplacement à effectuer. En effet, dans le cas où la forme erronée ne pouvait pas être rattachée de manière évidente à une seule forme existante, le changement ne s'opérait pas. Ainsi, certaines erreurs ont été conservées dans les textes : cela s'explique par le fait de vouloir rendre les textes les plus propres possible. De ce fait, remplacer une erreur d'OCR par une autre erreur ne permet pas de purifier les textes.

Cette modification peut néanmoins être considérée comme fructueuse pour différentes raisons. Tout d'abord, les changements opérés ont permis de limiter le nombre d'erreurs présentes dans les textes et de les purifier. Ensuite, le fait de ne pas modifier les fautes lorsque ces dernières n'avaient pas de remplacement évident n'a pas engendré de problèmes plus importants. Les erreurs étant déjà existantes, leur non-correction est un manque de purification mais pas un ajout d'erreurs supplémentaires. À ce titre, elles ne peuvent pas être considérées comme un échec de la mission.

2.3.2. Contenus indésirables

Le second aspect basé sur l'OCR concerne les contenus ayant été scannés sur les versions papier des romans mais venant parasiter les fichiers numériques. Ces contenus correspondent la plupart du temps à des titres apparaissant au milieu de phrases, ou bien à des numéros de pages et de paragraphes (cf. point « Mise en page »).

Ce type d'erreurs a nécessité énormément de temps afin de procéder au repérage des titres et des numéros indésirables. En effet, même si l'utilisation d'un script automatique a permis de trouver rapidement les éléments recherchés, les corrections via des expressions régulières ont demandé un certain temps. De plus, les titres pouvant apparaître en milieu de phrases ou en milieu de mots, une vérification manuelle a dû se faire pour repérer les cas rares et adapter les regex en fonction.

Cette étape est considérée comme une réussite également même si, comme pour la phase précédente, certains éléments n'ont pas pu être corrigés. En effet, le repérage des titres et numéros non désirés a été efficace grâce au script de recherche automatique dans tous les fichiers, mais leur correction n'a pas toujours été simple en termes de temps. Certains éléments à supprimer ont sûrement échappé à la recherche, et par conséquent à une correction. Cependant, la difficulté à construire les expressions régulières adaptées à chaque situation a permis de développer des compétences dans ce domaine.

3. Réalisation de l'alignement

La dernière mission de ce stage consistait à réaliser un alignement entre les romans en langue source et leur traduction en langue cible. Cet aspect a également déclenché quelques problèmes, et possède un bilan plutôt mitigé.

3.1. Nuages de points

La première phase de l'alignement résidait en la formation d'un nuage de points pour vérifier l'appariement d'un texte et de sa traduction (cf. point « Création de nuages de points »). L'alignement devait se baser sur le résultat de ce nuage de points.

Comme explicité dans le point « Le script d'alignement », le programme déjà constitué m'a été fourni et ne nécessitait que quelques modifications au niveau des fichiers en entrée et en sortie. Ces adaptations ont posé quelques problèmes et ont demandé un temps de réflexion assez important avant d'obtenir un traitement convenable. La formation des nuages de points a d'abord été réalisée en local afin de régler les différents paramètres, puis sur le serveur pour bénéficier de la rapidité d'exécution et de la création des résultats pour tous les fichiers.

Cette étape peut se rapporter à un succès car la version finale du script a bien produit des nuages de points permettant de contrôler l'appariement des romans et de leur traduction. Ce dernier a même permis de mettre à jour une paire de romans ne concordant pas, ce qui a engendré un rejet de cette paire du corpus français-allemand. Les difficultés d'adaptation du code fourni au fonctionnement souhaité peuvent également faire partie de la réussite de cette phase, car elles ont contribué à une amélioration des compétences dans cet aspect.

3.2. Dynamic Time Warping

Lors de la première phase de l'alignement, le script automatique n'a pas produit qu'un nuage de points pour chaque paire de romans analysée, mais également des fichiers au format

.txt, .ces et .tmx (cf. point « Gestion et description des sorties »). Ces différents fichiers offrent des informations supplémentaires sur les romans étudiés, à savoir le couplage phrase par phrase entre le roman en langue source et celui en langue cible.

Après avoir obtenu et vérifié les sorties produites par le premier lancement du script d'alignement, il a fallu ajouter un paramètre à son exécution afin de lancer une nouvelle mesure de similarité : celle du Dynamic Time Warping. Ce dernier, comme indiqué dans le point « Utilisation du Dynamic Time Warping », doit permettre de calculer un score de similarité entre le roman en langue source et sa traduction en langue cible plus précis que le nuage de points, qui reste assez sommaire.

Lors des premières tentatives de lancement du procédé, plusieurs échecs liés au nombre de ressources nécessaires ainsi qu'à la mémoire disponible sur le serveur ont été constatés. Après avoir remédié à ces problèmes (cf. point « Problème rencontré »), l'algorithme a pu être exécuté et a amené de nouveaux défis. En effet, chaque paire de textes n'a pas obtenu de score de similarité pour différentes raisons connues et inconnues. Celles-ci feront l'objet d'un point dans la partie « Chapitre 8. Perspectives ».

Étant donné les résultats obtenus grâce à ce procédé, cette phase ne peut être considérée ni comme un échec ni comme une réussite car, bien que certains scores aient été récupérés, un nombre important de paires n'a pas obtenu de score, et d'autres se sont retrouvées avec des scores infinis. Les scores manquants sont au nombre de 25 et les scores infinis concernent 106 paires. Sur un corpus de 238 paires, 131 d'entre elles n'ont pas obtenu de score de similarité. Cela signifie que seulement 107 possèdent un score, soit environ 45% du corpus. Ces résultats ne sont pas négligeables, mais des améliorations des techniques d'alignement et de traitement de la similarité des paires sont encore à prévoir afin d'obtenir de meilleurs résultats à l'avenir.

Ces différents retours sur expérience permettent de développer des pistes d'amélioration, sujet du chapitre suivant.

Chapitre 8. Perspectives

Au vu des actions réalisées lors de ce stage, des mises en contact avec des outils et des modules de Python, ainsi que des résultats obtenus des différents traitements, plusieurs pistes peuvent être abordées dans le but de compléter et d'améliorer certaines tâches. Tout d'abord, il semble important de préciser les étapes suivantes prévues, ainsi que l'objectif visé à ce stade du projet, puis quelques évolutions seront présentées.

1. Suite du projet

En premier lieu, un point sur la suite des étapes après mon intervention dans ce projet afin d'éclaircir à la fois le but global et la manière de l'atteindre. Une description des premières conclusions d'une autre stagiaire engagée sur ce projet ainsi que la présentation d'un outil servant à l'annotation composeront ce point.

1.1. Stage Elnaz Jalilian

Tout d'abord, une autre stagiaire, Elnaz Jalilian, provenant de la filière MIASHS (Mathématiques et Informatique Appliquées aux Sciences Humaines et Sociales) a rejoint l'équipe du projet quelques mois après moi afin de réaliser de nouveaux traitements sur les corpus. Lors de la réunion du 3 juillet 2023, conviant tous les participants au projet, Elnaz a présenté les résultats des premiers traitements qu'elle a réalisés.

1.1.1. But de son stage

L'objectif d'Elnaz lors de son stage est de mesurer la similarité entre les PPI dans les parties dialoguées. Pour cela, elle se base sur le modèle de langage BERT, présenté dans la partie « Outils de traitement automatique des langues » point « Bert », dans le but de créer des vecteurs tout en prenant en compte le contexte. La prise en compte du contexte est un nouvel élément dans l'analyse de similarité.

Selon Elnaz, il faut bien différencier la similarité syntaxique de la similarité sémantique. En effet, la similarité syntaxique correspond à une structure de phrases proche, tandis que la similarité sémantique correspond à une signification en contexte proche. Concrètement, deux phrases ayant une construction syntaxique avoisinante n'ont pas nécessairement une similarité sémantique, c'est-à-dire un sens presque identique. À l'inverse, deux phrases ayant un sens semblable ne possèdent pas forcément une structure syntaxique équivalente. Le modèle doit

donc être capable de dissocier chaque type de similarité rencontrée, car les traitements à effectuer par la suite ne seront pas les mêmes pour l'une et l'autre.

Afin de mener à bien sa partie dans ce projet, elle prévoit d'extraire des modèles syntaxiques du corpus et de repérer les similarités pour entraîner le modèle à différencier les deux types précédemment cités. Ainsi, les phrases « c'est bien » et « c'est froid » possèdent une similarité syntaxique mais pas sémantique que le modèle devra détecter et analyser correctement.

1.1.2. Traitements réalisés

Elnaz a ensuite présenté les quelques traitements qu'elle a déjà effectués au niveau du repérage des similarités dans les corpus. Elle a tout d'abord utilisé le **Lexicoscope**[41] pour recueillir des verbes composant généralement les PPI. Le Lexicoscope est un outil mis au point par Olivier Kraif, permettant d'effectuer des recherches dans un corpus aligné afin de comparer des mots ou expressions dans 2 langues différentes. Le principe est simple : sélectionner laquelle des 2 langues comparées est la langue source, puis entrer un mot ou une phrase dans la barre de recherche. Le résultat renverra un tableau contenant les expressions relevées dans le corpus en langue source à gauche, et leur équivalent en langue cible à droite.

Selon elle, il est important de bien choisir les schémas syntaxiques à repérer ainsi que les corpus, mais également de regrouper les données sémantiquement. Le modèle devant être entraîné, il faut commencer par lui simplifier la tâche afin qu'il enregistre les ressemblances et qu'il apprenne de lui-même à les détecter dans les textes.

BERT possède plusieurs versions différentes en fonction des langues des textes à analyser. Le français possède deux modèles basés sur BERT, appelés **FlauBERT**[42] et **CamemBERT**[43]. Ces modèles sont également des Transformers, c'est-à-dire qu'ils fonctionnent exactement comme BERT en cachant aléatoirement des mots dans les textes et en essayant ensuite de les prédire. Ces modèles fonctionnent assez bien pour les corpus écrits mais sont très peu efficaces pour les corpus oraux. Une différence a cependant été constatée par Elnaz lors de ses traitements : le modèle CamemBERT (Martin et al. (2019)) tient compte de la succession des phrases mais pas le modèle FlauBERT (Le et al. (2019)), ce qui a tendance à

[41] http://phraseotext.univ-grenoble-alpes.fr/lexicoscope_2.0/
[42] https://huggingface.co/flaubert/flaubert_base_uncased
[43] https://huggingface.co/camembert-base

donner des résultats différents. C'est une information à prendre en compte lors de l'observation des sorties produites par les modèles.

Grâce à l'utilisation de ces modèles Transformers, Elnaz a réalisé des tâches de prédiction de mots masqués et de création de représentations vectorielles afin de mesurer la performance du modèle sur la tâche demandée. Pour cela, elle a utilisé de nouvelles bibliothèques proposées par Python qui sont *Transformers*, *PyTorch* et *NumPy*, dont le fonctionnement sera explicité dans le point suivant.

En se servant de ces différentes techniques pour entraîner les modèles à repérer les similarités syntaxiques et sémantiques entre des PPI, Elnaz a réalisé des mesures de clustering (regroupement en français) pour pouvoir analyser les vecteurs d'*embedding* obtenus. Un *embedding* est une représentation vectorielle d'un mot sous forme numérique permettant de récupérer des informations sémantiques et syntaxiques sur ce mot, mais aussi d'être utilisé par la suite pour améliorer les performances des modèles. Elle a utilisé plusieurs méthodes pour cela, dont une mesure de similarité Cosinus, une mesure avec l'algorithme DBSCAN, et une mesure de K-Means. Leur fonctionnement est avancé dans le point suivant, tandis que les résultats de ses observations sont développés dans le point de conclusion.

1.1.3. Définitions des méthodes utilisées

La bibliothèque *Transformers* est spécialisée dans le domaine du TAL et favorise l'utilisation des modèles pré-entraînés pour effectuer des tâches précises. Elle est construite sur la bibliothèque *PyTorch* et fournit des implémentations pré-entraînées de nombreux modèles d'apprentissage profond tel que BERT. Elle ne permet pas d'entraîner les modèles, mais se base sur ces derniers pour bénéficier des avancées existantes dans le domaine du TAL sans avoir à partir de zéro. *Transformers* est considérée comme facile d'utilisation, flexible et performante car elle possède des fonctionnalités pour optimiser les modèles existants, et son interface est intuitive.

La bibliothèque *PyTorch* permet de développer et d'entraîner des modèles basés sur des réseaux de neurones profonds. Elle fonctionne grâce à des calculs tensoriels qui sont nécessaires pour l'entraînement du modèle, mais aussi des constructions de graphiques dynamiques permettant à la fois de traiter des segments de longueur variable plus efficacement, et de modifier la structure du modèle pendant l'exécution. L'emploi de l'apprentissage automatique dans ce cas sert à faciliter l'entraînement du modèle et à optimiser les traitements. *PyTorch* est

également compatible avec plusieurs interfaces de programmation, ce qui la rend utile pour la production de modèles entraînés dans différentes applications.

La bibliothèque *NumPy* est différente car elle permet de faire des opérations vectorielles et matricielles mais uniquement à base de graphiques statiques. Elle sert à traiter les données et à faire des calculs numériques de base, mais n'est pas responsable de l'entraînement ni de l'apprentissage du modèle car elle n'utilise pas de mécanisme d'apprentissage automatique. Cependant, le cumul des deux bibliothèques est nécessaire et offre des avantages considérables dans le cadre des calculs de similarité sémantique et syntaxique.

La mesure de similarité Cosinus utilisée par Elnaz consiste à calculer le produit scalaire des deux vecteurs et diviser le résultat par le produit des normes des deux vecteurs. Plus le résultat obtenu est élevé, plus les PPI comparées sont similaires.

L'algorithme DBSCAN (density-based spatial clustering of applications with noise) est un algorithme simple qui définit des clusters en utilisant l'estimation de la densité locale. Il est basé sur deux paramètres : la distance et le nombre de points devant se trouver dans un périmètre pour être considérés comme un cluster. Si les PPI appartiennent au même cluster, elles sont considérées comme similaires. Cependant, cette méthode ne s'est pas révélée très efficace lors des traitements faits par Elnaz car beaucoup de bruit a été repéré.

La méthode K-Means, quant à elle, est un algorithme de regroupement non supervisé utilisé pour partitionner un ensemble de données en clusters. Elle cherche à trouver des centres de cluster représentatifs pour minimiser la variance intra-cluster. Chaque point de donnée est attribué au cluster le plus proche, puis son centre est recalculé jusqu'à ce que les centres des clusters convergent vers une configuration stable. Cette méthode s'est révélée plutôt efficace au niveau des traitements faits par Elnaz.

1.1.4. Premières conclusions

Au vu des résultats obtenus, la méthode la plus performante dans le repérage et la création de groupes de PPI similaires, selon Elnaz, est la méthode K-Means. En effet, c'est celle qui a obtenu les meilleures sorties. Elle souligne cependant la nécessité d'optimiser l'algorithme pour améliorer encore les résultats.

De manière générale, toutes les observations d'Elnaz l'ont amenée à conclure qu'une optimisation des modèles et des algorithmes sont à envisager pour la suite des opérations concernant ce projet.

De plus, ses analyses lui ont permis de constater l'importance de se concentrer sur la similarité entre différentes expressions dans différentes phrases. En effet, l'analyse et la comparaison des PPI devant se faire en prenant en compte le contexte, il est indispensable de ne pas se focaliser uniquement sur une phrase puis une autre, mais au contraire de les prendre dans leur globalité.

Enfin, Elnaz a souligné la difficulté de traitement des expressions polysémiques du français. En effet, la similarité syntaxique peut être aisée à repérer même dans ces cas, mais la similarité sémantique le sera beaucoup moins en raison des différents sens que peut prendre une même expression. La prise en compte du contexte peut évidemment faciliter la compréhension du contexte pour le modèle, mais ce qui est évident à comprendre pour un humain est assez difficile à enseigner à une machine. Cette phase mérite donc une importante considération afin de permettre au modèle d'apprendre la complexité des expressions polysémiques de la langue.

1.2. Inception

En parallèle de l'analyse des similarités sémantiques et syntaxiques des PPI, l'utilisation d'un outil d'annotation appelé **Inception**[44] est envisagée. Ce dernier a pour but de mutualiser les annotations des PPI présentes dans les romans afin de gagner du temps sur le repérage.

1.2.1. Description

Inception est un outil permettant de faire une annotation collaborative en ligne de manière manuelle par le biais d'interventions humaines. L'annotation réalisée est sémantique, c'est-à-dire qu'elle cible les entités nommées, et multi-niveaux. De plus, l'outil est capable de proposer des suggestions et des recommandations grâce à un mécanisme d'apprentissage automatique. Le format proposé par l'interface est un format ressemblant au format ConLL.

Plusieurs couches d'annotations sont possibles avec Inception, par exemple au niveau du lemme, des PPI, ou encore de la partie du discours. Ce n'est pas un outil spécialisé pour les PPI, mais dans le cadre de ce projet, seules les PPI seront annotées. Il existe trois types

[44] https://inception.atilf.fr/

d'annotations différentes proposées par Inception : l'annotation de l'unité, l'annotation des relations, et l'annotation des chaînes.

Le but de l'emploi d'Inception dans le cadre de ce projet est d'identifier à la fois les PPI, leur modalité d'énonciation, leur type sémantico-pragmatique, et de leur affecter une étiquette sémantique pour une caractérisation minimale du sens (cf. Annexe 33). Ainsi, les PPI pourront être repérées, catégorisées et analysées.

De plus, un mécanisme d'adjudication permettra par la suite de comparer le travail de deux annotateurs et de visualiser les différences de traitement. Il faut pour cela avoir défini au préalable l'annotation de référence. Ce procédé est utile pour repérer instantanément les désaccords entre les participants au projet, et amener à des discussions dans le but de trouver un compromis convenant à tous.

1.2.2. Difficultés

Malgré l'efficacité de l'outil pour repérer et annoter les PPI, plusieurs difficultés restent d'actualité. Premièrement, l'explication des différentes annotations prévues pour les PPI est déjà compliquée du fait de son nombre. En effet, pas moins d'une dizaine de types est disponible, contrairement aux 4 types (méta discursif, réactif, situationnel et pragmatème) mis à jour par Agnès Tutin (2019) dans les corpus oraux de CLAPI. La première difficulté est donc d'associer la PPI au type qui lui correspond.

Ensuite, il a été prouvé qu'une même PPI peut avoir plusieurs types. La version actuelle de l'outil ne prend pas en compte cette possibilité, mais les paramètres peuvent être adaptés à cette éventualité. Ce point a été soulevé lors de la réunion du 3 juillet 2023 regroupant tous les membres du projet, dont Céline Poudat, enseignant chercheur à l'université de Nice Côte d'Azur et responsable de l'outil Inception.

Enfin, il est essentiel de souligner la difficulté de se mettre d'accord pour les membres du projet sur l'annotation à faire dans certains cas. Concrètement, la réunion du 3 juillet 2023 a ouvert un débat entre les protagonistes sur l'autonomie des PPI. Pour certains, lorsque la PPI est considérée comme autonome elle peut être annotée, tandis que si celle-ci est rattachée à un élément de la phrase elle ne doit pas l'être. C'est la différence entre une PPI et une expression référentielle. Le débat est resté ouvert quant à l'attitude à adopter lorsqu'une expression référentielle est rencontrée, et fera sûrement l'objet de prochaines discussions.

2. Objectif poursuivi

À ce stade du projet, les objectifs concernent la réalisation du modèle, la précision de ce qui doit être repéré automatiquement et des points de discussions autour des PPI. Ces aspects visent à spécifier de manière efficace la direction que doit prendre la suite du projet.

2.1. Modèle et fonctions

Au sujet du modèle, la réunion du 3 juillet 2023 a permis de définir les différents niveaux que celui-ci doit être capable de repérer sur les PPI.

Premièrement, le modèle doit pouvoir comprendre le niveau de représentation des formes de base des PPI. Ensuite, il doit se familiariser avec le niveau de variantes de surface quand elles existent. Puis, le niveau des constructions syntaxiques globales des PPI doit à son tour être pris en compte par le modèle, suivi du niveau du schéma lexico-sémantique et du niveau pragma sémantique.

Concrètement, le modèle doit être capable de repérer les PPI dans les romans, en se basant sur leur construction syntaxique et en prenant en compte leur aspect sémantique. C'est pour cela qu'Elnaz travaille sur l'entraînement des modèles à la fois sur les similarités syntaxiques et sémantiques des PPI.

De plus, le modèle doit être apte à repérer automatiquement de nombreuses autres caractéristiques que les différents niveaux précédemment cités. Parmi elles, il y a les spécificités des rôles des participants aux interactions. Celles-ci devront être détectées à l'aide d'attributs. La position des PPI dans l'interaction doit également être extraite par le modèle, ainsi que leur position à l'intérieur de l'intervention (ou tour de parole). Le fonctionnement des PPI par rapport aux autres énoncés fait aussi l'objet d'un repérage automatique souhaité, afin de comprendre la procédure de ces phrases préfabriquées.

La question du modèle à employer pour les corpus oraux et les corpus écrits s'est posée lors de la réunion du 3 juillet 2023. Comme avancé plus tôt, précisément dans la partie « Stage Elnaz Jalilian » point « Traitements réalisés », les modèles issus de BERT sont efficaces sur les corpus écrits mais peu sur les corpus oraux. De ce fait, l'utilisation d'un même modèle pour les deux types de corpus ne semble pas très convaincante si celui-ci se base sur BERT. Dans le cas où un nouveau modèle de langage performant à la fois pour l'oral et l'écrit fait son apparition

dans ce projet, ce dit modèle pourra être l'unique prototype utilisé. Actuellement, ce n'est pas envisageable.

2.2. Discussions autour des PPI

La réunion du 3 juillet 2023 a donné lieu à des mises au point concernant les PPI car, comme indiqué dans le point « Difficultés », il est difficile de se mettre d'accord sur ce qui doit être annoté et comment cela doit être annoté. Cette réunion a également été l'occasion de présenter les premières constatations faites par chacun des membres du projet, permettant ainsi d'affiner les traitements à réaliser.

Ainsi, il a été constaté que le corpus Wiki Discussion ne contient que peu de PPI et beaucoup de bruit. Ce corpus pourra néanmoins être utilisé dans le cadre des comparaisons prévues, mais il faut souligner que le nombre de PPI en provenant est bien inférieur à ce qui sera extrait des autres corpus.

Ensuite, certains aspects des PPI ont été abordés de manière à clarifier les cas dans lesquels elles peuvent être présentes. De ce fait, il a été démontré que les PPI peuvent être adressées ou non à quelqu'un selon leur position dans l'interaction. En effet, si celle-ci apparaît dans un monologue, elle n'est adressée à personne et est considérée comme autonome, tandis que lorsqu'elle est présente dans une interaction elle est forcément adressée à un interlocuteur. De même, une PPI peut avoir une portée ou non.

De plus, il a été prouvé que les PPI peuvent se répondre dans une interaction. C'est le cas par exemple d'un premier locuteur qui dit « Désolé » et d'un second qui répond « C'est rien ». Dans cette situation, les deux énoncés sont des PPI. Qui plus est, les PPI peuvent être non verbales. C'est le cas des discussions où l'un des locuteurs est énervé par exemple, et répond par un soupir ou une mimique.

Il a également été décidé de se focaliser sur l'aspect expressif des PPI, afin de compléter l'étude réalisée par Agnès Tutin (2020) portant sur la fonction expressive des PPI dans différents dictionnaires. De même, le rôle social des locuteurs dans la situation de communication fera l'objet d'une surveillance afin de déterminer si celui-ci influe sur la présence ou non des PPI.

Enfin, il est prévu que les PPI soient recueillies dans une base de données créée sur **PhPMyAdmin**[45]. Ainsi, elles pourront être sauvegardées et consultées tout au long de l'étude pour effectuer les comparaisons et analyser les différents aspects précédemment cités.

3. Évolutions proposées

Dans cette partie, différentes évolutions seront proposées en fonction des problèmes rencontrés lors de ce stage. Certains aspects concernent spécifiquement des erreurs ayant nécessité des traitements plus ou moins importants, d'autres abordent des remarques pouvant améliorer les interventions futures.

3.1. Évolutions liées à XML

La première évolution à citer dans le cadre de ce projet concerne les fichiers XML. En effet, pas mal de soucis ont été observés et ont dû être corrigés avant même de pouvoir entamer les traitements prévus. Un contrôle de la syntaxe par le biais de la bibliothèque *lxml* de Python peut permettre d'éviter le temps d'intervention initial qui a été indispensable ici (cf. point « Correction »), mais d'autres améliorations sont également envisageables.

3.1.1. Amélioration de l'étape de conversion

Tout d'abord, une amélioration de la conversion du format .xml au format .txt permettrait d'éviter des traitements supplémentaires ayant eu lieu pour chaque corpus traité lors de ce stage. En effet, les variantes du programme *text_suppr_n.py* n'auraient pas été nécessaires si la conversion n'avait pas conservé des balises dans les textes, des caractères propres aux XML comme des « \n » symbolisant des retours à la ligne, ou encore des entités non définies correspondant à des caractères spéciaux.

Ensuite, un autre aspect important lors de la conversion concerne la mise en page obtenue. En effet, les phrases contenues dans les paragraphes des fichiers XML pouvaient s'arrêter net en plein milieu, voire au milieu d'un mot, et continuaient alors dans la phrase suivante (cf. Annexe 34 et Annexe 35). Cela engendrait des problèmes au niveau de la mise en page, car le texte contenait des retours à la ligne à chaque fois qu'une phrase arrivait à son terme selon la syntaxe XML (c'est-à-dire dès que la balise <dc> était rencontrée, cf. point « Conversion »).

[45] https://www.phpmyadmin.net

De ce fait, une phrase pouvait commencer sans majuscule ni réel début dans les fichiers texte. L'idée a donc été de rassembler les phrases ayant été éclatées à cause de la syntaxe XML. Cela a alors déclenché un nouveau problème : le changement du nombre de lignes entre le fichier d'origine et le fichier texte. Comme avancé dans le point « Nettoyage », une modification du nombre de lignes engendre un décalage dans le fichier final contenant le format .conllu inséré dans le format .xml.

