StaySmart & RealReview

Optimisation intelligente du choix de logements

via le NLP et les modèles LLMs

Réalisé par :

Mohamed Elyes Maalel

elyes.maalel@dauphine.tn

Donia Tekaya

donia.tekaya@dauphine.tn

Professeure : Mme Anna Papa

Lien Github: StaySmart-RealReview-Assistant

Année académique : 2024–2025

"Final Feedback, Real Description, Better Choice"

Table des matières

Liste des tableaux

Table des figures

Introduction Générale

Les plateformes de location de courte durée, telles que *Airbnb*,*VRBO* ou *Booking.com*, jouent aujourd'hui un rôle central dans l'organisation de séjours touristiques ou professionnels. La description des logements proposée par les hôtes constitue souvent le premier point de contact entre l'offre et les voyageurs. Elle influence directement la perception du bien, le taux de réservation et la satisfaction des clients. Cependant, ces descriptions sont fréquemment idéalisées, voire partiales, cherchant à valoriser au maximum le logement sans toujours refléter fidèlement l'expérience réelle des visiteurs. De plus, elles peuvent être biaisées ou tronquées, ne mettant pas en avant certains aspects essentiels du logement.

Ce projet vise à résoudre ce déséquilibre en exploitant le potentiel du traitement automatique du langage naturel (NLP), et plus particulièrement les modèles avancés de type LLM (Large Language Models) et Transformers. Il a pour objectif de générer automatiquement une description plus réaliste et représentative d'un logement en s'appuyant sur les commentaires laissés par les clients. Il s'agit de croiser la description initiale fournie par le propriétaire avec les retours d'expérience des utilisateurs afin de construire une description enrichie, nuancée et alignée avec la réalité du séjour.

L'enjeu majeur est d'optimiser les choix de logements pour les voyageurs en leur fournissant une information plus fiable et objective. Pour ce faire, nous avons développé une application interactive avec Streamlit, articulée autour de deux axes majeurs :

L'analyse d'un Corpus scraper contenant les descriptions et les reviews de logements.

Le scraping d'une annonce VRBO en temps réel, permettant d'appliquer le même processus sur une page précise.

Notre approche permet non seulement de générer une nouvelle description affinée et plus fidèle à la réalité, mais aussi d'offrir une fonctionnalité clé supplémentaire : un feedback pertinent basé sur l'analyse des avis clients. Ces deux features générées et fournies par notre assistant renforcent la transparence et la fiabilité de l'information pour les futurs voyageurs, facilitant ainsi leur processus de prise de décision et améliorant leur expérience utilisateur.

Pipeline de projet :

Le pipeline commence par un scraping des données pour collecter les descriptions et les commentaires des logements. Ensuite, un prétraitement et un nettoyage des données sont effectués afin d'obtenir un corpus structuré, Dans la phase suivante, la détection de langue est réalisée et suivie d'une traduction pour uniformiser les données. Ensuite, DistillBERT est utilisé pour associer les commentaires les plus pertinents pour que le modèle T5 génère un résumé des commentaires sous forme de feedback final. Enfin, les LLM produisent une nouvelle description optimisée et plus réaliste du logement.

L'objectif final est de générer des descriptions améliorées et des Feedback , pour un meilleur choix pour les utuilsateur .

la figure ci-dessous présente le pipeline du projet :

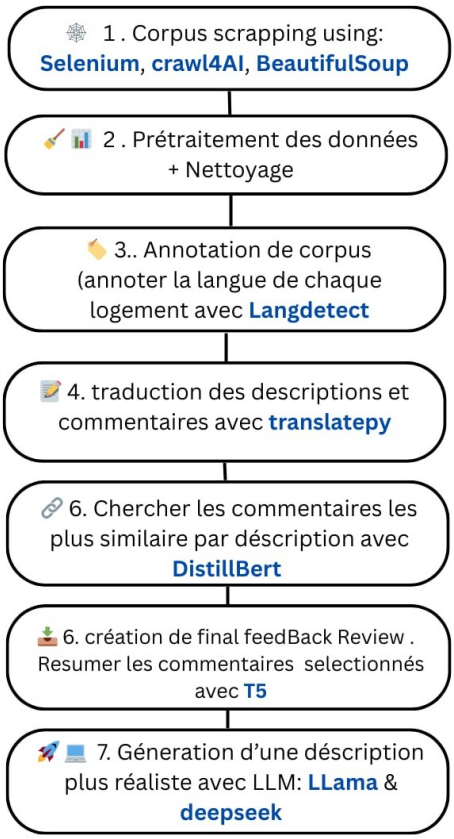

FIGURE1 – Pipeline du projet

CHAPITRE 1

Corpus Scraping et Nettoyage des données

Dans ce chapitre, nous allons nous concentrer sur les deux premières étapes : **Corpus et Nettoyage.**

1.1 Implementattion de corpus

Dans le cadre du développement de notre chatbot **StaySmart and Real Review Assistant**, la construction d'un corpus constitue une **étape cruciale** pour atteindre notre objectif.

Des plateformes comme VRBO et Airbnb disposent d'ensembles de données volumineux, regroupant des millions de publications, de descriptions de logements et de commentaires de visiteurs. Il est essentiel d'exploiter ces commentaires d'une manière innovante, différente de l'approche classique, afin d'obtenir un **feedback global** qui synthétise l'ensemble des avis et génère une **nouvelle description optimisée** du logement.

Cette approche permettra d'**optimiser et de faciliter le choix des logements** en exploitant des technologies avancées telles que le **NLP (Natural Language Processing)**, le **web scraping** et les modèles **LLM (Large Language Models)**.

Pour conclure, VRBO et Airbnb seront notre support fiable pour la collecte et l'exploitation de toutes les informations disponibles.

1.2 Crawling et Scraping :

crawling : désigne l'action de parcourir automatiquement un site web à l'aide d'un programme dans le but de repérer, visiter et collecter des liens vers des pages d'intérêt. Il s'agit de simuler une navigation humaine de manière automatisée.

scraping :, quant à lui, est l'étape suivante : elle consiste à extraire les informations textuelles ou visuelles du code HTML d'une page web. Le scraping permet donc d'aspirer des données structurées (textes, dates, commentaires, etc.) à partir d'éléments spécifiques du DOM d'une page.

1.2.1 Technologies utilisées

Nous avons adopté deux approches distinctes pour effectuer le crawling et le scraping sur les sites www.vrbo.com et https ://www.airbnb.fr :

Première approche : Selenium et BeautifulSoup

— **Selenium** : permet d'automatiser un navigateur web (ici Firefox) afin de simuler les clics, le scroll et la navigation humaine.

— **BeautifulSoup** : sert à parser et extraire facilement les contenus HTML de la page une fois chargée.

Deuxième approche : Crawl4AI

— **Crawl4AI** : bibliothèque Python open-source optimisée pour la collecte automatisée de données. Conçue pour les modèles de langage de grande taille (LLM), les agents d'IA et les pipelines de données, elle permet d'extraire rapidement et efficacement des informations à partir de sites web.

L'automatisation du navigateur Firefox est rendue possible grâce à GeckoDriver, qui assure la compatibilité avec Selenium.

1.3 Étapes du crawling sur VRBO

Pour collecter les données, nous avons ciblé la page de recherche suivante sur VRBO, correspondant à la région de Paris et ses environs :

`https://www.vrbo.com/search?destination=Paris...`

Le script de crawling a été conçu pour exécuter les étapes suivantes :

1. Initialisation d'un navigateur Firefox via Selenium.
2. Attente du chargement complet de la page à l'aide de `WebDriverWait`.
3. Identification des éléments à partir du CSS `a[data-stid='open-hotel-information']`.
4. Extraction de la liste des URL correspondant aux logements disponibles.

Les liens ainsi obtenus sont ensuite transmis à l'étape de scraping pour collecter les informations détaillées des logements.

1.4 Scraping des données

1.4.1 Extraction des descriptions

Une fois sur la page de détail d'un logement, nous souhaitons extraire la description principale du bien, qui présente des informations essentielles telles que les caractéristiques de l'hébergement, ses équipements etc.. :

FIGURE1.1 – Exemple Summary

Identification de la balise contenant la description : Nous recherchons la balise qui contient la description principale du logement.

