CentraleSupélec

Rapport de fonctionnement

Multi Model Agent Translator

22 mai 2025

Table des matières

Introduction

Ce rapport de fonctionnement vise à présenter la structure et les différentes fonctions de notre projet pour faciliter le travail de l'alternant de SOCOTEC qui reprendra notre projet. Puisque notre code s'appuie sur beaucoup de librairies, nous avons tâché de lister toutes les dépendances pour faciliter l'installation du dépôt.

1 Présentation

Multi-Modal-Agent-Translator est un projet Python permettant d'extraire, de traduire et d'évaluer du contenu provenant de documents PDF ou Word. Il s'appuie sur une architecture d'agents pour chaque étape du traitement.

2 Installation

1. **Cloner le dépôt** :

   ```
   git clone https://github.com/AlaBoussoffara/Multi-Modal-Agent-Translator
   cd Multi-Modal-Agent-Translator
   ```

2. **Installer les dépendances** :

   ```
   pip install -r requirements.txt
   ```

   Assurez-vous d'avoir également Tesseract OCR et Poppler installés sur votre système pour l'OCR et la conversion PDF.

3. **(Optionnel) Lancer l'interface Streamlit** :

   ```
   streamlit run streamlit_app.py
   ```

3 Utilisation

— Placez vos documents sources dans `src_documents/`.
— Utilisez les scripts ou l'interface Streamlit pour extraire, traduire et évaluer les documents.
— Les résultats sont enregistrés dans `mt_outputs/` (traductions) et `comet_scores/` (évaluations).

4 Architecture

4.1 Arborescence du projet

```
agents/
    extractor_agent.py
    generator_agent.py
    translator_agent.py
    evaluation_agent.py
    type_detection_agent.py
comet_scores/
    evaluation_results.csv
glossary/
    dictgeniecivil.pdf
    glossary_engfr_metadata.pickle
    glossary_engfr.faiss
    glossary_freng_metadata.pickle
    glossary_freng.faiss
mt_outputs/
    ... (traductions générées)
ref_translations/
```

```
    ... (traductions de référence)
src_documents/
    ... (documents sources)
streamlit_app.py
extraction.py
mmat_langgraph_version.py
requirements.txt
readme.md
```

4.2 Agents

Le projet est organisé autour d'agents spécialisés.

4.2.1 type_detection_agent.py

Ce fichier sert à détecter automatiquement le type d'un fichier en fonction de son extension. Une classe `TypeDetectionAgent` propose une méthode `detect(filepath)` : elle analyse l'extension du fichier (par exemple .pdf, .docx, .html, .txt) et retourne une chaîne indiquant le type de fichier correspondant.

4.2.2 extractor_agent.py et generator_agent.py

Le fichier `extractor_agent.py` regroupe l'ensemble des classes et fonctions nécessaires à l'extraction de contenu structuré à partir de différents types de fichiers : PDF, DOCX, HTML et TXT. Il propose des librairies capables de lire un document, d'en extraire les paragraphes, la mise en forme (police, taille, couleur, etc.) et de standardiser ces informations dans un dictionnaire. Ce module constitue la première étape du pipeline, en produisant des données prêtes à être traduites ou analysées.

Le fichier `generator_agent.py` est responsable de la génération des fichiers de sortie contenant le contenu traduit, tout en préservant la mise en page et le format d'origine. Il est capable de réinsérer le texte traduit dans la structure originale en rédigeant le texte source dans un PDF puis en insérant le texte traduit à la même position. Ce module fonctionne en complémentarité avec `extractor_agent.py` : alors que ce dernier extrait et structure le contenu des documents sources, `generator_agent.py` prend ce contenu structuré (y compris les métadonnées de mise en forme) pour produire un document final fidèle à l'original, mais dans la langue cible.