De même pour les espaces abusifs devant les points ou les virgules ayant été conservés lors de la conversion. Ces derniers ont été supprimés dans un premier temps, puis maintenus pour cause de modification du nombre de lignes et de décalage dans les versions finales. Ces changements avaient pour but de rendre les textes obtenus après conversion plus propres. Mais les traitements à effectuer par la suite étaient plus importants que la purification des textes, la décision d'abandonner ces modifications a alors été prise.

Une amélioration de la conversion pourrait donc permettre de prendre en compte les éléments précédemment cités, soit lors de la conversion elle-même, soit en modifiant le contenu des fichiers XML directement. La deuxième solution contribue de manière plus sûre à éliminer à la fois les phrases coupées en deux et les espaces abusifs, car la correction se fera directement au niveau de la source. Les textes obtenus seront alors plus propres, et le choix de la purification ou du bon nombre de lignes ne se posera plus.

3.1.2. Amélioration de la gestion de ces fichiers

En ce qui concerne ce qui peut être considéré comme la gestion des fichiers XML, ces améliorations visent à réduire le temps de traitement supplémentaire ayant été nécessaire pour les corpus du projet. Le premier point lié à cet aspect est la présence de paragraphes vides dans les fichiers XML d'origine. Cela amène plusieurs problèmes qui ne sont pas évidents au départ. En effet, les traitements suivants permettant de détecter le discours direct dans les phrases et de les annoter en conséquence, un paragraphe vide provoque un décalage au niveau de l'insertion du format .conllu dans le format .xml. De plus, la présence d'un paragraphe vide cause le maintien d'un paragraphe vide en fin de fichier après l'étape d'insertion, car le nombre de paragraphes présents dans le fichier XML ne correspond pas au nombre de paragraphes contenus dans le fichier .conllu.

Les paragraphes vides ont donc dû être repérés et supprimés avant de pratiquer l'insertion (cf. point « Suppression des contenus inadaptés »). D'une part, ce phénomène a

demandé la création d'un script supplémentaire pour gérer ce problème, et d'autre part la suppression des paragraphes vides a engendré un décalage au niveau des identifiants des paragraphes dans les fichiers XML concernés. Ce nouveau souci a également nécessité un traitement afin de rectifier cet écart, donc à nouveau du temps.

Une vérification des fichiers XML avant transmission pour un quelconque traitement permettrait de mettre à jour les paragraphes vides et de les supprimer, ou bien une recherche du motif d'apparition de ces paragraphes vides servirait à corriger ce phénomène. Ainsi, un supplément de traitements non prévus serait évité. Dans tous les cas, pour être efficace, la modification doit agir sur les fichiers XML d'origine.

3.2. Évolutions liées aux analyses

Le second type d'évolution à proposer fait suite aux différentes analyses menées durant les tâches de ce projet. Ces propositions ont été amorcées dans les parties qui leur correspondent, et concernent principalement la technique d'OCR et les analyses faites par Stanza.

3.2.1. Évolutions de la technique d'OCR

Au sujet de l'OCR dont le procédé a été utilisé pour obtenir une version numérique des romans, la partie « Erreurs spécifiques dans les textes » point « OCR » a révélé les différentes erreurs observées. Il est évident que ce système, bien que très efficace, n'est pas encore totalement au point. En effet, les erreurs telles que la reconnaissance d'un « 1 » à la place d'un « I » engendre des problèmes au niveau de l'analyse de Stanza qui peut avoir du mal à reconstituer le mot original.

De plus, la vérification et la correction des erreurs liées à l'OCR nécessitent également du temps de traitement humain et automatique supplémentaire. En effet, dans le cadre de ce projet, un passage par un fichier Excel recensant les différentes erreurs détectées ainsi que leurs corrections pour les textes en français a été indispensable, car une modification manuelle de tous les fichiers était simplement inenvisageable compte tenu du nombre de fichiers à contrôler. L'analyse des mots inconnus avec l'outil NooJ, la création du fichier Excel de correction, ainsi que la conception du programme gérant les remplacements automatiques a demandé des ressources auxiliaires. Ces dernières pourraient devenir inutiles une fois le processus de l'OCR devenu plus performant.

Concernant les textes en allemand, les erreurs peuvent être corrigées de la même manière que pour les textes en français. Cependant, les connaissances en allemand n'étant pas suffisantes à Grenoble, et les équipes de l'ATILF n'étant pas spécialistes du traitement automatique des langues, il a été convenu que les erreurs seraient corrigées grâce au correcteur orthographique de Word. Cette hypothèse nécessite un temps énorme et une intervention humaine, ce qui est loin d'être la meilleure solution.

Afin de pouvoir se passer des traitements manuels et automatiques rectifiant les erreurs d'OCR repérées dans ces corpus, il faut que le procédé d'océrisation devienne plus performant. Les traitements réalisés ici n'ont pas permis de prendre en compte la totalité des cas nécessitant des corrections, et certaines erreurs ont donc été malheureusement maintenues. À ce stade, la correction complète des erreurs résultant de l'OCR n'est pas imaginable, et la puissance de ce procédé n'est pas une technique sur laquelle agir directement est une possibilité.

3.2.2. Évolutions de Stanza

À propos de l'analyse de Stanza, de nombreux problèmes ont été soulevés dans le point « Modification des sorties obtenues », mais également dans le point « Vérification des lemmes de Stanza ». Il est certain que, de même que pour le procédé d'océrisation, les analyses de Stanza sont loin d'être parfaites.

Le premier souci constaté concernait la création de doublons analysés dans certains textes. Ces doublons ont forcément nécessité des traitements complémentaires afin de supprimer ces passages non désirés et d'aligner le texte restant. S'ils n'avaient pas été traités, ces derniers auraient été responsables d'un décalage au niveau des paragraphes dans les fichiers XML.

Ensuite, un problème a été remarqué à la suite de l'exécution du script automatique permettant de supprimer les doublons dans les fichiers. Comme précisé précédemment, seuls certains fichiers étaient concernés par la présence de doublons. De ce fait, l'action du programme sur les fichiers non impliqués supprimait totalement leur contenu. Une vérification manuelle des fichiers vides après le lancement du précédent script a donc été indispensable. Ce problème étant directement lié à la présence des doublons, la correction de ce dernier sera immédiate dès lors que la création de doublons sera abrogée.

Puis, des erreurs au niveau de la catégorisation faite par Stanza ont été repérées dans les sorties. En effet, certains noms propres n'étaient pas considérés comme tel par Stanza, mais

plutôt comme des noms communs. Ce fut le cas de « Blanche Neige » dont le nom n'a pas été reconnu. Cela peut être dû à un manque de ressource du côté de Stanza, de même pour les erreurs de lemmatisation retrouvées. En effet, des erreurs au niveau de la lemmatisation de Stanza ont également été aperçues dans les sorties. La provenance de ces erreurs est inconnue, mais celles-ci ont été traitées grâce à des dictionnaires de formes fléchies servant de référence. Une amélioration des ressources de Stanza, qui ne relève pas de nos compétences, pourrait limiter ces types d'erreurs et de corrections.

Enfin, il semble important de souligner la lenteur d'exécution des analyses de Stanza. Bien que ces analyses soient puissantes et assez complètes sans être parfaites, elles nécessitent un temps d'action conséquent. Lors des différentes analyses réalisées dans le cadre de ce projet, la plupart des analyses lancées ont mis environ 24h pour produire tous les résultats. Évidemment, ce temps est proportionnel au nombre de textes composant le corpus, mais il faut en tenir compte lors de projets composés de gros, voire de très gros, corpus.

Une optimisation de Stanza dans sa globalité permettrait à la fois d'éviter la création de programmes devant agir par-dessus l'analyse, de régler plusieurs erreurs, et de diminuer le temps de traitement. Cela ne peut pas se faire directement du côté des utilisateurs, mais peut constituer des pistes d'améliorations pour les créateurs et les développeurs de l'outil, afin de proposer de nouvelles versions plus performantes.

3.2.3. Évolutions du DTW

Concernant le Dynamic Time Warping, plusieurs améliorations sont également envisageables car les résultats présentés dans le point « Résultats obtenus » démontrent de nombreux soucis.

Le premier aspect problématique est le non-retour de scores. En effet, dans 25 cas, le DTW a échoué et n'a proposé aucun score de similarité pour les paires de textes. Dans ces cas-ci, il est possible que les textes associés soient effectivement trop éloignés l'un de l'autre pour qu'une correspondance soit trouvée par l'algorithme. Lorsque le nuage de points est inexistant, il est évidemment impossible de faire correspondre les textes. Néanmoins, lorsque le nuage de points est correct mais qu'un trou se situe dans la diagonale, le script d'alignement gérant le lancement du DTW pourrait être amélioré afin de prendre en compte ces cas. Le score obtenu serait certainement mauvais et très élevé, mais il serait présent et permettrait de vérifier la précision de la traduction.

De même pour le second aspect à relever, à savoir les scores infinis. Ces derniers sont au nombre de 106 dans le corpus, et correspondent à peu près à des scores nuls car aucune correspondance n'a pu être établie pour les paires de textes concernés. Or, en comparant le score avec le nuage de points de ces paires, il arrive que ces deux éléments ne soient pas compatibles. En effet, lorsque le nuage est mauvais, il est logique de constater un score perturbé. Mais dans le cas où le nuage est bon et le score est infini, il y a un problème. Ainsi, une modification des paramètres compris dans le script d'alignement pourrait permettre de limiter les scores infinis abusifs et d'obtenir plus de scores corrects.

Pareillement à Stanza, le DTW possède une lenteur d'exécution importante. L'analyse des 238 paires concernées dans ce projet a également pris environ 24h, un aspect non négligeable à prendre en compte lors de l'analyse de gros corpus. De plus, une optimisation globale du script d'alignement permettrait d'obtenir de meilleurs résultats au niveau des scores de similarité. En effet, cela pourrait limiter le nombre de scores infinis recueillis lorsque les conditions sont pourtant réunies pour obtenir un score d'une part, et pourrait prendre en compte les trous présents dans les nuages de points d'autre part afin d'obtenir malgré tout un score. Ainsi, les résultats seraient plus nombreux, pas forcément meilleurs, et la qualité de la traduction pourrait concerner plus de paires de textes qu'actuellement (celles-ci représentent 45% du corpus).

Enfin, un moyen plus automatisé pour récupérer les scores obtenus dans une liste serait également considéré comme une amélioration du DTW. En effet, la version actuelle récupère bien les scores perçus dans un fichier texte, mais ne prend pas en compte la paire de textes concernée : il faut alors l'ajouter manuellement grâce à la liste des paires et à un fichier contenant les paires dont l'analyse n'a pas été possible (cf. point « Récupération des scores du DTW »). Plus les étapes sont automatiques et rapides, plus elles sont efficaces. Le moins d'intervention manuelle possible est toujours recherché pour améliorer les programmes.

3.3. Autres remarques

D'autres aspects plus généraux observés lors de ce stage peuvent faire l'objet d'améliorations. Il s'agit des scripts de manière générale, et du serveur dont le recours a permis de diminuer grandement le temps nécessaire à l'exécution de nombreux programmes.

3.3.1. Scripts

Concernant les différents scripts utilisés lors de ce stage, que ce soit dans le cadre du corpus Phraseobase ou bien du corpus GLFA, plusieurs d'entre eux font des actions similaires. En effet, les traitements à effectuer sur les textes sont à peu près les mêmes dans les deux corpus, ce qui amène à la création de nombreux programmes agissant sur des fichiers dont le chemin d'accès est différent. C'est le cas des romans annotés et non annotés du corpus Phraseobase, où les manipulations sont presque identiques, mais également des romans français et allemands du corpus de l'ATILF, bien qu'ici les traitements soient plus éloignés.

Une amélioration possible au niveau de ce nombre de programmes assez important pour des agissements assez proches serait un regroupement de ces scripts en un nombre plus petit. En effet, l'exécution de programmes les uns après les autres revient à lancer un seul programme (dans le meilleur des cas) exécutant les mêmes actions les unes après les autres. Le résultat serait un programme énorme mais unique, à ne lancer qu'une fois pour obtenir tous les traitements souhaités, sans devoir retenir un ordre d'exécution comme c'est le cas actuellement. Dans le cadre de ce stage, au vu des erreurs à corriger manuellement et des programmes à n'exécuter que sur certains fichiers, il n'est pas possible de regrouper tous les programmes en un. Mais un regroupement en 4 scripts concernant les 4 catégories de romans analysés (l'alignement resterait à part) pourrait être considéré comme une évolution positive.

De plus, durant ce stage, des scripts déjà créés par d'autres personnes et dans d'autres circonstances ont été récupérés pour être réutilisés. La première étape lors de cette récupération est la compréhension de ce que fait le code, de comment il le fait et de comment il fonctionne. Cette phase peut s'avérer longue et difficile si le code en question n'est pas bien documenté. Elle est pourtant indispensable avant toute modification ou adaptation à un autre code. Une des améliorations possibles permettant de gagner du temps sur la compréhension de ces codes serait de prendre autant en compte le code et sa documentation. En effet, un code bien documenté évite une perte de temps lors de sa passation d'un programmeur à un autre. Dans n'importe quel projet, cet aspect est important et constitue un changement positif.

3.3.2. Serveur

Au niveau du serveur, celui-ci a été très utile pour exécuter les différents programmes plus rapidement que sur la machine en local, mais quelques aspects qui lui sont propres ont néanmoins posé problème par moments.

Tout d'abord, la plupart des scripts utilisés lors de ce stage ont nécessité plusieurs versions. En effet, certaines ont servi de test, puis ont été complétées avant de devenir la version optimale utilisée sur les corpus. Cependant, certaines erreurs soulevées par des versions problématiques étaient directement liées au serveur lui-même, car elles demandaient trop de ressources ou de mémoire par rapport à ce que le serveur pouvait fournir. Il a donc fallu s'adapter constamment et réduire le coût en ressources lorsque c'était nécessaire pour éviter de faire bugger le système.

Les propositions de regroupement des scripts annoncées dans le point précédent ne peuvent être possibles que si la mémoire du serveur est capable de traiter un programme conséquent, ce qui, selon les tests effectués, n'est pas le cas actuellement. Cela signifie que chaque programme est tributaire de la mémoire disponible sur le serveur. Dès lors qu'une erreur de type « killed » est renvoyée par le terminal, deux solutions sont accessibles : modifier le code actuel pour tenter de limiter son besoin en ressources, ou augmenter la mémoire du serveur.

Enfin, le fait de travailler sur un serveur distant tel que MIAI Serveur engendre parfois des problèmes inattendus. En effet, il arrive que le serveur soit indisponible pour une raison inconnue. Dans ce cas, il est impossible d'accéder aux fichiers, aux scripts et encore moins de les exécuter. Cela peut également se produire si l'un des outils Cisco AnyConnect ou FileZilla n'est pas disponible car, comme présenté dans la partie « Description des outils utilisés » point « Autres outils », ces derniers sont indispensables à l'accès au serveur. Comme tout outil informatique, il n'est pas à l'abri d'une panne et cela peut malheureusement se produire à n'importe quel moment. Aucune amélioration n'est envisagée sur ce point, mais il me semble important d'en être conscient.

Après ce développement des quelques améliorations possibles selon les cas observés, le dernier chapitre présente les bilans.

Chapitre 9. Bilans

Arrivée au terme de ce stage, il est temps de se pencher sur les différentes compétences acquises ou développées durant ces quelques mois. Ces dernières sont évidemment nombreuses, mais peuvent être regroupées en 3 catégories : le développement personnel, le développement de compétences, et enfin la collaboration.

1. Développement personnel

Le premier point marquant de ce stage est le développement personnel que ce dernier a engendré. En effet, les nombreuses tâches réalisées n'ont pas seulement permis d'accroître des connaissances théoriques, mais également de progresser sur le plan personnel.

1.1. Qualités personnelles

Le thème le plus important selon moi concerne les qualités personnelles dont j'ai pris conscience tout au long de ce stage. En effet, j'étais consciente des connaissances que j'avais acquises lors de ma formation, plus particulièrement lors de mon Master en Industries de la Langue, mais il m'était difficile de comprendre comment celles-ci allaient pouvoir être utilisées dans le cadre professionnel. Cependant, les différentes missions, ainsi que la création des nombreux scripts ayant permis d'arriver à ce qui était souhaité, m'ont aidé à réaliser que je possède des compétences non négligeables dans le domaine du traitement automatique des langues. Ce stage m'a permis de prendre confiance en moi de manière globale sur mes capacités.

De plus, je me suis découvert une grande patience durant ce stage notamment grâce aux nombreuses erreurs survenues lors des tests. En effet, les erreurs pouvant provenir de plusieurs sources, allant des erreurs de syntaxe dans les fichiers XML à la saturation de la mémoire du serveur, en passant par de multiples problèmes au sein même du code, il était indispensable de ne pas se décourager ni abandonner pour le bien du projet. De ce fait, il a fallu à de nombreuses reprises reprendre le code depuis le début, abandonner une voie pour en recommencer une nouvelle. Certaines tentatives ont duré plusieurs jours, voire plusieurs semaines, avant que le code ne fonctionne comme prévu.

Ensuite, ce stage m'a offert la possibilité de développer une grande autonomie grâce aux différentes missions qui m'ont été confiées. En effet, je connaissais les objectifs que je devais atteindre, mais aucune ligne directrice ne m'a été donnée pour y parvenir. J'ai donc pu créer mes propres programmes en utilisant les modules de mon choix, la syntaxe de mon choix, ou encore l'approche de mon choix. Ce qui importait ici était le résultat plus que la manière de l'obtenir, cette liberté m'a été très bénéfique pour explorer les différentes façons de procéder ainsi que pour développer mes capacités.

Enfin, les tâches à effectuer m'ont amené à devoir faire de nombreuses recherches personnelles pour surmonter les obstacles rencontrés. En effet, les solutions pouvant contrer les erreurs, ou bien permettre de les éviter par le biais d'un nouvel itinéraire quand cela était nécessaire, ont demandé plus ou moins de ressources extérieures. Énormément de bugs ont pu être résolus notamment grâce à **Stack Overflow**[46], un site de questions réponses entre programmeurs visant à trouver des solutions aux problèmes de codage, mais aussi grâce au site de documentation de **Python**[47]. Ces recherches m'ont à la fois permis de parfaire mes compétences, et de gagner en autonomie et en confiance, car chaque problème qui se pose possède une solution.

1.2. Organisation

Le second point à souligner dans le cadre du développement personnel est l'organisation suivie tout au long de ce stage. Il est vrai que l'organisation est indispensable dans n'importe quel projet, mais elle l'est d'autant plus lorsque plusieurs programmes devant réaliser le même type de traitement mais sur des fichiers quelque peu différents est d'actualité. De plus, il est important d'avoir une organisation rigoureuse pour garder une trace de chaque étape afin de faciliter la rédaction du mémoire.

Durant ce stage, l'organisation adoptée m'a permis de conserver énormément d'informations sur les tâches réalisées dans un document à part. Ce document contient des indications sur les erreurs rencontrées ainsi que les solutions envisagées, mais également sur les difficultés et les différentes sources requises pour le bon déroulé des opérations. De ce fait, toutes les étapes traversées pendant ces quelques mois ont été conservées pour éviter de perdre des informations importantes avec le temps. La rédaction du mémoire a ainsi été simplifiée,

[46] https://stackoverflow.com/
[47] https://docs.python.org/3/

mais également les étapes répétitives où les erreurs similaires ont nécessité un traitement similaire. Les solutions ayant déjà été inscrites dans le document, il suffisait de les retrouver pour gagner un temps précieux.

De plus, une organisation méthodique était indispensable au vu du nombre de scripts créés et de fichiers manipulés. En effet, chaque programme correspondait plus ou moins à une étape précise du traitement des données. Il fallait donc s'assurer que les scripts étaient exécutés dans le bon ordre pour ne pas se retrouver avec des erreurs ou des sorties indésirables. Ensuite, certains fichiers nécessitaient des traitements manuels à la suite des traitements automatiques réalisés. Les fichiers ayant subi plusieurs modifications et plusieurs traitements automatiques tout au long du stage, il fallait garder une trace de ceux qui devaient être modifiés manuellement pour ne pas avoir de mauvaise surprise (ces informations étaient conservées dans le document précédemment présenté).

Il fallait également indiquer quels scripts concernaient la totalité des fichiers des corpus, indépendamment de leur catégorie (annotés/non annotés et allemands/français), et lesquels étaient spécifiques à une catégorie particulière. Ces précisions étaient importantes lors du lancement de la totalité des programmes sur les fichiers, de même que les dossiers créés à la suite des traitements automatiques réalisés. En effet, pour pouvoir vérifier et contrôler la qualité des sorties obtenues après un traitement spécifique, il était indispensable de connaître la provenance des fichiers d'entrée du programme et la destination des fichiers de sortie. Ainsi, des tableaux contenant à la fois l'ordre des scripts, et l'emplacement des fichiers d'entrée et de sortie de ces derniers, étaient présents dans ce même document (cf. Annexe 36, Annexe 37 et Annexe 38).

2. Développement de compétences

Le deuxième point essentiel de ce stage est évidemment le développement de compétences non négligeables. En effet, les connaissances théoriques acquises lors d'une formation forment bien entendu une base importante, mais la mise en pratique et l'adaptation de ces connaissances au sein d'un projet permet d'asseoir ces savoirs d'une part, et d'en développer de nouveaux d'autre part.

2.1. Connaissances informatiques

Le premier aspect concernant les compétences développées lors de ce stage est celui des connaissances informatiques. Le Master dans lequel j'évolue depuis 2 ans est une formation

mêlant la linguistique et l'informatique. Je possédais donc déjà certaines bases en informatique grâce aux différents cours. Néanmoins, ces connaissances étaient loin d'être suffisantes pour réaliser mon stage. En effet, j'ai été confrontée à plusieurs nouveautés qu'il a fallu appréhender et apprivoiser petit à petit.

2.1.1. Système Linux

La première de ces nouveautés a été la découverte du système Linux, bien différent du système Windows avec lequel j'étais habituée à évoluer à la fois lors de ma formation et dans le domaine privé. De fait, l'ordinateur m'ayant été fourni pour la durée de mon stage, il a fallu tout d'abord comprendre son fonctionnement. Cette étape n'a pas demandé un temps important, contrairement à l'installation de logiciels et d'applications sur la machine. En effet, certains logiciels disponibles sur le système Windows ne le sont pas sur le système Linux. C'est le cas par exemple du logiciel **Notepad++**[48] dont j'avais l'habitude de me servir sur Windows, et qui m'a amenée à utiliser Geany pour ce projet.

De plus, l'installation d'applications est très différente d'un système à l'autre, et j'ai eu recours au site de documentation du système **Ubuntu**[49] présent sur la machine que j'ai utilisée. Ce dernier m'a permis d'installer Zoom afin de participer aux réunions régulières du projet, car il est indispensable de passer par les lignes de commande sous Linux pour pouvoir installer quoi que ce soit sur la machine, contrairement à Windows.

Ensuite, le fonctionnement des lignes de commande sous Linux est également un point sur lequel j'ai dû m'attarder pour la simple et bonne raison que je n'avais quasiment jamais utilisé le terminal des ordinateurs durant ma formation. Ce moyen étant le seul possible pour pouvoir exécuter les scripts sur le serveur, ou tout simplement communiquer avec ce dernier à distance, il a fallu se familiariser avec le terminal, la navigation depuis celui-ci dans différents dossiers, et les commandes existantes.

2.1.2. Serveur distant

La deuxième nouveauté concerne l'apprivoisement du serveur distant. Celui-ci se pilote de la même façon que le serveur local sur chaque machine, c'est-à-dire via un terminal et grâce à des lignes de commande. Cependant, lorsqu'un programme est créé et nécessite l'utilisation

[48] https://notepad-plus-plus.org/
[49] https://doc.ubuntu-fr.org/

d’un module spécifique, importé au préalable au début de chaque script, il faut également s’assurer que ce module est bien disponible sur le serveur.

Dans le cas où le module souhaité est déjà installé sur le serveur, le script sera exécuté sans problème. En revanche, il arrive que le module ne soit pas présent sur le serveur, ce qui engendre des erreurs d’exécution. Pour y remédier, il a fallu utiliser la commande « pip install » suivie du nom du module à installer. L’important est d’éviter d’installer trop de modules différents sur le serveur, car ces derniers prennent de la place mémoire. Il faut donc essayer d’utiliser le plus possible les mêmes ressources sur plusieurs programmes.

Il existe plusieurs modules proposant des actions similaires, c’est le cas par exemple des modules *lxml* et *Element Tree* cités dans la partie « Partie 2 - Réalisation concrète ». Un choix est souvent possible entre des modules équivalents, il faut alors se renseigner sur les fonctionnalités de chacun d’eux et cibler précisément ce qui est attendu avec le programme créé. Ainsi, plusieurs recherches ont été faites dans le but de comprendre exactement le fonctionnement des modules visés, ainsi que leur installation sur le terminal du serveur.

2.1.3. Découverte des outils

La troisième nouveauté abordée ici est celle des nombreux outils employés lors de ce stage. La plupart des outils ayant servi dans ce projet m’étaient plus ou moins inconnus, c’est le cas particulièrement de Stanza. Le premier point a donc été de se renseigner sur le fonctionnement global de chaque outil afin d’anticiper les sorties obtenables. S’en est suivi de nombreuses erreurs imprévues, comme indiqué dans la partie « Partie 2 - Réalisation concrète », qui m’ont confrontée aux différentes limites de ces outils.

La découverte des limites liées à chaque outil a donné lieu à de multiples actions pour tenter de contrer au maximum leur effet indésirable. Cet aspect a considérablement participé à améliorer les compétences déjà acquises dans le domaine de la programmation Python. Elles ont également permis de développer les différentes évolutions des techniques d’analyse proposées dans le point « Évolutions liées aux analyses ».

2.1.4. Autres compétences

Enfin, le dernier point à citer concerne d’autres compétences développées grâce aux différentes tâches réalisées dans le cadre de ce stage. Celles-ci représentent un apprentissage important dans le domaine de la programmation en langage Python, notamment avec les

modules et bibliothèques utilisés, mais également les multiples manières d'obtenir le résultat souhaité. En effet, lorsqu'une manière n'était pas fonctionnelle, une autre pouvait l'être en utilisant moins de ressources ou en étant plus efficace.