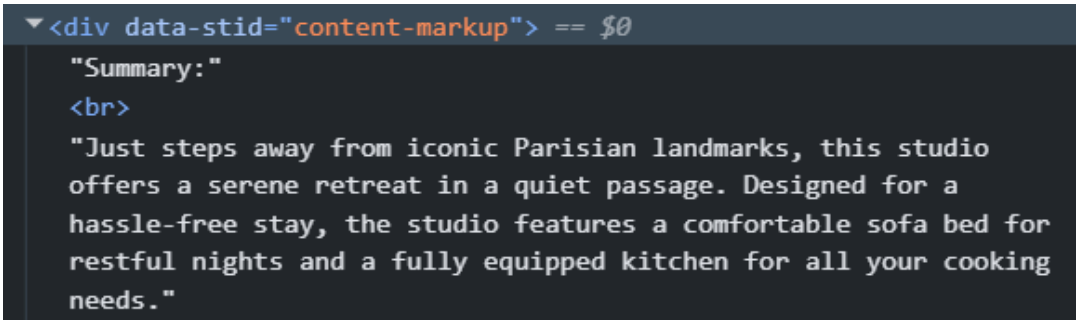

FIGURE1.2 – Summary balise

Le contenu de cette balise est ensuite nettoyé pour constituer la description du logement.

1.4.2 Extraction des commentaires

Le scraping des commentaires nécessite une interaction dynamique avec la page. Cela sera réalisé à l'aide de Selenium :

1. Clic automatique sur le bouton *"See all reviews"*

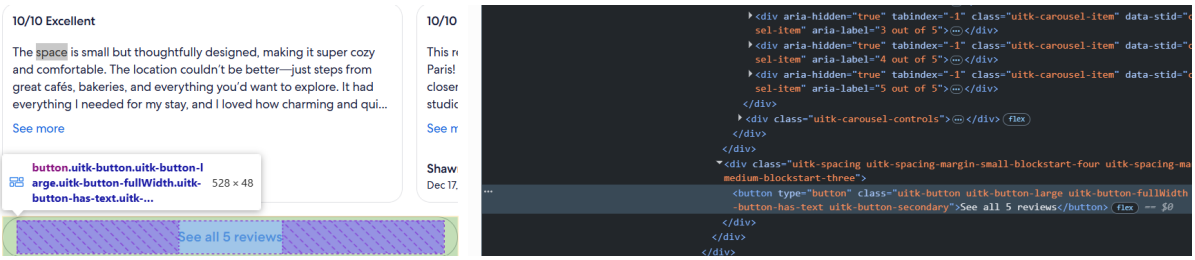

FIGURE1.3 – Boutton balise

2. Scroll progressif pour charger les commentaires supplémentaires
3. Clics répétés sur les boutons *"See more"* pour afficher les avis entiers
4. Extraction des balises suivantes :

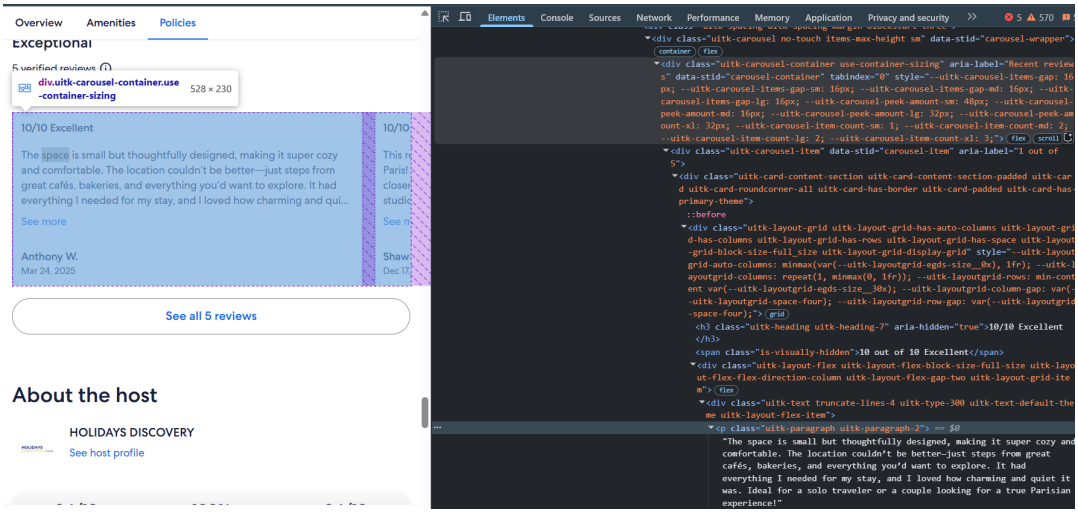

FIGURE1.4 – Summary balise

Chaque avis client est ainsi collecté et associé à la description du logement.

1.5 Format Final Brut de la Base de Données

Les données collectées sont organisées sous forme tabulaire dans un fichier Excel, qui contient les informations suivantes :

(10000, 3)

	listing_id	description	comments
0	3109	Lovely Appartment with one bedroom with a Quee...	Tout s'est bien déroulé. Merci bien. PG
1	3109	Lovely Appartment with one bedroom with a Quee...	Un petit nid fouiller douillet situé dans app...
2	3109	Lovely Appartment with one bedroom with a Quee...	Appartement spacieux, propre,clair, et calme à...
3	3109	Lovely Appartment with one bedroom with a Quee...	Appartement totalement rénové, en parfait état...
4	5396	NEW SOFA-BED SINCE JUNE 2023, Please disregard...	Perfect location!! Nasrine was a delight and m...
...	...	...	...
9995	568394	- In the heart of one of the oldest district o...	We loved staying in Alain's apartment in Paris...
9996	568394	- In the heart of one of the oldest district o...	The apartment was exactly as described and was...
9997	568394	- In the heart of one of the oldest district o...	Alain was a wonderful host and we were in awe ...
9998	568394	- In the heart of one of the oldest district o...	La estancia en el apartamento de Alain fue per...
9999	568394	- In the heart of one of the oldest district o...	Perfect location! Cozy and chic apartment! A+

10000 rows × 3 columns

FIGURE1.5 – Exemple de structure de la base de données brute

Les colonnes du fichier comprennent :

— `url` : L'URL du logement.

— `description` : La description rédigée par l'hôte du logement.

— `comment` : Un commentaire d'un client concernant le logement.

Chaque ligne du fichier représente donc une paire `(description, commentaire)`. Cette structure permet de traiter chaque combinaison de manière individuelle dans le cadre d'un processus de traitement NLP (Natural Language Processing), facilitant ainsi l'analyse et l'interprétation des données.

1.5.1 Difficultés rencontrées

Le scraping de VRBO s'est avéré complexe en raison de plusieurs facteurs :

— Chargement dynamique du contenu via JavaScript asynchrone

— Nécessité de simuler des clics utilisateurs (boutons, scrolls)

— Blocages potentiels liés à la protection anti-robots

— Temps de chargement variables → nécessité d'ajouter des `time.sleep()` et des `wait.until()`

Malgré cela, l'utilisation combinée de Selenium et BeautifulSoup a permis d'obtenir des résultats satisfaisants.

1.6 Nettoyage des Données et structuration

Avant d'entamer toute opération de similarité sémantique ou de résumé, les textes bruts ont été nettoyés et normalisés :

— **Nettoyage** : suppression des balises HTML, caractères spéciaux et de contrôle via expressions régulières.

```python
def remove_illegal_chars(text):
    if isinstance(text, str):
        # Supprimer les balises HTML comme <br />, < br />, etc.
        text = re.sub(r"<\s*br\s*/?\s*>", " ", text, flags=re.IGNORECASE)
        # Supprimer tous les caractères spéciaux sauf lettres, chiffres, ponctuation de base
        text = re.sub(r"[^\w\s.,!?'-]", " ", text)
        # Supprimer les caractères de contrôle
        text = re.sub(r'[\x00-\x1F\x7F-\x9F]', '', text)
        # Normaliser les espaces
        text = re.sub(r"\s+", " ", text)
        return text.strip().lower()
    return text
```

FIGURE 1.6 – Regx

— **suppression des stopwords** : a suppression des mots vides est une étape cruciale du prétraitement du texte dans l'analyse des sentiments. Elle consiste à supprimer les mots courants et non pertinents, c'est effectuée avec `spaCy`, combinée à `swifter` et `multiprocessing` pour un traitement accéléré.

