EXTRACTION INTELLIGENTE ET GÉNÉRATION DE CONNAISSANCES

ARCHITECTURE RAG POUR L'ANALYSE DES RAPPORTS SFCR

EQUIPE 10

Introduction

Dans le domaine du NLP, l'extraction d'informations pertinentes à partir de documents financiers au format PDF pose deux principaux défis : l’élimination des éléments indésirables (en-têtes, pieds de page) issus des OCR et la structuration des contenus complexes comme les tableaux et graphiques pour une analyse efficace."

Pour ce projet, il sera question pour nous de récupérer et traiter les données pertinentes des rapports (à l’aide de l’OCR) et de construire une architecture RAG pour répondre à des questions prédéfinies.

RÉCUPÉRATION ET STOCKAGE DES INFORMATIONS UTILISES DES RAPPORTS SFCR

ÉTAPE 1 : ACQUISITION ET CONVERSION DES RAPPORTS

Les rapports SFCR au format PDF sont analysés à l'aide de l'API Google Cloud Vision.

- L'OCR extrait le contenu brut des documents en générant des fichiers JSON, incluant :
  - Les blocs textuels,
  - Les informations sur la taille, la position des caractères, et la mise en page.

ÉTAPE 2 : TRANSFORMATION EN TABLEAU STRUCTURÉ

La fonction produce_brut() (fournie dans le fichier helper.py) est utilisée pour traiter ces fichiers JSON. Cette fonction :

- Extrait les blocs textuels et leurs informations associées (positions, tailles, etc.),
- Génère un tableau structuré (au format Excel) contenant des colonnes décrivant chaque bloc (texte, position, taille, etc.).

ÉTAPE 3 : TRANSFORMATION EN TABLEAU STRUCTURÉ(CONTRIBUTION DE L'ÉQUIPE)

La fonction produce_brut() (fournie dans le fichier helper.py) est utilisée pour traiter ces fichiers JSON. Cette fonction :

- Extrait les blocs textuels et leurs informations associées (positions, tailles, etc.),
- Génère un tableau structuré (au format Excel) contenant des colonnes décrivant chaque bloc (texte, position, taille, etc.).

Nous avons du ajouter la fonction " **Labeliser" pour catégoriser les informations dans les classes inutiles, utiles, paragraphe et titre et ensuite les intégrer les dans un dataframe, afin de l'utiliser avec le RAG.**

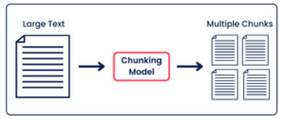
```
def labeliser(data):
    pages = data['num_page'].unique()  #nombre de pages
    labels = []
    for i in pages:
        df = data.loc[data['num_page'] == i]
        char_size = pd.to_numeric(df['char_size'], errors='coerce')
        mean_char_size = char_size.mean(skipna=True)
        #df = df.sort_values(by='pos_y').reset_index(drop=True)
        for j in range(df.shape[0]):
            ligne = df.iloc[j]
            if ligne['text'] == ' ' or ligne['text']=='':
              label = 'inutile'
            else :
              if ligne['pos_y'] < 0.0986 or ligne['pos_y'] > 0.92:
                  label = 'inutile'
              else :
                if ligne['area']<= 0.001 :
                    label = 'inutile'
                else :
                  if ligne['area']<=0.0012:
                    for ij in range(df.shape[0]):
                      if ij != j:
                        ligne2 = df.iloc[ij]
                        if abs(ligne2['pos_y'] -ligne['pos_y'])<= 0.008 :
                            label = 'inutile'
                            break
                  else :
                    if is_mostly_numeric(ligne['text']):
                      label = 'inutile'
                    else :
                      if '=' in ligne['text'] or ('/' in ligne['text'] and '(' in ligne['text']):
                        label = 'inutile'
                      else:
                        if ligne['width']<0.212 and ligne['height']<0.025 and ligne['pos_x']> 0.329  :
                          label = 'inutile'
                        else :
                          if ligne['width']<0.094 and ligne['height']<0.05 :
                            if ligne['chars']<70 :
                              l=0
                              label='utile'
                              for ij in range(df.shape[0]):
                                if ij != j:
                                  ligne2 = df.iloc[ij]
                                  if abs(ligne2['pos_y'] -ligne['pos_y'])<= 0.018 :
                                      l+=1
                                      break
                              if l>0 :
                                label = 'inutile'
                            else :
                              label = 'utile'
                          elif ligne['width']< 0.21 and ligne['height']<0.05  and 0.48<ligne['pos_x']<0.52 :
                            label = 'inutile'
                          else :
                            label = 'utile'
            if label=='utile' :
              if ligne['char_size']>=mean_char_size :
                if ligne['chars']<70 :
                  if ligne['height']<0.018 :
                    label = 'titre'
                  elif ligne['text'][0].isdigit() :
                    label = 'titre'
```

assureur	label
axa-output-1-to-71	inutile
axa-output-1-to-71	paragraphe
axa-output-1-to-71	inutile
axa-output-1-to-71	inutile
axa-output-1-to-71	inutile
...	...
axa-output-1-to-71	inutile
axa-output-1-to-71	inutile
axa-output-1-to-71	inutile
axa-output-1-to-71	inutile
axa-output-1-to-71	inutile

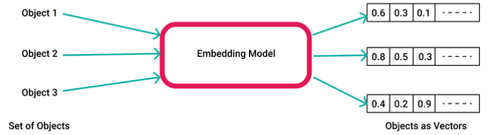
ARCHITECTURE RAG

L’architecture RAG (Retrieval-Augmented Generation) est un modèle d’IA qui combine :

Après avoir classifier les bonnes informations avec la methode “labeliser”, on les stock dans un dataframe.



La segmentation en chunks se fait sur labase des paragraphes jugé utiles avec la fonction “Chunk_report” qu’on a construit., et à partir du Dataframe.



L’embedding permet de la vectorisation des chunks

MISTRAL AI_

Le choix de Mistral comme modèle LLM pour notre projet repose sur sa Performance, son adaptabilité et sa grande capacité de contextualisation , idéal pour notre projet.

Lorsqu’une question est posée, les chunks pertinents sont récupérés et utilisés par le LLM pour produire une réponse précise, assurant une analyse efficace des rapports complexes.