De plus, ce stage m'a amenée à découvrir un nouveau format de fichier avec lequel je n'étais pas familière, car je ne l'avais jamais rencontré dans le cadre de ma formation. Il s'agit du format XML ConLL. Ce dernier est semblable au format XML classique en termes de syntaxe et de balisage, mais il est couplé avec le modèle de dépendances universelles UD à cause de Stanza. De ce fait, une première phase de mise à niveau a été nécessaire et m'a permis de compléter mes compétences.

2.2. Réactions aux problèmes

Le second aspect du développement de compétences concerne la réaction aux différents problèmes rencontrés. Ces derniers ont été multiples et possédaient des causes et des solutions totalement différentes, leur gestion a donc demandé quelques capacités.

2.2.1. Gestion des erreurs

La partie de gestion des erreurs fut la partie la plus gourmande en termes de temps, d'énergie et de compétences. En effet, j'ai été confrontée à plusieurs types d'erreurs, allant d'erreurs « bêtes » à des erreurs plus complexes. Il fallait donc s'adapter constamment, revoir son approche, et réinventer le code quand cela était nécessaire pour tenter de surmonter ces problèmes.

Dans la catégorie des erreurs « bêtes », il y a les erreurs de syntaxe retrouvées dans les fichiers XML d'origine et ayant empêché tout traitement. En effet, ces erreurs ont nécessité un temps de traitement supplémentaire afin de repérer tout d'abord l'emplacement et le type d'erreur, puis de la corriger (cf. point « Conversion »). Ces erreurs auraient pu être détectées avant, puisqu'aucun traitement informatique n'est possible sur des fichiers XML comportant des erreurs de syntaxe. Ces imprévus m'ont certes permis de développer des compétences, mais ils sont également responsables d'une perte de temps sur le projet.

Ensuite, les limites des conversions du format .xml au format .txt ont également généré de nombreuses erreurs ayant nécessité des traitements additionnels. En effet, comme précisé dans le point « Nettoyage », quelques balises ont été conservées lors de la conversion, ainsi que des *underscores* et d'autres erreurs. Ces problèmes ont aussi demandé un temps d'action non

prévu au départ, mais indispensable pour la suite des opérations. Leur résolution a permis de développer des capacités de programmation, mais également d'adaptation.

2.2.2. Adaptation

L'adaptation est un point tout aussi important dans le développement de compétences, car elle a été présente à chaque étape de ce stage. En effet, comme précisé dans le point précédent, la gestion des différentes erreurs rencontrées a nécessité une plus ou moins grande adaptation en fonction des situations. Cette adaptation m'a permis de mettre en pratique les nombreuses compétences acquises pendant mes études, en cherchant à obtenir le moyen le plus rapide, efficace et pertinent dans chaque situation.

De plus, il faut souligner l'aspect non contrôlable de la programmation lors de ce stage qui est la limite du serveur. En effet, certains programmes créés ne comportaient pas d'erreurs les empêchant de s'exécuter correctement, mais ils avaient besoin d'un grand nombre de ressources ou de mémoire. Dans certains cas, l'exécution de ces scripts menait à des échecs en raison d'une surcharge sur le serveur (cf. point « Serveur »). Une adaptation et un changement étaient alors obligatoires pour restreindre les ressources et se maintenir dans la limite permise.

3. Collaboration

Le dernier point soulevé dans ce bilan est celui de la collaboration. En effet, mon stage ayant pris place dans un projet de plus grande envergure, et regroupant plusieurs équipes de plusieurs domaines d'activités, il est indispensable de s'attarder sur les conséquences de cet aspect.

3.1. Au sein d'un projet

La participation à un projet d'une telle ampleur était une totale découverte pour moi. J'avais bien évidemment déjà participé à des projets de groupe lors de ma formation, mais cela n'a rien à voir avec l'organisation et le déroulé d'un projet tel que PREFAB. La première étape a été de se renseigner sur le but du projet, ainsi que les différentes étapes pour y parvenir. Cela m'a permis de développer de nouvelles connaissances sur les phrases préfabriquées.

Ensuite, j'ai découvert le mode de fonctionnement d'un gros projet rassemblant autant d'équipes à coordonner, impliquer, conduire et informer. Comme indiqué dans le point « Difficultés », il est compliqué de se mettre d'accord sur les annotations à effectuer et sur la définition concrète d'une PPI. De plus, il faut constamment s'assurer que les participants au

projet n'étant pas des spécialistes du domaine linguistique comprennent les enjeux et les solutions développées. Il est indispensable de conserver un niveau équivalent de compréhension pour chaque personne, une ligne directrice stricte et des dates à respecter afin de maintenir le projet dans les meilleures conditions.

Pour ma part, je n'ai pas découvert le travail d'équipe ni la communication grâce à ce projet, car j'y suis confrontée depuis plusieurs années dans ma vie personnelle. Cependant, le fait de devoir expliquer précisément ce que je fais, et pourquoi je le fais, à des membres totalement étrangers au domaine du traitement automatique des langues, m'a demandé un effort et m'a aidé à acquérir de nouvelles compétences. En effet, la vulgarisation de termes spécifiques ne faisait pas partie de mes aptitudes avant de participer à ce projet, car tous les projets auxquels j'ai participé auparavant ne regroupaient que des spécialistes du domaine. Il n'était donc pas nécessaire d'expliciter chaque terme de manière simplifiée. La vulgarisation fait maintenant partie de mes capacités.

Enfin, le fait de participer activement à un projet et d'y avoir sa place a également été quelque chose de nouveau pour moi. Comme précisé juste avant, j'ai déjà participé à des projets en équipe dans ma vie personnelle, mais chaque participant était considéré au même niveau car, même si les rôles différaient, tous étaient importants. Au début de mon stage, j'étais persuadée que je n'étais qu'une stagiaire qui n'allait participer au projet que pendant quelque mois, et ne réaliser que quelques traitements minimes. Finalement, j'ai été intégrée au projet de manière plus intense que je ne l'avais pensé. J'ai vraiment fait partie d'une équipe, j'ai participé à des réunions, j'ai dû faire des comptes-rendus à mes tuteurs, j'ai eu une vraie place pendant ces quelques mois. Le travail que j'ai pu fournir a été apprécié et reconnu par mes tuteurs de stage, Olivier Kraif et Agnès Tutin. Ce qui ne peut que me rendre fière et reconnaissante de cette confiance qu'ils ont placée en moi lorsqu'ils m'ont proposé de les rejoindre pour mon stage de fin d'études.

3.2. Avec une équipe

Le travail en collaboration avec une équipe que je ne connaissais pas avant de débuter mon stage fait également partie des nouveautés auxquelles j'ai été confrontée. En effet, comme précisé dans le point précédent, il a fallu que je m'adapte aux personnes avec qui je communiquais sur l'avancée de mes tâches, car certaines étaient considérées comme des spécialistes dans le domaine (mes tuteurs) et d'autres avaient besoin de plus d'explications.

Le fait de communiquer régulièrement avec mes tuteurs me permettait d'utiliser du vocabulaire « expert », ce qui me demandait moins d'efforts a priori, mais qui n'était pas forcément plus simple. En effet, je me devais de leur fournir des comptes-rendus précis lors de réunions régulières, mais également de leur expliquer rigoureusement les problèmes rencontrés. Je travaillais de manière autonome durant ce stage, mais mes tuteurs étaient toujours disponibles par mail au cas où un problème était vraiment retors. Dans ce cas, les explications fournies devaient être extrêmement précises, d'où la difficulté de communication d'une part mais le développement de nouvelles qualifications d'autre part.

Il faut également citer la participation de Zhiyuan Qiu lors de ce stage, car elle a fourni certains scripts permettant d'annoter les passages de discours direct dans les fichiers XML d'origine (cf. point « Annotation du discours direct »). Olivier Kraif m'a aussi partagé des codes qu'il a lui-même créés pour la réalisation de l'alignement. Je considère que ce mode de fonctionnement basé sur la passation de programmes peut être perçu comme une sorte de travail d'équipe, même si je ne suis pas entrée en contact direct avec Zhiyuan Qiu. De plus, comme précisé dans la partie « Autres remarques » point « Scripts », la récupération de scripts créés par d'autres personnes, et devant être réutilisés, demande certaines connaissances afin de comprendre le fonctionnement de ceux-ci et de pouvoir les adapter à l'objectif visé. Cela participe donc au développement de compétences.

Enfin, ma participation à la réunion du 3 juillet 2023 m'a permis de présenter la quasi-totalité des missions que j'avais accomplies à l'ensemble des équipes participant au projet (il ne manquait que la mesure du Dynamic Time Warping). Cette étape m'a confrontée à la fois à devoir résumer ces quelques mois de stage en quelques minutes, tout en expliquant de manière simplifiée pour que les non spécialistes me comprennent, et à la fois à m'exprimer publiquement devant des personnes totalement inconnues (à part mes tuteurs). Cette phase a été importante en termes de résumé concis mais concret, accessible à tous, mais aussi de présentation orale. Encore une fois, ces compétences ont été renforcées grâce à ce stage.

Conclusion

En conclusion, j'ai effectué mon stage de fin d'études de Master 2 en Sciences du Langage parcours Industries de la Langue en tant que stagiaire en traitement de données linguistiques au sein du laboratoire LIDILEM. Lors de ce stage de 4 mois, j'ai pu mettre en pratique mes connaissances théoriques acquises durant ma formation, et me suis confrontée aux aspects impondérables de l'analyse de données ainsi qu'aux difficultés se produisant dans le cadre d'un projet.

Après ma rapide intégration dans l'équipe, j'ai eu l'occasion de développer plusieurs programmes informatiques et de réaliser des actions concrètes sur les données.

Ce stage a été très enrichissant car il m'a permis de mettre en pratique de façon concrète les différentes compétences que j'avais assimilées tout au long de mon enseignement, mais également d'en développer de nouvelles grâce aux missions précises à effectuer. De plus, j'ai pu découvrir le mode de fonctionnement d'un projet tel que le projet PREFAB, coordonnant plusieurs équipes de différents domaines. Cela m'a amenée à parfaire mes compétences en matière de travail d'équipe et de communication, mais également à en acquérir de nouvelles au niveau de la vulgarisation de termes spécifiques.

Cette expérience m'a également apporté de nombreuses compétences personnelles. Celles-ci peuvent se décliner notamment autour de la confiance, de la patience, de l'autonomie, de l'adaptation ou encore de l'organisation. Autant de qualités qui me suivront dans le monde du travail, et me permettront de surmonter bien des obstacles.

Durant mon stage, j'ai également manipulé quelques outils informatiques existants et puissants dans le domaine du traitement automatique des langues, ce qui m'a permis de me familiariser avec leurs atouts mais aussi leurs limites. De plus, l'adaptation au système Linux, aux lignes de commande, ainsi que l'utilisation d'un serveur distant m'a amenée à développer de nouveaux savoir-faire pouvant m'être utiles dans le monde du travail. De ce fait, j'ai réalisé que la programmation et la création de scripts automatiques me passionne.

À la fin de mon stage, il a été démontré que les mesures de similarité obtenues grâce au procédé de Dynamic Time Warping ne sont efficaces qu'à 45% sur le corpus étudié. Le script utilisé pour l'alignement était en cours de développement et a depuis été modifié pour régler certains problèmes que j'ai pu rencontrer. Cette conclusion offre néanmoins une ouverture sur le développement de nouvelles techniques d'alignement, ou bien sur le renforcement des

techniques actuelles afin d'obtenir de meilleurs résultats. De cette façon, les analyses contrastives n'en seront que plus complètes.

Bibliographie

Akbik, A., Bergmann, T., Blythe, D., Rasul, K., Schweter, S., & Vollgraf, R. (2019). FLAIR: An Easy-to-Use Framework for State-of-the-Art NLP. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, 54-59.

Bird, S., Klein, E., Loper, E., & Baldridge, J. (2008). Multidisciplinary Instruction with the Natural Language Toolkit. *Proceedings of the Third Workshop on Issues in Teaching Computational Linguistics (TeachCL-08)*, 62-70.

Devlin, J., Chang, M-W., Lee, K., & Toutanova, K. (2019). BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 4171-4186.

Jalilian, E. (2023). *Journée PREFAB. Convergence entre syntaxe et sémantique.* (Présentation PowerPoint). Université Grenoble Alpes.

Lavignasse, S. (2005). La DTD et son langage XML: Une application pour la lexicographie contemporaine. *Éla. Études de linguistique appliquée*, n°137, 73-94.

Le, H., Vial, L., Frej, J., Segonne, V., Coavoux, M., Lecouteux, B., Allauzen, A., Crabbé, B., Besacier, L., & Schwab, D. (2019). FlauBERT: Unsupervised Language Model Pre-training for French.

Martin, L., Muller, B., Ortiz Suàrez, P. J., Dupont, Y., Romary, L., Villemonte de la Clergerie, É., Seddah, D., & Sagot, B. (2019). CamemBERT: a Tasty French Language Model.

Neumann, M., King, D., Beltagy, I., & Ammar, W. (2019). ScispaCy: Fast and Robust Models for Biomedical Natural Language Processing. *Proceedings of the 18th BioNLP Workshop and Shared Task*, 319-327.

Nivre, J., De Marneffe, M-C., Ginter, F., Hajic, J., D. Manning, C., Pyysalo, S., Schuster, S., Tyers, F., & Zeman, D. (2020). Universal Dependencies v2: An Evergrowing Multilingual Treebank Collection. *Proceedings of the Twelfth Language Resources and Evaluation Conference*, 4034–4043.

Obeid, O., Zalmout, N., Khalifa, S., Taji, D., Oudah, M., Alhafni, B., Inoue, G., Eryani, F., Erdmann, A., & Habash, N. (2020). CAMeL Tools: An Open Source Python Toolkit for Arabic Natural Language Processing. *Proceedings of the Twelfth Language Resources and Evaluation Conference*, 7022-7032.

Qi, P., Zhang, Y., Zhang, Y., Bolton, J., & D. Manning, C. (2020). Stanza: A Python Natural Language Processing Toolkit for Many Human Languages. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, 101-108.

Rodrigo, A., Monteleone, M., & Reyes, S. (2018). A Pedagogical Application of NooJ in Language Teaching: the Adjective in Spanish and Italian. *Proceedings of the First Workshop on Linguistic Resources for Natural Language Processing*, 47-56.

Roux, C. (2004). Annoter les documents XML avec un outil d'analyse syntaxique. *Actes de la 11ème conférence sur le Traitement Automatique des Langues Naturelles. Articles longs*, 289-298.

Silberztein, M. (2005). NooJ: A Linguistic Annotation System For Corpus Processing. *Proceedings of HLT/EMNLP 2005 Demonstration Abstracts*, 10-11.

Silberztein, M. (2015). *La formalisation des langues, l'approche de NooJ*. ISTE Editions. repéré sur ResearchGate.

Silberztein, M., & Tutin, A. (2005). NooJ, un outil TAL pour l'enseignement des langues. Application pour l'étude de la morphologie lexicale en FLE. *Open Edition Journals, 8 (2)*, 123-134.

Tutin, A. (2021). Constructions des phrases PREFABriquées dans les interactions langagières (PREFAB). *AAPG2021*.

Tutin, A. (2019). Phrases préfabriquées des interactions : quelques observations sur le corpus CLAPI. *Cahiers de Lexicologie, 2019, Les phrases préfabriquées : Sens, fonctions, usages*, 114, 63-91.

Tutin, A. (2023). *PREFAB Réunion d'étape 03/07/23.* (Présentation PowerPoint). Université Grenoble Alpes.

Tutin, A. (2020). Tu parles! Et puis quoi encore! Phrases préfabriquées à fonction expressive dans les dictionnaires français. Congrès Mondial de Linguistique Française CMLF 2020, Jul 2020, Montpellier (en ligne), France.

Verner, J., & Vernerová, A. (2020). PyValLex: A Processing System for Valency Lexicon Data. *Proceedings of the Twelfth Language Resources and Evaluation Conference*, 7187-7193.

Sitographie

Agence Nationale de la Recherche. (2022). *Constructions des phrases préfabriquées dans les interactions langagières (PREFAB) – PREFAB.* https://anr.fr/Projet-ANR-22-CE54-0013

Agence Nationale de la Recherche. (2008). *Corpus prosodique de référence en français parlé – RHAPSODIE*. https://anr.fr/Projet-ANR-07-CORP-0030

Agence Nationale de la Recherche. (2013). *Outils et Recherches sur le Français Écrit et Oral – ORFEO*. https://anr.fr/Projet-ANR-12-CORP-0005

Agence Nationale de la Recherche. (2015). *Segmentation de corpus oraux – SegCor*. https://anr.fr/Projet-ANR-15-FRAL-0004

Association for Computers and the Humanities, Association for Computational Linguistics, & Association for Literary and Linguistic Computing. (1987). Text Encoding Initiative. https://tei-c.org/

Atwood, J. & Spolsky, J. (2008). Stack Overflow. https://stackoverflow.com/

Baier, T. (2014). NooJ cmd. https://github.com/numblr/nooj-cmd

Bird, S. & Klein, E. (2001). Natural Language Toolkit. https://www.nltk.org/

CLLE. (2013). GLAFF, Un Gros Lexique À Tout Faire du Français. http://redac.univ-tlse2.fr/lexiques/glaff.html

Dib, F. (2019). Regex101. https://regex101.com/

Edmonds, M. (2020). *A brief history of Optical Character Recognition (OCR)*. https://www.pitneybowes.com/content/dam/pitneybowes/uk/en/shipping-and-mailing/e-invoicing/Blog_E-invoicing-The_Brief_History_of_OCR.pdf?itid=lk_inline_enhanced-template

Google. (2018). BERT. https://github.com/google-research/bert

Google. (2006). Google Translate. https://translate.google.fr/?hl=fr

Google. (2021). LaBSE. https://tfhub.dev/google/LaBSE/2

Google. (2021). universal-sentence-encoder-cmlm/multilingual-preprocess. https://tfhub.dev/google/universal-sentence-encoder-cmlm/multilingual-preprocess/2

Ho, D. (2003). Notepad++. https://notepad-plus-plus.org/

Honnibal, M. (2015). SpaCy. https://spacy.io/

Hugging Face. (2020). CamemBERT. https://huggingface.co/camembert-base

Hugging Face. (2019). FlauBERT. https://huggingface.co/flaubert/flaubert_base_uncased

Klie, J.-C., Bugert, M., Boullosa, B., Eckart de Castilho, R. & Gurevych, I. (2018). Inception. https://inception.atilf.fr/

Kosse, T. (2001). FileZilla. https://filezilla-project.org/index.php

Kraif, O. & Diwersy, S. (2011). Lexicoscope. http://phraseotext.univ-grenoble-alpes.fr/lexicoscope_2.0/

Krzyżanowska, A., & Grossmann, F. (2018). *Pragmatèmes en contraste : de la modélisation linguistique au codage lexicographique – PRAGMALEX*. https://www.umcs.pl/fr/descriptif-du-projet,15298.htm

LATTICE, MODYCO, ATILF, LIF, LORIA, CLLE, & ICAR. (2013). *ORFEO, Corpus d'Etude pour le Français Contemporain (CEFC)*. https://repository.ortolang.fr/api/content/cefc-orfeo/11/documentation/site-orfeo/home/index.html

Leibniz Institut für Deutsche Spräche. (2014). DeReWo – Korpusbasierte Grund-/Wortformenlisten. https://www.ids-mannheim.de/digspra/kl/projekte/methoden/derewo/

Lerner, S., & Bosack, L. (1984). Cisco Systems. https://www.cisco.com/c/fr_be/products/security/anyconnect-secure-mobility-client/index.html

Nivre, J., McDonald, R., & al. (2014). Universal Dependencies. https://universaldependencies.org/

Python Software Foundation. (2001). Python Docs. https://docs.python.org/3/

Ratschiller, T. & Delisle, M. (1998). https://www.phpmyadmin.net

Shuttleworth, M. (2004). Documentation Ubuntu. https://doc.ubuntu-fr.org/

Silberztein, M. (2002). NooJ. https://nooj.univ-fcomte.fr/

Silberztein, M. (1993). INTEX. http://intex.univ-fcomte.fr/

Stanford NLP Group. (2020). Stanza. https://stanfordnlp.github.io/stanza/

Stanford University. (2000). The Stanford Natural Language Processing Group. https://nlp.stanford.edu/

Université Côte d'Azur. (2000). Bases, Corpus, Langage (BCL). https://univ-cotedazur.fr/laboratoires/bases-corpus-langage-bcl

Université de Lorraine. (2001). ATILF. https://www.atilf.fr/

Université de Lorraine. (2001). Axe Lexicographie franco-allemande. https://www.atilf.fr/recherche/equipes/lexique/lexicographie-franco-allemande/

Université d'Orléans. (2008). Eslo Enquêtes Sociolinguistiques à Orléans. http://eslo.huma-num.fr/index.php/pagecorpus/pagepresentationcorpus

Université Grenoble Alpes. (2019). Institut MIAI Grenoble Alpes. https://miai.univ-grenoble-alpes.fr/institut-miai/

Université Grenoble Alpes. (2019). Institut MIAI Grenoble Alpes. miai-server.u-ga.fr

Université Grenoble Alpes. (1987). Lidilem. https://lidilem.univ-grenoble-alpes.fr/

Université Grenoble Alpes. (2016). PhraseoRom – La phraséologie du roman. https://phraseorom.univ-grenoble-alpes.fr/fr

Université Grenoble Alpes. (2019). PhraseoBase. http://phraseotext.univ-grenoble-alpes.fr/phraseobase/index.html

Université Lyon 2. (2003). ICAR. http://icar.cnrs.fr/

Van Rossum, G. (1991). Python. https://www.python.org/

Wendling, C., Treleaven, N., Brush, M., Tröger, E., Lanitz, F., Trotman, L., & Techet, J. (2005). Geany. https://www.geany.org/

Willems, L. (2015). *Tutoriel pour maîtriser les expressions régulières*. https://www.lucaswillems.com/fr/articles/25/tutoriel-pour-maitriser-les-expressions-regulieres

Yuan, E. S. (2011). Zoom. https://zoom.us/fr

Zalando Research. (2018). Flair. https://github.com/flairNLP/flair

Glossaire

Booléen : désigne un type de variable ne pouvant avoir que 2 valeurs, soit vrai soit faux.

Cluster : désigne un agglomérat de valeurs ayant des similarités, synonyme de regroupement.

Débogage : désigne l'action de correction des bugs présents dans les programmes informatiques.

Embedding : désigne une représentation vectorielle présentée sous forme numérique permettant de récupérer des informations sur le sujet étudié.

Entité nommée : désigne des objets textuels (mot, ou groupe de mots) catégorisables dans des classes telles que noms de personnes, noms d'organisations ou d'entreprises, noms de lieux…

Expression régulière : désigne une modélisation d'une chaîne de caractères, grâce à une syntaxe précise, représentant un ensemble de chaînes de caractères possibles.

Lemme : désigne une entrée d'un lexique d'une langue, par exemple un verbe à l'infinitif ou un nom au masculin singulier, qui correspond à la forme canonique d'un mot.

Lemmatisation : désigne le traitement lexical permettant d'appliquer un lemme à une forme.

Match : désigne le résultat renvoyé par une expression régulière.

Océrisation : désigne le fait d'extraire les données des documents grâce à la reconnaissance des caractères. Technique qui tire son nom de l'optical character recognition (OCR).

Parser / parsing : désigne le fait de parcourir le contenu d'un texte ou d'un fichier en l'analysant pour expliciter sa syntaxe ou en extraire des éléments.

Token : désigne généralement un mot, mais peut aussi désigner un symbole spécifique dans un texte (par exemple une ponctuation ou des guillemets).

Tokenisation : désigne le traitement lexical permettant de convertir une chaîne de caractères (un texte) en tokens.