— **Stemming** : chaque mot est réduit à sa racine avec `PorterStemmer` (NLTK) pour réduire la variabilité lexicale.

```python
def remove_illegal_chars(text):
    if isinstance(text, str):
        # Supprimer les balises HTML comme <br />, < br />, etc.
        text = re.sub(r"<\s*br\s*/?\s*>", " ", text, flags=re.IGNORECASE)
        # Supprimer tous les caractères spéciaux sauf lettres, chiffres, ponctuation de base
        text = re.sub(r"[^\w\s.,!?'-]", " ", text)
        # Supprimer les caractères de contrôle
        text = re.sub(r'[\x00-\x1F\x7F-\x9F]', '', text)
        # Normaliser les espaces
        text = re.sub(r"\s+", " ", text)
        return text.strip().lower()
    return text
```

FIGURE1.7 – Stemming

```python
# Vérifier le résultat
print(merged_data[['description_stemmed', 'comments_stemmed']].head(10))
```

```
                                description_stemmed  \
0  [modern, stylish, apart, complet, renov, 2011]

                                    comments_stemmed
0  [terrif, locat, nice, clean, neat, apart, exce...
1  [locat, facil, apart, amaz, meet, host, extrem...
```

FIGURE1.8 – Exemple output Stemming

Conclusion

Le scraping des descriptions de logements et des avis des visiteurs est effectué grâce à Selenium, BeautifulSoup, et Crawl4AI, ce qui a permis de constituer un corpus riche en informations. L'étape suivante consiste à nettoyer ce corpus, afin de procéder à l'annotation et à la détection de langue, en vue d'atteindre l'objectif final. Cet objectif est de résumer les avis des clients, d'obtenir un retour d'information complet, et de réviser la description initiale en générant une nouvelle description plus réaliste.

CHAPITRE 2

Annotation et traduction du corpus

Dans ce chapitre, nous allons nous focaliser sur deux étapes clés du pipeline de notre projet :

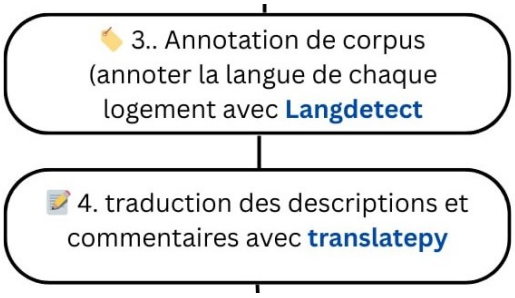

FIGURE2.1 – chap 2 pipeline

2.1 Annotation du corpus

En explorant notre corpus, nous avons détecté la présence de plusieurs langues, ce qui pose un défi majeur pour la traduction. Afin de garantir une traduction optimale et performante, nous mettons en place une phase d'annotation permettant d'identifier précisément la langue de chaque texte. Cette étape est essentielle pour structurer le corpus et assurer une traduction optimale.

La figure ci-dessous illustre la présence de plusieurs langues dans notre corpus, principalement le français et l'anglais, mais aussi d'autres langues telles que l'espagnol et l'italien.

3	3109	Lovely Appartment with one bedroom with a Quee...	Appartement totalement rénové, en parfait état...
4	5396	NEW SOFA-BED SINCE JUNE 2023, Please disregard...	Perfect location!! Nasrine was a delight and m...
...	...	...	...
9995	568394	- In the heart of one of the oldest district o...	We loved staying in Alain's apartment in Paris...

FIGURE2.2 – Exemple de multilangue

2.1.1 Détection de langue avec `langdetect`

Pour assurer un traitement homogène des textes en NLP, notamment dans le cadre de la traduction, nous avons appliqué une détection automatique de langue sur les descriptions et commentaires collectés.

Outil utilisé : `langdetect`

`langdetect` est une bibliothèque Python basée sur la librairie de détection de langue de Google. Elle repose sur un algorithme d'apprentissage supervisé utilisant des **n-grammes de caractères** pour identifier la langue d'un texte.

Fonctionnement :

— Le texte est segmenté en séquences de caractères (n-grammes).

— Chaque langue possède un modèle statistique de probabilité basé sur ces n-grammes.

— L'algorithme calcule la vraisemblance du texte d'appartenir à chaque langue.

— La langue avec la plus haute probabilité est retournée.

Avantages :

— Rapide et léger

— Supporte plus de 55 langues

— Performant sur des textes courts (titres, avis)

Dans notre projet, nous avons utilisé `langdetect` pour annoter chaque description et commentaire par une langue. Cette annotation nous a permis de filtrer les textes non anglophones et de les traduire en anglais avant d'appliquer les traitements NLP (similarité, résumé, génération).

2.1.2 Resultat :

La figure ci-dessous présente les résultats obtenus après l'application de `LangDetect` sur le corpus des commentaires, Cette analyse met en évidence la diversité linguistique des donnees collectés, avec une prédominance du français et de l'anglais, ainsi que la présence d'autres langues telles que l'espagnol et l'italien. L'annotation automatique des langues constitue une étape essentielle pour assurer une traduction cohérente

	comments	comments_lang
0	Tout s'est bien déroulé. Merci bien. PG	fr
1	Un petit nid fouiller douillet situé dans app...	fr
2	Appartement spacieux, propre,clair, et calme à...	fr
3	Appartement totalement rénové, en parfait état...	fr
4	Perfect location!! Nasrine was a delight and m...	en
5	This is a nice place in a great location in Pa...	en
6	Nice studio, very clean, very quiet, in a grea...	en
7	Superb location, great studio. \r A slice ...	en
8	Perfect place to be in Paris, walking home eve...	en
9	Wonderful hosts - very accommodating, friendly...	en

FIGURE2.3 – Exemple output du langDetect sur les commentaires

pour plus comprendre notre corpus annoter L'image en dessous présente un tableau indiquant la répartition des langues des commentaires.

- en (anglais) : 7347 commentaires

- fr (français) : 1698 commentaires

- es (espagnol) : 288 commentaires

- de (allemand) : 166 commentaires

```
Distribution des langues pour les commentaires :
comments_lang
en         7347
fr         1698
es          288
de          166
```

FIGURE2.4 – Distribution des langues dans les commentaires

2.2 Traduction du Corpus

Après l'annotation du corpus parla langue, cette etape constiste a traduire les descriptions et commentaires non anglophones vers l'anglais afin d'uniformiser les textes en vue d'un traitement NLP (calcul de similarité, résumé, génération). pour cela on a aopter a choisr parmi plusiers approche.

FIGURE2.5 – Traduction pipeline

2.2.1 selection de la methode du traduction

Dans cette section, nous abordons le choix du modèle de traduction le plus adapté pour une meilleure compréhension et une analyse plus de notre corpus, et donc pour traduire notre corpus multilingue (français, espagnol, allemand) vers l'anglais, quatre méthodes ont été évaluées : `translatepy`, **mBART**, **MarianMT**, et **M2M-100** :

1. **MarianMT (via `googletrans` et Hugging Face) :**

   MarianMT est une famille de modèles de traduction , Utilise une architecture Transformer optimisée pour la traduction, avec des modèles pré-entraînés pour des paires de langues spécifiques (ex. Helsinki-NLP)

   — **Avantages :**

   — Les modèles MarianMT sont plus légers que T5 ou mBART, ce qui les rend plus rapides à exécuter, même sur des machines avec des ressources limitées.

- Très rapide et fiable.

— **Inconvénients :**

- aires de langues spécifiques : Chaque modèle MarianMT est entraîné pour une paire de langues spécifique.

2. **TranslatePy et DeepL :**

translatepy s'appuie sur des API comme DeepL et Google Translate, qui sont souvent plus performantes pour les langues courantes grâce à leur entraînement sur des données massives et leur optimisation continue. DeepL, en particulier, excelle dans la préservation des nuances et des expressions idiomatiques.

— **Avantages :**

- Traductions de très haute qualité, notamment pour les textes longs et techniques.
- Tonalité plus naturelle que Google Translate.

— **Inconvénients :**

- API payante après une période gratuite limitée.
- Moins de langues supportées que Google.

3. **mBART (Multilingual BART) :**

mBART est une extension multilingue de BART (Bidirectional and Auto-Regressive Transformer), développée par Facebook AI. Il est pré-entraîné sur un corpus multilingue (25 langues) et conçu pour des tâches comme la traduction automatique. Architecture : Basé sur une architecture Transformer, mBART utilise un pré-entraînement denoising (reconstruction de texte bruité) pour apprendre des représentations multilingues, ce qui le rend efficace pour la traduction.