4.2.3 translator_agent.py

Ce module implémente la classe `TranslatorAgent`, qui opère la traduction de paragraphes structurés en s'appuyant sur un modèle de langage (LLM). On détaille plus particulièrement ce fichier puisqu'il est central.

1. Initialisation et chargement des ressources

- `_load_faiss_index()` lit l'index FAISS (créé par `extraction.py`, cf. plus bas) pour permettre la recherche rapide de termes similaires
- `_load_glossary_metadata()` charge la liste des paires (terme original, terme traduit) associées à l'index.
- SentenceTransformer est chargé pour transformer les textes en vecteurs numériques

2. Recherche de termes du glossaire

- `_get_relevant_glossary_terms(source_text, max_terms=10)` :
  - Encode le texte source en embedding.
  - Recherche les `max_terms` termes les plus proches dans l'index FAISS.

— Retourne les paires (original, traduit) triées par pertinence (distance dans l'espace des embeddings).

3. Traduction des paragraphes

— `translate(paragraphs, ...)` : Pour chaque paragraphe :
  — Découpe le texte en blocs de taille maximale (`max_chunk_words`) pour éviter de dépasser les limites du modèle.
  — Pour chaque bloc :
    — Récupère le contexte : derniers mots du bloc précédent, premiers mots du bloc suivant
    — Recherche les termes du glossaire pertinents pour ce bloc.
    — Construit le prompt qui inclut le contexte, les instructions pour préserver la structure, et le glossaire.
    — Envoie ce prompt au modèle de langage pour obtenir la traduction du bloc.
  — Reconstitue le paragraphe traduit en concaténant les blocs

4.2.4 evaluation_agent.py

Le fichier `evaluation_agent.py` contient l'agent d'évaluation des traductions. Il utilise le modèle COMET pour calculer automatiquement des scores de qualité de traduction à partir de triplets : texte source, traduction de notre programme et traduction de référence. L'agent prend en charge le chargement du modèle, l'évaluation d'une liste de triplets, l'association des scores aux fichiers correspondants et la sauvegarde des résultats dans un fichier CSV. Ce module permet d'objectiver la qualité des traductions produites par le système.

4.3 mmat_langgraph_version.py

Le fichier `mmat_langgraph_version.py` est le fichier qui met bout à bout les agents pour constituer la pipeline à l'aide de la librairie LangGraph. Grâce à LangGraph, la pipeline peut être facilement modifié ou adapté (par exemple si l'on souhaite évaluer la traduction ou non). On propose aussi dans `main()` quelques exemples d'utilisation.

4.4 Parsing du glossaire pour le RAG

Le fichier `extraction.py` sert à extraire automatiquement les termes d'un glossaire bilingue à partir d'un PDF et à créer des bases de données FAISS pour permettre une recherche rapide de correspondances de termes.

1. La fonction `extract_glossary` parcourt les pages du PDF spécifiées, identifie les termes originaux (souvent en gras) et leurs traductions, nettoie les indications grammaticales, puis construit une liste de paires (terme original, terme traduit)
2. Pour chaque terme original extrait, le script utilise la librairie SentenceTransformer pour générer un vecteur d'embedding, c'est-à-dire une représentation numérique du terme
3. Les embeddings sont ajoutés à un index FAISS (IndexFlatL2), qui permet ensuite de retrouver rapidement le terme le plus proche lors d'une recherche par similarité

4.5 OCR

La partie OCR du fichier `extractor_agent.py` est assurée par la fonction `ocrize_pdf`. Cette fonction prend en entrée le chemin d'un fichier PDF et, pour chaque page, convertit la page en image, applique l'OCR avec Tesseract pour extraire le texte, puis reconstitue un nouveau PDF contenant le texte reconnu. Le PDF OCRisé est enregistré dans le même dossier

que l'original, avec le suffixe `_ocr.pdf`. Cette étape permet de rendre le contenu textuel accessible à l'extraction, il suffit donc après d'utiliser la pipeline habituelle.