Sigles et abréviations utilisés

ALR : Arbres Lexico-syntaxiques Récurrents

ANR : Agence Nationale de la Recherche

API : Alphabet Phonétique International

ATILF : Analyse et Traitement Informatique de la Langue Française

BERT : Bidirectional Encoder Representations from Transformers

BCL : Bases, Corpus, Langage

CLAPI : Corpus de LAngue Parlée en Interaction

CPP : Construction de Phrases Préfabriquées

DBSCAN : Density-Based Spatial Clustering of Applications with Noise

DEREKO : DEutsche REferenzKOrpus

DTD : Document Type Definition (Description de Type de Document)

DTW : Dynamic Time Warping

FTP : File Transfer Protocol (protocole de transfert de fichier)

GLAFF : Gros Lexique À Tout Faire du Français

GLFA : Groupe de Lexicographie Franco-Allemande

HTML : HyperText Markup Language

IA : Intelligence Artificielle

ICAR : Interactions, Corpus, Apprentissages, Représentations

LABSE : Language-specific BERT Sentence Embedding

LIDILEM : Linguistique et Didactique des Langues Étrangères et Maternelles

MIAI : Multidisciplinary Institute in Artificial Intelligence

MIASHS : Mathématiques et Informatique Appliquées aux Sciences Humaines et Sociales

NLP : Natural Language Processing

NLTK : Natural Language Toolkit

OCR : Optical Character Recognition (reconnaissance optique de caractères)

PNG : Portable Network Graphics

POS : Part Of Speech (parties du discours)

PPI : Phrases Préfabriquées des Interactions

PREFAB : Constructions des phrases préfabriquées dans les interactions langagières

REGEX : REGular EXpression (expression régulière)

SAMPA : Speech Assessment Methods Phonetic Alphabet

SFTP : Secure File Transfer Protocol (protocole de transfert de fichier sécurisé)

SSH : Secure SHell

TAL : Traitement Automatique des Langues

TEI : Text Encoding Initiative

TMX : Translation Memory eXchange

TSV : Tabulation-Separated Values

UD : Universal Dependencies

UMR : Unité Mixte de Recherche

VPN : Virtual Private Network (réseau privé virtuel)

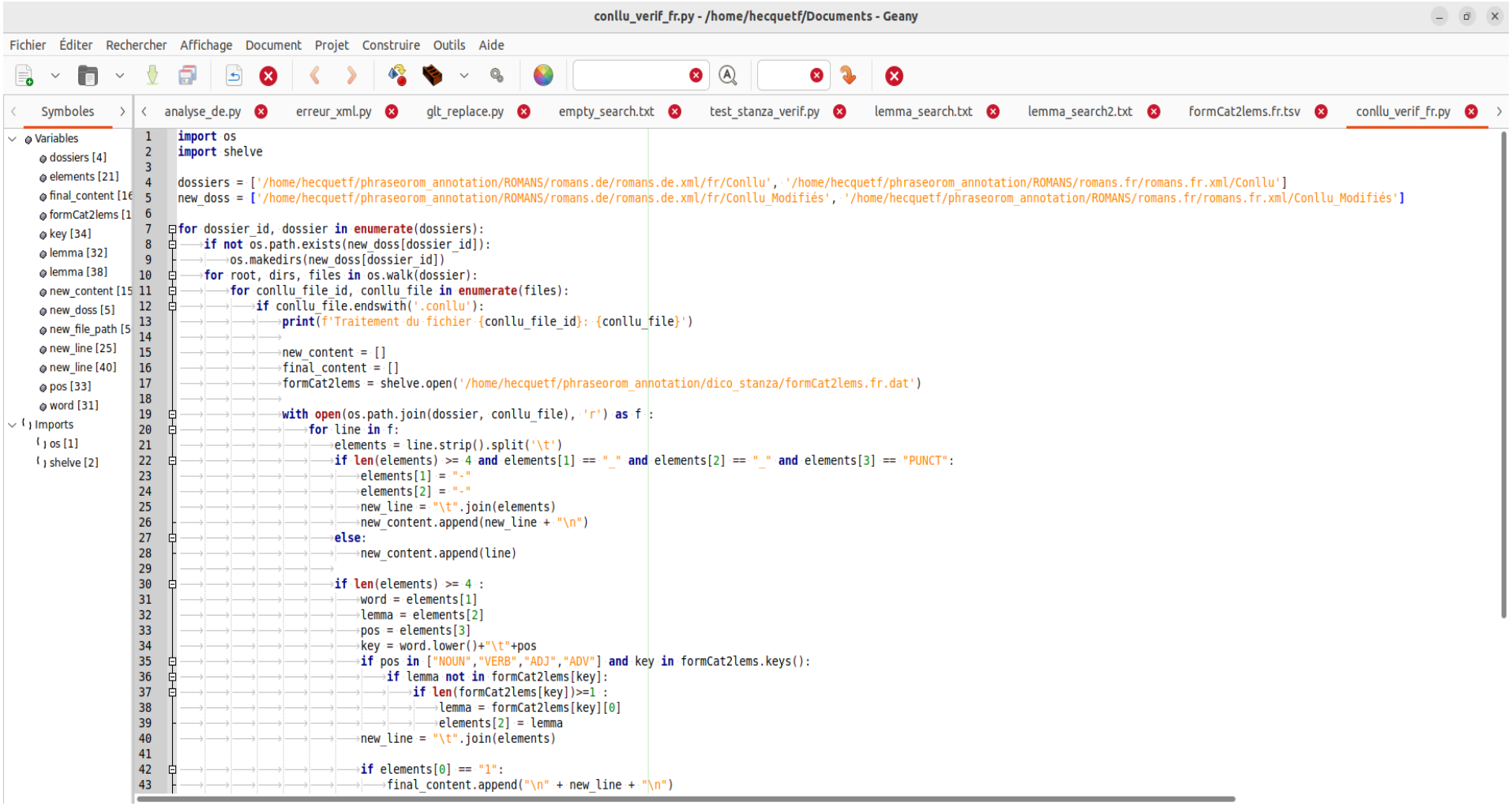
XIP : Xerox Incremental Parser

XML : eXtensible Markup Language

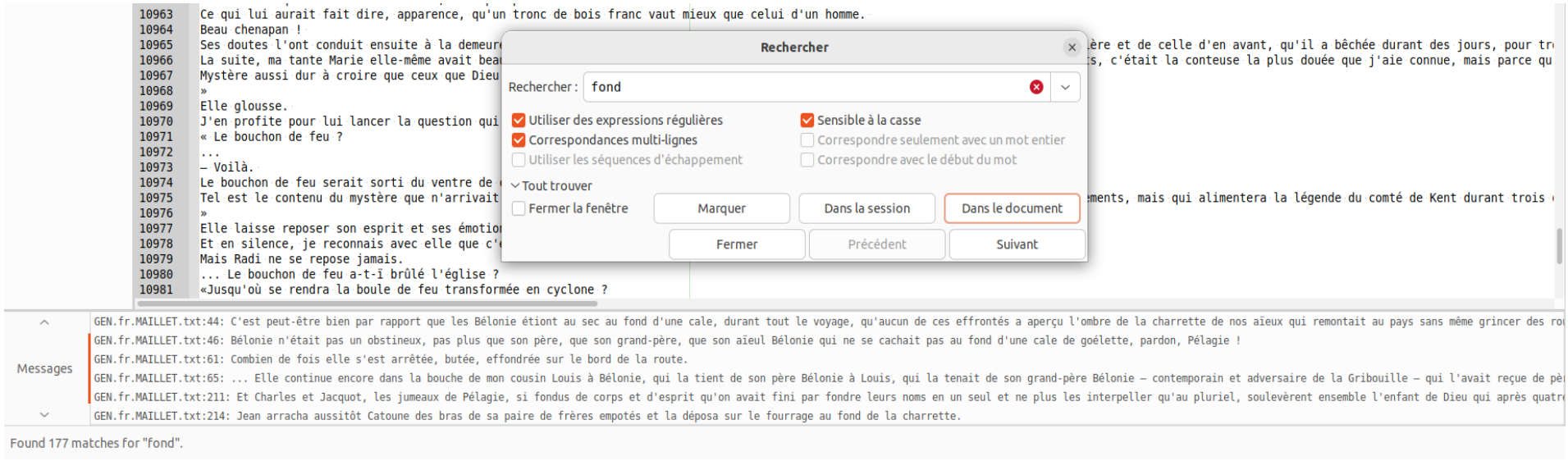
Table des illustrations

Table des annexes

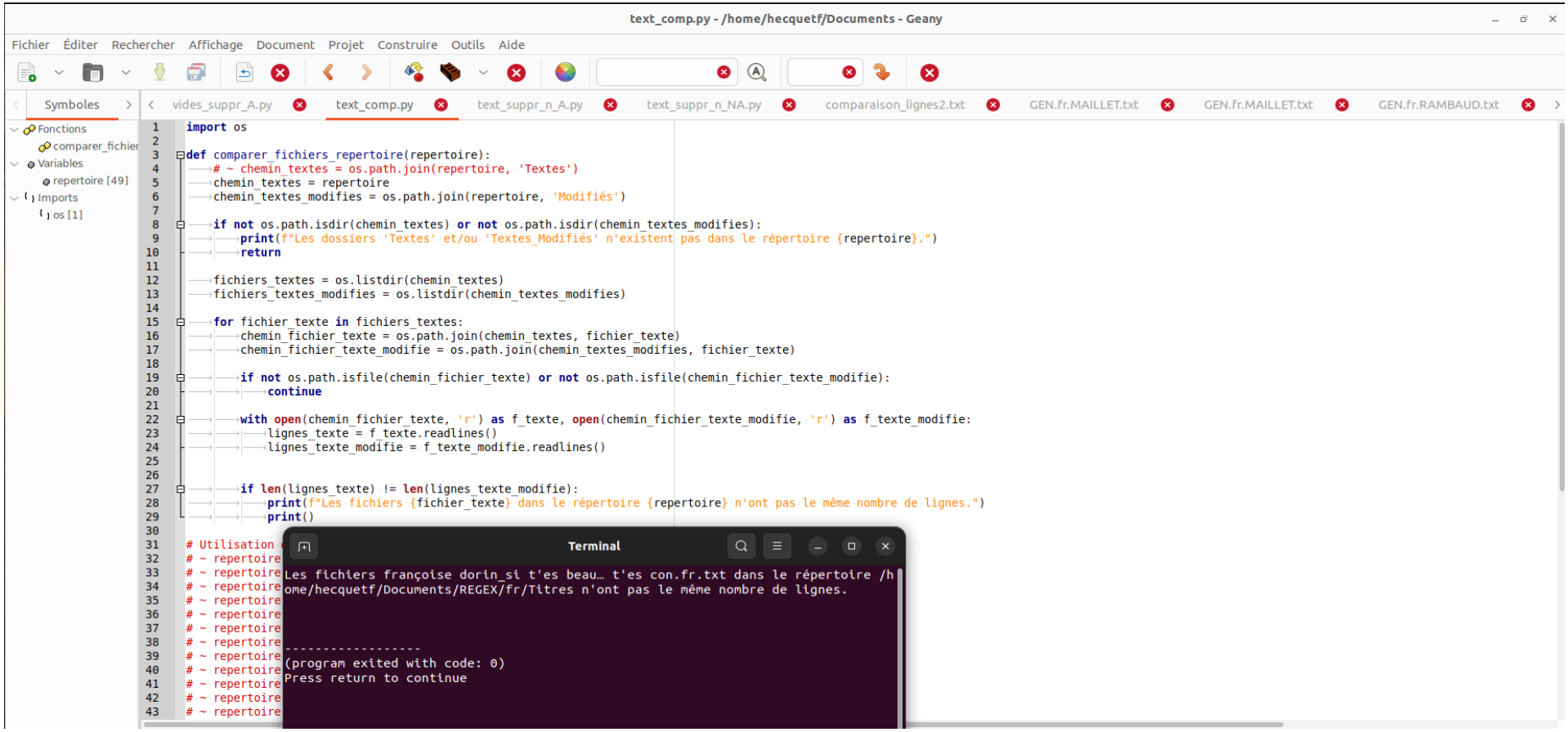
Annexe 1
Interface de Geany



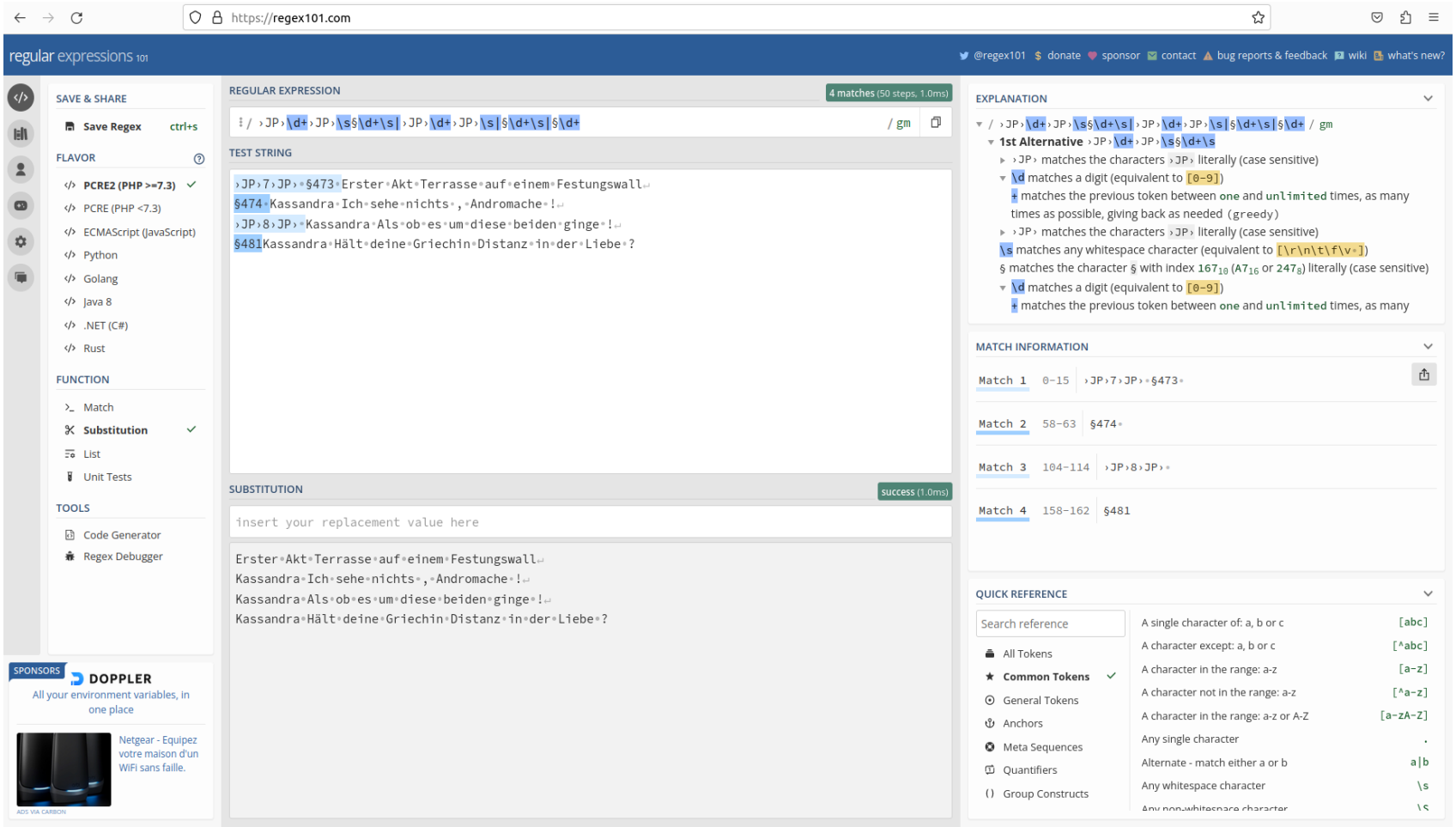
Annexe 2
Résultat d'une recherche sur Geany



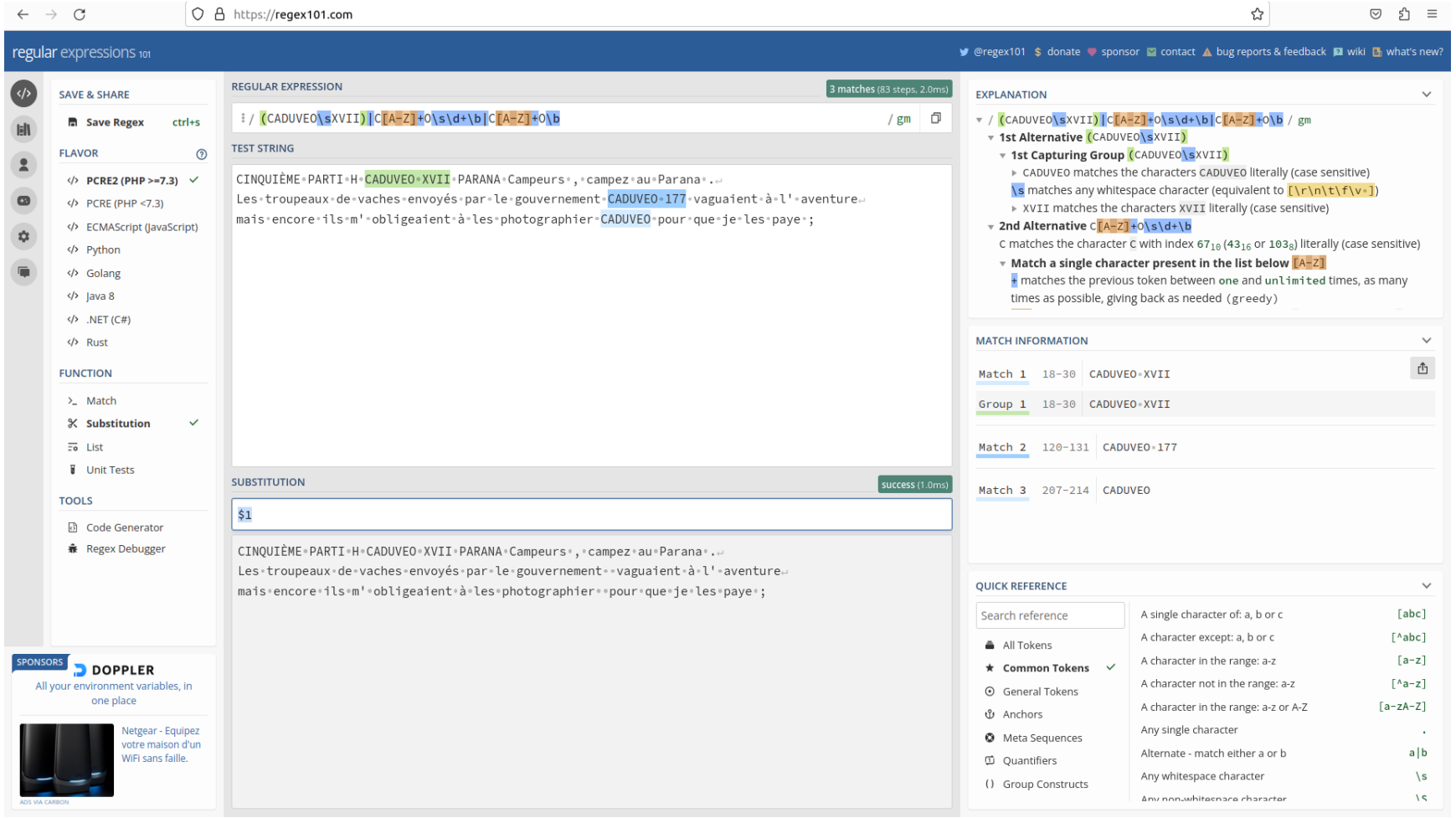
Annexe 3
Exécution d'un script en local



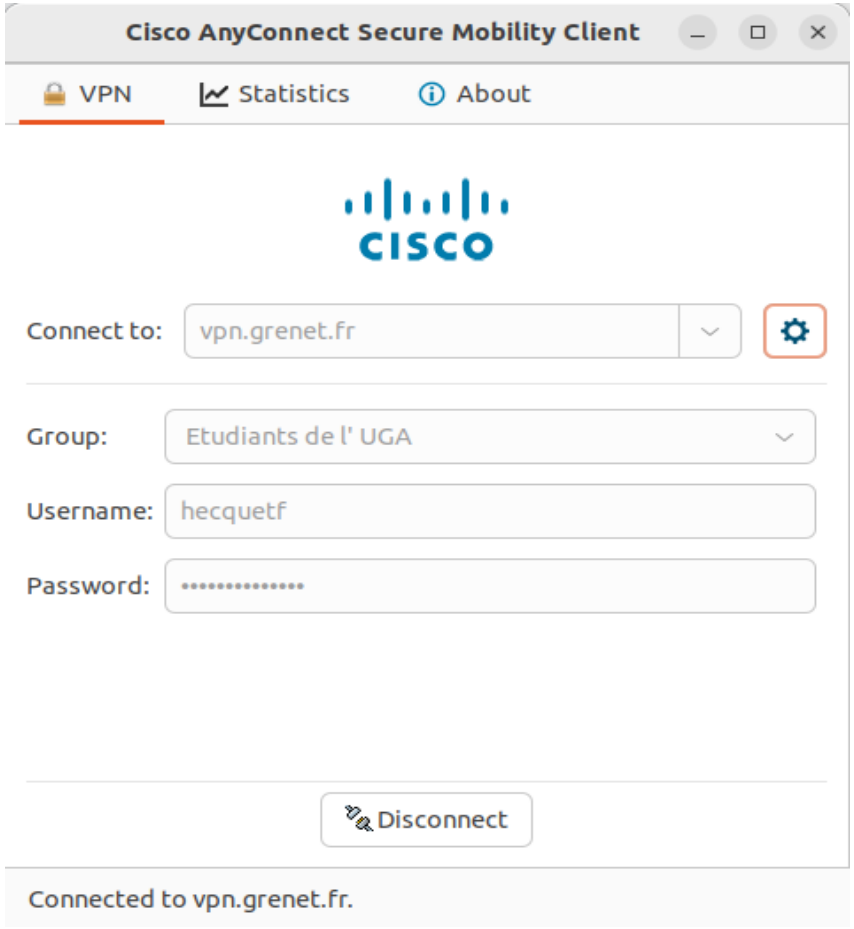
Annexe 4
Exemple de regex de suppression