— **Avantages :**

- Capacité multilingue : mBART est pré-entraîné sur 25 langues, ce qui lui permet de gérer des paires de langues comme français $\rightarrow$ anglais, espagnol $\rightarrow$ anglais, ou allemand $\rightarrow$ anglais sans nécessiter un fine-tuning intensif.

- **Inconvénients :**
  - Consommation de tokens élevée.
  - Plus lent qu'une API dédiée.
  - Nécessite une clé API (payante à grande échelle).

4. **M2M100 de Facebook / NLLB (No Language Left Behind) :**

   M2M-100, développé par Meta AI, est un modèle de traduction multilingue capable de traduire directement entre 100 langues sans passer par une langue pivot (comme l'anglais). Il est conçu pour des traductions "many-to-many" (de n'importe quelle langue source vers n'importe quelle langue cible).

   - Compatible avec la bibliothèque `transformers` :

```
from transformers import M2M100ForConditionalGeneration

from transformers import M2M100Tokenizer
```

5. **Avantages :**
   - Architecture : Basé sur une architecture Transformer, M2M-100 est pré-entraîné sur un corpus massif de 7,5 milliards de phrases couvrant 100 langues.
   - Pas de langue pivot : Contrairement à certains modèles qui traduisent d'abord vers l'anglais avant de passer à une autre langue, M2M-100 traduit directement, réduisant les erreurs cumulatives.
6. **Inconvénients :**
   - Nécessite des ressources GPU pour de bonnes performances.
   - Bien qu'accessible via Hugging Face, il peut être plus difficile à configurer pour des utilisateurs novices (ex. gestion des tokens de langue).

2.2.2 Résumé comparatif

— `translatepy` : Bibliothèque Python s'appuyant sur DeepL et Google Translate, offrant une excellente qualité pour les langues courantes.

— **mBART** : Conçu pour la traduction multilingue (25 langues), avec une bonne qualité pour les langues courantes, mais lourd en ressources.

— **MarianMT** : Spécialisé pour la traduction, léger et précis pour des paires de langues spécifiques, idéal pour des ressources limitées.

— **M2M-100** : Traduit directement entre 100 langues, unifié et performant, mais nécessite un GPU puissant.

Critère	translatepy	mBART	MarianMT	M2M-100
Spécialisation	Traduction via API	Traduction multilingue	Traduction spécifique	Traduction multilingue
Nombre de langues	Très large (via API)	25 langues	Centaines de paires	100 langues
Qualité	Très bonne (avec DeepL)	Bonne	Très bonne	Bonne à très bonne
Ressources	Faibles	Élevées	Modérées	Très élevées
Intégration	Très facile	Moyenne	Facile	Moyenne

TABLE 2.1 – Comparaison des méthodes de traduction pour le projet VRBO.

Pour notre projet, on a opter a choisir `translatepy` pour sa simplicité d'integration et sa qualité (via DeepL), qui offre une meilleure qualité pour les langues courantes.

2.2.3 traduction avec translatepy

Bibliothèque utilisée : `translatepy`

Fonctionnement : Traduction réalisée langue par langue, par lots (*batch*), afin d'optimiser les ressources et de réduire la charge.

— Détection automatique des codes de langue (par ex., `zh-tw` → `zh-TW`, `iw` → `he`).

— En cas d'erreur lors de la traduction, le texte original est préservé tel quel.

Exemple de journal de traduction :

listings

```
[1/50] fr  en : "Trs bel appartement, bien situ..."  "Very nice
   apartment, well located..."
[2/50] es  en : "Muy limpio y cerca del metro..."  "Very clean and
   close to the subway..."
```

La figure ci-dessous montre les résultats de traduction, qui seront examinés en détail plus tard dans une interface Streamlit, illustrant l'utilisation de notre chatbot. Vous trouverez également une présentation détaillée de cette interface.

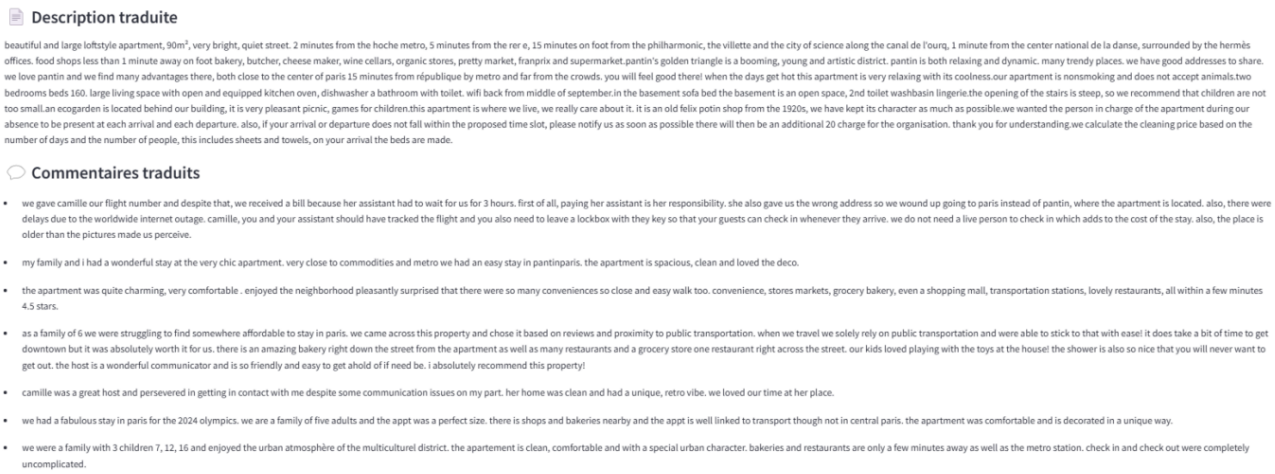

FIGURE2.6 – Exemple output de traduction dans l'interface streamlit

Conclusion

Dans cette partie, nous avons annoté notre corpus à l'aide de Langdetect , puis effectué la traduction avec Translatepy . Ces deux étapes sont nécessaires et fondamentales dans notre pipeline pour atteindre l'objectif final, qui consiste à utiliser **des modèles de similarité NLP** et à **générer un feedback final** ainsi qu'une **description**.

CHAPITRE 3

Similarite sémantique, résumé et géneration

Dans ce chapitre, nous nous concentrerons sur les trois dernières étapes, qui constituent le cœur du pipeline de notre projet :

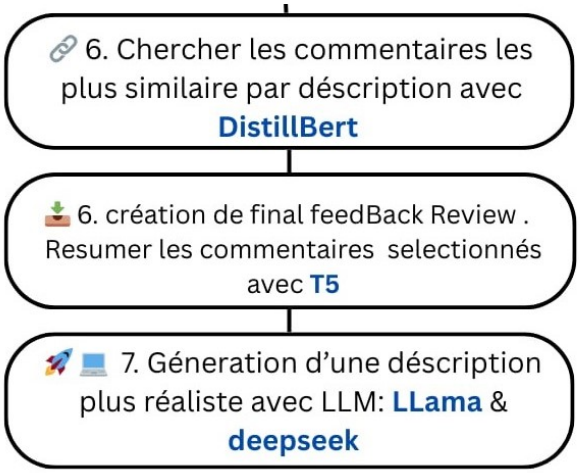

FIGURE3.1 – chapitre 3 pipeline

3.1 Sélection des commentaires les plus Similaires

Dans cette première partie, notre objectif est de sélectionner les 5 commentaires les plus similaires entre eux et à la description. Cela représente un défi, notamment dans le choix des modèles de transformation de phrases optimaux pour la génération des embeddings. Cette approche nous permettra de calculer la similarité tout en préservant la sémantique au niveau de la phrase, en créant des embeddings qui capturent le contexte.

FIGURE3.2 – sentence transformers

3.1.1 Comparaison des Méthodes de Similarité

Trois approches ont été mises en œuvre pour la transformation des phrases en embeddings et la mesure de la similarité entre les descriptions de logements et leurs commentaires :

1. **DistilBERT :**

   Basé sur une version "distillée" de BERT (Bidirectional Encoder Representations from Transformers), DistilBERT est un modèle plus léger (40

   - **Avantages :**
     - Maintien d'une bonne performance en captant la semantique et le contexte.
     - Réduction du temps d'inférence grâce à un modèle plus compact.
   - **Inconvénients :**
     - Moins efficace pour capturer des dépendances longues dans les phrases.