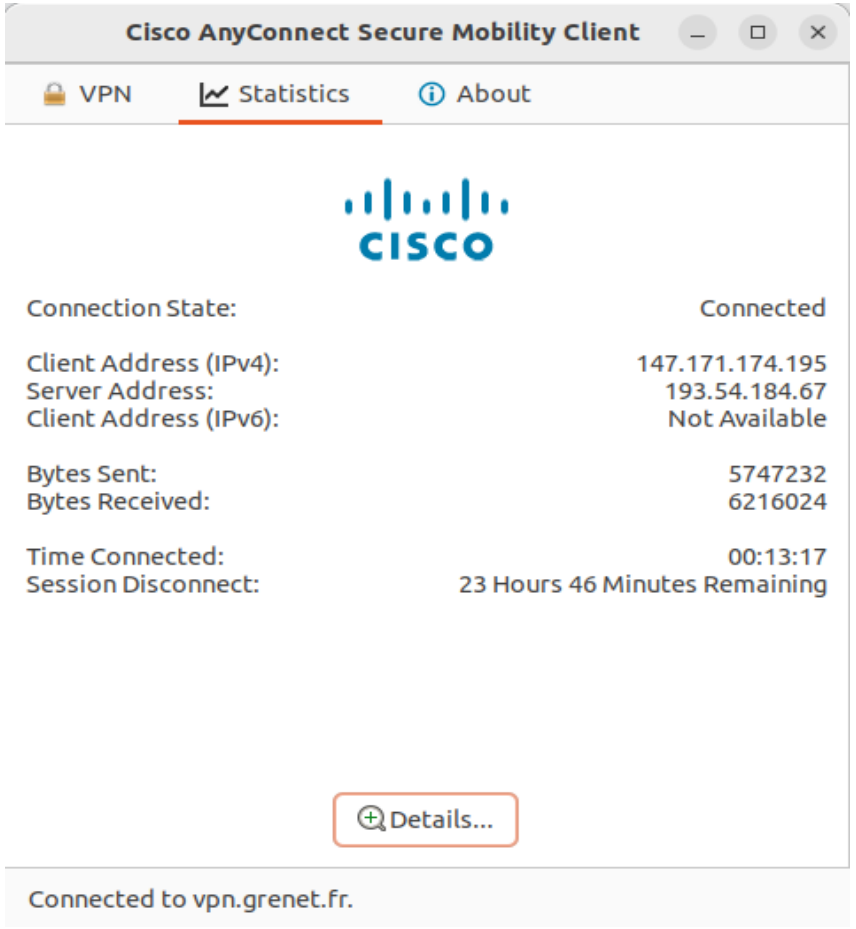


Annexe 5
Exemple de regex de remplacement

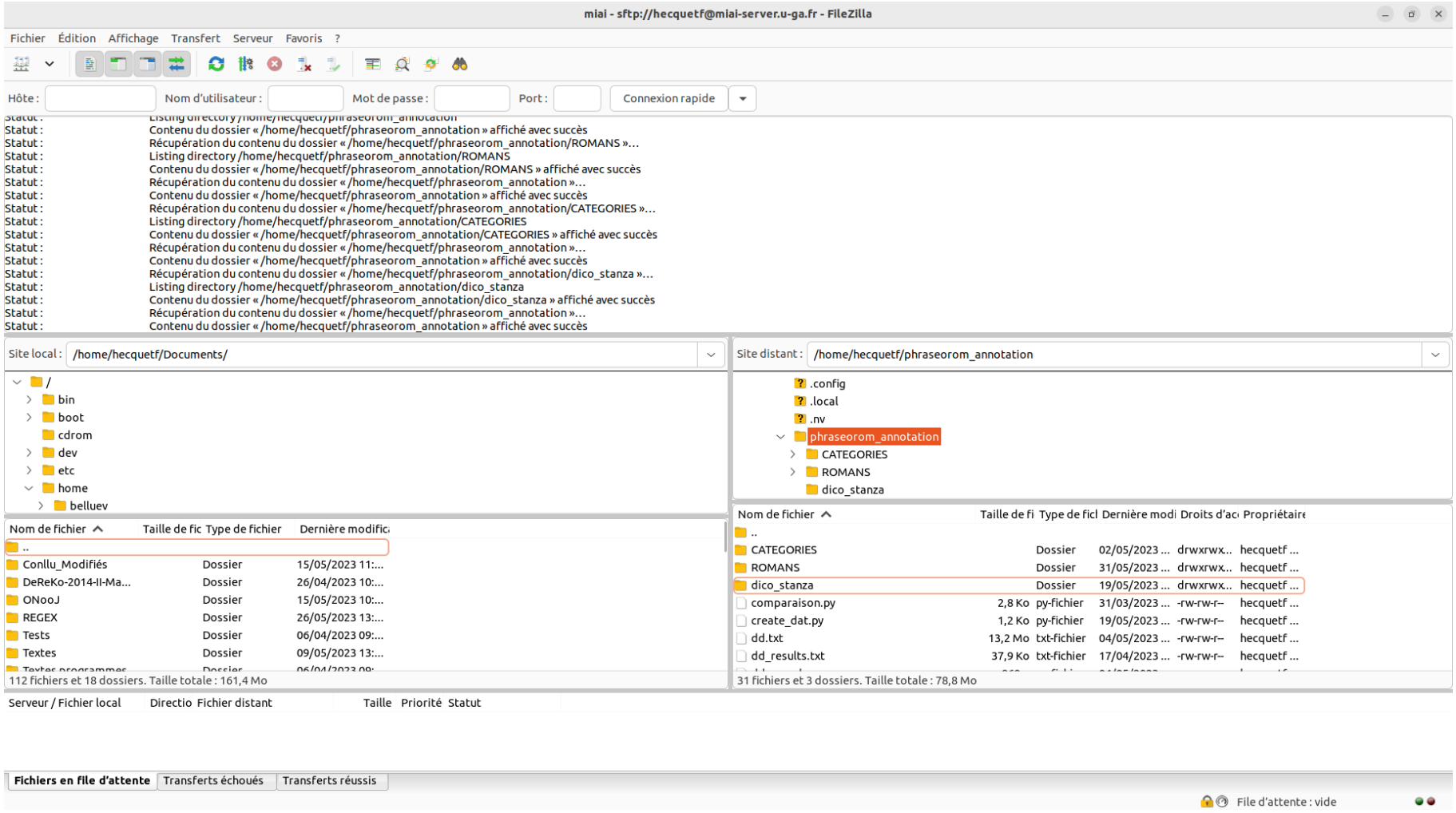


Annexe 6
Connexion au VPN via Cisco AnyConnect

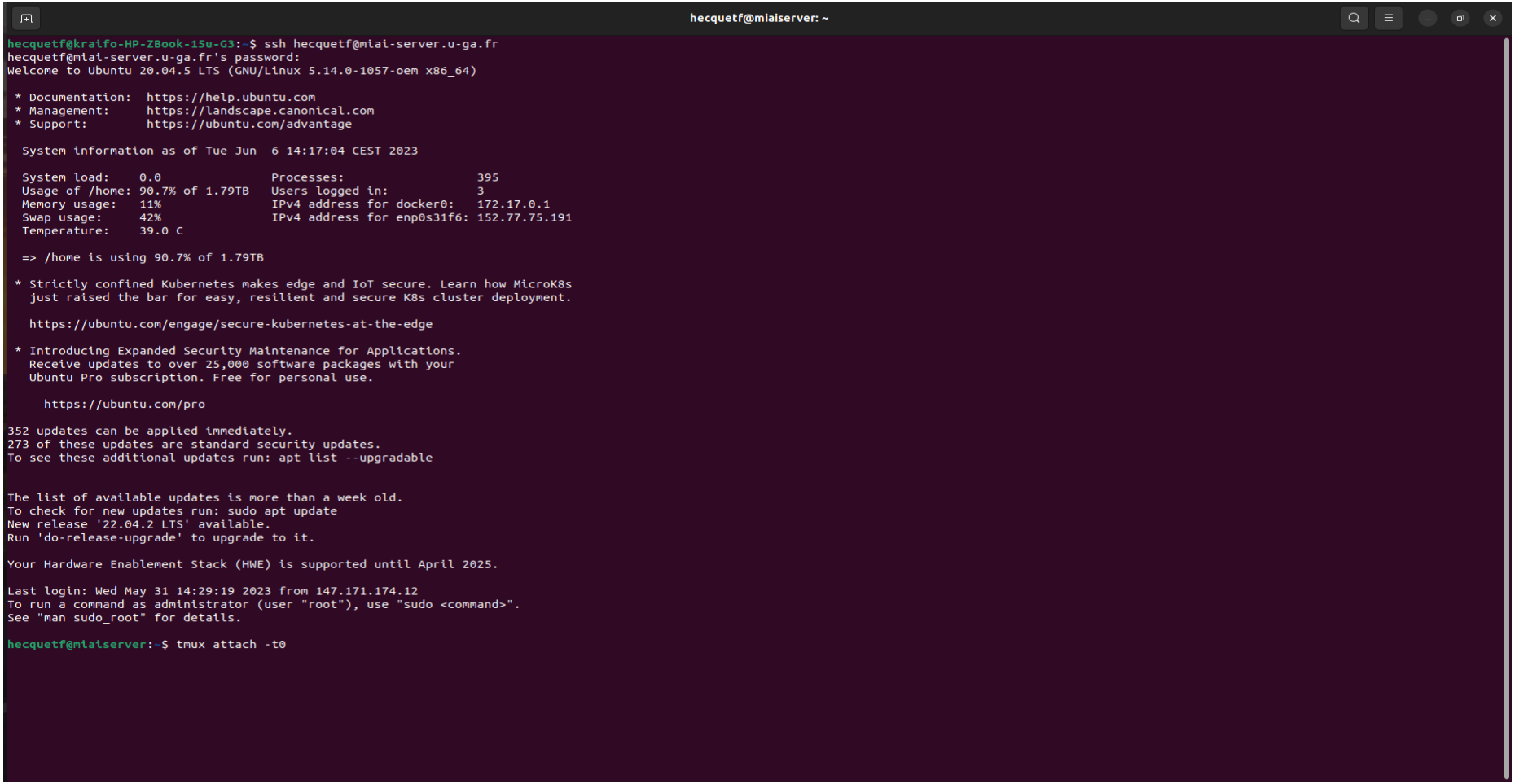




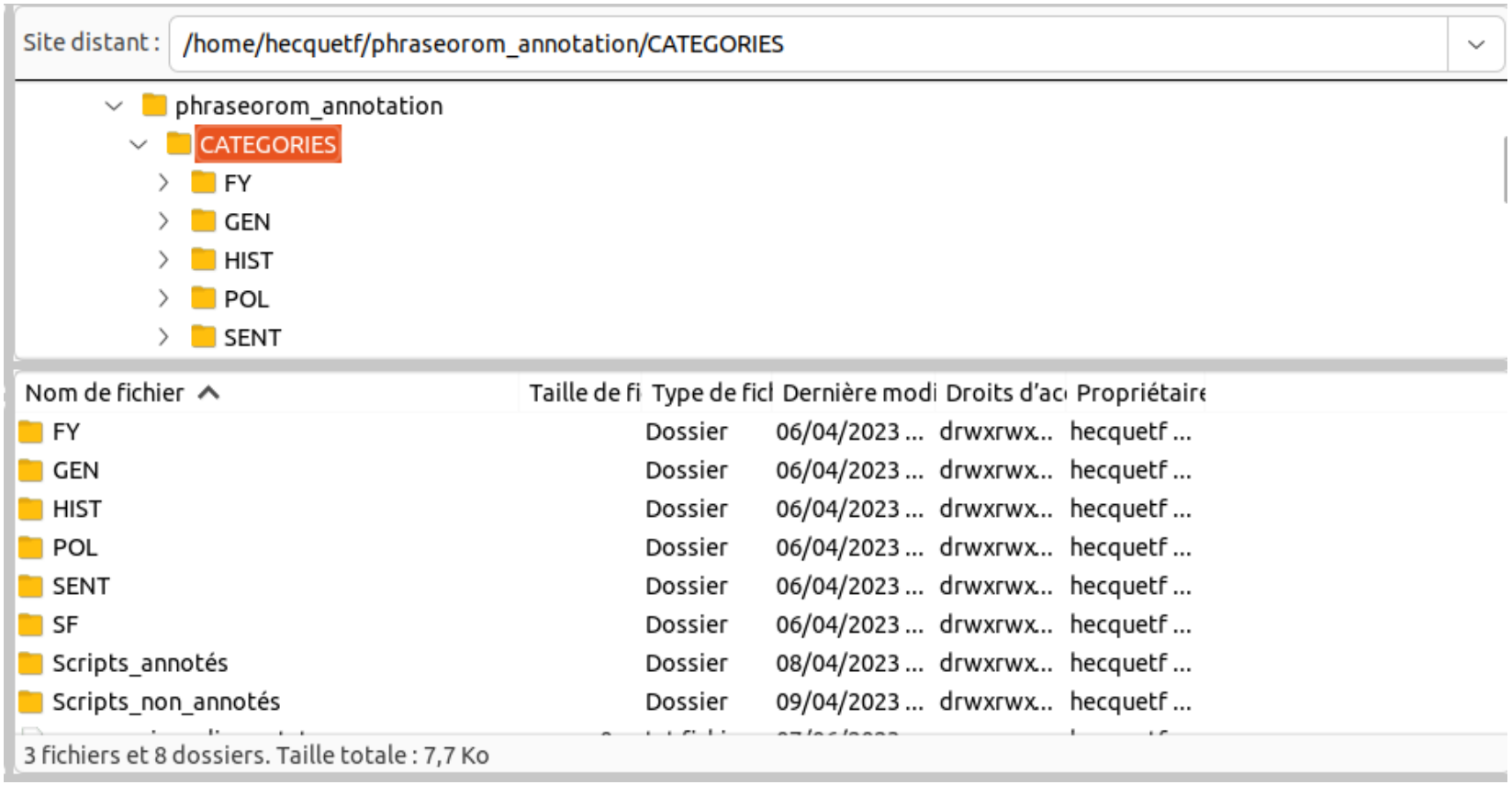
Annexe 7
Interface de FileZilla



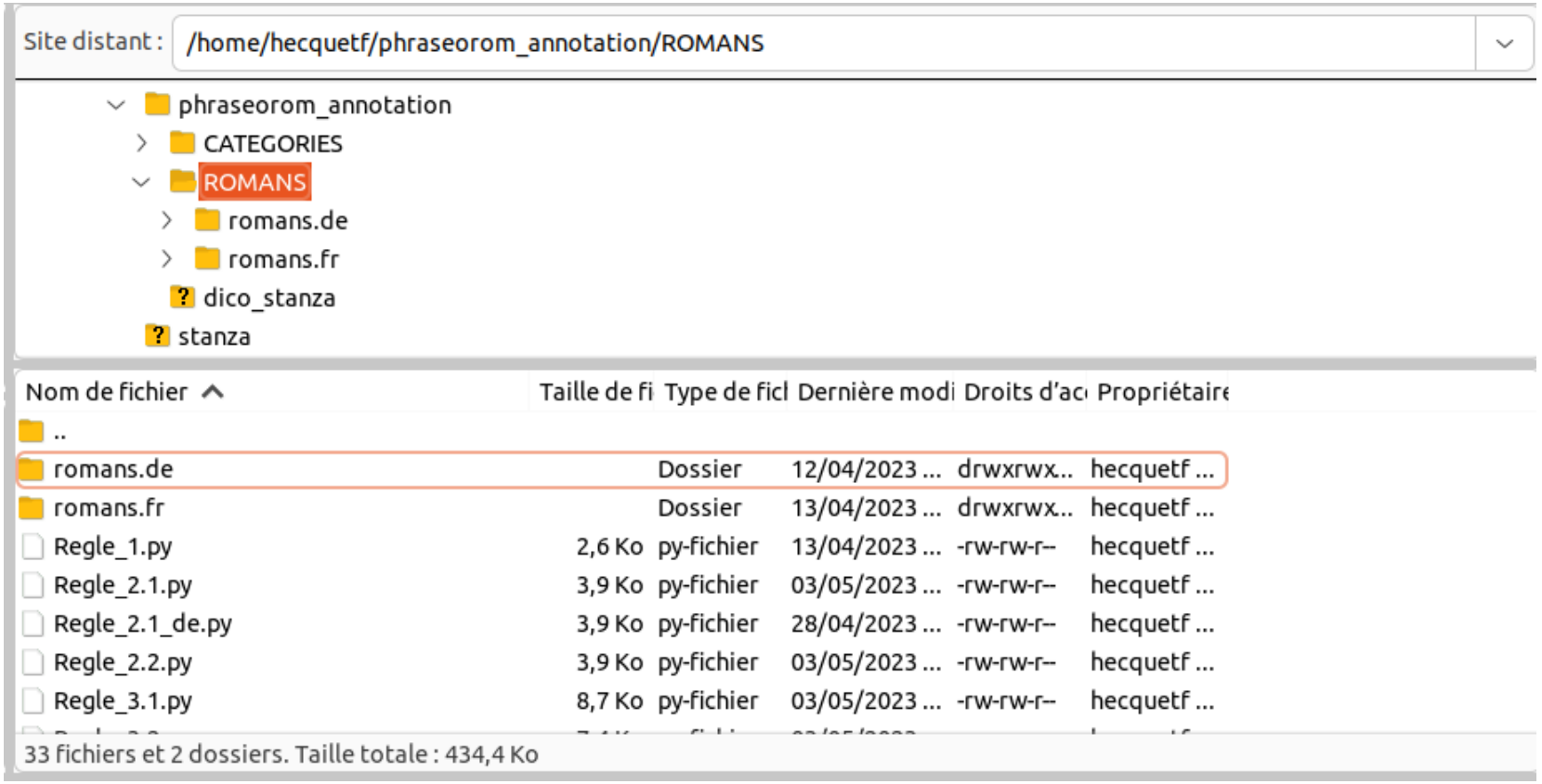
Annexe 8
Connexion au serveur MIAI avec SSH



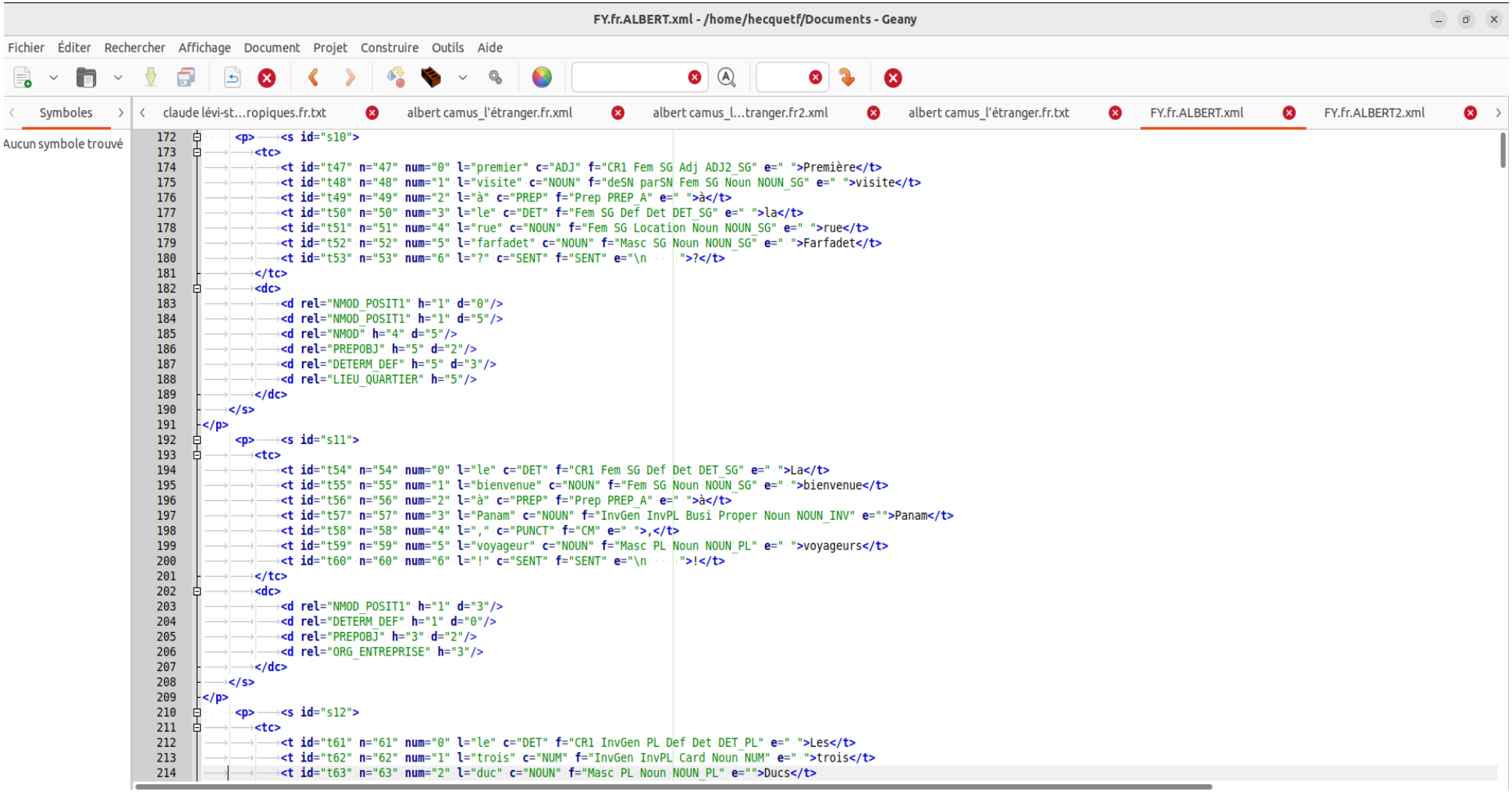
Annexe 9
Corpus Phraseobase



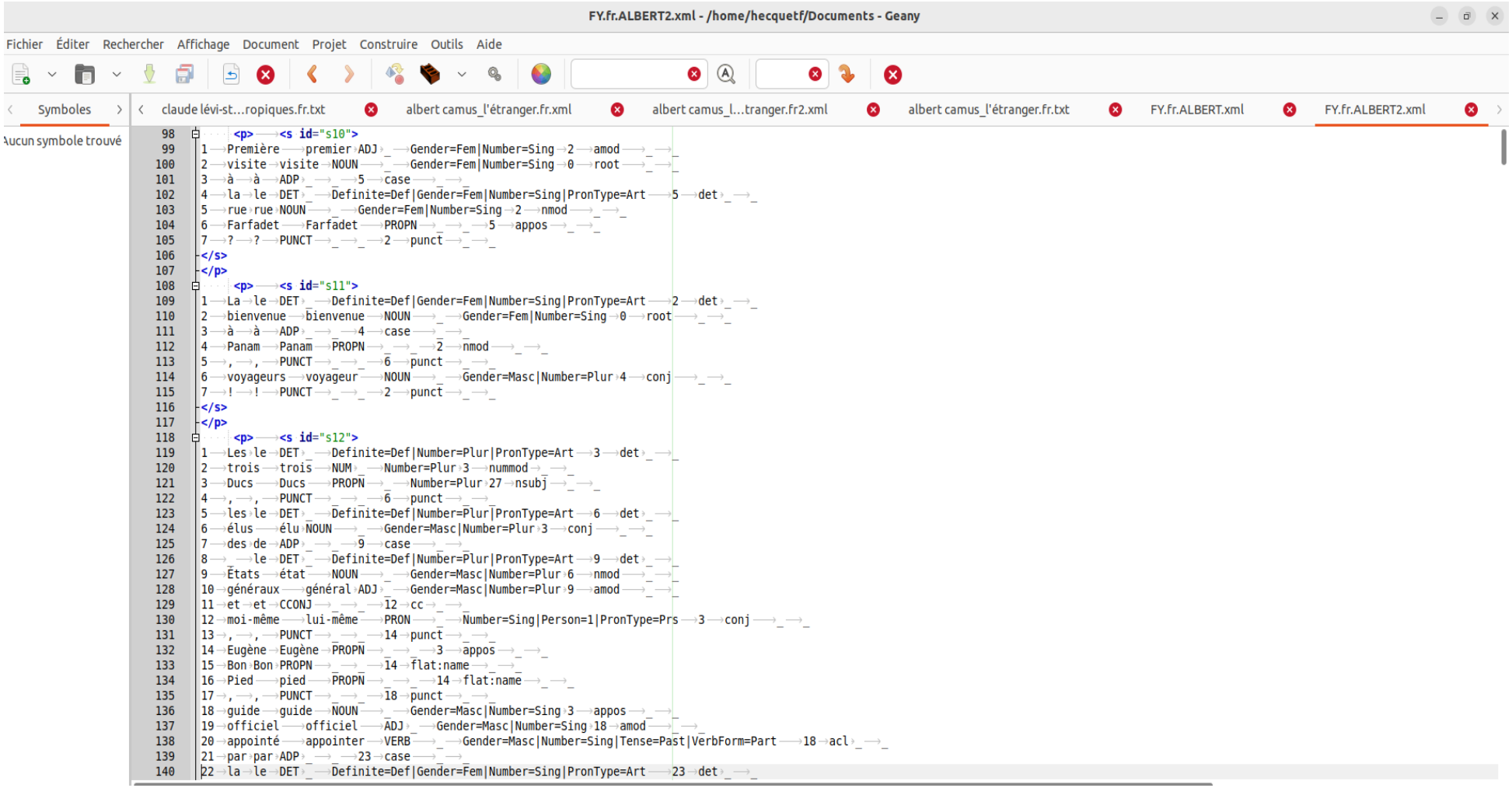
Annexe 10
Corpus GLFA



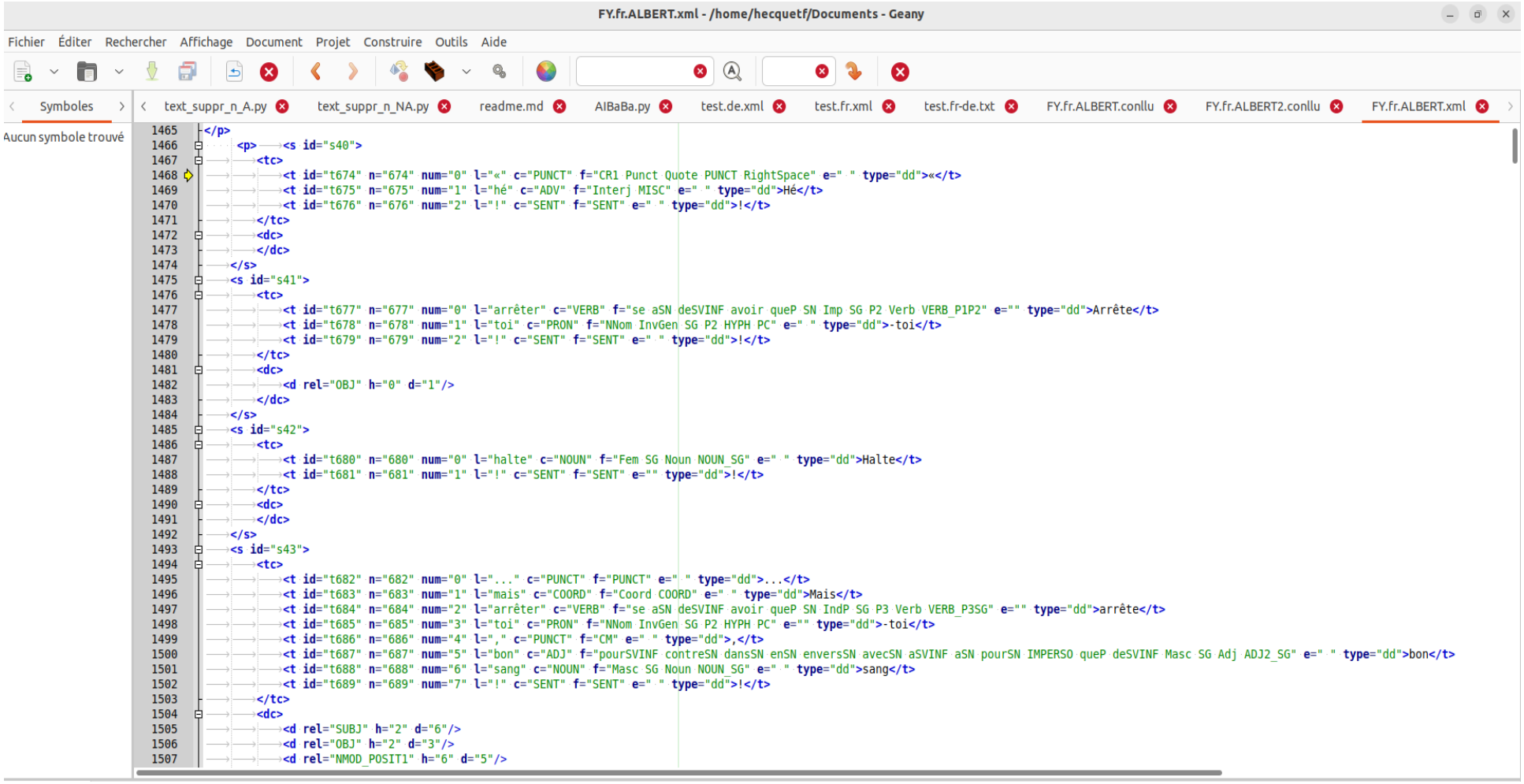
Annexe 11
Format XML d'origine



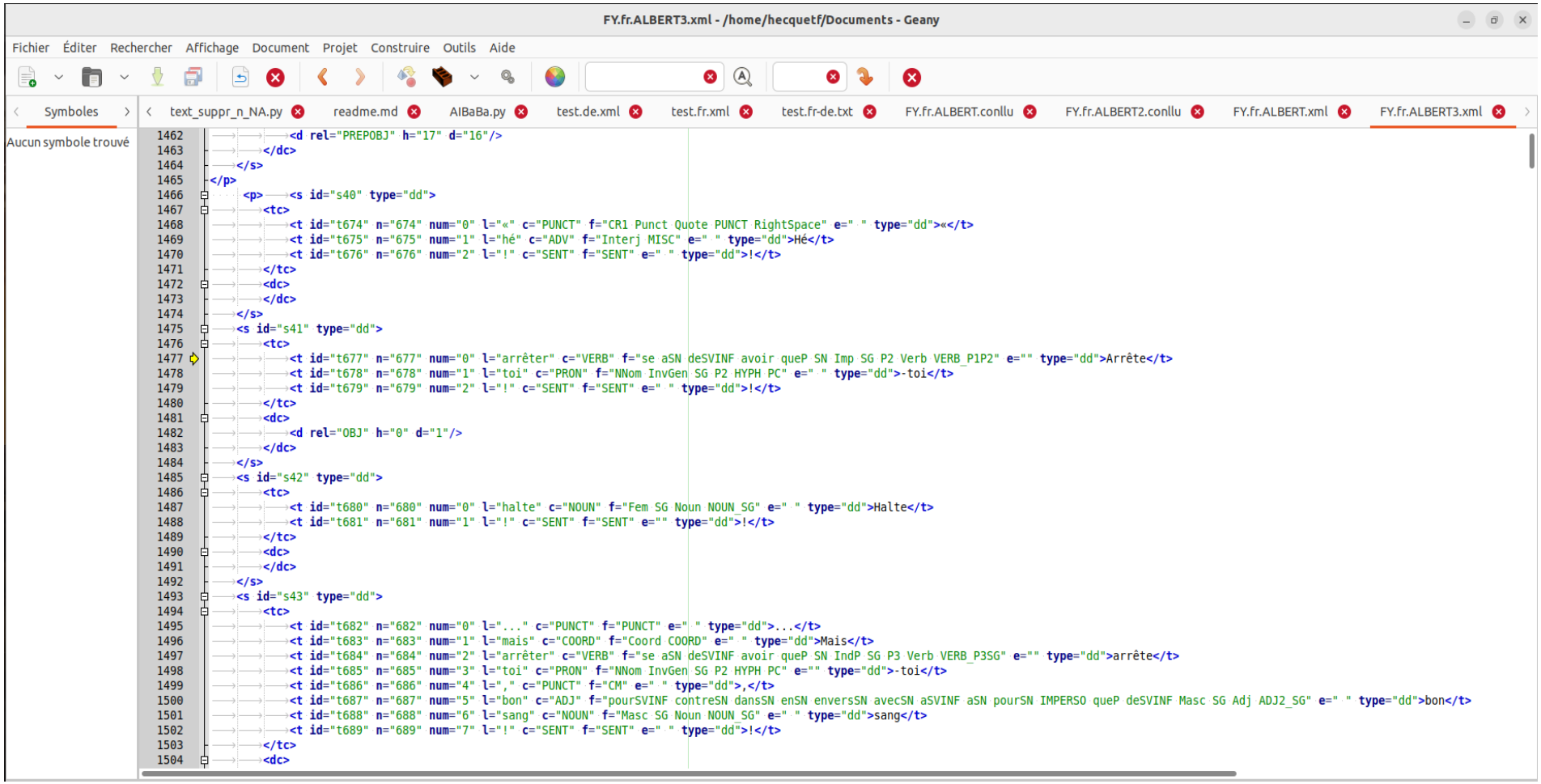
Annexe 12
Format XML ConLL



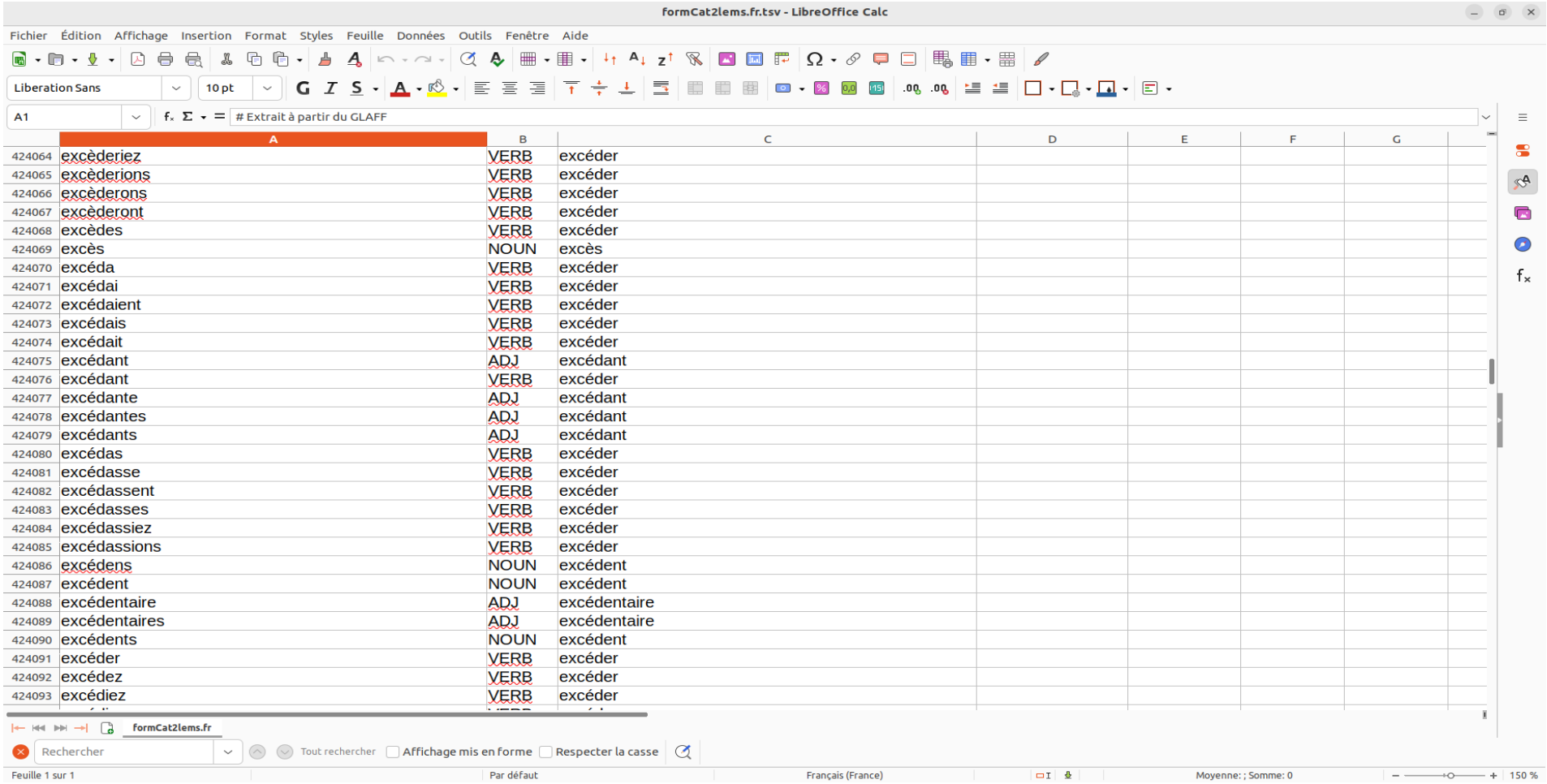
Annexe 13
Annotation de discours direct au niveau des tokens



Annexe 14
Annotation de discours direct au niveau des phrases

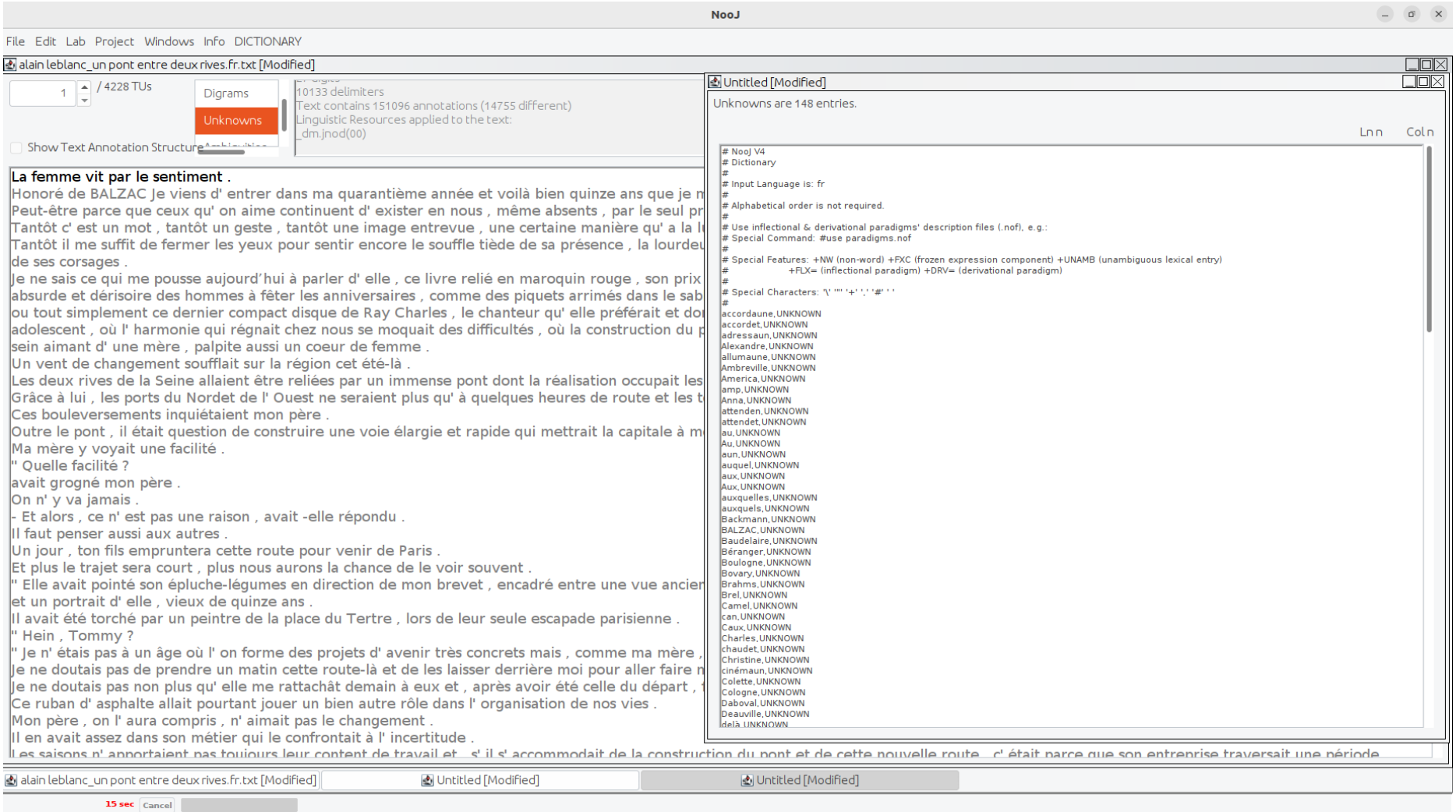


Annexe 15
Extrait du dictionnaire de formes fléchies du français

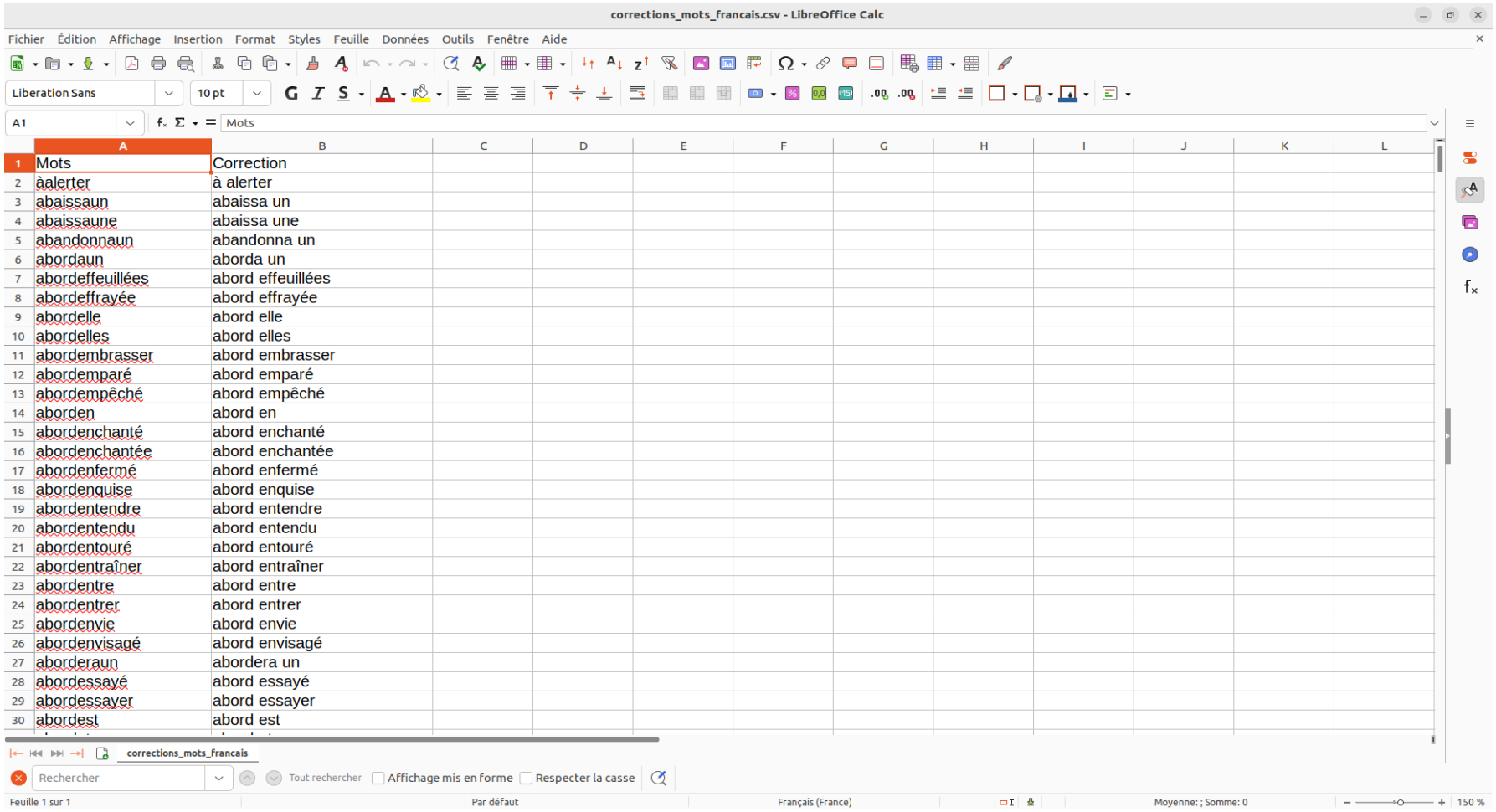


	A	B	C
424064	excèderiez	VERB	excéder
424065	excèderions	VERB	excéder
424066	excèderons	VERB	excéder
424067	excèderont	VERB	excéder
424068	excèdes	VERB	excéder
424069	excès	NOUN	excès
424070	excéda	VERB	excéder
424071	excédai	VERB	excéder
424072	excédaient	VERB	excéder
424073	excédais	VERB	excéder
424074	excédait	VERB	excéder
424075	excédant	ADJ	excédant
424076	excédant	VERB	excéder
424077	excédante	ADJ	excédant
424078	excédantes	ADJ	excédant
424079	excédants	ADJ	excédant
424080	excédas	VERB	excéder
424081	excédasse	VERB	excéder
424082	excédassent	VERB	excéder
424083	excédasses	VERB	excéder
424084	excédassiez	VERB	excéder
424085	excédassions	VERB	excéder
424086	excédens	NOUN	excédent
424087	excédent	NOUN	excédent
424088	excédentaire	ADJ	excédentaire
424089	excédentaires	ADJ	excédentaire
424090	excédents	NOUN	excédent
424091	excéder	VERB	excéder
424092	excédez	VERB	excéder
424093	excédiez	VERB	excéder

Annexe 16
Exemple d'analyse de mots inconnus avec NooJ

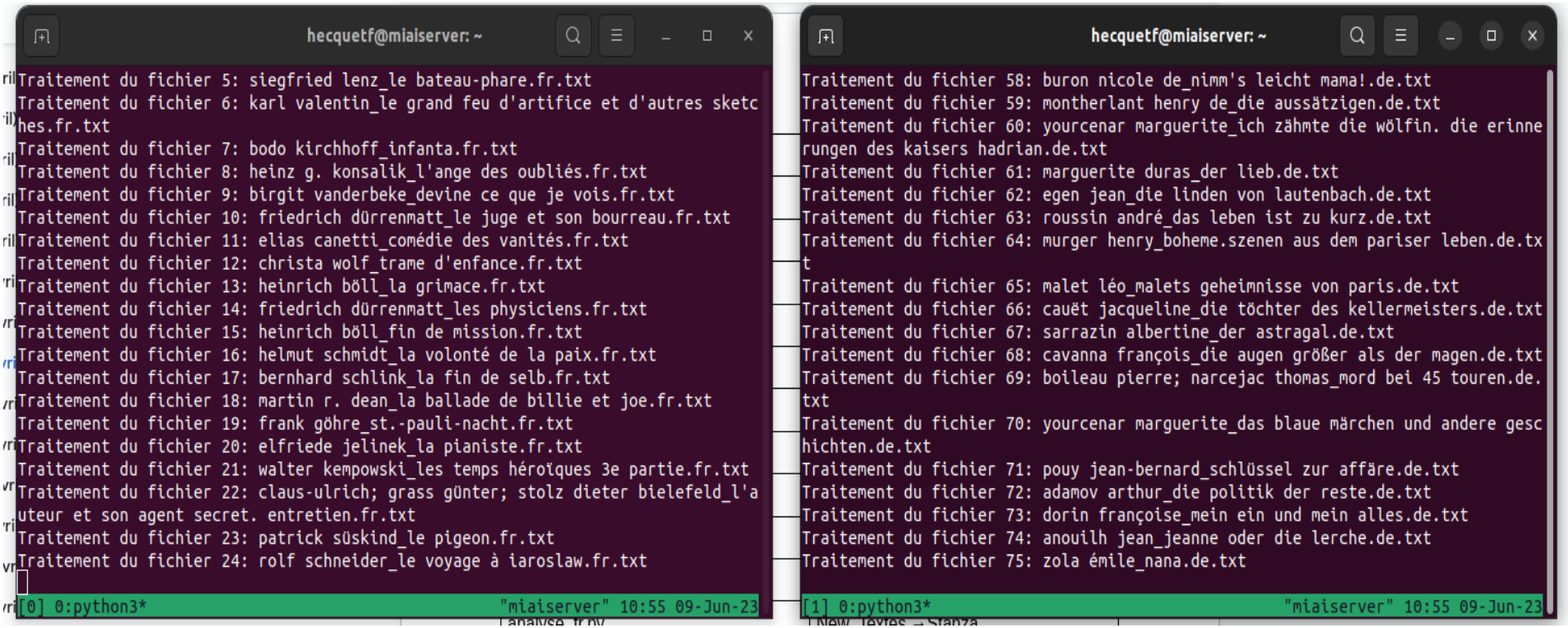


Annexe 17
Extrait du fichier de correction des mots inconnus (OCR)



	A	B
1	Mots	Correction
2	àalerter	à alerter
3	abaissaun	abaissa un
4	abaissaune	abaissa une
5	abandonnaun	abandonna un
6	abordaun	aborda un
7	abordeffeuillées	abord effeuillées
8	abordeffrayée	abord effrayée
9	abordelle	abord elle
10	abordelles	abord elles
11	abordembrasser	abord embrasser
12	abordemparé	abord emparé
13	abordempêché	abord empêché
14	aborden	abord en
15	abordenchanté	abord enchanté
16	abordenchantée	abord enchantée
17	abordenfermé	abord enfermé
18	abordenquise	abord enquise
19	abordentendre	abord entendre
20	abordentendu	abord entendu
21	abordentouré	abord entouré
22	abordentraîner	abord entraîner
23	abordentre	abord entre
24	abordentrer	abord entrer
25	abordenvie	abord envie
26	abordenvisagé	abord envisagé
27	aborderaun	abordera un
28	abordessayé	abord essayé
29	abordessayer	abord essayer
30	abordest	abord est