2. **DistilRoBERTa :**

   Version distillée de RoBERTa (Robustly Optimized BERT Approach), qui optimise les performances de BERT en supprimant l'objectif "Next Sentence Prediction" et en utilisant davantage de données d'entraînement.

   - **Avantages :**
     - plus perferment que distilBERT grâce à un pré-entraînement plus robuste.
     - Apprentissage plus rapide grâce à la distillation du modèle.
     - Excellente performance sur des tâches de similarit textuelle.
   - **Inconvénients :**
     - Consomme encore plus de ressources que DistilBERT.

3. **Word2Vec :**

   Algorithme d'apprentissage de représentations vectorielles basé sur un réseau de neurones peu profond (shallow neural network), avec deux approches principales : CBOW (Continuous Bag of Words) et Skip-gram.

— **Avantages :**

  — Génération rapide de vecteurs de mots à partir de grands corpus.

  — Moins gourmand en ressources que les modèles transformers.

— **Inconvénients :**

  — Ne prend pas en compte le contexte et la semantique d'une phrase entière (problème de désambiguïsation).

  — Moins efficace pour capturer les relations entre phrases longues.

  — Représentation statique des mots (un mot a toujours le même vecteur, indépendamment du contexte).

3.1.2 Resultat et Interpretation

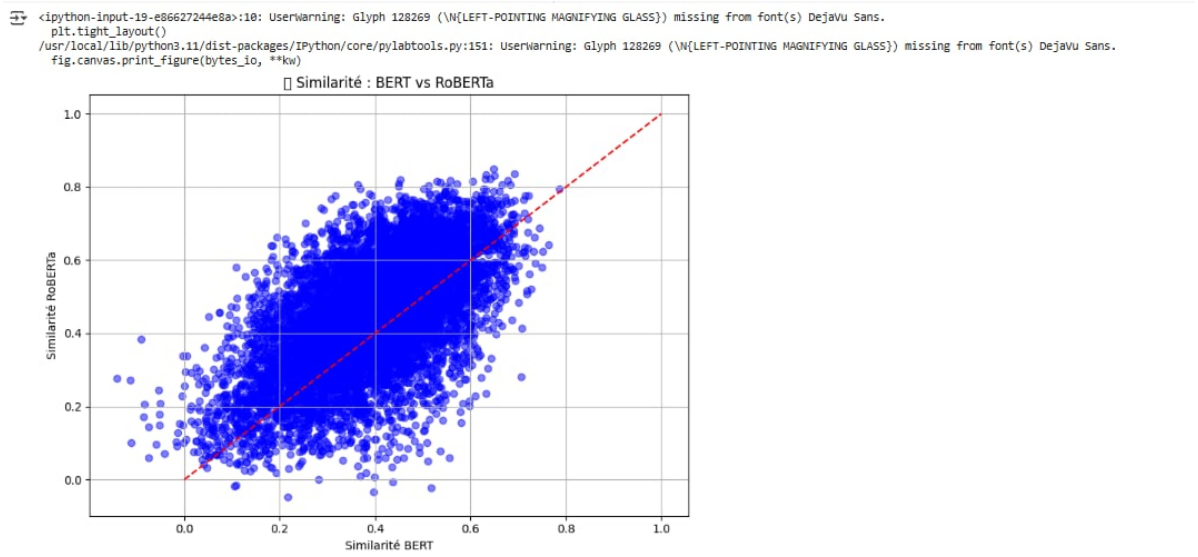

FIGURE3.3 – BERT et RoBERTa comparaison

La figure illustre la comparaison des scores de similarité calculés par BERT et RoBERTa pour des paires de descriptions et de commentaires. Chaque point bleu représente une paire, où l'axe X indique la similarité obtenue avec BERT et l'axe Y celle obtenue avec RoBERTa. La ligne rouge en pointillés (y = x) sert de référence : si un point est au-dessus, cela signifie

que RoBERTa attribue une similarité plus élevée que BERT pour cette paire, et inversement si le point est en dessous.

Globalement, les scores des deux modèles sont corrélés, suivant une tendance diagonale, ce qui est attendu puisqu'ils sont tous deux basés sur des architectures Transformer et entraînés sur des tâches similaires. Cependant, RoBERTa a tendance à attribuer des similarités plus élevées que BERT, suggérant qu'il capte mieux les nuances sémantiques dans les commentaires et descriptions analysés.

On observe également une dispersion plus marquée des points lorsque BERT attribue des scores élevés (supérieurs à 0.7). Dans ces cas, RoBERTa présente une plus grande variabilité, avec des scores allant de 0.3 à 0.9. Cette différence suggère que RoBERTa pourrait être plus sensible aux subtilités du langage, offrant ainsi une meilleure discrimination sémantique entre des phrases proches mais non identiques.

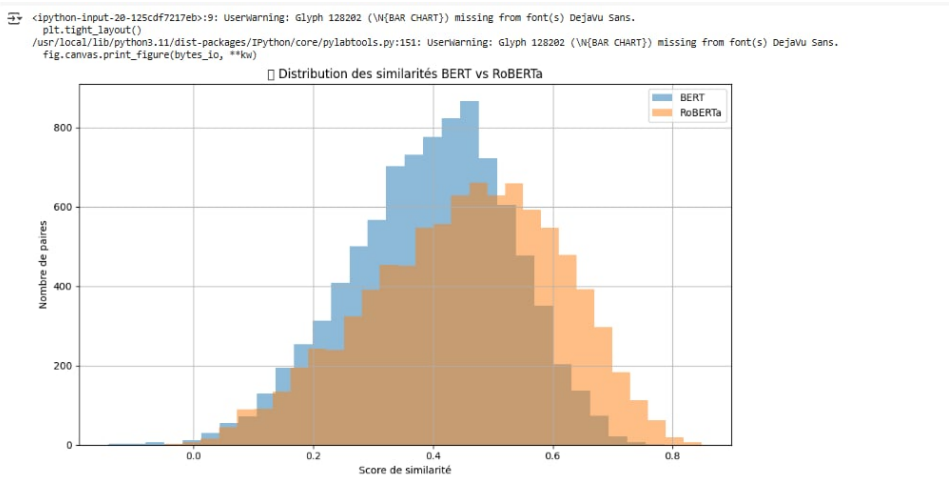

FIGURE3.4 – Distribution des similarites entre Bert et Roberta

BERT attribue des scores de similarité globalement plus bas, avec une moyenne centrée autour de 0.4. Cela signifie qu'il évalue la proximité sémantique de manière plus prudente et conservatrice, accordant rarement des scores élevés.

À l'inverse, RoBERTa attribue des similarités plus élevées en moyenne, avec un pic situé

entre 0.5 et 0.6. Il considère donc les paires de phrases comme étant plus proches sémantiquement que ne le fait BERT.

De plus, la distribution des scores de RoBERTa est plus étalée vers des valeurs élevées (> 0.7), suggérant qu'il capte mieux certaines similarités fortes. Cela peut indiquer une meilleure sensibilité sémantique, lui permettant d'identifier plus précisément les liens entre les descriptions et les commentaires.

En résumé, BERT adopte une approche plus conservatrice et modérée dans l'évaluation des similarités

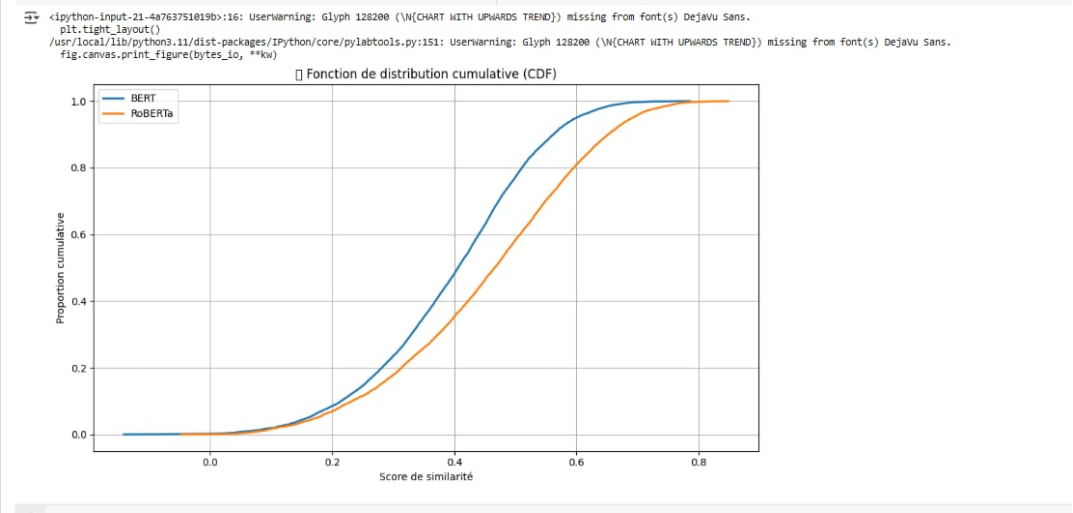

FIGURE3.5 – Fonction de distribution cumulative

BERT est plus conservateur La courbe de BERT (en bleu) monte plus rapidement que celle de RoBERTa (en orange) avant 0.5. Cela signifie que plus de 60

RoBERTa capte mieux les fortes similarités Au-delà de 0.6, la courbe de RoBERTa dépasse celle de BERT. Cela indique que RoBERTa détecte davantage de paires avec une forte similarité. Il semble donc plus sensible aux liens sémantiques profonds entre les phrases.

conclusion finale :

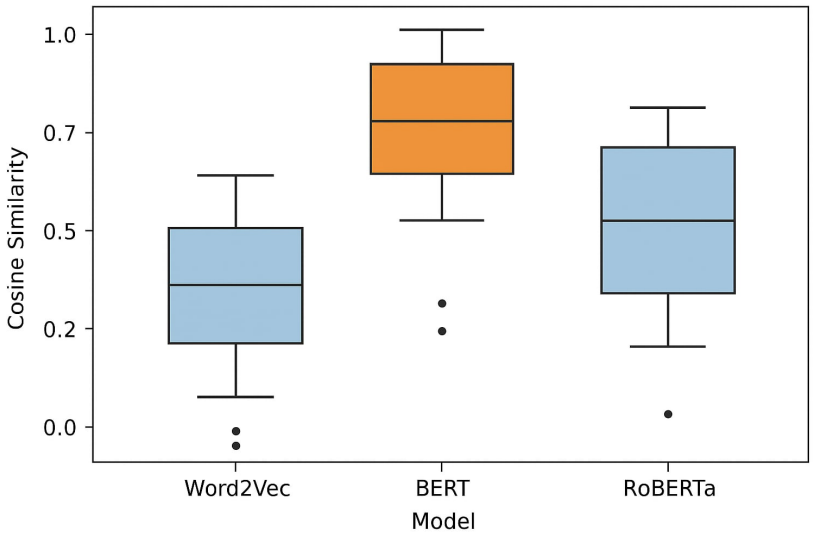

FIGURE3.6 – boxplot comparaison

Les résultats indiquent que DistilBERT adopte une approche plus prudente et modérée dans l'évaluation des similarités. Il génère des scores plus équilibrés et cohérents, garantissant une meilleure fiabilité dans l'identification des commentaires pertinents. Grâce à sa capacité à capturer le contexte et la sémantique, il parvient à établir des liens significatifs entre une description et des commentaires, même en l'absence d'un vocabulaire commun. Cette aptitude lui permet ainsi de mieux discerner la proximité réelle des phrases sur le plan du contexte.

En conclusion, nous avons choisi d'utiliser DistilBERT pour sélectionner les cinq commentaires les plus similaires entre eux et à la description. Ce choix repose sur sa capacité à capturer à la fois le contexte et la sémantique, garantissant ainsi une évaluation plus précise des similarités.

3.2 Creation Final FeedBack (résumé des commentaires)

Dans cette section, nous allons aborder la génération du feedback final, qui servira de repère aux utilisateurs des sites de location. Ce feedback leur permettra d'obtenir une synthèse des avis sans avoir à lire l'ensemble des commentaires.

Pour cela, nous avons exploré deux approches et nous choisirons la meilleure à la fin :

1. LSTM (Long Short Term Memory) avec Seq2Seq

2. T5 (Text-to-Text Transfer Transformer) / Bart

3.2.1 LSTM (Long Short Term Memory) avec Seq2Seq et resultat :

Encoder plusieurs commentaires d'une description dans un LSTM, puis décoder ces informations pour générer un résumé pertinent. L'objectif du modèle LSTM (Long Short-Term Memory) avec une architecture Seq2Seq est de transformer un texte d'entrée (par exemple, des avis clients) en un texte de sortie (comme un résumé de feedback). Cette approche repose sur deux composants principaux : un encodeur et un décodeur, tous deux basés sur des réseaux de neurones.

— **Architecture Seq2Seq avec LSTM et mécanisme d'attention :**

  — Encodeur (LSTM) : L'encodeur est généralement un LSTM (Long Short-Term Memory) qui parcourt la séquence d'entrée, mot par mot, pour générer une représentation interne de la séquence. À chaque étape, l'encodeur produit un état caché qui capture l'information contextuelle.

    Dans le cadre classique de Seq2Seq, l'encodeur génère un seul vecteur de contexte (le dernier état caché) résumant toute la séquence d'entrée.

  — Mécanisme d'attention : Au lieu de se limiter à un seul vecteur de contexte, l'attention permet au décodeur de "regarder" différentes parties de l'encodeur à chaque étape de la génération du texte.

Le mécanisme d'attention calcule un poids d'attention pour chaque état caché de l'encodeur. Ces poids déterminent l'importance relative de chaque élément de la séquence d'entrée pour générer la sortie à chaque étape.

- Décodeur (LSTM) : Le décodeur, également un LSTM, utilise l'état caché de l'encodeur et les poids d'attention pour générer la séquence de sortie

- **Avantages :**
  - Le mécanisme d'attention permet au modèle de se souvenir de différentes parties de la séquence d'entrée, même pour des séquences longues. Cela évite la perte d'informations importantes qui pourrait se produire avec un vecteur de contexte unique dans le modèle classique Seq2Seq.
  - L'attention permet au modèle de se concentrer spécifiquement sur des parties de la séquence d'entrée qui sont les plus importantes à chaque étape du décodeur. Cela améliore la précision du modèle.
- **Inconvénients :**
  - Le calcul des poids d'attention à chaque étape est coûteux en termes de calcul. Cette augmentation de la complexité peut rendre l'entraînement du modèle plus lent et plus difficile, surtout lorsque la séquence d'entrée est longue.
  - LSTM, bien que mieux adapté aux séquences que les RNN traditionnels, reste séquentiel dans son traitement des données.

implementation LSTM + Seq2Seq et resultat :

Tokenizer :a tokenisation est effectuée en utilisant la classe Tokenizer de Keras.

```
# Tokenisation
tokenizer = Tokenizer(num_words=VOCAB_SIZE, oov_token="<OOV>")
tokenizer.fit_on_texts(df_grouped['all_comments'])
input_seq = tokenizer.texts_to_sequences(df_grouped['all_comments'])
input_padded = pad_sequences(input_seq, maxlen=MAX_INPUT_LENGTH, padding='post')
```

FIGURE3.7 – Tokenizer

Architecture Seq2Seq avec LSTM : L'encodeur prend les commentaires (texte d'entrée) et les transforme en une représentation interne sous forme de vecteurs ou états cachés qui résument l'information contextuelle du texte. L'encodeur traite l'entrée de manière séquentielle et encode l'information en un vecteur compact qui sera ensuite utilisé par le décodeur.

```
# Modèle Seq2Seq avec LSTM et mécanisme d'attention
# Encodeur
enc_input = Input(shape=(50,))
enc_emb = Embedding(5000, 128, mask_zero=True)(enc_input)
enc_lstm, state_h, state_c = LSTM(128, return_sequences=True, return_state=True)(enc_emb)
```

FIGURE3.8 – Encodeur

Le décodeur prend les états cachés de l'encodeur et génère un résumé (texte de sortie). Le décodeur utilise l'information contextuelle fournie par l'encodeur pour produire une séquence de mots représentant un résumé pertinent du texte d'entrée.

```
# Décodeur
dec_input = Input(shape=(20,))
dec_emb = Embedding(5000, 128, mask_zero=True)(dec_input)
dec_lstm, _, _ = LSTM(128, return_sequences=True, return_state=True)(dec_emb, initial_state=[state_h, state_c])
```