Annexe 18
Traitement en parallèle de romans français et allemands sur le serveur

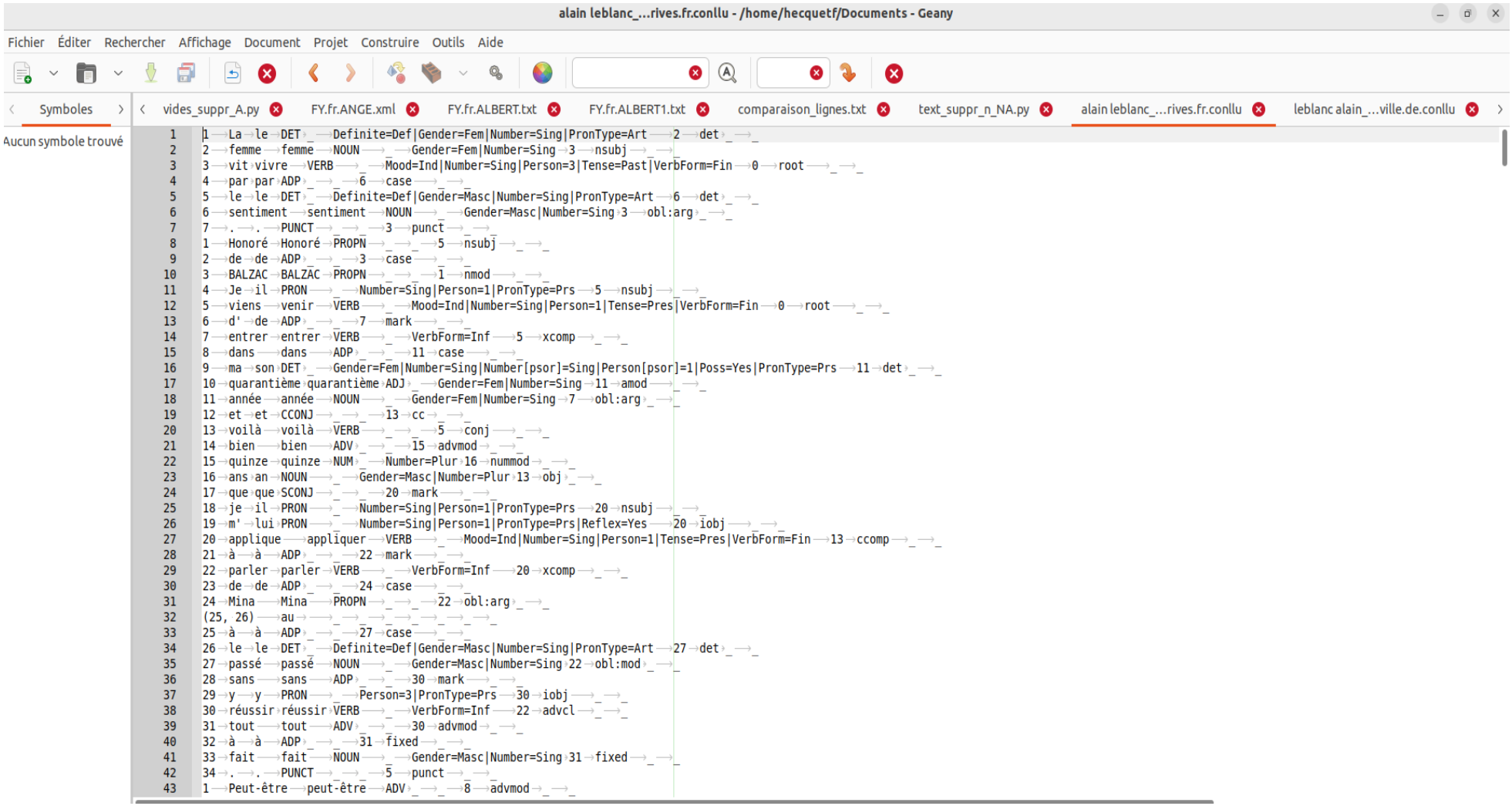


Annexe 19
Entités de 3 lettres en allemand

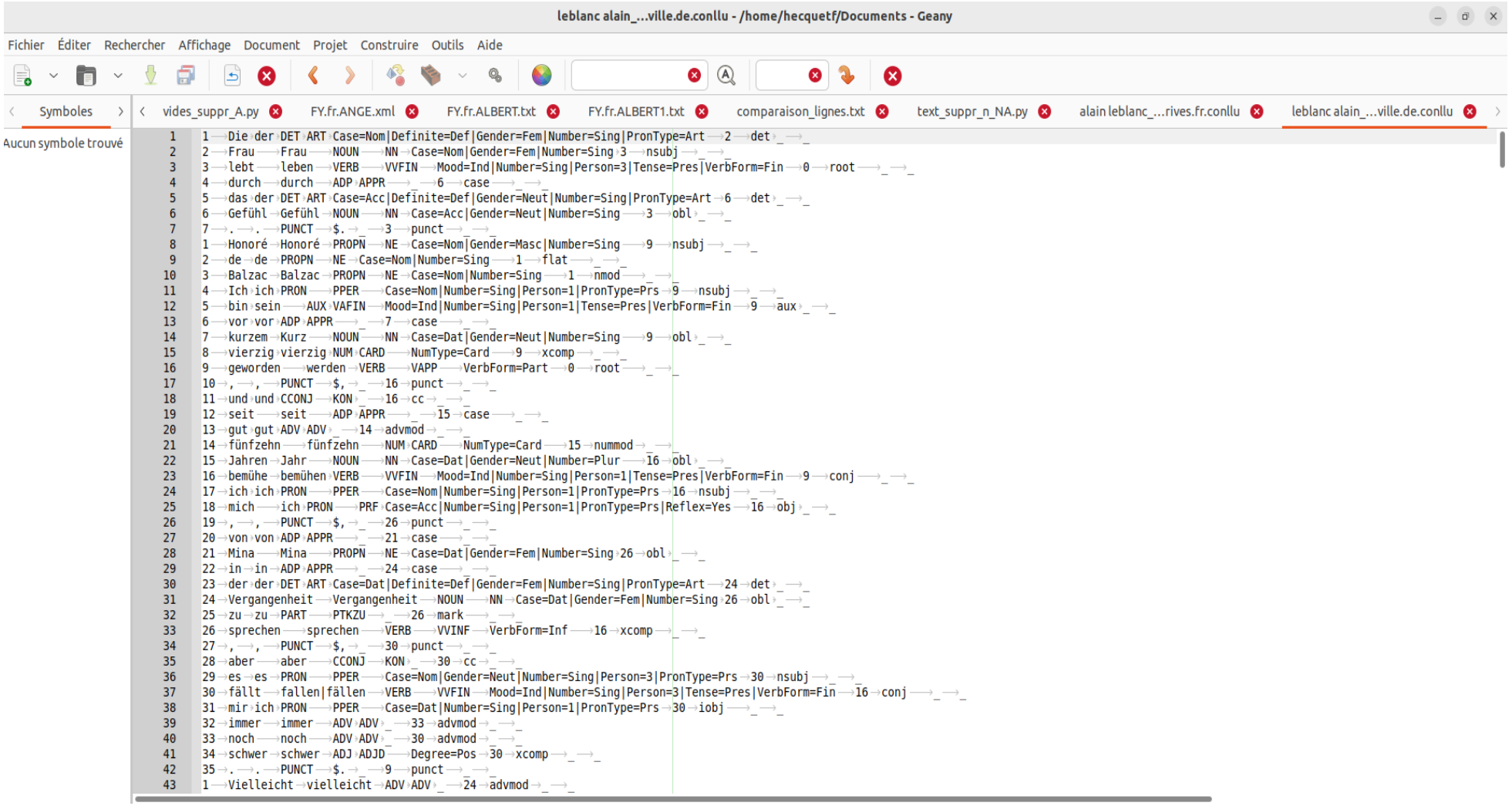
```
I M DSGCHRFCHEN CLAQUEBUE kam eines Tages eine grHOPne Stute zur Welt .
Das GrHOPn war nicht etwa das genugsam bekannte PissegrHOPn , das bei weiáfelligen MTNDhren als eine Begleiterscheinung des Alterns und HinfTNDlligwerdens auftritt , nein , es war ein wohltuend anspr
Als Jules Haudouin das Tierchen an s Licht kommen sah , wollte er seinen eigenen Augen und auch denen seiner Frau nicht trauen .
aDas ist doch nicht mCCHglich" , staunte er .
"Da hTNDtte ich viel zu groáes GlHOPck . ~
Fur Haudouin , Landwirt und RoáhTNDndler , hatte es sich bisher nie gelohnt , daá er ein SchlTNDuling , ein LHOPgenbold und ein knickeriger Raffer gewesen war .
Seine KHOPhe krepierten ihm paarweise , die Schweine gleich halbdutzendweise , und das Korn keimte ihm in den STNDcken .
Mit seinen Kindern erging es ihm auch nicht viel besser ; er hatte ihrer drei , und umso viele zu behalten , hatte er sechse in die Welt setzen mHOPssen .
Aber Kinder verschmerzte Haudouin leichter als sein Vieh .
Er weinte am Beerdigungstag sein Schneuztuch tropfnaá , ging dann nach Hause , drHOPckte es aus und hTNDngte es zum Trocknen an die Schnur .
Das Jahr HOPber hatte er dann so oft und reichlich Gelegenheit , seine Frau zu herzen und zu karessieren , daá er's fertigbrachte , ihr jedesmal wieder eines zu hecken .
Das ist ja beim Kindersegen das Bequeme , und in dieser Hinsicht õ8 konnte Haudouin auch nicht klagen .
Er hatte drei quicklebendige SCCHhne und dazu drei TCCHchter , die drauáen auf dem Friedhof lagen , und das war gerade ungefTNDhr das , was er brauchte .
Eine grHOPne Stute , das war eine unerharte Neuigkeit , etwas , was seit Menschengedenken noch gar nie dagewesen war .
Das Vorkommnis schien bemerkenswert , denn in Claquebue ereignete sich sonst nie etwas .
Man erzTNDhlte sich , Maloret pflege seine TCCHchter um ihre Tungfernschaft zu erleichtern ; aber die Geschichte hatte lTNDngst jegliche Zugkraft verloren , weil sie nTNDmlich schon seit hundert Jahr
Die Maloret hatten es seit jeher mit ihren TCCHchtern so gehalten , und man war nachgerade daran gewCCHhnt .
Von Zeit zu Zeit machten sich die Republikaner , ein halbes Dutzend insgesamt , eine finstere , mondlose Nacht zunutze und zogen vor das Pfarrhaus , wo sie unter den Fenstern des Pfarrers die Carmagnol
johlten .
Davon abgesehen , geschah nie , gar nie irgend etwas Aufregendes .
Und so langweilte man sich eben .
Und weil die Zeit nicht verging , starben auch die alten Leute nicht .
Es lebten achtundzwanzig HundertjTNDhrige in der Kirchgemeinde , nicht eingerechnet die Alten zwischen siebzig und hundert , die rund die HTNDlfte der BevCCHlkerung ausmachten .
Freilich hatte man ein paar von ihnen kreuzlahm oder gar tot geschlagen , doch konnten solcherlei TTNDtlichkeiten und handgreifliche Abhilfe einzig und allein dank privater Initiative zustandekommen ,
Die Neuigkeit wischte aus dem Stall ins Freie , lief im Zickzack zwischen dem Wald und dem Bach hin und her , machte dreimal die Runde um Claquebue und ging dann auf dem Platz vor der BHOPrgermeistere
Augenblicklich rannte alt und jung , das ganze Dorf , Jules Haudouins Haus zu , die einen im Laufschritt oder gar im Galopp , andere wieder hinkend , hoppelnd oder an KrHOPcken humpelnd .
Man lief sich regelrecht die AbsTNDtze ab , weil ein jeder zuerst dort sein wollte , und die alten , klapprigen Mummelgreise waren kein biáchen verstTNDndiger õ9 als die Weibsbilder und mischten ihre
"Es tut sich etwas !
Es tut sich etwas !"
Im Hofe beim RoátTNDuscher erreichte der Tumult seinen HCCHhepunkt , denn die Einwohner von Claquebue hatten bereits wieder , wie in alten Zeiten , etwas gefunden , worHOPber sie sich in die Haare ger
Die wHOPtigsten FrCCHmmler lagen dem Pfarrer eifernd in beiden Ohren , er mHOPsse unverzHOPglich dem grHOPnfarbigen Pferdchen den unreinen Geist austreiben , und die sechs Republikaner , die seine Ge
Und zwar ganz unverhohlen , ohne sich HOPberhaupt Zwang anzutun , schrien sie ihm das einfach mitten ins Gesicht .
Nun ging eine gelinde Keilerei los , der BHOPrgermeister bekam einen Fuátritt in den Hintern , so daá ihm eine Ansprache den Hals hinaufrutschte .
Die jungen Frauen wehklagten , weil man sie zwickte , und die alten jammerten desgleichen , weil sie nicht gekniffen wurden , und die Rotzbuben plTNDrrten und erhoben ein Wehgeheul .
Schuld daran waren die Ohrfeigen und Maulschellen , die sie ausgeteilt bekamen .
Endlich tauchte Jules Haudouin unter der StalltHOPr auf .
QuietschvergnHOPgt und mit blutbesudelten HTNDnden stand er da und bestTNDtigte : "GrHOPn wie ein Apfel ist das Fohlen ! "
Einschallendes GelTNDchter lief durch die Menge , und dann sah man einen ururalten Mann mit beiden Armen umsich schlagen und aui der Stelle mausetot umfallen , inseinem hundertundachten Lebensjahr .
Daraufhinschwoll das GelTNDchter der Zuschauer ins Ungeheuerliche an ; alles hielt sich den Bauch mit beiden HTNDnden und lachte , bis man nicht mehr konnte .
In weniger als einer halben Stunde muáten sieben HundertjTNDhrige drei Neunziger und ein Achtziger dran glauben und das Zeitliche segnen .
õ10 Darunter waren ein paar , die schon lTNDngst nicht mehr so recht wohlauf gewesen waren .
Unter der StalltHOPr stand Haudouin und dachte an seinen hochbetagten Vater , der fHOPr viere fraá , er wandte sich zu seiner Frau und klCCHnte , am meisten seien ja nicht die zu beklagen , die von hi
Der Pfarrer hatte alle HTNDnde voll zu tun , um die Sterbenden mit der letzten TrCCHstung zu versehen .
```

Annexe 20
Reconnaissance des caractères accentués en allemand

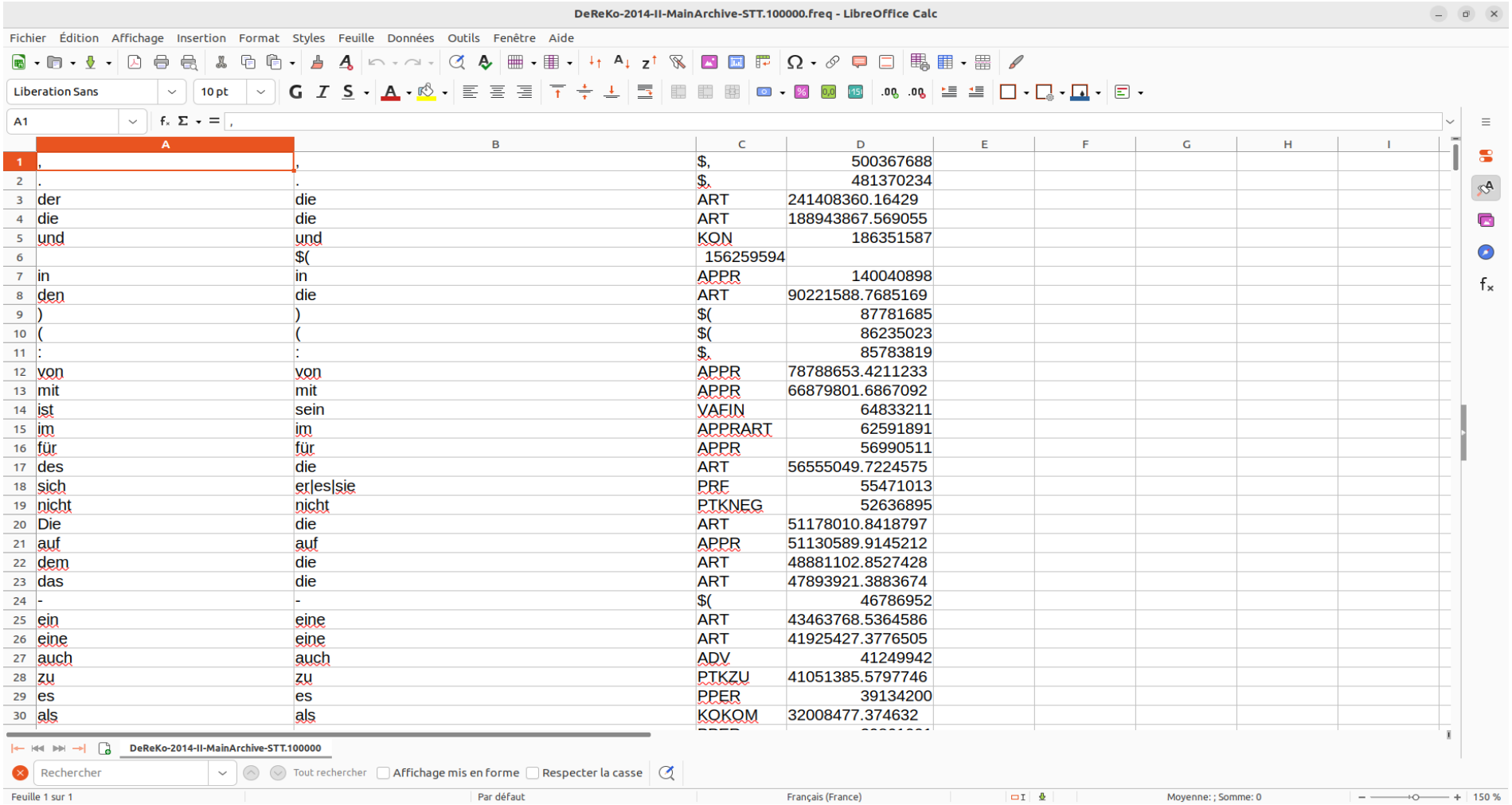
Annexe 21
Sortie de Stanza d'un roman en français au format .conllu



Annexe 22
Sortie de Stanza d'un roman en allemand au format .conllu

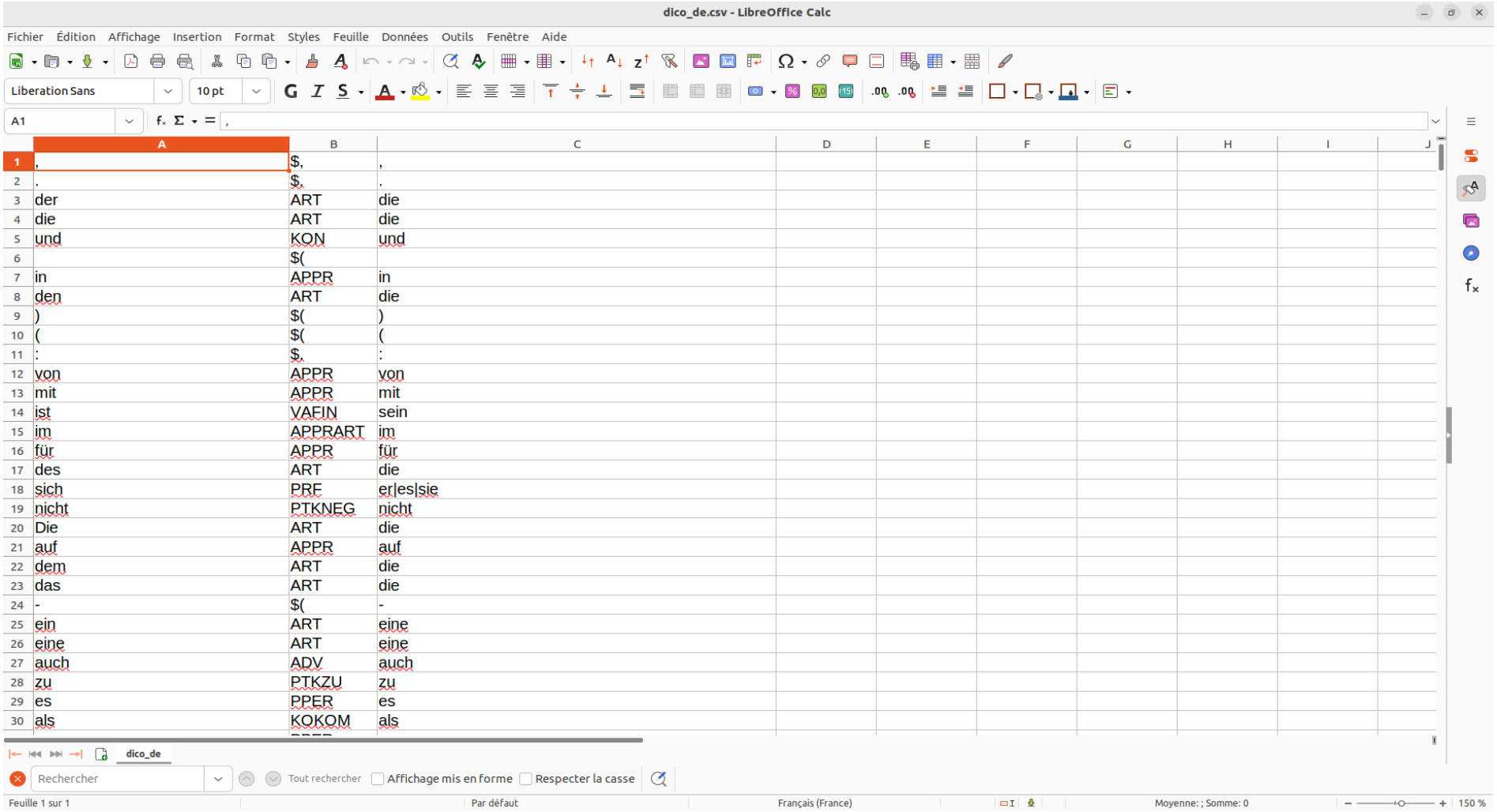


Annexe 23
Version de base du Dereko



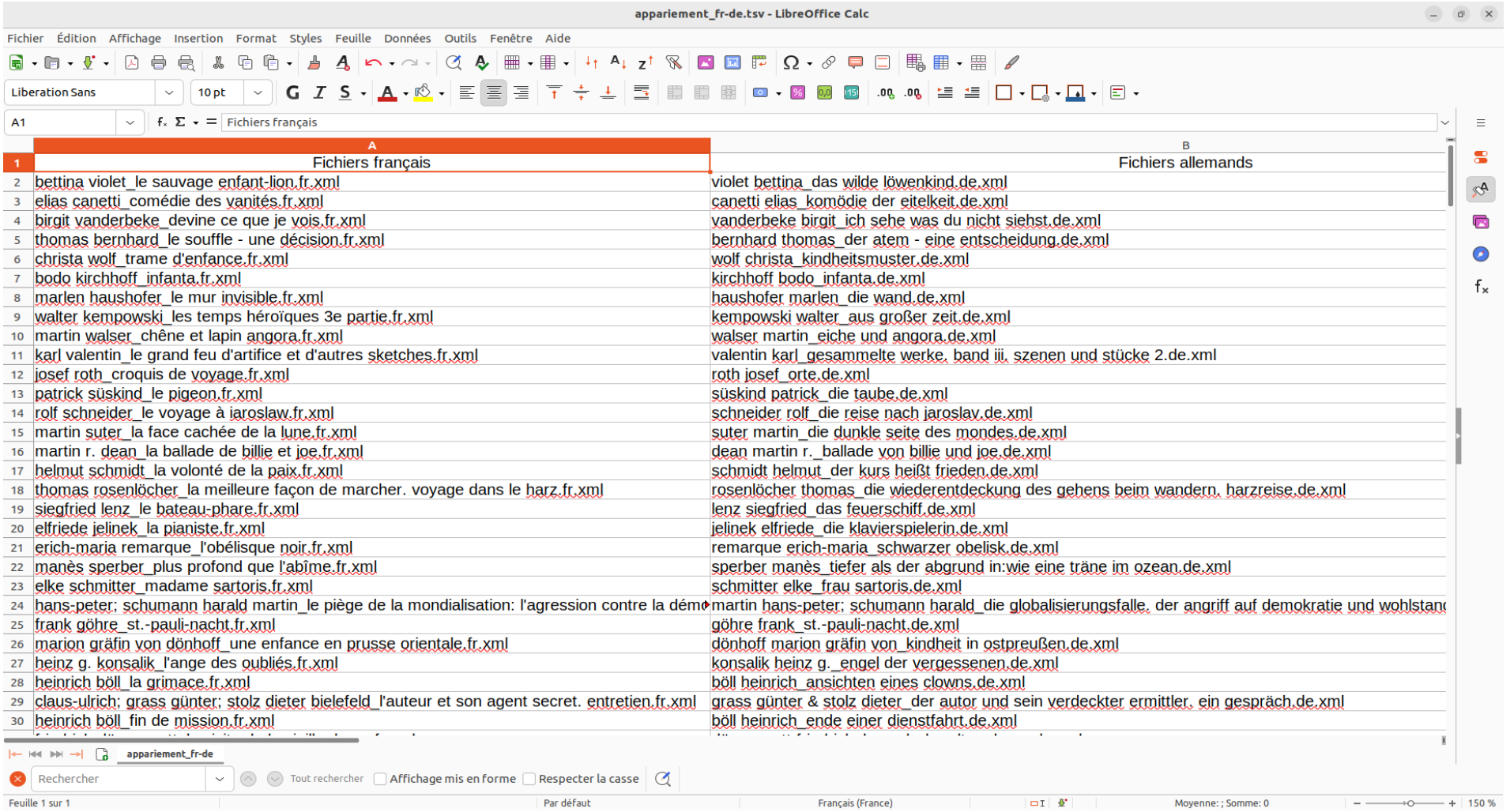
	A	B	C	D	E	F	G	H	I
1	,	,	$,	500367688					
2	.	.	$.	481370234					
3	der	die	ART	241408360.16429					
4	die	die	ART	188943867.569055					
5	und	und	KON	186351587					
6		$(	156259594						
7	in	in	APPR	140040898					
8	den	die	ART	90221588.7685169					
9	)	)	$(	87781685					
10	(	(	$(	86235023					
11	:	:	$.	85783819					
12	von	von	APPR	78788653.4211233					
13	mit	mit	APPR	66879801.6867092					
14	ist	sein	VAFIN	64833211					
15	im	im	APPRART	62591891					
16	für	für	APPR	56990511					
17	des	die	ART	56555049.7224575					
18	sich	er|es|sie	PRF	55471013					
19	nicht	nicht	PTKNEG	52636895					
20	Die	die	ART	51178010.8418797					
21	auf	auf	APPR	51130589.9145212					
22	dem	die	ART	48881102.8527428					
23	das	die	ART	47893921.3883674					
24	-	-	$(	46786952					
25	ein	eine	ART	43463768.5364586					
26	eine	eine	ART	41925427.3776505					
27	auch	auch	ADV	41249942					
28	zu	zu	PTKZU	41051385.5797746					
29	es	es	PPER	39134200					
30	als	als	KOKOM	32008477.374632					

Annexe 24
Extrait du dictionnaire de formes fléchies de l'allemand

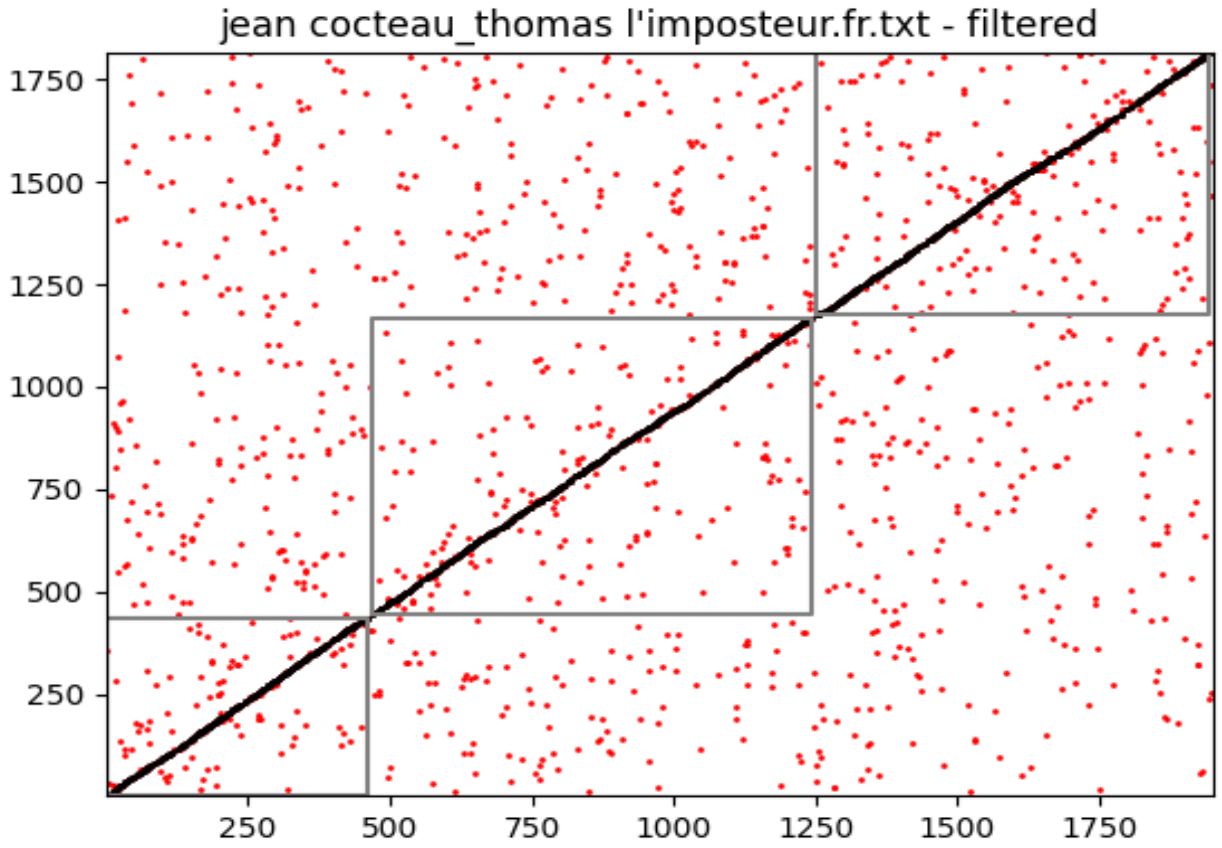


	A	B	C
1	,	$,	,
2	.	$.	.
3	der	ART	die
4	die	ART	die
5	und	KON	und
6		$(	
7	in	APPR	in
8	den	ART	die
9	)	$(	)
10	(	$(	(
11	:	$.	:
12	von	APPR	von
13	mit	APPR	mit
14	ist	VAFIN	sein
15	im	APPRART	im
16	für	APPR	für
17	des	ART	die
18	sich	PRF	er|es|sie
19	nicht	PTKNEG	nicht
20	Die	ART	die
21	auf	APPR	auf
22	dem	ART	die
23	das	ART	die
24	-	$(	-
25	ein	ART	eine
26	eine	ART	eine
27	auch	ADV	auch
28	zu	PTKZU	zu
29	es	PPER	es
30	als	KOKOM	als

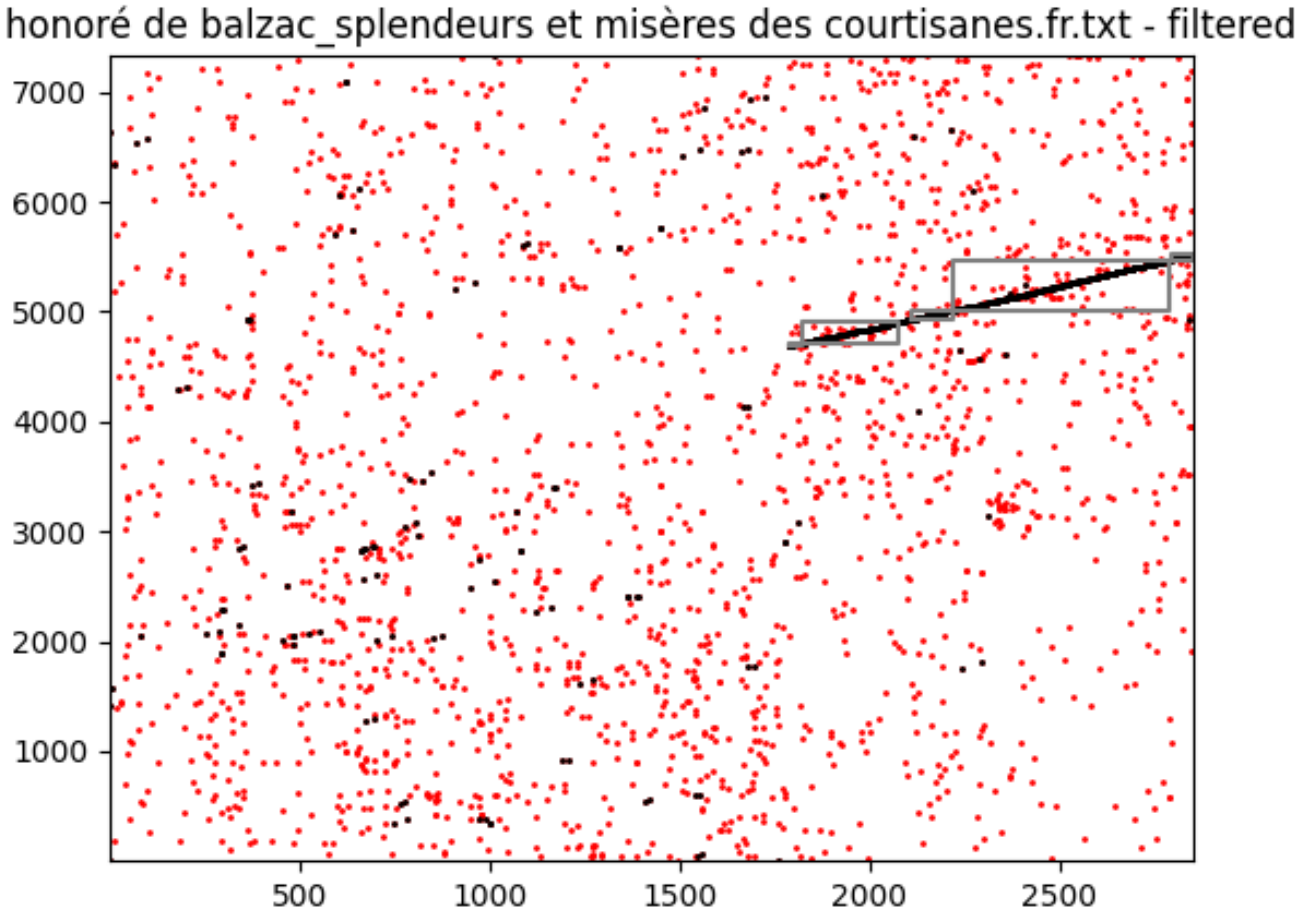
Annexe 25
Extrait de la liste des fichiers appariés pour l'alignement