FIGURE3.9 – decodeur

Mécanisme d'Attention :le mécanisme d'attention est appliqué entre les sorties de l'encodeur et du décodeur. Les scores d'attention sont calculés via une couche Dot et sont normalisés par une fonction softmax

```
# Calcul des scores d'attention
attn_score = Dot(axes=[2, 2])([dec_lstm, enc_lstm])
attn_weights = Activation('softmax')(attn_score)
context = Dot(axes=[2, 1])([attn_weights, enc_lstm])
```

FIGURE3.10 – Ajout de Mecanisme d'Attention

combinaison du contexte et de la sortie du décodeur La concaténation du contexte (extrait par attention) et de la sortie du LSTM décodeur permet d'intégrer les informations des deux sources : l'entrée et l'état interne du modèle, cela produit une entrée enrichie qui est ensuite utilisée pour générer la prédiction du mot suivant dans la séquence de sortie final FeedBack, cette approche permet d'augmenter la précision du modèle.

```
# Combinaison du contexte et de la sortie du décodeur
dec_combined = Concatenate(axis=-1)([context, dec_lstm])
dec_dense = Dense(5000, activation='softmax')(dec_combined)
```

FIGURE3.11 – Combianaison Lstm+Attention

La figure ci-dessous inclut les résultats d'entraînement du modèle LSTM :

```
Epoch 1/50
10/10 ━━━━━━━━━━━━━━━━━━━━ 22s 580ms/step - loss: 8.5061
Epoch 2/50
10/10 ━━━━━━━━━━━━━━━━━━━━ 7s 701ms/step - loss: 7.7454
Epoch 3/50
10/10 ━━━━━━━━━━━━━━━━━━━━ 8s 444ms/step - loss: 5.8620
Epoch 4/50
10/10 ━━━━━━━━━━━━━━━━━━━━ 6s 467ms/step - loss: 5.6629
Epoch 5/50
10/10 ━━━━━━━━━━━━━━━━━━━━ 3s 301ms/step - loss: 5.4622
```

FIGURE3.12 – Entrainament du modele Lstm

Conclusion :suite a le faite que Les séquences longues (commentaires concaténés) ont rapidement saturé la mémoire du modèle. Le vocabulaire généré était limité et répétitif, avec des sorties souvent peu informatives (ex. : bank...). De plus, l'entraînement nécessitait une grande quantité de données et un temps élevé. Enfin, l'LSTM restait limité dans la modélisation du contexte global à long terme, même avec l'intégration d'une couche d'attention.

Face à ces défis, nous avons opté pour l'utilisation de modèles basés sur Transformers tels que BART et T5 , qui offrent des capacités de traitement plus performantes pour la gestion du contexte global et la génération de texte plus cohérente et variée.

Résumé visuel de l'architecture LSTM + Attention

Ce schéma montre le pipeline de génération séquentielle avec focus dynamique via attention. Chaque mot généré est influencé par les parties les plus importantes de l'entrée, identifiées par des poids d'attention.

3.2.2 Transition vers les modèles Transformer T5 et BART

Afin de dépasser les limitations du modèle Seq2Seq, nous avons utilisé des modèles de type **Transformer pré-entraînés**, qui ont montré d'excellents résultats pour la résumé.

— **BART (Bidirectional and Auto-Regressive Transformer)** : ce modèle combine un encodeur bidirectionnel et un décodeur autoregressif . Cela lui permet de capturer à la fois le contexte global d'un texte et de générer des sorties séquentielles. Le résumé.

— **T5 (Text-to-Text Transfer Transformer)** : repose sur une architecture encoder-decoder optimisée pour les tâches NLP sous un format uniforme "input → output".

Critère	**BART**	**T5**
Architecture	Transformer standard avec encodeur bidirectionnel (BERT) et décodeur autoregressif (GPT).	Architecture encoder-decoder optimisée pour des tâches text-to-text.
Pré-entraînement	Corruption de texte (supprime, remplace, réordonne) pour reconstruire le texte original.	Entraînement sur une grande diversité de tâches NLP sous format "input → output".
Stratégie de Résumé	Optimisé pour l'extraction et la réécriture. Utilisé pour le résumé extractif et abstrait.	Entraîné pour générer des résumés cohérents et fluides directement à partir des corpus.
Qualité du résumé	Résumés plus fidèles au texte original, conservant le contexte et la structure.	Résumés plus fluides et naturels avec une meilleure compréhension sémantique.
Capacité à capturer le contexte	Très bon pour comprendre et reformuler des phrases avec fidélité.	Excellente capacité à reformuler et synthétiser le contenu sémantique.
Taille et flexibilité	Versions multiples (BART-Large, BART-Base) selon les ressources.	Plusieurs tailles (T5-Small, T5-Base, T5-Large, T5-XL, T5-XXL), adaptées à différentes applications NLP.
Type de Résumé	Excellente reformulation tout en respectant la structure d'origine.	Génère des textes proches du langage humain naturel.

TABLE 3.1 – Comparaison entre BART et T5

Approche utiliser : combinaison du modèle T5 et BART

pour générer des résumés de textes on a opter exploiter et benificier de ces deux modèles : BART et T5. L'objectif est de produire des résumés plus cohérents en combinant les résultats des deux modèles.

Stratégies d'Ensemble :

Trois stratégies sont proposées pour combiner les résumés :

Best : Sélectionne le résumé avec le meilleur score.

Fusion : Fusionne les résumés générés par les deux modèles.

Weighted : Pondère les résumés selon les scores ROUGE obtenus par chaque modèle.

resumer avec BART : Cette méthode génère un résumé en utilisant le modèle BART. Elle effectue la tokenisation du texte d'entrée, puis génère un résumé en utilisant la méthode generate de BART. Les paramètres comme la longueur du résumé, la pénalité de longueur et la taille des faisceaux sont ajustés pour optimiser la qualité du résumé.

```python
def bart_summary(self, text: str, max_length: int = 150, min_length: int = 40, num_beams: int = 4)
    inputs = self.bart_tokenizer.encode(
        text,
        return_tensors="pt",
        max_length=1024,
        truncation=True
    ).to(self.device)

    summary_ids = self.bart_model.generate(
        inputs,
        num_beams=num_beams,
        length_penalty=2.0,
        max_length=max_length,
        min_length=min_length,
        no_repeat_ngram_size=3,
        early_stopping=True
    )
```

FIGURE3.13 – partie du code Bart

resumer avec T5 : cette méthode fonctionne de manière similaire à BART, mais on ajoute le préfixe "summarize : " pour la tâche de résumé. T5 génère un résumé à l'aide de son propre tokenizer et modèl.

```python
def t5_summary(self, text: str, max_length: int = 150, min_length: int = 40, num_beams
    input_text = "summarize: " + text
    inputs = self.t5_tokenizer.encode(
        input_text,
        return_tensors="pt",
        max_length=1024,
        truncation=True
    ).to(self.device)

    summary_ids = self.t5_model.generate(
        inputs,
        num_beams=num_beams,
        length_penalty=2.0,
        max_length=max_length,
        min_length=min_length,
        no_repeat_ngram_size=3,
        early_stopping=True
    )
```

FIGURE3.14 – partie du code T5

Fusion des résumés pour avoir Final FeedBack : ette méthode combine les résumés générés par BART et T5 en fonction de la stratégie choisie pour avoir le FeedBack.

```python
def ensemble_summary(
    self,
    text: str,
    max_length: int = 150,
    min_length: int = 40,
    strategy: str = 'fusion',  # 'best', 'fusion', 'weighted'
    weights: Tuple[float, float] = (0.5, 0.5)  # Poids pour BART et T5
) -> Dict[str, Any]:
    bart_summary = self.bart_summary(text, max_length, min_length)
    t5_summary = self.t5_summary(text, max_length, min_length)

    # Évaluation des résumés
    bart_scores = self._evaluate_summary(text, bart_summary)
    t5_scores = self._evaluate_summary(text, t5_summary)
```

FIGURE3.15 – partie du code su fusion T5 et Bart

Resultat dans l'interface streamlit du chatbot :

L'image ci-dessous présente les résultats du résumé FINAL FEEDBACK, qui seront analysés en profondeur dans une interface Streamlit, mettant en lumière l'utilisation de notre chatbot. Vous trouverez également une présentation détaillée de cette interface.

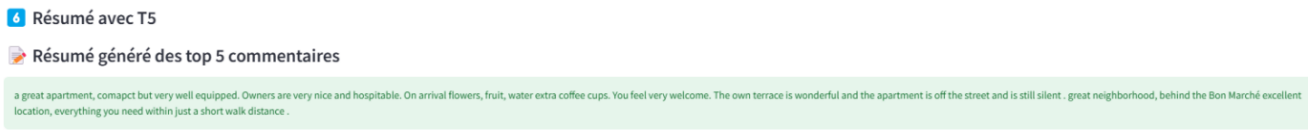

FIGURE3.16 – Feedback

3.3 Generation de la description avec LLM LLama :

l'objective est d'améliorer une description initiale en tenant compte des avis des utilisateurs. Il utilise un modèle LLaMA-3.3-70B pour générer une nouvelle description en s'appuyant sur un résumé des avis les plus pertinents.

Pour fournir une description plus réaliste et fidèle à la réalité du bien en exploitant les commentaires clients plutôt que la description fournie par l'hôte via LLAMA.

1. **Modèle utilisé :**

   il est cruciale de utiliser les capaciter des llm sur tout dans les phases de generation de texte apres avour uriliser les modeles transformers dans cette partie on va uluiliser les llm pour la generation de la nouvelle description .

   Le modèle utilisé est LLaMA-3.3-70B-Versatile, un Large Language Model (LLM) développé par Meta. Voici ses principales caractéristiques :

   — Nombre de paramètres : 70 milliards (70B), ce qui en fait un modèle très puissant pour la compréhension et la génération de texte.
   — Capacité à résumer, reformuler et générer des descriptions réalistes et prise en compte du contexte via le prompt.

```
chat_completion = client.chat.completions.create(
    messages=[{"role": "user", "content": prompt}],
    model="llama-3.3-70b-versatile",
)
st.markdown("### 🧠 Description générée via Groq")
st.success(chat_completion.choices[0].message.content)
```

FIGURE3.17 – Llama model

2. **prompt :**

   n prompt est l'entrée (ou l'instruction) donnée à un LLM pour lui dire quoi faire. Il sert à guider le modèle pour générer une réponse en fonction du contexte.

   Dans ce cas, le prompt est construit en combinant plusieurs éléments :

   — Feed prompt : On fournit au modèle de langage trois éléments essentiels : La description initiale (desc_en), $quireprésenteladescriptiondebasefournieparl'hote.$

Les cinq commentaires les plus pertinents (top5), sélectionnés en fonction de leur proximité avec la description initiale.

Un résumé de ces commentaires (FeedBack), qui synthétise les points clés mentionnés par les clients.

```
prompt = f"Voici une description initiale :\n{desc_en}\n\n"
prompt += "Voici les 5 commentaires les plus proches :\n" + "\n".join([c[0] for c in top5]) + "\n\n"
prompt += f"Voici un résumé de ces commentaires :\n{t5_summary}\n\n"
```

FIGURE3.18 – prompt fournit

— Contexte et rôle du modèle → Le modèle est présenté comme un expert en réécriture de descriptions pour améliorer leur réalisme et leur attrait.

— instructions explicites au modèle

— Analyser la description originale et les avis clients.

— Identifier les forces et les critiques soulevées par les utilisateurs.

— Rédiger une nouvelle description améliorée qui :

— Conserve l'essence de la description initiale.

— Met en avant les points positifs relevés par les utilisateurs.

```
    def generate_description(
        des commentaires et un résumé.

        Args:
            original_desc: Description originale
            comments: Liste des commentaires pertinents
            Résumé:  FeedBack
            temperature: Température pour la génération (contrôle la créativité)
            max_tokens: Nombre maximum de tokens à générer

        Returns:
            Résultat de la génération et métadonnées
        """
        # Construire le prompt avec des instructions claires
        prompt = f"""Tu es un expert en réécriture de descriptions de produits ou services pour les rendre plus réalistes

DESCRIPTION ORIGINALE:
{original_desc}

COMMENTAIRES PERTINENTS:
{chr(10).join([f"- {comment}" for comment in comments])}

RÉSUMÉ DES COMMENTAIRES:
{summary}

INSTRUCTIONS:
1. Analyse la description originale et les commentaires des utilisateurs
2. Identifie les points forts et les critiques mentionnés dans les commentaires
3. Crée une nouvelle description qui:
   - Reste fidèle au sujet principal de la description originale
   - Intègre les retours positifs des commentaires
   - Adresse de manière subtile les points faibles identifiés
   - Utilise un ton professionnel et convaincant
```

FIGURE3.19 – prompt avec instruction détaillée

Resultat finale et conclusion :

L'image ci-dessous illustre les résultats de la génération d'une description plus réaliste via LLaMA, en s'appuyant sur un prompt détaillé. Ce processus permet d'optimiser le choix du logement pour les utilisateurs en exploitant deux **Features clés générées et fournies par notre assistant : une description affinée et un feedback pertinent.**

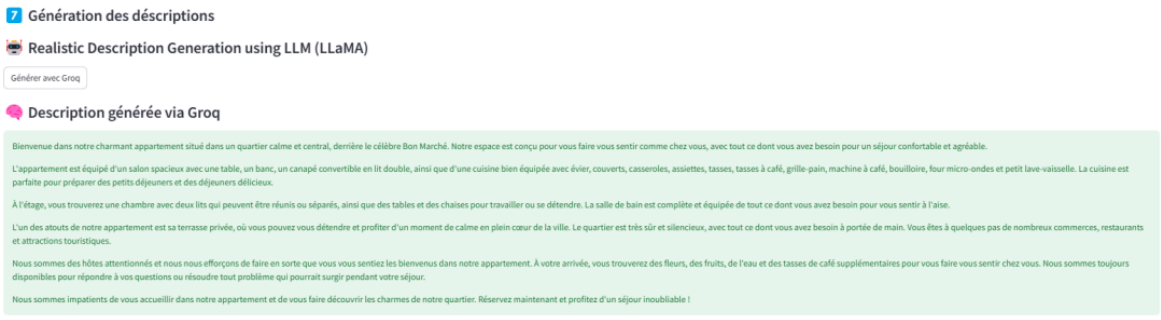

FIGURE3.20 – Nouvelle description generer

Ces aspects sont analysés en profondeur au sein d'une interface Streamlit, mettant en avant l'utilisation de notre chatbot.

Conclusion génerale :

Ce projet a permis de concevoir un assistant intelligent dédié à l'optimisation du choix des logements, en exploitant les avis clients et en générant des descriptions plus réalistes et objectives. Il vise à remédier aux biais des descriptions fournies par les hôtes en s'appuyant sur les avancées du traitement automatique du langage naturel (NLP) et des modèles de generative IA (LLMs). L'objectif principal est de produire automatiquement une description enrichie et représentative d'un logement en croisant les informations initiales avec les retours d'expérience des voyageurs.

Grâce à une approche combinant web scraping, analyse sémantique et génération de texte, notre solution réduit l'écart entre les descriptions idéalisées et la réalité perçue par les utilisateurs. Nous utilisons Transformers pour mesurer la similarité entre avis et description, et pour extraire et synthétiser les avis les plus pertinents,enfin un LLM pour générer des descriptions affinées, plus objectives et alignées avec les attentes des voyageurs.

Cette approche novatrice améliore la transparence des plateformes de location et permet aux utilisateurs de faire des choix plus éclairés et adaptés à leurs besoins. À terme, ce travail pourrait être étendu en intégrant de nouveaux critères d'évaluation (rapport qualité-prix, etc.) pour offrir une personnalisation encore plus fine des recommandations.

Résumé :

Ce projet vise à optimiser le choix des logements en exploitant à la fois les avis clients et les descriptions fournies par les hôtes. Il propose deux fonctionnalités clés : "Final Feedback" et "New Real Description", qui constituent de nouveaux éléments générés par notre assistant pour les utilisateurs des plateformes de location. Grâce à l'intégration du NLP et des LLMs, notre solution combine le scraping, l'analyse de similarité sémantique (DistilBERT), le résumé automatique (T5 et BART), ainsi que la génération de descriptions améliorées (Groq-LLaMA 3.3). L'objectif est de fournir des descriptions plus fidèles à la réalité, renforçant ainsi la transparence et permettant aux utilisateurs de faire des choix mieux informés.

3.4 Mots-clés :

Scraping, NLP,Transformers, DistilBERT, T5, LLM, cral4Ai, Sémantique,LLaMA, Analyse des avis, Génération de texte.