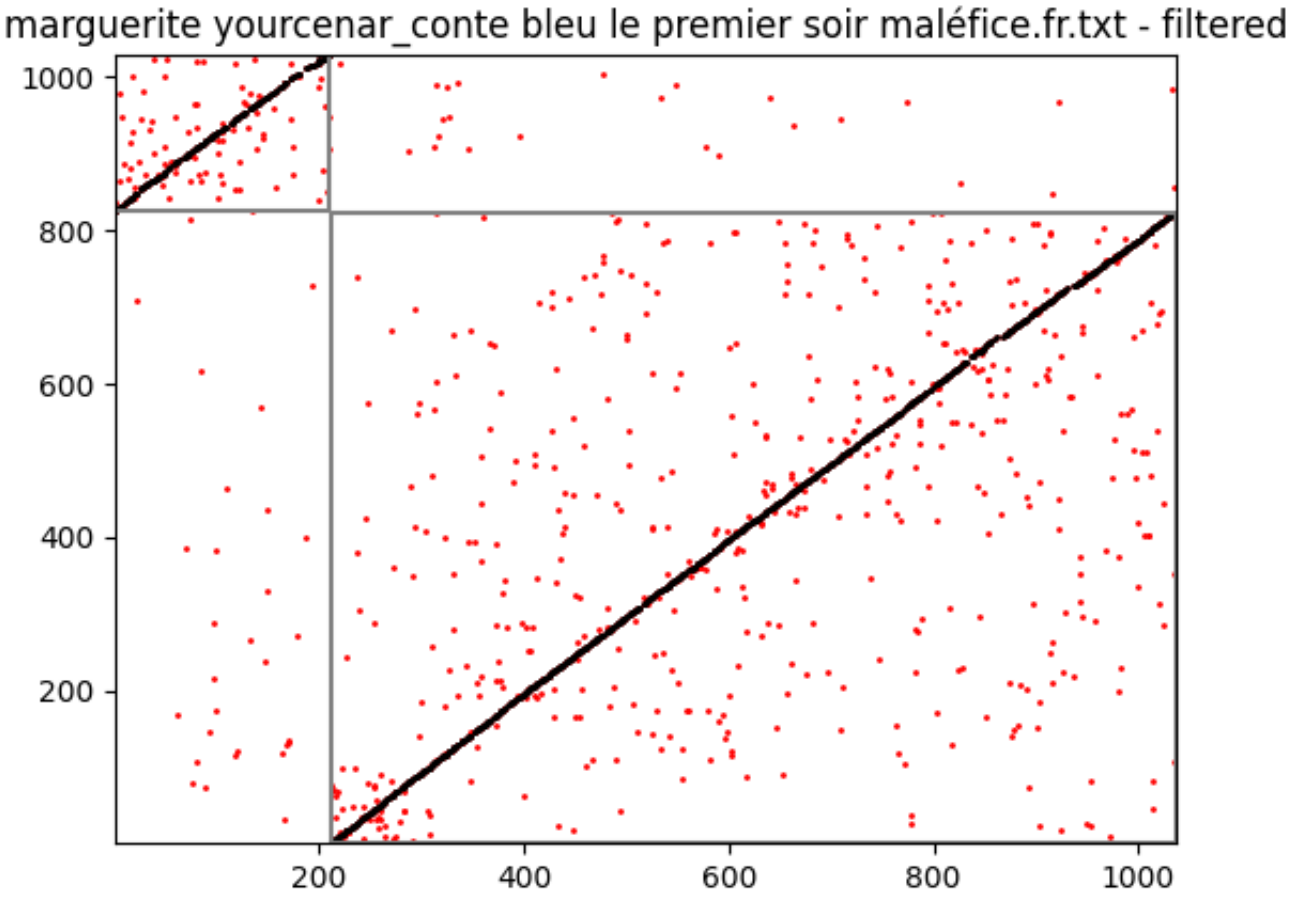
Annexe 26
Exemple d'un nuage de points de textes bien appariés



Annexe 27
Exemple d'un nuage de points de textes mal appariés



Annexe 28
Exemple d'un nuage de points de textes dont la correspondance n'est pas identique



Annexe 29
Exemple de sortie d'une paire de textes alignés au format .tmx

create_dat.py | ailign.py | lire_tsv.py | ailign2.py | tri.py | ailign.py | alain leblanc_...s.fr.anchor.txt | alain leblanc_...s.fr.anchor.ces | alain leblanc_...s.fr.anchor.tmx

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE tmx SYSTEM "tmx14.dtd">
<tmx version="1.4">
  <header
    creationtool="Alignable"
    creationtoolversion="1.0"
    datatype="unknown"
    segtype="sentence"
  >
  </header>
  <body>
<tu>
	<tuv xml:lang="en">
		<seg>&lt;start&gt;</seg>
	</tuv>
	<tuv xml:lang="de">
		<seg>&lt;start&gt;</seg>
	</tuv>
</tu>
<tu>
	<tuv xml:lang="en">
		<seg>La femme vit par le sentiment .</seg>
	</tuv>
	<tuv xml:lang="de">
		<seg>Die Frau lebt durch das Gefühl .</seg>
	</tuv>
</tu>
<tu>
	<tuv xml:lang="en">
		<seg>Peut-être parce que ceux qu' on aime continuent d' exister en nous , même absents , par le seul projecteur de la pensée qui les empêche de s' évanouir dans le néant .</seg>
	</tuv>
	<tuv xml:lang="de">
		<seg>Vielleicht weil Menschen , die wir lieben , auch nach ihrem Tod in uns weiterleben , einzig durch den Projektor unserer Erinnerung davor bewahrt , ins _ Nichts zu verschwinden .</seg>
	</tuv>
</tu>
<tu>
	<tuv xml:lang="en">
		<seg>Tantôt c' est un mot , tantôt un geste , tantôt une image entrevue , une certaine manière qu' a la lumière de frapper les murs à une certaine heure .</seg>
	</tuv>
	<tuv xml:lang="de">
		<seg>Manchmal ist es ein Wort , manchmal eine Geste , manchmal ein flüchtig auftauchendes Bild oder die Art , wie das Licht zu einer bestimmten Tageszeit auf die Wände fällt .</seg>
	</tuv>
```

Annexe 30
Exemple de sortie d'une paire de textes alignés au format .ces

create_dat.py | ailign.py | lire_tsv.py | ailign2.py | tri.py | ailign.py | alain leblanc_...s.fr.anchor.txt | alain leblanc_...s.fr.anchor.ces | alain leblanc_...s.fr.anchor.tmx

```
<?xml version="1.0" encoding="utf-8"?>

<cesAlign type="seg" version="1.6">

<cesHeader version="2.3">
    <translations>
        <translation lang="en" />
        <translation lang="*" />
    </translations>
</cesHeader>

<linkList>
    <linkGrp targType="seg">

        <link xtargets="1 ; 1"/>
        <link xtargets="2 ; 2"/>
        <link xtargets="4 ; 4"/>
        <link xtargets="5 ; 5"/>
        <link xtargets="6 ; 7"/>
        <link xtargets="7 ; 11"/>
        <link xtargets="8 ; 12"/>
        <link xtargets="9 ; 13"/>
        <link xtargets="10 ; 14"/>
        <link xtargets="12 ; 15"/>
        <link xtargets="18 ; 21"/>
        <link xtargets="19 ; 22"/>
        <link xtargets="25 ; 27"/>
        <link xtargets="26 ; 29"/>
        <link xtargets="29 ; 31"/>
        <link xtargets="31 ; 33"/>
        <link xtargets="32 ; 34"/>
        <link xtargets="33 ; 35"/>
        <link xtargets="34 ; 36"/>
        <link xtargets="35 ; 37"/>
        <link xtargets="36 ; 38"/>
        <link xtargets="40 ; 42"/>
        <link xtargets="41 ; 43"/>
        <link xtargets="43 ; 44"/>
        <link xtargets="45 ; 46"/>
        <link xtargets="47 ; 48"/>
        <link xtargets="48 ; 49"/>
        <link xtargets="49 ; 49"/>
        <link xtargets="50 ; 50"/>
```

Annexe 31
Exemple de sortie d'une paire de textes alignés au format .txt

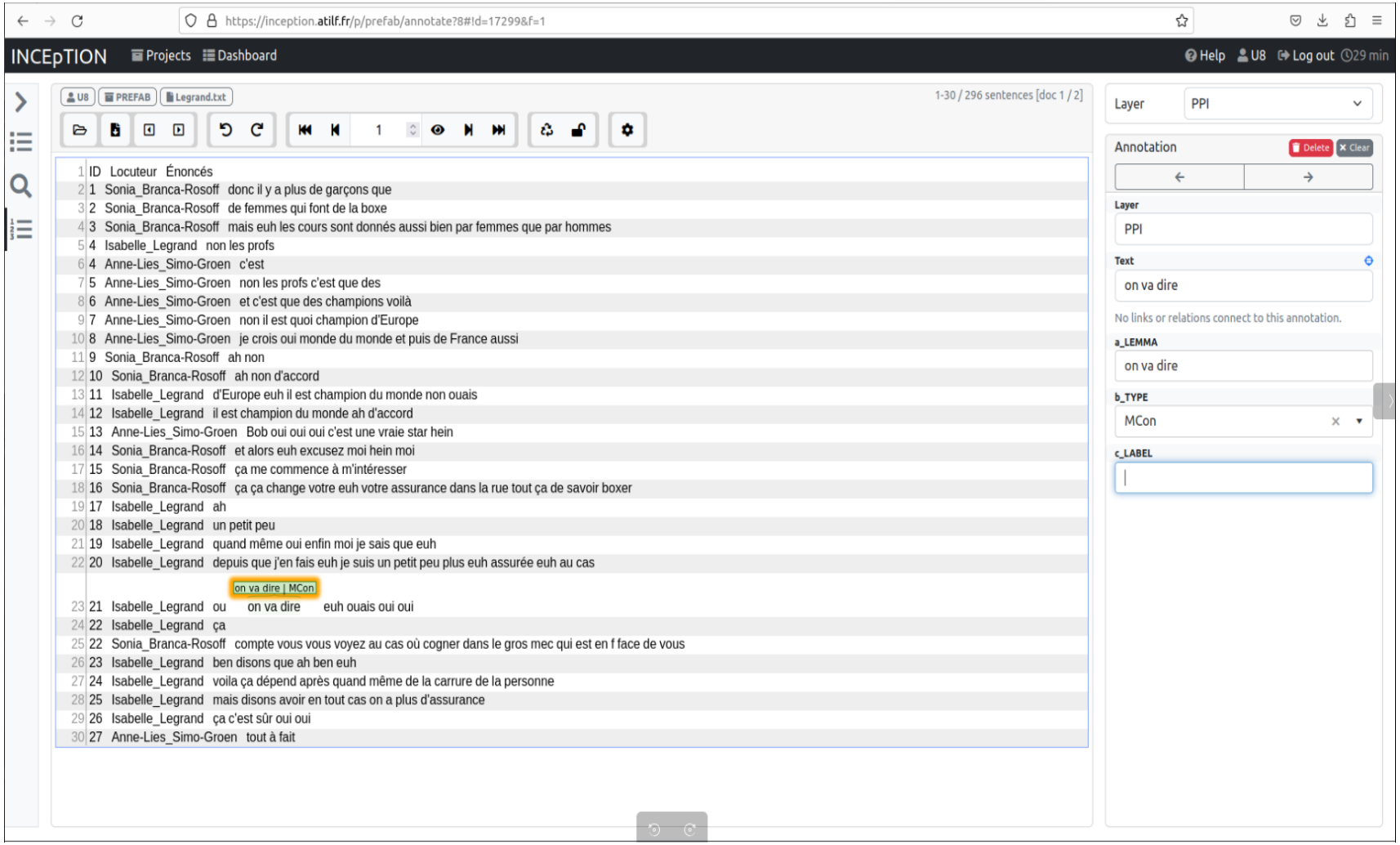
comparaison.py | FY.fr.ALBERT.xml | AIBaBa.py | analyse_fr.py | create_dat.py | ailign.py | lire_tsv.py | ailign2.py | tri.py | ailign.py | alain leblanc_...s.fr.anchor.txt

```
<start>
<start>

La femme vit par le sentiment .
Die Frau lebt durch das Gefühl .

Peut-être parce que ceux qu' on aime continuent d' exister en nous , même absents , par le seul projecteur de la pensée qui les empêche de s' évanouir dans le néant .
Vielleicht weil Menschen , die wir lieben , auch nach ihrem Tod in uns weiterleben , einzig durch den Projektor unserer Erinnerung davor bewahrt , ins _ Nichts zu verschwinden .

Tantôt c' est un mot , tantôt un geste , tantôt une image entrevue , une certaine manière qu' a la lumière de frapper les murs à une certaine heure .
Manchmal ist es ein Wort , manchmal eine Geste , manchmal ein flüchtig auftauchendes Bild oder die Art , wie das Licht zu einer bestimmten Tageszeit auf die Wände fällt .

Tantôt il me suffit de fermer les yeux pour sentir encore le souffle tiède de sa présence , la lourdeur de son bras appuyé sur le mien , les effluves de son eau de Cologne imprégnée dans le coton lége
Ich spüre ihren Atem , den Druck ihres Arms auf meinem Arm , ich rieche das Eau de Cologne , nach dem ihre Sommerkleider dufteten .

Je ne sais ce qui me pousse aujourd'hui à parler d' elle , ce livre relié en maroquin rouge , son prix d' honneur de l' année 40 que je croyais perdu et que j' ai depuis peu retrouvé , cette tradition
Ziemlich absurd , unser Brauch , Gedenktage zu feiern - Pfosten , in den Sand gerammt , um die Jahre festzuhalten , die uns unter den Füßen wegrieseln und verschwinden wie Wanderdünen ... Vielleicht l

Un vent de changement soufflait sur la région cet été -là .
I In jenem Sommer begann in unserer Gegend ein neuer Wind zu wehen .

Les deux rives de la Seine allaient être reliées par un immense pont dont la réalisation occupait les conversations et les couvertures des _ journaux .
Die beiden Seineufer sollten durch eine gewaltige Brücke miteinander verbunden werden , und dieses Projekt beherrschte die Gespräche der Leute und die Titelseiten der Zeitungen .

Grâce à lui , les ports du _ Nord et de l' Ouest ne seraient plus qu' à quelques heures de route et les touristes se risqueraient plus nombreux dans nos contrées .
Die Brücke würde den Weg zu den Hafenstädten im _ Norden und Westen erheblich verkürzen und dem Fremdenverkehr Auftrieb geben .

Outre le pont , il était question de construire une voie élargie et rapide qui mettrait la capitale à moins de deux heures .
Außer der Brücke sollte auch noch eine Autobahn gebaut werden , über die man Paris in weniger als zwei Stunden erreichen konnte .

Il faut penser aussi aux _ autres .
" Man muß auch an die anderen denken .

Un jour , ton fils empruntera cette route pour venir de Paris .
Eines Tages wird unser Junge vielleicht in Paris wohnen , und dann können wir froh sein um eine gute Verbindung .

" Je n' étais pas à un âge où l' on forme des _ projets d' avenir très concrets mais , comme ma mère , je ne me figurais pas rester plus tard à Y .
Ich war noch zu jung , um konkrete Zukunftspläne zu schmieden , aber darin waren Mutter und ich uns einig :

Je ne doutais pas de prendre un matin cette route -là et de les laisser derrière moi pour aller faire mon chemin ailleurs .
Eines Tages würde ich meinen Eltern Lebewohl sagen und auf dieser neuen Straße davonfahren , um anderswo mein Glück zu machen , davon war ich fest überzeugt .
```

Annexe 32
Extrait du fichier texte contenant les scores du DTW

```
bettina violet : 426.8884663820258
elias canetti : 4171.987653478939
birgit vanderbeke : 330.10268402695624
thomas bernhard : inf
christa wolf : inf
bodo kirchhoff : 4806.312201255436
marlen haushofer : 1887.3206707596885
martin walser : 2348.501743578911
josef roth : inf
patrick süskind : 376.46297239065115
rolf schneider : inf
martin suter : 2514.7330885529495
helmut schmidt : inf
thomas rosenlöcher : 783.9557787828155
siegfried lenz : 1512.4979704082077
elfriede jelinek : 3100.955324321965
erich maria remarque : inf
manès sperber : inf
elke schmitter : 892.6014014184511
hans peter : inf
frank göhre : inf
heinz g konsalik : inf
heinrich böll la grimace : inf
claus ulrich : inf
heinrich böll fin de mission : inf
friedrich dürrenmatt la visite de la vieille : inf
friedrich dürrenmatt romulus : 2539.0035838283434
friedrich dürrenmatt les physiciens : inf
friedrich dürrenmatt le juge et son bourreau : 1592.659488296512
arthur adamov : 814.2800907909921
albert camus la peste : inf
albert camus l'étranger : inf
```

Annexe 33
Interface d'Inception



Annexe 34
Mauvais découpage de phrase dans un fichier XML

```
373374                <t id="t173587" n="79650" num="1" l="mais" c="COORD" f="Coord COORD" e=" " type="dd">Mais</t>
373375                <t id="t173588" n="79651" num="2" l="ça" c="PRON" f="Masc SG Dem Pro PRON" e=" " type="dd">ça</t>
373376                <t id="t173589" n="79652" num="3" l="être" c="VERB" f="en pourSN aSVINF il deSN queP avoir aSN SADJdeSVINF SADJ contreSN locSN deSVINF SN Cond SG P3 Verb VAUX_P3SG"
373377                <t id="t173590" n="79653" num="4" l="quand même" c="ADV" f="Adv ADV" e=" " type="dd">quand même</t>
373378                <t id="t173591" n="79654" num="5" l="dommage" c="NOUN" f="deSN Masc SG Noun NOUN_SG" e=" " type="dd">dommage</t>
373379                <t id="t173592" n="79655" num="6" l="de" c="PREP" f="Prep PREP_DE" e=" " type="dd">de</t>
373380                <t id="t173593" n="79656" num="7" l="donner" c="VERB" f="dansSN avoir SN aSVINF pourSN se aSN surSN Inf Verb VERB_INF" e=" " type="dd">donner</t>
373381                <t id="t173594" n="79657" num="8" l="à" c="PREP" f="Prep PREP_A" e=" " type="dd">à</t>
373382                <t id="t173595" n="79658" num="9" l="un" c="DET" f="Fem SG Indef Det DET_SG" e=" " type="dd">une</t>
373383                <t id="t173596" n="79659" num="10" l="si" c="CONJ" f="SConj CONN" e=" " type="dd">si</t>
373384                <t id="t173597" n="79660" num="11" l="rare" c="ADJ" f="IMPERSO queP deSVINF InvGen SG Adj ADJ_SG" e=" " type="dd">rare</t>
373385                <t id="t173598" n="79661" num="12" l="aventurer" c="VERB" f="se aSVINF SN avoir IndP SG P3 Verb VERB_P3SG" e=" " type="dd">aventure</t>
373386                <t id="t173599" n="79662" num="13" l="d" c="NOUN" f="InvGen InvPL Noun NOUN_INV" e="\n      " type="dd">d</t>
373387            </tc>
373388            <dc>
373389                <d rel="SUBJ" h="3" d="2"/>
373390                <d rel="OBJ_SPRED" h="3" d="5"/>
373391                <d rel="VMOD_POSIT1" h="3" d="4"/>
373392                <d rel="VMOD" h="3" d="7"/>
373393                <d rel="VMOD_POSIT1_SUBORD" h="7" d="12"/>
373394                <d rel="VMOD_POSIT1" h="7" d="9"/>
373395                <d rel="PREPOBJ" h="7" d="6"/>
373396                <d rel="PREPOBJ" h="9" d="8"/>
373397                <d rel="SUBJ" h="12" d="13"/>
373398                <d rel="CONNECT" h="12" d="10"/>
373399            </dc>
373400        </s>
373401    </p>
373402        <p>    <s id="s10956">
373403            <tc>
373404                <t id="t173600" n="79663" num="0" l="e" c="NOUN" f="CR1 InvGen InvPL Noun NOUN_INV" e=" ">e</t>
373405                <t id="t173601" n="79664" num="1" l="le" c="DET" f="InvGen SG Def Det DET_SG" e="">l'</t>
373406                <t id="t173602" n="79665" num="2" l="âme" c="NOUN" f="Fem SG Noun NOUN_SG" e=" ">âme</t>
373407                <t id="t173603" n="79666" num="3" l="un" c="DET" f="Masc SG Indef Det DET_SG" e=" ">un</t>
373408                <t id="t173604" n="79667" num="4" l="nom" c="NOUN" f="Masc SG Noun NOUN_SG" e=" ">nom</t>
373409                <t id="t173605" n="79668" num="5" l="de" c="PREP" f="Prep PREP_DE" e=" ">de</t>
373410                <t id="t173606" n="79669" num="6" l="maladie" c="NOUN" f="Fem SG Noun NOUN_SG" e="">maladie</t>
373411                <t id="t173607" n="79670" num="7" l="." c="SENT" f="SENT" e=" ">.</t>
373412            </tc>
373413            <dc>
```

Annexe 35
Mauvais découpage de phrase dans un fichier texte

```
10954   » Elle balaie d'un seul oeil mon visage étonné avant d'ajouter :
10955   « Mais ça serait quand même dommage de donner à une si rare aventure d
10956   e l'âme un nom de maladie.
```

Annexe 36
Tableau d'organisation des fichiers annotés du corpus Phraseobase

Ordre des fichiers Python	Entrée → Sortie
conversion_A.py	Origine → Textes
text_suppr_n_A.py	Textes → Textes_Modifiés
analyse_A.py	Textes_Modifiés → Stanza
stanza_modif_A.py	Stanza → Stanza_Modifiés
mep_A.py	Stanza_Modifiés → Conllu
conllu_verif.py	Conllu → Conllu_Modifiés
ajout_type_dd_A.py	Origine → Origine_Modifiés
vides_suppr_A.py	Origine_Modifiés → Ready
suppr_xml_A.py	Ready → Final
conllu_to_xml_A.py	Final → Ultime

Annexe 37
Tableau d'organisation des fichiers non annotés du corpus Phraseobase

Ordre des fichiers Python	Entrée → Sortie
Regle_1.py	Origine → DD_annotation
Regle_2.1.py	DD_annotation → DD_suite
Regle_2.2.py	DD_suite → Suppr_crochet
Regle_3.1.py	Suppr_crochet → Suppr_pronom
Regle_3.2.py	Suppr_pronom → Suppr_verbe
conversion_NA.py	Suppr_verbe → Textes
text_suppr_n_NA.py	Textes → Textes_Modifiés
analyse_NA.py	Textes_Modifiés → Stanza
stanza_modif_NA.py	Stanza → Stanza_Modifiés
mep_NA.py	Stanza_Modifiés → Conllu
conllu_verif.py	Conllu → Conllu_Modifiés
ajout_type_dd_NA.py	Suppr_verbe → Origine_Modifiés
vides_suppr_NA.py	Origine_Modifiés → Ready
suppr_xml_NA.py	Ready → Final
conllu_to_xml_NA.py	Final → Ultime

Annexe 38
Tableau d'organisation des fichiers du corpus GLFA

Ordre des fichiers Python	Entrée → Sortie
Regle_1.py	Origine → Annotation
Regle_2.1.py	Annotation → Annotation_suite
Regle_2.1_de.py	Annotation → Annotation_suite
Regle_2.2.py	Annotation_suite → Suppr_crochets
Regle_3.1.py	Suppr_crochets → Suppr_pronoms
Regle_3.2.py	Suppr_pronoms → Suppr_verbes
conversion.py	Suppr_verbes → Textes
text_suppr_n.py	Textes → Textes_Modifiés
correction_OCR_fr.py	Textes_Modifiés → New_Textes
titre_suppr_fr.py	New_Textes → New_Textes
text_modif_de.py	Textes → Textes_Modifiés
titre_suppr_de.py	Textes_Modifiés → Textes_Modifiés
analyse_fr.py	New_Textes → Stanza
analyse_de.py	Textes_Modifiés → Stanza
stanza_modif.py	Stanza → Stanza_Modifiés
mep.py	Stanza_Modifiés → Conllu
conllu_verif_fr.py	Conllu → Conllu_Modifiés
conllu_verif_de.py	Conllu → Conllu_Modifiés
ajout_type_dd.py	Suppr_verbes → Origine_Modifiés
vides_suppr.py	Origine_Modifiés → Ready
suppr_xml.py	Ready → Final
conllu_to_xml.py	Final → Ultime

Sommaire

MOTS-CLÉS : phrases préfabriquées, corpus parallèle, traduction, analyse de données, calcul de similarité, traitement automatique des langues

RÉSUMÉ

Ce mémoire présente une partie des actions indispensables dans le cadre du projet PREFAB, basé sur l'analyse de corpus bilingues. Il met en lumière le fonctionnement de différents outils existants pour réaliser des analyses puissantes, ainsi que les limites rencontrées et les corrections nécessaires. Il souligne également la fusion prometteuse entre la linguistique et le traitement automatique des langues pour modéliser un phénomène linguistique peu documenté. Le mémoire offre une description globale du projet et une présentation de la suite des opérations en cours. Parmi les nombreuses réalisations, le repérage, l'annotation, l'analyse et l'inventaire des phrases préfabriquées du français ont été initiés avec succès. Le projet PREFAB, entamé en octobre 2022 pour une durée de 48 mois, ouvre de nouvelles perspectives pour la compréhension des phrases préfabriquées et suscite un fort intérêt pour l'avenir de la recherche en linguistique et traitement automatique des langues.

KEYWORDS : prefabricated sentences, parallel corpus, translation, data analysis, similarity calculation, natural language processing

ABSTRACT

This dissertation presents some of the actions required as part of the PREFAB project, based on the analysis of bilingual corpora. It highlights the workings of various existing tools for performing powerful analyses, as well as the limitations encountered, and corrections needed. It also highlights the promising fusion between linguistics and natural language processing to model a little-documented linguistic phenomenon. The dissertation offers an overall description of the project and a presentation of the work in progress. Among the many achievements, the identification, annotation, analysis, and inventory of French prefabricated sentences have been successfully initiated. The PREFAB project, which began in October 2022 and will run for 48 months, is opening new perspectives for the understanding of prefabricated sentences and is of great interest for future research in linguistics and natural language processing.